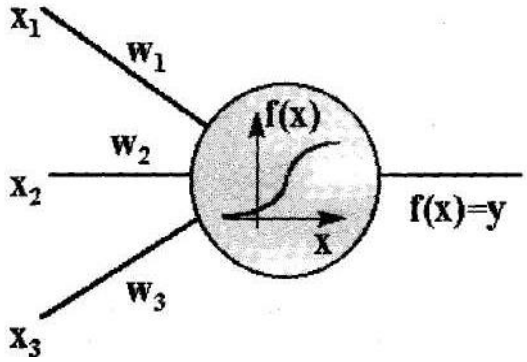
А.Ю.Кононюк

# НЕЙРОННІ МЕРЕЖІ

# І

# ГЕНЕТИЧНІ АЛГОРИТМИ




Київ

«Корнійчук»

2008

**УДК 51(075.8)**
**ББК 30.17**
**К65**

Рецензент: *М.К.Печурін* – д-р техн. наук, проф.
(Національний авіаційний університет).

**Кононюк А.Ю.**
**К65 Нейроні мережі і генетичні алгоритми – К.:«Корнійчук», 2008. – 446 с.**
**ISBN 978-966-7599-50-**

Розглянуті питання розробки структур одно і багатошарових нейроних мереж і алгоритми їх навчання. Пиведені основи побудови та реалізації генетичних алгоритмів.

Наведені приклади і задачі можуть бути використані на практичних занаттях, в курсовому і дипломному проектувані.

Книга виявиться дуже корисною для фахівців різних спеціальностей, економістам, фізикам, мамематикам і спеціалістам в галузі інформатики, а також студентам цих і суміжних з ними галузей знань.

**ББК 30.17**

**ISBN 978-966-7599-50-** 

**Зміст**

# Передмова

Проблематика нейроних мереж і генетичних алгоритмів, і особливо комбінації цих методів - це одна з областей досліджень, що найбільше інтенсивно розвиваються в даний час, яка одержала назва *«Обчислювальні технології».* Її можна вважати сучасним відгалуженням інформатики *(Computer Science),* зв'язаним з методами штучного інтелекту *(Artificial Intelligence),* хоча і принципово відмінним від класичного підходу, застосовуваного адептами цього напрямку.

Дисципліна *«Обчислювальні технології»* досліджує інтелектуальні методи розв'язання різних задач і знаходить застосування в різних областях, у тому числі в промисловості, економіці, медицині й ін. Нейроні мережі, генетичні алгоритми доповнюють класичні експертні системи, що вважаються одним з головних напрямків додатка штучного інтелекту, а також у деяких випадках виконують функції цих систем шляхом реалізації так званих інтелектуальних обчислювальних систем. Останні являють собою об'єднання нейроних мереж і генетичних алгоритмів, взаємодія яких дозволяє розв'язувати різні задачі, але найважливіше - вони стають універсальним інструментом для обробки інформації. Та сама система може застосовуватися для розв'язання різних задач, що вважається безсумнівним достоїнством у порівнянні з класичними експертними системами, орієнтованими, як правило, на досить вузьку проблему, наприклад, на медичну діагностику конкретного захворювання. Інтелектуальні обчислювальні системи можна легко «перепрограмувати» на розв'язання іншої задачі, причому роль такого програмування грає навчання. Таким чином, ці системи мають здатність до навчання, що вважається головним атрибутом інтелектуальності.

Теоретичні основи методів, застосовуваних у сфері *«Обчислювальні технології»,* закладені дослідженнями, проведеними в області *«Soft Computing».* Ця галузь інформатики займається так називаними м'якими обчисленнями, що характеризуються неповнотою даних і відсутністю точності. До них відносяться нечіткі системи, деякі розділи теорії ймовірностей, а також нейроні мережі і генетичні алгоритми.

По тематиці Soft Computing публікуються численні роботи, присвячені генетичним алгоритмам і нейроним мережам, їхнім гібридам і практичним додаткам. Проводяться різні конференції, на яких представляються новітні результати досліджень методів і інтелектуальних технологій.

Книга «Нейроні мережі і генетичні алгоритми», як ми вважаємо, займе належне місце серед наукової і навчальної літератури в галузі новітніх обчислювальних тухнологій і їхніх об'єднань. Книга виявиться особливо корисною магістрам різних спеціальностей, економістам, фізикам, математикам і спеціалістам з інформатики , а також студентам цих і суміжних областей знання.

# 1. Введення в аналітичні інформаційні технології

## 1.1. Основні поняття і означення

Обчислювальні комп'ютерні системи (ОКС) знаходять усе більше застосування в техніку, економіці, соціальних системах і є найважливішим компонентом у функціонуванні виробництва, різного профілю державних і громадських організаціях, установах охорони здоров'я й ін. і є частиною багатьох освітніх програм навчання. ОКС допомагають людству враховувати у своїй діяльності непередбачені прояви навколишнього середовища. Ефективність ОКС залежить від можливостей доступу, обробки й аналізу інформації. Передача даних по комп'ютерним мережам надають велику кількість інформації. Для ефективного співробітництва з користувачем, ОКС повинні мати зачатки людського інтелекту, щоб кваліфіковано зберігати й обробляти великі обсяги інформації, використовуючи аналоги природних засобів комунікацій. Дослідження розробників нових інтелектуальних інформаційних систем (ІІС) спрямовані на розробку методів і засобів об'єднання функцій людського інтелекту і ОКС. Для рішення задач аналізу і переробки великих обсягів інформації створені і стрімко розвиваються так називані аналітичні інформаційні технології. Аналітичні інформаційні технології (АІТ) - це процеси, що на основі побудованих моделей, алгоритмів, математичних виразів дозволяють по відомим даним визначити значення невідомих характеристик і параметрів. Найпростішим прикладом АІТ служить теорема Піфагора, що дозволяє визначити значення довжини гіпотенузи за відомим значенням довжини катетів по відповідній формулі

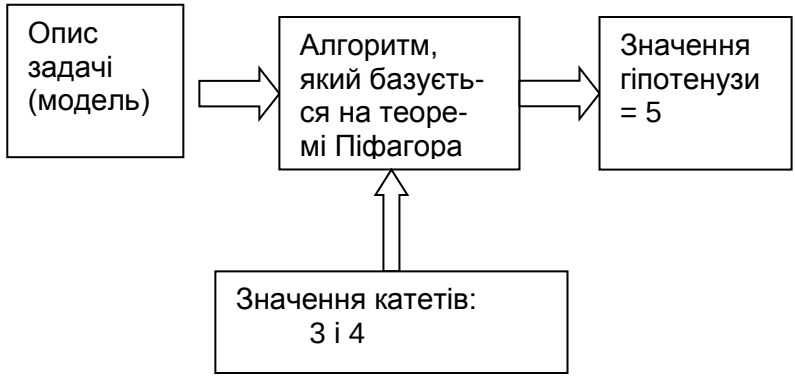
$$c^2=a^2+b^2.$$

Іншим прикладом АІТ можна назвати алгоритм обробки інформації людським мозком. Навіть мозок дитини може розв’язувати задачі, непідвласні сучасним комп'ютерам, наприклад, розпізнавання знайомих осіб у юрбі або ефективне

керування декількома десятками м'язів при грі у футбол. Унікальністю мозку є здібності до розв'язання нових задач - гри в шахи, водінню автомобіля і т.д. Але при цьому, мозок погано пристосований до обробки великих обсягів числової інформації - людина не може перемножити два багатозначних числа, не використовуючи калькулятора або алгоритму обчислення в стовпчик. Реальні задачі з числами набагато складніше, ніж множення і людині для розв'язання таких задач необхідні додаткові методики і засоби. АІТ потрібні в першу чергу людям, що приймають рішення - керівникам, аналітикам, експертам, консультантам. Ефективність функціонування всіх сфер людської діяльності в більшому степені визначається якістю цих рішень - точністю прогнозів, оптимальністю обраних стратегій. І від якості цих рішень залежить ефективність функціонування різних організацій. Використання АІТ допомагає вирішувати проблеми прогнозування (курсів валют, цін на сировину, попиту, доходу компанії, рівня безробіття, числа страхових випадків) і оптимізації (розкладів, маршрутів, плану закупівель, плану інвестицій, стратегії розвитку). Для багатьох задач бізнесу і виробництва не існує чітких алгоритмів їхнього розв'язання, тому керівникам і експертам приходиться розв'язувати ці задачі керуючись своїм досвідом і інтуїцією. Часто існуючі методики виявляються малоефективними для розв'язання багатьох практичних задач, оскільки неможливо точно описати, що стоїть перед особою яка приймає рішення, задачу за допомогою невеликого числа параметрів моделі, або розрахунок моделі займає занадто багато часу й обчислювальних ресурсів. АІТ дозволяють створювати моделі, що істотно підвищують ефективність прийнятих рішень. Розрізняють АІТ з детермінованими і стохастичними (ймовірностними) процесами.

**Детерміновані АІТ.** Аналітичні інформаційні технології типу теореми Пифагора використовуються людством уже багато сторіч. За цей час була створена величезна кількість формул, теорем і алгоритмів для розв'язання різного класу математичних задач - визначення об'ємів, розв'язання систем лінійних рівнянь, пошуку коренів многочленів. Розроблено складні й ефективні методи для розв'язання задач оптимального керування, розв'язання диференціальних рівнянь і т.д. Усі ці методи діють по одній і тій же схемі.


Для застосування алгоритму необхідно, щоб поставлена задача цілком описувалася визначеною детермінованною моделю (деяким набором відомих функцій і значень їхніх параметрів). У такому випадку алгоритм дає точну відповідь. Наприклад, для застосування теореми Пифагора потрібно перевірити, що трикутник – прямокутний:



**Ймовірнісні технології АІТ.** У виробничій і практичної діяльності перед особою, яка приймає рішення часто зустрічаються задачі, зв'язані з використанням випадкових величин - наприклад, задача прогнозування курсу акцій. Для подібних задач важко будувати детермінованні моделі, а отже, і детермінованні технологічні процеси, тому необхідно застосовувати принципово інший, ймовірнисний підхід, при розв'язанні виникаючих задач. Параметри ймовірнисних моделей - це розподіл випадкових величин, їхні середні значення, дисперсії, моменти і т.д. Як правило, ці параметри заздалегідь невідомі, а для їхньої оцінки використовуються статистичні методи, застосовувані до вибірок зафіксованих значень даних (історичні дані).


Такого роду методи припускають, що відома деяка ймовірнісна модель задачі. Наприклад, у задачі прогнозування курсу акцій можна припустити, що завтрашній курс акцій залежить тільки від їхнього курсу за останні 2 дні (авторегресивна модель). Якщо це вірно, то спостереження курсу акцій протягом декількох місяців дозволяє з великою ймовірністю оцінити значення коефіцієнтів цієї залежності і прогнозувати курс акцій у майбутньому.

В останні 10 років відбувається бурхливий розвиток аналітичних інформаційних систем нового типу. У їхній основі лежать технології штучного інтелекту, що імітують природні процеси, наприклад, діяльність нейронів мозку або процес природного добору особей. При розробці АІТ враховуються їхні здібності:

- розуміти задачі, загальні процеси і знання можливостей інших систем і людей, що приймають участь у взаємодії з АІТ;
- налагоджувати зв'язки з користувачами на основі розуміння природної мови, графічних зображень, знаків і ін.;
- використовувати знання, засновані на логічному мисленні;
- координувати прийняття рішень, планування і керування;
- навчатися з використанням попереднього досвіду й адаптувати своє поводження до навколишнього середовища.

Розуміння цих можливостей у людях і втілення їх при розробці програм є центральним у створеннях новітніх АІТ, здатних здобувати і використовувати знання. Від зростання потужностей для проведення інформаційного аналізу, ухвалення рішення, гнучкого проектування і виробництва залежить національна конкурентноздатність. При додаванні інтелекту до комп'ютерних систем усуваються багато обмежень у розв’язанні реальних задач.

## 1.2. Data Mining - інтелектуальний аналізатор даних.

Комп'ютерні інформаційні технології з використанням штучного інтелекту для виконання процесів числень переживають свій розквіт. В даний час відбувається стрімке зростання числа програмних продуктів, що використовують нові технології, а також типів задач, де їхнє застосування дає значний ефект, у тому числі й економічний. Процеси автоматичної обробки й аналізу даних, що називають Data Mining (видобуток знань) стають невід'ємною частиною концепції електронних сховищ даних і організації обчислень з використанням штучного

інтелекту. Існуючий доступ користувача до сховища даних забезпечує тільки одержання відповідей на задані питання, у той час як технологія data mining дозволяє побачити ("знайти") сховані правила і закономірності в наборах даних, що користувач не може передбачити, і знання яких може сприяти збільшенню ефективності діяльності підприємства. Data Mining переводиться як "видобуток" або "розкопка даних". Часто, поряд з терміном Data Mining застосовують вираз "інтелектуальний аналіз даних". Як відомо, людський розум не пристосований для сприйняття великих масивів різнорідної інформації. Людина, як правило, не здатна уловлювати більш двох-трьох взаємозв'язків навіть у невеликих вибірках. Сучасна математична статистика, що тривалий час претендувала на роль основного інструмента аналізу даних, також не завжди підходить при розв’язанні задач з реального різноманітного життя. Вона оперує усередненими характеристиками вибірки, що часто виявляються фіктивними величинами (середній ріст юнаків у школі, середня вага жінок на підприємстві і т.п.). Тому методи математичної статистики виявляються корисними, головним чином, для перевірки заздалегідь сформульованих гіпотез.

У процесі своєї діяльності підприємства й організації збирають величезні масиви даних з яких вони хочуть одержати корисну інформацію. Як можна довідатися з даних про те, що є більш істотним для організації клієнтів, як розмістити ресурси ефективним чином або як мінімізувати втрати? Для розв’язання цих проблем створюються сучасні технології інтелектуального аналізу, що використовуються для знаходження моделей ситуацій і відношень, схованих у базі даних - моделей, що не можуть бути побудовані існуючими методами. Модель ситуації, як і карта місцевості - це абстрактне представлення реальності. На карті може бути зазначено шлях від аеропорту до будинку, але на ній важко в рельном режимі часу показати аварію, що створила пробку, або ремонтні роботи, що ведуться в даний момент і вимагають об'їзду. Доти поки модель ситуації не відповідає існуючим реально відношенням, неможливо одержати успішний результат подій.

Розрізняють два види моделей ситуації: *прогнозуючі й описові.* Перші використовують один набір даних з відомими результатами для побудови моделей, що явно прогнозують результати для інших наборів, а другі описують залежності в існуючих даних, що у свою чергу використовуються для прийняття рішень або дій (впливів).

Безумовно, компанія, що тривалий час знаходиться на ринку і знаюча своїх клієнтів користується множиною моделей ситуацій. Технології інтелектуального аналізу даних можуть не тільки підтвердити ці емпіричні спостереження, але і знайти нові, невідомі раніше моделі ситуацій. Спочатку це може дати користувачеві незначний ефект (мікроефект), але якщо його об'єднати по кожному товару і кожному клієнту, це приведе до великого сумарного ефекту, що буде набагато більшим стосовно того ефекту, що отриманий без допомоги технології Data Mining. З іншого боку, за допомогою методів Data Mining можна знайти таку модель ситуації, що приведе до радикального поліпшення у фінансовому і ринковому положенні компанії.

Data Mining - це набір технологічних операцій і засобів їхньої реалізації. Вона не виключає необхідності знання аналітиком бізнесу і розуміння даних або аналітичних методів. Ця технологія допомагає аналітикам у знаходжені моделей ситуацій і відношень у даних, але вона не говорить про цінності цих моделей для організації. Кожна модель повинна перевірятися в реальному середовищі. Хоча інструментарій інтелектуального аналізу і звільняє користувача від можливих складностей у застосуванні статистичних методів, він усе-таки жадає від нього розуміння роботи інструментарію й алгоритмів, на яких він базується. Крім цього, технологія знаходження нового знання в базі даних не може дати відповіді на не задані питання. Вона не заміняє аналітиків або менеджерів, а дає їм сучасний, могутній інструмент для поліпшення виконуваної роботи. Сучасні технології інтелектуального аналізу переробляють інформацію з метою автоматичного пошуку прототипів, характерних для яких-небудь фрагментів неоднорідних багатомірних даних. Вага формулювання гіпотез і виявлення незвичайних прототипів переведена з людини на комп'ютер. У постановці задачі Data

Mining важко відшукати щось нове. Фахівці протягом декількох останніх десятків років розв’язували подібні задачі, але тільки зараз виникла можливість їхнього розв’язання в реальному режимі часу, тому що нинішні засоби і методи обробки даних стали доступні і швидкодіючими, а їхні результати зрозумілими навіть для фахівців низької кваліфікації.

Для успішного проведення процесу знаходження нового знання необхідною умовою є наявність сховища даних.

*Сховище даних* - це предметно-орієнтовний, інтегрований, прив'язаний до часу, незмінний набір даних, необхідних для підтримки процесу прийняття рішень.

*Предметна* орієнтація означає, що дані об'єднані в категорії і зберігаються відповідно областям, що вони описують, а не до застосувань, що їх використовують. *Інтегрованність* означає, що дані задовольняють вимогам усього підприємства, а не однієї функції бізнесу. Цим сховище даних гарантує, що однакові звіти, які згенеровані для різних аналітиків, будуть містити однакові результати. *Прив'язка в часі* означає, що сховище можна розглядати як сукупність "історичних" даних: можна відновити картину на будь-який момент часу. Атрибут часу завжди явно є присутнім у структурах сховища даних. *Незмінність* означає, що, потрапивши один раз у сховище, дані там зберігаються і не змінюються. У сховище дані лише додаються. Для організації й експлуатації інформаційного сховища створюється спеціалізоване програмне забезпечення, що забезпечує ефективну взаємодію з користувачем.

Ключовою можливістю застосування новітніх технологій стало значне падіння ціни на пристрої збереження інформації. Це істотно удешевило і збільшило можливості збору і збереження великих обсягів інформації. Падіння цін на процесори з одночасним збільшенням їхньої швидкодії і значним збільшенням обсягів оперативної пам'яті посприяло розвиткові технологій, зв'язаних з обробкою величезних масивів інформації. У результаті цього були переборені значні труднощі, що

зустрічаються при перебуванні нового знання в сховищах інформації.

Клієнт-серверна архітектура також є необхідним атрибутом технології інтелектуального аналізу даних. Такий підхід надає можливості виконувати найбільш трудомісткі процедури обробки даних на високопродуктивному сервері як розроблювачам проектів, так і користувачам. На цьому ж сервері можуть зберігатися, і по запитах клієнтів, виконуватися корпоративні проекти.

Сфера застосування технологій обчислень, які побудовані на використанні штучного интелекта, нічим не обмежена - вона застосовна скрізь, де є які-небудь дані. Але в першу чергу методи Data Mining сьогодні зацікавили комерційні підприємства, що розробляють проекти на основі інформаційних сховищ даних. Досвід багатьох таких підприємств показує, що віддача від використання технологій інтелектуального аналізу даних може досягати 1000%. Технології Data Mining надають велику допомогу для керівників і аналітиків у їхній повсякденній діяльності, оскільки вони можуть одержати відчутні переваги в конкурентній боротьбі.

Коротко охарактеризуємо деякі можливі бізнеси-застосування технологій інтелектуального аналізу даних і обчислень.

Застосування інтелектуальних інформаційних технологій поширено в широкому спектрі індустрій. Методи Data Mining поширені в багатьох організаціях, оскільки вони сприяють збільшенню доходів. Ці методи можуть використовуватися для керування взаєминами з клієнтами. Визначаючи характеристики клієнтів, що можуть звернутися до конкурентів, компанія може починати дії для їхнього утримання, тому що зберегти клієнта завжди дешевше, ніж придбати нового.

- Маркетинг даних - визначення за допомогою методів Data Mining кола кандидатів для розсилання цільової реклами,

що дозволяє збільшити продаж, зменшивши при цьому витрати на проведення такої реклами.

- Телекомунікаційні компанії, страхові компанії і фондові біржі застосовують ці технології для визначення втрат клієнтів.
- Компанії, що діють на фінансовому ринку, визначають ринкові і галузеві характеристики для передбачення індивідуальних і фондових переваг у найближчому майбутньому.
- Медицина - визначає ефективність застосування медикаментів, хірургічних процедур і медичних тестів.
- Фармацевтичні фірми використовують сховища даних по хімічних сполуках для перебудови комбінацій з'єднань, що надалі можна буде використовувати як ліки для лікування різних захворювань.
- Супермаркети визначають, які продукти продавати і як їх розташовувати усередині магазина для досягнення більшої кількості продажів.

Ключем до успішного застосування методів інтелектуальних обчислень служить не просто вибір алгоритму, а майстерність фахівця, що створює модель ситуації і можливості програми, що моделює процес. Існують дві сторони успіху в пошуку даних. По-перше - чітке і ясне формулювання задачі, що підлягає рішенню. По-друге - використання коректних даних. Після вибору даних із усіх доступних джерел (або одержання даних із зовнішніх джерел) необхідно їх перетворити або згрупувати у визначеному порядку. Чим більше аналітик може "грати" з даними, будувати різні моделі, оцінювати результати, тим вище може бути результат. Робота з даними стає ефективною при інтеграції наступних компонентів: візуалізації, графичного інструментарію, засобів формування запитів, оперативної аналітичної обробки, що дозволяє зрозуміти дані й інтерпретувати результати.

# 1.3. Основні моделі ситуацій і методи технологій обчислень, з використанням штучного интелекта

## Методи знаходження знань в інформаційному сховищі

Розглянемо основні види методів, що використовуються для знаходження нового знання на основі даних інформаційного сховища. Метою інтелектуальних інформаційних технологій є знаходження нового знання, що користувач може надалі застосувати для поліпшення результатів своєї діяльності. Результат моделювання - це виявлення типу відношень у даних.

Можна визначити шість методів виявлення й аналізу знань:

- класифікація,
- регресія,
- прогнозування тимчасових послідовностей (рядів),
- кластерізація,
- асоціація,
- послідовність.

Перші три методи використовуються головним чином для прогнозування , у той час як останні зручні для опису існуючих закономірностей у даних.

*Класифікація* - найпоширеніша модель інтелектуального аналізу даних. З її допомогою виявляються ознаки, що характеризують групу, до якої належить той або інший об'єкт. Це робиться за допомогою аналізу вже класифікованих об'єктів і формулювання деякого набору правил.

У багатьох видах бізнесу проблемою є втрата сталих клієнтів. Класифікація допомагає знайти характеристики "хитливих" покупців і створити модель ситуації, що прогнозує, при якій покупець схильний піти до іншого продавця. Використовуючи модель, можна сформувати ефективні види

знижок, а також інші привабливі пропозиції по обслуговуванню покупців, що будуть робити сприятливі впливи на покупців і утримувати їх від переходу до іншого продавця. Один раз визначений ефективний класифікатор використовується для класифікації нових записів у базі даних у вже існуючі класи й у цьому випадку він здобуває характер прогнозу. *Наприклад, класифікатор, що вміє ідентифікувати ризик віддачі позики, може бути використаний для ухвалення рішення, чи великий ризик надання позики визначеному клієнтові*. Тобто, класифікатор використовується для прогнозування ймовірності повернення позики.

*Регресійний* аналіз використовується, коли *відношення між змінними* можуть бути виражені кількісно у виді деякої комбінації цих змінних. Отримана комбінація використовується для прогнозування чисельного значення, що може приймати цільова (залежна) змінна, що обчислюється на заданому наборі значень вхідних (незалежних) змінних. У найпростішому випадку, для цього використовуються стандартні статистичні методи, такі як лінійна регресія, але більшість реальних моделей ситуації не укладаються в її рамки. Наприклад, розміри продажів або фондових цін складні для прогнозування, так як можуть залежати від комплексу взаємин змінних.

*Прогнозування тимчасових послідовностей*. Основою для будь-яких систем прогнозування служить історична інформація, збережена в інформаційних сховищах у виді тимчасових рядів. Якщо можна побудувати математичну модель і знайти прототипи, що адекватно відображають цю динаміку, є ймовірність, що з їх допомогою можна прогнозувати поводження системи в майбутньому. Прогнозування тимчасових послідовностей дозволяє на основі аналізу поводження тимчасових рядів оцінювати майбутні значення прогнозованих змінних. Ці моделі повинні містити в собі особливі ознаки часу: ієрархія періодів (місяць-квартал-рік), особливі відрізки часу (п'яти- шести- або семиденний робочий тиждень), сезонність, свята й ін.

*Кластерізація* відрізняється від класифікації тим, що класи заздалегідь не задані і за допомогою моделі кластерізації засобу інтелектуальних обчислень самостійно створюють однорідні групи даних.

*Асоціація* сприяє здійснювати аналіз структур і застосовується, коли кілька подій зв'язані між собою. Класичний приклад аналізу структури покупок відноситься до представлення придбання якої-небудь кількості товарів як одиночної економічної операції (транзакції). Оскільки велика кількість покупок здійснюється в супермаркетах, а покупці для зручності використовують кошики, куди і складають товар, що покупається, то для пошуку асоціацій служать результати аналіз вмісту кошика. Метою використання даного підходу є пошук трендів (однакових ділянок) серед великого числа транзакцій, які можна використовувати для пояснення поводження покупців. Така інформація може бути використана для регулювання запасів, зміни розміщення товарів на території магазина й ухвалення рішення по проведенню рекламної кампанії для збільшення продажів або для просування визначеного виду продукції. Наприклад, дослідження, проведене в супермаркеті, може показати, що 65% покупців картопляних чіпсів, беруть також і "кока-колу", а при наявності знижки за такий комплект "колу" беруть у 85% випадків. Маючі такі дані, менеджерам легко оцінити, наскільки ефективна надана знижка. Хоча цей підхід прийшов з роздрібної торгівлі, він може також застосовуватися у фінансовій сфері для аналізу портфеля цінних паперів і пошуку наборів фінансових послуг, що клієнти часто беруть разом. Це може використовуватися для створення деякого набору послуг, як частини кампанії по стимулюванню продажів.

*Послідовність* має місце, якщо існує ланцюжок зв'язаних у часі подій. Традиційний аналіз структури покупок має справу з набором товарів, що представляють одну транзакцію. Варіант такого аналізу зустрічається, якщо існує додаткова інформація (номер кредитної карти клієнта або номер його банківського рахунку) для зв'язування різних покупок у єдину тимчасову серію. У такій ситуації важливо не тільки співіснування даних усередині однієї транзакції, але і порядок, у якому ці дані

з'являються в різних транзакціях і час між цими транзакціями. Правила, що встановлюють ці відношення, можуть бути використані для визначення типового набору попередніх продажів, що можуть повести за собою наступні продажі визначеного товару. Установлено, що після покупки будинку в 45% випадків протягом місяця купується і нова кухонна плита, а в продовж наступних двох тижнів 60% новоселів обзаводяться холодильником.

## Методи (алгоритми) інтелектуальних обчислень

- нейроні мережі;
- дерева рішень;
- системи розмірковувань на основі аналогічних випадків;
- алгоритми визначення асоціацій і послідовностей;
- нечітка логіка;
- генетичні алгоритми;
- еволюційне програмування;
- візуалізація даних.

*Нейроні мережі* це системи, які мають таку архітектуру, використовування якої дозволяє умовно імітувати роботу нейронів. Математична модель нейрона являє собою деякий універсальний нелінійний елемент із можливістю широкої зміни і настроювання його характеристик (параметрів). Нейроні мережі являють собою сукупність зв'язаних між собою шарів нейронів, що одержують вхідні дані, здійснюють їх обробку і генерують результат, який видають на виході. Між вузлами видимих вхідного і вихідного шарів може знаходитися визначене число схованих шарів. Нейроні мережі реалізують непрозорий процес. Це означає, що побудована модель, як правило, не має чіткої інтерпретації. Багато пакетів комп'ютерних програм, що реалізують алгоритми нейроних мереж, застосовуються в сфері обробки комерційної інформації, при розпізнаванні образів, розшифровки рукописного тексту, інтерпретації кардіограм.

Апаратні і програмні засоби реалізації алгоритмів нейромереж називають нейрокомп'ютером.

- Нейрокомп'ютери забезпечують використання існуючих методів для розв'язання багатьох оригінальних задач. І неважливо, що спеціалізована обчислювальна машина дозволяє краще вирішувати один клас задач. Важливіше, що один нейрокомп'ютер розв'яже і цю задачу, і іншу, і третю і не треба щораз проектувати і створювати спеціалізовану ЕОМ, нейрокомп'ютер зробить усе сам і навіть не гірше.
- Замість програмування - навчання. Нейрокомп'ютер учиться, потрібно лише сформувати навчальні множини. Робота програміста заміняється новою роботою вчителя. Краще це чи гірше? Не те, не інше. Програміст указує машині всі послідовності виконання роботи, а вчитель створює "навчальне середовище", до якого пристосовується нейрокомп'ютер. З'являються нові можливості для роботи.
- Нейрокомп'ютери ефективні там, де потрібний аналог людського способу мислення (інтуїції), зокрема, для розпізнавання образів, читання рукописних текстів, підготовки аналітичних прогнозів, перекладу з однієї мови на іншу і т.п. Саме для таких задач звичайно важко скласти явний алгоритм.
- Нейроні мережі дозволяють створити ефективне програмне і математичне забезпечення для комп'ютерів з високим ступенем распараллелювання обробки (обчислень).
- Нейрокомп'ютеры "демократичні", вони прості, як текстові процесори, тому з ними може працювати низькокваліфікованний користувач.

*Дерева рішень* – цей метод широко застосовується в різних областях виробництва, фінансів і бізнесу, де часто зустрічаються задачі числового прогнозу. У результаті застосування цього методу, для навчальної вибірки даних створюється ієрархічна структура правил класифікації типу, "ЯКЩО... ТОДІ...", що мають вид дерева. Для того щоб вирішити, до якого класу віднести деякий об'єкт або ситуацію, треба відповісти на запитання, що стоїть у вузлах цього дерева, починаючи з його кореня. Питання можуть мати вигляд "Значення параметра А більше Х ?" або вид "Значення змінної В належить підмножині ознак С ? ". Якщо

відповідь позитивна, перехід до правого вузла наступного рівня, якщо негативна - то до лівого вузла; потім знову здійснюється процедура відповіді на питання, яке зв'язане з відповідним вузлом. Таким чином, зрештою, можна дійти до одного з кінцевих вузлів, де визначений клас об'єкта. Цей метод гарний наочним представленням правил і його легко зрозуміти. В даний час збільшується інтересу до задач, що використовують для свого рішення дерева рішень. В основному це зв’язано з тим, що більшість виробничих і економічних проблем вирішується ними швидше, ніж алгоритмами нейронних мереж, вони простіші і зрозуміліші для користувачів. У той же час не можна сказати, що дерева рішень завжди діють безвідмовно: для визначених типів даних вони можуть виявитися неприйнятними. Справа в тім, що окремим вузлам на кожній галузі приділяється менше число записів даних - дерево може сегментувати дані на велику кількість окремих випадків. Чим більше таких окремих випадків, тим менше навчальних прикладів попадає в кожен такий окремий випадок, і їхня класифікація стає менш надійною. Якщо дерево занадто "гіллясте" - складається з великого числа дрібних галузей - воно не буде давати статистично обґрунтованих відповідей. Як показує практика, у більшості систем, що використовують дерева рішень, ця проблема не знаходить задовільного рішення.

*Системи розмірковувань на основі аналогічних випадків*. Ідея алгоритму досить проста. Для того, щоб зробити прогноз на майбутнє або вибрати правильне рішення, системи знаходять у минулому близькі аналоги наявної ситуації і вибирають ту ж відповідь, що і була для них правильною. Тому цей метод ще називають методом "найближчого сусіда". Системи розмірковувань на основі аналогічних випадків дають гарні результати в різних задачах. Головний їхній недолік полягає в тому, що вони не створюють яких-небудь моделей або правил, що узагальнюють попередній досвід, - у виборі рішення вони базуються на всьому масиві доступних історичних даних, тому неможливо сказати, на основі яких конкретно факторів ці системи будують свої відповіді.

*Алгоритми визначення асоціацій* знаходять правила про окремі предмети, що з'являються разом в одній транзакції, наприклад, в одній покупці.

*Послідовність* - це теж асоціація, але залежна від часу. Асоціація записується як А>Б, де А називається передумовою, Б - наслідком. Частота появи кожного окремого предмета або групи предметів, визначається в такий спосіб - підраховується кількість появ цього предмета у всіх подіях (покупках) і ділиться на загальну кількість подій. Ця величина вимірюється у відсотках і зветься "поширеність". Низький рівень поширеності (менш одного тисячної відсотка) говорить про неістотність асоціації. Для визначення важливості кожного отриманого асоціативного правила необхідно одержати величину, що зветься "довірчість А до Б" (взаємозв'язок А і Б). Ця величина показує, як часто з появою А з'являється Б і розраховується як відношення частоти появи (поширеності) А і Б разом до поширеності А. Тобто, якщо довірчість А до Б дорівнює 20%, то це означає, що при покупці товару А в кожнім п'ятому випадку купують і товар Б. Якщо поширеність А не дорівнює поширеності Б, то і довірчість А до Б не дорівнює довірчості Б до А. Справді, покупка комп'ютера частіше веде до покупки дискет, чим покупка дискети до покупки комп'ютера. Ще одною важливою характеристикою асоціації є потужність асоціації. Чим більше потужність, тим сильніший вплив, що поява А робить на появу Б. Потужність розраховується по формулі: (довірчість А до Б) / (поширеність Б).

Деякі алгоритми пошуку асоціацій спочатку сортують дані і тільки після цього визначають взаємозв'язок і поширеність. Єдиною розбіжністю таких алгоритмів є швидкість або ефективність перебування асоціацій. Це важливо, у зв'язку з величезною кількістю комбінацій, яку необхідно перебрати для знаходження більш значимих правил. Алгоритми пошуку асоціацій можуть створювати свої бази даних поширеності, довірчості і потужності, до яких можна звертатися при запиті. Наприклад: "Знайти всі асоціації, у яких для товару X довірчість більш 50% і поширеність не менш 2,5%". При знаходженні послідовностей додається змінна часу, що дозволяє працювати

із серією подій для знаходження послідовних асоціацій протягом деякого періоду часу.

Підводячи підсумки цьому методу аналізу, необхідно сказати, що може виникнути така ситуація, коли товари в супермаркеті будуть згруповані за допомогою знайдених моделей, але це, замість очікуваного прибутку, дасть зворотний ефект. Це може відбутися через те, що клієнт не буде довго ходити по магазину в пошуках бажаного товару, купуючи при цьому ще щось, що потрапляється на очі, і те, що він ніколи не планував купувати.

*Нечітка логіка* застосовується для масивів даних, у яких приналежність даних до якої-небудь групи є ймовірністю в інтервалі від 0 до 1. Чітка логіка маніпулює результатами, що можуть бути або істиною, або неправдою. Нечітка логіка застосовується в тих випадках, коли існує "може бути" у доповненні до "так" або "ні".

Областю використання алгоритмів нечіткої логіки є будь-як аналітичні системи, у тому числі :

- нелінійний контроль за процесами (виробництво);
- удосконалення стратегій керування і координації дій, наприклад, складне промислове виробництво;
- самонавчальні системи (або класифікатори);
- дослідження ризикованих і критичних ситуацій;
- розпізнавання образів;
- фінансовий аналіз (ринки цінних паперів);
- дослідження даних (корпоративні сховища).

У Японії цей напрямок переживає бум. Тут функціонує спеціально створена лабораторія Laboratory for International Fuzzy Engineering Research ( LIFE ). Програмою організації є створення більш близьких до людини обчислювальних пристроїв. LIFE поєднує 48 компаній у числі яких знаходяться: Hitachi, Mitsubishi, NEC, Sharp, Sony, Honda, Mazda, Toyota. З

іноземних учасників LIFE можна виділити: IBM, Fuji Xerox, до діяльності LIFE також виявляє інтерес NASA.

Потужність і інтуїтивна простота нечіткої логіки як методології рішення виникаючих проблем гарантує її успішне використання у вбудованих системах контролю й аналізу інформації. При цьому відбувається підключення людської інтуїції і досвіду оператора. На відміну від традиційної математики, що вимагає на кожнім кроці моделювання точних і однозначних формулювань закономірностей, нечітка логіка пропонує зовсім інший рівень мислення, завдяки чому творчий процес моделювання відбувається на високому рівні абстракцій, при якому постулується лише мінімальний набір закономірностей.

*Генетичні алгоритми* є могутнім засобом рішення різних комбінаторних задач і задач оптимізації. Проте, генетичні алгоритми ввійшли зараз у стандартний інструментарій методів інтелектуальних обчислень. Цей метод названий так тому, що в якомусь степені імітує процес природного добору в природі. Нехай нам треба знайти рішення задачі, найбільш оптимальної з погляду деякого критерію, де кожне рішення цілком описується визначеним набором чисел або величин нечислової природи. Наприклад, якщо нам треба вибрати сукупність фіксованого числа параметрів ринку, що істотно впливають на його динаміку, це буде масив імен цих параметрів. Про цей масив можна говорити як про сукупності хромосом, що визначають якості індивіда - даного рішення поставленої задачі. Значення параметрів, що визначають рішення, називаються генами. Пошук оптимального рішення при цьому схожий на еволюцію популяції індивідів, представлених наборами хромосом. В еволюції діють три механізми: по-перше, добір найсильніших - наборів хромосом, яким відповідають найбільш оптимальні рішення; по-друге, схрещування - виробництво нових індивідів за допомогою змішування хромосомних масивів відібраних індивідів; і, по-третє, мутації - випадкові зміни генів у деяких індивідів популяції. У результаті зміни поколінь виробляється рішення поставленої задачі, що уже не може бути далі поліпшено.

*Еволюційне програмування* наймолодша область інтелектуальних обчислень. Гіпотези про вид залежності цільової змінної від інших змінних формулюються системою у виді програм на деякій внутрішній мові програмування. Якщо це універсальна мова, то теоретично на ній можна виразити залежність будь-якого виду. Процес побудови таких програм будується як еволюція у світі програм (цим метод трохи схожий на генетичні алгоритми). Якщо система знаходить програму, що точно виражає шукану залежність, вона починає вносити в неї невеликі модифікації і відбирає серед побудованих у такий спосіб дочірніх програм ті, котрі підвищують точність її функціонування. Система "вирощує" кілька генетичних ліній програм, що конкурують між собою в точності знаходження шуканої залежності. Спеціальний модуль, що транслює, переводить знайдені залежності з внутрішньої мови системи на мову користувача (математичні формули, таблиці й ін.), роблячи їх легкодоступними. Для того, щоб зробити отримані результати більш зрозумілими для користувача математики, існує великий арсенал різноманітних засобів візуалізації виявлених залежностей. Пошук залежності цільових змінних від інших проводиться у формі функцій якого-небудь визначеного виду. Наприклад, в одному з найбільш вдалих алгоритмів цього типу - методі групового обліку аргументів (МГУА) залежність шукають у формі поліномів. Причому складні поліноми заміняються декількома простими, враховуючі лише деякі ознаки (групи аргументів). Зазвичай використовуються попарні об'єднання ознак. Цей метод не має великих переваг у порівнянні з нейроними мережами з готовим набором стандартних нелінійних функцій, але отримані формули залежності, у принципі, піддаються аналізові й інтерпретації.

*Програми візуалізації даних* у визначеному змісті не є засобом аналізу інформації, оскільки вони тільки представляють її користувачеві. Проте, візуальне представлення, скажемо, відразу чотирьох змінних наочно узагальнює величенні обсяги даних.

*Комбіновані методи*. Часто виробники з'єднують зазначені підходи. Об'єднання алгоритмів нейроних мереж і технології

дерев рішень сприяє побудові більш точної моделі і підвищенню швидкодії. Для рішення кожної проблеми варто шукати свій оптимальний метод.

## 1.4. Процес пошуку нового знання

Пошук нового знання - це процес, що складається з декількох операцій, кожна з яких повинна забезпечувати ефективне використання засобів інтелектуальних обчислень. Основними складовими етапами цього процесу є:

1. Визначення проблеми (постановка задачі).
2. Збір і підготовка даних:
   - аналіз даних,
   - групування і коректування даних,
   - добір даних,
   - перетворення даних.
3. Побудова моделі ситуації:
   - оцінка моделі й інтерпретація ситуації,
   - зовнішня перевірка моделі.
4. Використання моделі
5. Спостереження за поводженням моделі

Розглянемо докладно кожну з цих складових:

1. **Визначення проблеми**. Для ефективного використання всіх переваг інтелектуальних інформаційних технологій необхідно ясно представити мета майбутнього аналізу. Змістовна постановка задачі здійснюється в залежності від мети. Якщо необхідно збільшити прибуток торговельної організації, то для цілей: "збільшення кількості продажів" і "збільшення ефективності реклами" необхідно формулювати різні задачі. На цьому ж етапі визначаються способи оцінювання результатів майбутнього проекту і можливі витрати на його реалізацію.

2. **Збір і підготовка даних**. Цей етап самий тривалий з усього процесу пошуку нового знання: він може займати від 50% до 85% часу всього процесу пошуку нового знання. На цьому

етапі необхідно визначити джерела одержання даних. Це можуть бути дані, які накопичені самою організацією або зовнішні дані від доступних джерел (зведення про погоду або перепис населення) або приватних джерел (різні архівні дані, бази нотаріальних контор і ін.). Зібрані дані підлягають аналізові

*а. Аналіз даних*. При побудові моделі ситуації необхідно пам'ятати дуже важливе правило, що оцінює коректність вхідних даних: "Якщо на вхід задачі надходить інформаційне "сміття", то і результатом теж буде "сміття". Вхідні дані можуть знаходитися в одній базі даних або в декількох. Перед "завантаженням" даних у сховище необхідно врахувати, що різні джерела даних можуть бути спроектовані під визначені задачі і, відповідно, виникають проблеми, зв'язані з об'єднанням даних: різні формати представлення числових даних (наприклад, цілі або дійсні); різне кодування даних (наприклад, різний формат дат); різні способи збереження даних; різні одиниці виміру (дюйми і сантиметри); а також частота збору даних і дата останнього відновлення. Навіть, якщо дані знаходяться в одній базі даних, то все рівно треба звертати увагу на пропущені значення величини і значення нереальної величини, так називані "викиди". Аналітик повинний завжди знати, як, де і при яких умовах збираються дані, і бути упевненим, що всі дані, використовувані для проведення аналізу обмірювані однаковим способом. Проаналізовані дані необхідно об'єднати і відкоригувати.

б. *Групування* і коригування *даних*. На цьому етапі створюють сховища даних для подальшої їхньої обробки, здійснюють групування даних і наповняють ними сховища або додають до нього дані, які відібрані на попередніх етапах. У цей же час відбувається їхнє корегування, тобто виправлення усіх виявлених помилок. Існують різні аспекти корегування даних. Усі вони спрямовані на знаходження і виправлення помилок, яких припускалися на етапі збору інформації. Помилкою в даних можуть вважатися:

- пропущене числове значення,
- неможлива подія (невірно набране значення - "викид") і ін.

Корегування здійснюють керуючись здоровим глуздом, використанням правил корегування і/або з залученням експерта, що добре знає предметну область. Запис у базі даних, у якій є помилка, повинна бути виправлена або, у спірних випадках, виключена з подальшого розгляду. Після перевірки й оцінки даних, виконують операції перетворення і форматування даних відповідно результатам їхнього оцінювання. Це робиться для більш ефективного використання і спостереження за даними. Дані дискретних подій перетворюють у спеціально розроблену або стандартну форму, у якій відбиває час і опис подій. Якщо користувачі будуть легко розбиратися в цій формі, вони зможуть швидко вивчити опісані події, що були закладені в основу побудови цієї форми. Може показатися, що цей крок дублює етап збору даних, але насправді це два різних етапи. На першому з них відбувається добір даних для прискорення машинної обробки інформації без утрати якості, на другому - дані приводяться до виду, зручному для візуального контролю користувачем. Фахівець, що виконує аналіз даних, може повніше уявити собі вхідні дані. Це необхідно для різного роду звітів, коли потрібно коротко охарактеризувати вхідні дані, використовувані для їхнього аналізу.

*в*. *Добір даних*. Якщо сховище сформоване і визначені типи моделей ситуацій, що будуть побудовані для рішення поставленої задачі, здійснюють добір даних, що необхідні для використання їхній у цих моделей. Мається на увазі не тільки зменшення кількості записів у базі по визначеній умові, але також і зміна кількості полів, злиття різних таблиць в одну, або, навпаки, створення на основі однієї таблиці декількох. Тобто, перетворення відбувається в "трьох вимірах": по кількості записів, по кількості полів і за структурою.

*г*. *Перетворення даних*. Процес перетворення даних здійснює збагачення отриманої бази даних, а саме: додає необхідні відношення на основі існуючих полів (не просто "ціна" і "кількість", а їхній добуток - "загальна сума", не борг і дохід, а відношення боргу до доходу), додає інтервали (по номеру місяця можна поставити номер кварталу, а відсоток виконання плану

можна доповнити характеристиками "добре", "задовільно"), додає критическе значення (максимум, середнє, мінімум).

**3. Побудова моделі ситуації**. Метод побудови моделі ситуації являє собою ітераційний процес, тобто, виконує створення ряду моделей для знаходження однієї (оптимальної), що найбільшою мірою задовольняє поставленим цілям.

Моделі можна розділити на дві групи, кожна з яких визначає свій тип моделі:

- Контрольовані (моделі класифікації, регресії, прогнозування тимчасових послідовностей);
- Неконтрольовані (кластерізація, асоціація і послідовність).

Після того, як визначено тип моделі, необхідно вибрати (розробити) алгоритм побудови моделі або технологію пошуку знання. Сутність процесу побудови контрольованої моделі зводиться до знаходження залежностей на одній частині даних ("навчання моделі") і перевірки цих залежностей на іншій частині даних (оцінка точності). Модель вважається побудованою, якщо завершується цикл "навчання" і перевірок. Якщо точність моделі при чергових ітераціях не поліпшується, то це говорить про завершення процесу побудови моделі.

Оскільки "навчальні" і тестові дані знаходяться в одній базі даних, то часто виникає необхідність у третьому наборі даних - контрольному, котрий вибирається з таких даних, що не перетинаються з "навчальними" і тестовими. Він потрібний для незалежного оцінювання точності моделі. Як правило, усі три набори даних належать множині даних, необхідній для реалізації визначеного проекту. Найбільш відомий тестовий метод називається простою оцінкою. У цьому випадку розподіл даних на два набори відбувається випадковим чином. Відношення кількості тестових даних до кількості даних, на яких відбувається побудова моделі повинна бути в межах від 5% до 33%. Після побудови моделі, її використовують для передбачення значень на тестовому наборі. Мірою точності моделі вважають

відношення кількості вдалих результатів до загальної кількості прикладів у тестовому наборі (можна використовувати таку змінну, як міра неточності, що дорівнює 1 - "міра точності"). Якщо для побудови моделі використовується не дуже велика база даних, то застосовується так називана перехресна оцінка точності. У цьому випадку дані випадковим чином поділяються на дві приблизно рівні частини. Після цього модель буде будуватися на одній з них, а інша використовується для визначення точності. Потім частини бази міняються ролями. Отримані дві незалежні оцінки точності поєднуються (як середнє арифметичне або іншим способом) для найкращої оцінки точності моделі, побудованої на всій базі. Для ще менших баз, у кілька тисяч записів, використовується n-перехресна оцінка точності. У цьому випадку база поділяється на n приблизно рівних непересічних груп. Далі перша з цих груп стає тестовим набором, а інші групи поєднуються, і на їхній основі відбувається побудова моделі. Отримана модель використовується для передбачення значень для тестового набору й у такий спосіб виходить перше значення точності. Аналогічним чином одержують усі n незалежні значення точності. Середнє з них є точністю всієї моделі.

Ще один спосіб використовується для знаходження точності в малих базах даних. У цьому випадку модель будується на основі даних усієї бази. Після цього випадковим чином із записів бази створюється множина тестових наборів (мінімум 200, а іноді навіть більше 1000). Один запис може бути присутнім у різних тестових наборах. Для кожної з них визначається точність. Середнє з них є точністю всієї моделі.

Після того, як побудова моделі довершена, можна побудувати модель, використовуючи інші параметри або навіть змінити алгоритм побудови моделі, так як ніколи не можна сказати, який алгоритм, яка технологія пошуку знання дасть кращі результати. Не можна бути упевненим, що обрана технологія буде працювати найкраще. Часто приходиться будувати велику кількість моделей і оцінювати кожну для знаходження кращої. Крім цього, для різних моделей необхідна різна підготовка даних і неминуче повторення кроків. Усе це

збільшує час знаходження кращої моделі, тому необхідно застосовувати технології паралельних обчислень.

*а*. *Оцінка моделі й інтерпретація ситуації*. Після побудови моделі необхідно оцінити результати і пояснити (інтерпретувати) їхню значимість. При оцінці моделі обчислюється точність, але треба пам'ятати, що це значення вірне лише для даних, на яких модель побудована і бути готовим, що нові дані, до яких надалі буде застосовуватися модель, можуть відрізнятися від вихідних невідомим чином.

*б*. *Зовнішня перевірка моделі*. Висока точність моделі не є гарантією того, що модель правильно відбиває реальну ситуацію. Однією з причин неправильного відображення ситуації, є існування так званих неявних припущень у моделі. Так, наприклад, коефіцієнт інфляції сам по собі не може бути частиною моделі, що пояснює схильність покупців до покупки того іншого товару, але різка зміна цього коефіцієнта з 3% до 20% уже, напевно, може пояснити таке поводження.

Інша причина - це існування проблем з даними, що приводять до некоректності моделі, тому дуже важливо перевірити модель у реальній ситуації. Наприклад, якщо модель використовується для добору кандидатів для цільової реклами, то можна зробити тестове розсилання для перевірки моделі на невеликому обсязі даних. Якщо модель використовується для передбачення ризику неповернення кредиту, то варто випробувати цю модель на невеликій кількості претендентів на позичку. Чим більше ризик, зв'язаний з некоректністю моделі, тим важливіше провести попередні експерименти для перевірки моделі перед її експлуатацією.

**4. Використання моделі**. Після побудови й оцінки моделі, її можна використовувати різними способами. Наприклад, аналітик може подивитися групу, що визначила модель кластерізації, графіки ефективності моделі або отриманих правил. Іноді аналітик може використовувати модель для вибору деяких записів з бази даних для проведення додаткового аналізу. Базуючись на результатах використання моделі, аналітик може

рекомендувати дії, які можна починати в діловій сфері. Однак, часто технології інтелектуальних обчислень - це частина автоматизованої системи (наприклад, знаходження кредитних ризиків, визначення можливості втрати клієнтів і ін.), тобто модель вбудовується в систему, яку аналітик або менеджер можуть застосовувати для ухвалення рішення. З іншого боку, модель можна включати в систему, що генерує деяку дію (наказ), коли прогнозована величина починає виходити за межі якихось значень. Загалом, методи інтелектуальних обчислень, це невелика, хоча і важлива частина кінцевого програмного продукту. Процедура пошуку знання за допомогою цих методів може сполучатися зі знаннями експертів і застосовуватися до даних у базах.

**5. Спостереження за моделлю**. Коли модель починає працювати в реальній ситуації, то необхідно вимірювати точність моделі на реальних даних. Однак, навіть якщо модель працює добре, і можна вважати, що робота на цьому закінчується, то все рівно необхідно продовжувати спостереження за моделлю. Усі системи мають властивість розвиватися, і отримані дані (їхня структура, точність, періодичність) теж міняються. Зовнішня змінна, така як коефіцієнт інфляції, своєю зміною теж можуть впливати на поводження людей і на фактори, що впливають на цю зміну. Час від часу модель необхідно піддавати повторному тестуванню, і навіть реконструюванню. Найпростішим способом спостереження за функціонуванням моделі є графіки розбіжностей між передбаченими величинами і реальними значеннями. Вони прості для побудови і розуміння і можуть вбудовуватися в програмні продукти. Така автоматизована система може стежити сама за собою і сповіщати користувача, коли величина цих розбіжностей починає виходити за визначений граничний рівень.

# 2. Нейроні мережі

## 2.1 Введення

В останні роки великий інтерес викликає проблематика нейроних мереж, генетичних алгоритмів і нечітких систем. Ці напрямки відносяться до наукової області, обумовленої в англомовній літературі терміном Computational Intelligence. На рис. 2.1. видно, що задачі нейроних мереж, генетичних алгоритмів і нечітких систем можуть розглядатися поза зв'язком між собою, однак їхня взаємозалежність виявляється надзвичайно важливою. Зокрема, генетичні алгоритми можна застосовувати для підбора ваг і топології нейроної мережі, а також для формування бази правил і функцій приналежності нечіткої системи

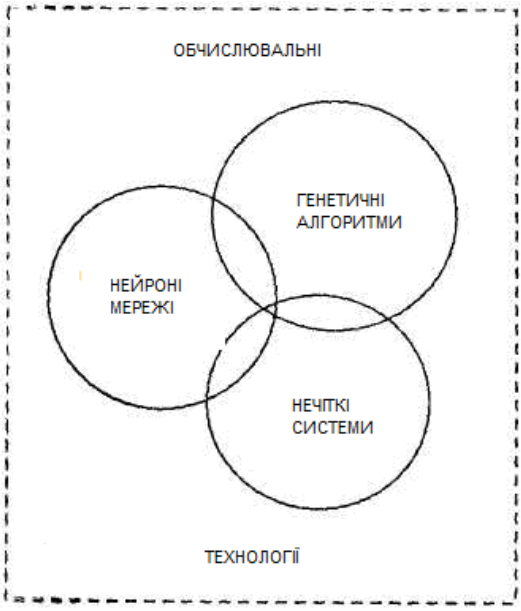



Рис. 2.1. Взаємозв'язки між нейроними мережами, генетичними алгоритмами і нечіткими системами.

У свою чергу, нейроні мережі дозволяють вибирати відповідні параметри для самих генетичних алгоритмів (параметри схрещування і мутації); саму філософію нейроних мереж можна закласти у фундамент нечітких систем, що у результаті знаходять здатність до навчання.

Крім того, методи теорії нечітких множин дозволяють підбирати як згадані вище параметри генетичних алгоритмів, так і коефіцієнти, що визначають швидкість навчання нейроних мереж. Одним з популярних напрямків Artificial Intelligence є теорія нейроних мереж (neuron nets). Даний курс є систематизованим вступним курсом у цей напрямок. Нашою метою є познайомити користувачів з основними нейроно-мережми парадигмами, показати область застосування цього напрямку.

Людей завжди цікавило їхнє власне мислення. Це самопитання, думання мозку самого про себе є, можливо, відмітною рисою людини. Нейробіологи і нейроанатоми досягли в цій області значного прогресу. Ретельно вивчаючи структуру і функції нервової системи людини, вони багато чого зрозуміли в «електропроводці» мозку, але мало довідалися про його функціонування. У процесі нагромадження ними знань з'ясувалося, що мозок має приголомшуючу складність. Сотні мільярдів нейронів, кожний з яких з'єднаний із сотнями або тисячами інших, утворюють систему, що далеко перевершує наші самі сміливі мрії про суперкомп'ютери. На сьогоднішній день існують дві взаємно збагачуючі одна одну мети нейронного моделювання: перша – зрозуміти функціонування нервової системи людини на рівні фізіології і психології і друга – створити обчислювальні системи (штучні нейроні мережі), що виконують функції, подібні до функцій мозку.

Нейроні мережі можна розглядати як сучасні обчислювальні системи, що перетворять інформацію з образа процесів, що відбуваються в мозку людини. Оброблювана інформація має числовний характер, що дозволяє використовувати нейрону мережа, наприклад, як модель об'єкта з зовсім невідомими характеристиками. Інші типові додатки нейроних мереж охоплюють задачі розпізнавання, класифікації, аналізу і стиску образів.

Понад 80 % усіх додатків нейроних мереж відноситься до так називаних багатошарових мереж без зворотних зв'язків. У них сигнал пересилається в напрямку від вхідного шару через сховані шари (якщо вони існують) до вихідного шару.

Хотілося б підкреслити, що в посібнику представляється досить важливий із прикладної точки зору фрагмент надзвичайно великої області знань, що одержала назву «штучні нейроні мережі».

Інтелектуальні системи на основі штучних нейроних мереж дозволяють з успіхом розв'язувати проблеми розпізнавання образів, виконання прогнозів, оптимізації, асоціативній пам'яті і керування. Традиційні підходи до розв'язання цих проблем не завжди дають необхідну гнучкість і багато додатків виграють від використання нейромереж. Штучні нейромережи є електронними моделями нейроної структури мозку, який, головним чином, учиться на досвіді. Природна аналогія доводить, що множина проблем, що не піддаються розв'язанню традиційними комп'ютерами, можуть бути ефективно вирішені за допомогою нейромереж.

Тривалий період еволюції додав мозкові людини багато якостей, відсутніх у сучасних комп'ютерах з архітектурою фон Неймана. До них відносяться:

- розподілене представлення інформації і паралельні обчислення
- здатність до навчання й узагальнення
- адаптивність
- толерантність до помилок
- низьке енергоспоживання

Системи, побудовані на принципах біологічних нейронів, мають перераховані характеристики, які можна вважати істотним досягненням в індустрії обробки даних. Досягнення в області нейрофізіології дають початкове розуміння механізму природного мислення, де збереження інформації відбувається у виді складних образів. Процес збереження інформації як образів, використання образів і розв'язання поставленої

проблеми визначають нову область в обробці даних, що, не використовуючи традиційного програмування, забезпечує створення паралельних мереж і їхнє навчання. У лексиконі розроблювачів і користувачів нейромереж присутні слова, відмінні від традиційної обробки даних, зокрема, "поводитися", "реагувати", "самоорганізовувати", "навчати", "узагальнювати" і "забувати".

## 2.1. Історія створення нейроних мереж

Вивченню людського мозку - тисячі років. З появою сучасної електроніки, почалися спроби апаратного відтворення процесу мислення. Першою спробою створення і дослідження штучних нейроних мереж вважається робота Дж. Маккалока (J. McCulloch) і У. Питтса (W. Pitts) "Логічне обчислення ідей, що відносяться до нервової діяльності" (1943 р.), у якій були сформульовані основні принципи побудови штучних нейронів і нейроних мереж. І хоча ця робота була лише першим етапом, багато ідей, які описані тут, залишаються актуальними і на сьогоднішній день. Величезний внесок у нейронауку внесла детекторна теорія. Її основоположником вважається Дж. Маккалок, що із групою своїх співробітників у 1959 році опублікував статтю за назвою "Про що ока жаби говорять мозкові жаби", де вперше було введене поняття нейрона-детектора. Робота зацікавила багатьох дослідників, найбільш успішні з яких - Р. Хьюберт і Т. Визела, об'єднавши гістохімічні і нейрофізіологічні методи, показали, що нейрони зорової кори кішки виборчо реагують на лінії визначеного нахилу. Об'єднані у вертикальні стовпчики з загальним для них нахилом, ці нейрони утворюють аналізатор, що визначає нахил лінії в локальній ділянці простору. Таким чином, поняття аналізатора, уведене І.П. Павловим на підставі вивчення умовних рефлексів, знайшло в рамках детекторної теорії свою нейрону основу.

Великим проривом в області нейроінтелекту стало створення нейрофізіологом Френком Розенблатом у 1962 р. моделі одношарової нейроної мережі, названої персептроном. Вона була використана для такого широкого класу задач, як прогнозування погоди, аналіз електрокардіограм і штучний зір.

Перші успіхи викликали сплеск оптимізму і послужили стимулом для продовження досліджень. Однак незабаром з'ясувалося, що створені мережі не здатні розв’язувати деякі задачі, що істотно не відрізняються від тих, котрі вони розв’язували успішно. Пізніше Марвин Мінський, використовуючи точні математичні методи, строго довів ряд теорем, показавши, що використовувані одношарові мережі теоретично не здатні розв’язувати багато простих задач, наприклад, реалізувати логічну функцію "виключаюче АБО". Бездоганність доказів Мінського, підкріплена його авторитетом в учених колах, з'явилася однієї з причин затримки розвитку нейроінтелекта майже на два десятиліття. Однак ряд найбільш наполегливих учених, таких як Кохонен, Гроссберг, Андерсон продовжили дослідження, поступово створюючи теоретичні основи для побудови і застосування штучних нейроних мереж. Як з'ясувалося, Мінський був занадто песимістичний у своїх прогнозах і багато з задач, описаних їм як не розв'язувані, зараз розв'язуються нейроними мережами з використанням стандартних процедур. В даний час теорія про нейроінтелект придбала новий подих. Було запропоновано багато різних розробок, таких, наприклад, як когнітон, здатний з високою вірогідністю розпізнавати досить складні образи (наприклад, ієрогліфи) незалежно від повороту і масштабу зображення. Автором когнітона є японський учений К. Фукушима (K. Fukushima). У 1982 році американський біофізик Дж. Хопфилд (J. Hopfield) запропонував цікаву модель мережі, що одержала в майбутньому його ім'я. Пізніше було розроблено ряд ефективних алгоритмів: мережа зустрічного потоку (R. Hecht-Neilsen), двонаправлена асоціативна пам'ять (B. Kosko) і інші. Сьогодні обговорення штучних нейроних мереж відбуваються скрізь. Перспектива їхнього використання здається досить яркою, у світлі розв’язання нетрадиційних проблем і є ключем до створення нових технологій. В даний час дослідження і розробки спрямовані на програмні й апаратні засоби реалізації нейромереж. Компанії працюють над створенням трьох типів нейрочипів: цифрових, аналогових і оптичних.

## 2.3. Нейрон і його моделі

Розвиток штучних нейроних мереж надихається біологією. Тобто, розглядаючи мережні конфігурації й алгоритми, дослідники роблять це, використовуючи терміни, які характерні для опису організації мозкової діяльності. Але на цьому аналогія, мабуть, закінчується. Наші знання про роботу мозку настільки обмежені, що мало б знайшлося орієнтирів для тих, хто став би йому наслідувати. Тому розроблювачам мереж приходиться виходити за межі сучасних біологічних знань у пошуках структур, здатних виконувати корисні функції.

Нервова система і мозок людини складаються з нейронів, з'єднаних між собою нервовими волокнами. Нервові волокна здатні передавати електричні імпульси між нейронами. Усі процеси передачі роздратувань від нашої шкіри, вух і очей до мозку, процеси мислення і керування діями - усе це реалізовано в живому організмі як передача тектричних імпульсів між нейронами.

Базовим елементом мозку людини є специфічні клітки, відомі як нейрони, здатні запам'ятовувати, думати і застосовувати попередній досвід до кожної дії, що відрізняє їх від інших кліток тіла. Кора головного мозку людини є плоскої, утвореною з нейронів поверхнею, товщиною від 2 до 3 мм площею близько 2200 $см^2$. Кора головного мозку містить близько $10^{11}$ нейронів, що приблизно дорівнює числу зірок Молочного шляхи. Кожен нейрон зв'язаний з $10^3$ - $10^4$ іншими нейронами. У цілому мозок людини має приблизно від $10^{14}$ до $10^{15}$ взаємозв'язків. Сила людського розуму залежить від числа базових компонентів, різноманіття з'єднань між ними, а також від генетичного програмування і навчання.

Індивідуальний нейрон є складним, має свої складові, підсистеми і механізми керування і передає інформацію через велику кількість електрохімічних зв'язків. Нараховують біля сотні різних класів нейронів. Разом нейрони і з'єднання між ними формують недвійковий, нестійкий і несинхронний процес, що відрізняється від процесу обчислень традиційних комп'ютерів.

Розглянемо будівлю біологічного нейрона. Базовий елемент нервової системи - це нервова клітка, яку називають *нейроном* (рис. 2.2)


Рис. 2.2. Схематичне зображення нервової клітки – нейрона.

У нейроні можна виділити тіло клітки, яке називають *сомою,* а також вихідні з нього два види відростків:

а) по яких у нейрон надходить інформація - *дендрити* і

б) по якому нейрон передає інформацію - *аксон.*

Кожен нейрон має тільки один вихідний відросток, по якому він може передавати імпульс декільком іншим нейронам. Одиничний нейрон приймає збудження від величезної кількості нейронів (їхнє число може досягати тисячі). Кожен нейрон передає збудження іншим нейронам через нервові стики, називані *синапсами,* при цьому процес передачі сигналів має складну електрохімічну природу. Синапс є функціональним вузлом між двома нейронами (волокно аксона одного нейрона і дендрит іншого). Коли імпульс досягає синаптичного закінчення, виробляються хімічні речовини, називані нейротрансмиттерами. Нейротрансмиттери проходять через синаптичну щілину, і в залежності від типу синапса, збуджуючи або гальмуючи здатність нейрона-приймача генерувати електричні імпульси. Результативність синапса настроюється сигналами, які через нього проходять, тому синапси навчаються в залежності від

активності процесів, у яких вони беруть участь. Нейрони взаємодіють за допомогою короткої серії імпульсів. Повідомлення передається за допомогою частотно-імпульсної модуляції.

Синапси впливають на силу імпульсу. Можна вважати, що при проходженні синапса сила імпульсу міняється у визначене число раз, що ми будемо називати вагою синапса. Імпульси, що надійшли до нейрона одночасно по декількох дендритах, підсумовуються. Якщо сумарний імпульс перевищує деякий поріг, нейрон збуджується, формує власний імпульс і передає його далі по аксоні. Важливо відзначити, що ваги синапсів можуть змінюватися згодом, а звідси виходить, що змінюється і поводження відповідного нейрона. Синапси відіграють роль репитерів інформації, у результаті функціонування яких збудження може підсилюватися або послаблятися. Як наслідок, до нейрона приходять сигнали, одна частина з яких робить збудливий, а друга - гальмуючий вплив. Нейрон підсумовує збудливі і гальмуючі імпульси. Якщо їхня алгебраїчна сума перевищує деяке граничне значення, то сигнал з виходу нейрона пересилається за допомогою аксона до інших нейронів.

Побудуємо математичну модель описаного процесу (рис. 2.3).

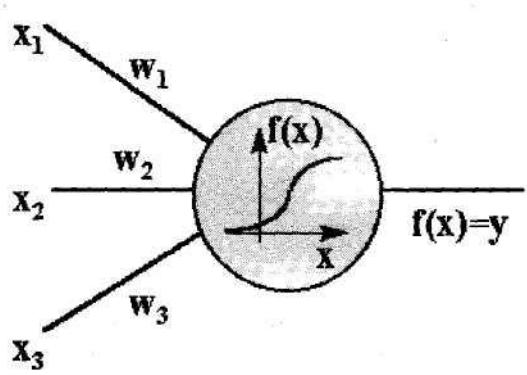



Рис. 2.3. Спрощена модель нейрона.

На рисунку зображена модель нейрона з трьома входами (дендритами), причому синапси цих дендритів мають ваги $w_1$, $w_2$, $w_3$. Нехай до синапсів надходять імпульси сили $x_1$, $x_2$, $x_3$ відповідно, тоді після проходження синапсів і дендритів до нейрона надходять імпульси $w_1 x_1$, $w_2 x_2$, $w_3 x_3$. Нейрон перетворить отриманий сумарний імпульс

$$x = w_1 x_1 + w_2 x_2 + w_3 x_3$$

відповідно до деякої передатної функції $f(x)$. Сила вихідного імпульсу дорівнює

$$y = f(x) = f(w_1 x_2 + w_2 x_2 + w_3 x_3).$$

Таким чином, нейрон цілком описується своїми вагами $w_k$ і передатною функцією $f(x)$. Одержавши набір чисел (вектор) $w_k$ як входи, нейрон видає деяке число $y$ на виході.

Розглянемо узагальнену модель нейрона (рис. 2.4), зв'язану з першими спробами формалізувати опис функціонування нервової клітки.

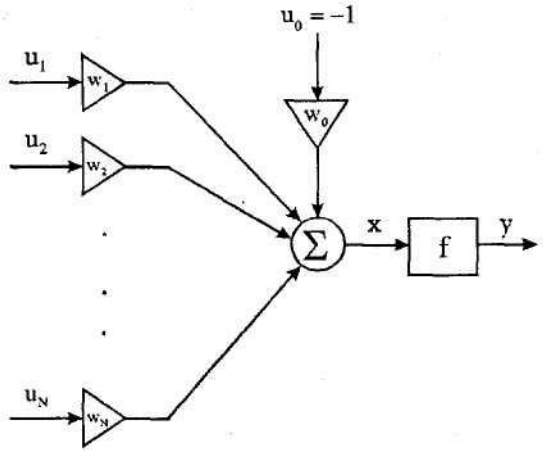



Рис. 2.4. Узагальнена модель нейрона.

Уведемо наступні позначення:

$u_1, \ldots, u_N$ - вхідні сигнали даного нейрона, що приходять від інших нейронів;

$w_1 \ldots . w_N$ - синаптичні ваги;

$y$ - вихідний сигнал нейрона;

$v$- граничне значення.

Формула, що описує функціонування нейрона, має вигляд

$$y = \begin{cases} 1 \quad при \quad \sum_{i=1}^{N} w_1 u_1 \geq v, \\ 0 \quad при \quad \sum_{i=1}^{N} w_1 u_1 < v. \end{cases} \tag{2.1}$$

Модель (2.1) може бути представлена у виді

$$y = f\left( \sum_{i=0}^{N} w_i u_i \right), \tag{2.2}$$

де

$$f(x) = \begin{cases} 1 \quad при \quad x \geq 0, \\ 0 \quad при \quad x < 0. \end{cases} \tag{2.3}$$

а також $w_0 = v$, $u_0 = 1$.

Формула (2.2) описує модель нейрона, представлену на рис. 2.4. Ця модель була запропонована в 1943 р. Маккаллоком і Питтсом . В якості функції f може прийматися не тільки одинична функція (2.3), але й інші граничні функції виду

$$f(x) = \begin{cases} 1 \quad при \quad x \geq 0, \\ -1 \quad при \quad x < 0. \end{cases} \tag{2.4}$$

Або

$$f(x) = \begin{cases} 1 \quad при \quad x > 1, \\ -1 \quad при \quad x < -1, \\ x \quad при \quad |x| \leq 1. \end{cases} \tag{2.5}$$

На початковій фазі моделювання біологічних нейроних мереж застосовувалися граничні функції (2.3), (2.4) і (2.5). В даний час найчастіше використовується сігмоїдальна функція, яка обумовлена виразом

$$f(x) = \frac{1}{1 + e^{-\beta x}} > 0. \tag{2.6}$$

Відзначимо, що при β→∞ характеристика (2.6) прагне до граничної уніполярної функції (2.3). Як альтернативу застосовується функція гіперболічного тангенса

$$f(x) = th\left(\frac{\alpha x}{2}\right) = \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} > 0. \qquad (2.7)$$

У цьому випадку характеристика (2.7) прагне до граничної біполярної функції (2.4) при α→∞. Приклади функції f у моделі (2.2) показані на рис. 2.5.

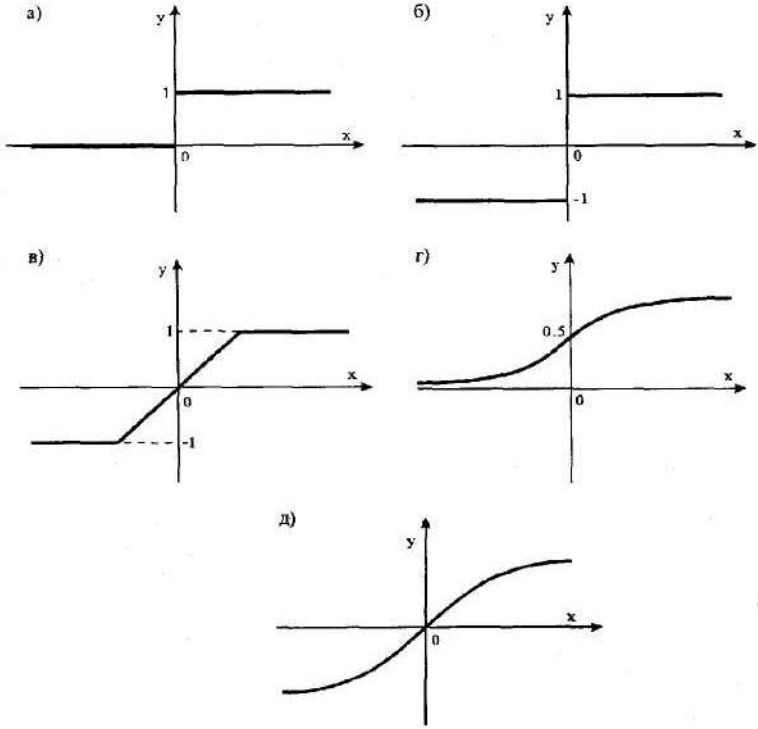



Рис. 2.5. Приклади функції f.

У наявним зараз пакетах програм штучні нейрони називаються "елементами обробки" і мають більше можливостей, чим простий штучний нейрон, описаний вище. На рис. 2.6 зображена детальна схема спрощеного штучного нейрона.

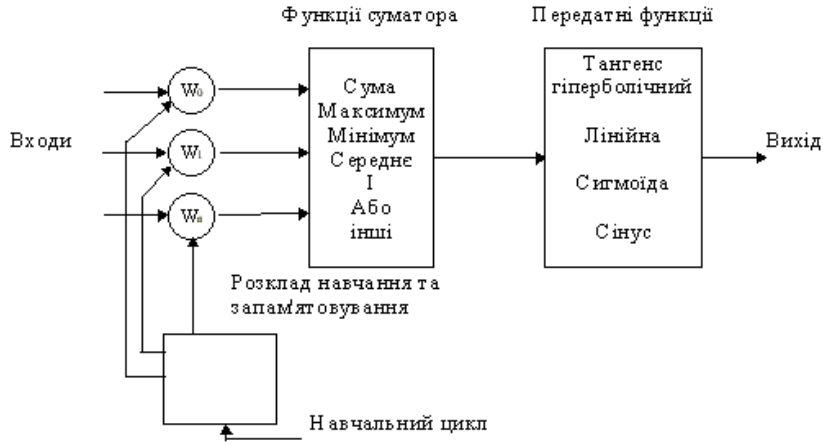



Рис. 2.6. Схема спрощеного штучного нейрона.

Модифіковані входи передаються на функцію підсумовування, що переважно тільки підсумовує добутки. Можна вибрати різні операції, такі як середнє арифметичне, найбільше, найменше, OR, AND, і т.п., що виробляють різні значення. Більшість комерційних програм дозволяють інженерам-програмістам створювати власні функції сумматора за допомогою підпрограм, закодованих мовою високого рівня. Іноді функція підсумовування ускладнюється додаванням функції активації, що дозволяє функції підсумовування діяти в часі. У кожнім з цих випадків, вихід функції підсумовування проходить через передатну функцію на вихід (0 або 1, -1 або 1, або яке-небудь інше число) за допомогою визначеного алгоритму. В існуючих нейромережах як передатні функції, як ми

вже говорили, можуть бути використані сігмоїда, синус, гіперболічний тангенс і ін. Приклад того, як працює передатна функція показано на рис. 2.7.

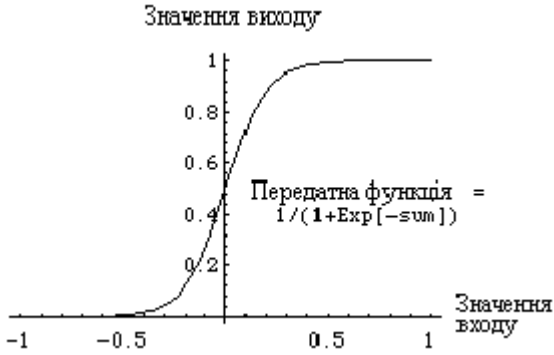



Рис. 2.7. Сігмоїдна передатна функція

Усі штучні нейромережі конструюються з базового блоку - штучного нейрона. Існуючі розмаїтості і відмінності, є підставою для мистецтва талановитих розроблювачів при реалізації ефективних нейромереж.

## 2.4. Нейроні мережі

**Штучна нейрона мережа (ШНМ)** — це математична модель, а також пристрій паралельних обчислень, що представляють собою систему з'єднаних і взаємодіючих між собою простих процесорів (штучних нейронів).Як математична модель штучна нейрона мережа являє собою окремий випадок методів распізнавання образів або дискриминантного аналізу. Такі процесори зазвичай досить прості, особливо в порівнянні з процесорами, використовуваними в персональних комп'ютерах. Кожен процесор подібної мережі має справу тільки із сигналами, що він періодично одержує, і сигналами, що він періодично посилає іншим процесорам. І проте, будучи з'єднаними в досить велику мережу з керованою взаємодією, такі локально прості

процесори разом здатні виконувати досить складні задачі. Поняття виникло при вивченні процесів, що протікають у мізку при міслені, і при спробі змоделювати ці процеси. Отримані моделі називаються штучними нейроними мережами (ШНМ), у яких використовується велика кількість зв'язків, які зв’язують окремі нейрони. Групування в мозку людини відбувається так, що інформація обробляється динамічним, інтерактивним і самоорганізуючим шляхом. Біологічні нейроні мережі створені в тривимірному просторі з мікроскопічних компонентів і здатні до різноманітних з'єднань, а для створеної людиною мережі існують фізичні обмеження.

З іншого боку, штучна нейрона мережа – це набір штучних нейронів, з'єднаних між собою. Як правило, передатні функції всіх нейронів у мережі фіксовані, а ваги є параметрами мережі і можуть змінюватися. Деякі входи нейронів позначені як зовнішні входи мережі. Деякі виходи - як зовнішні виходи мережі. Подаючи будь-які числа на входи мережі, ми одержуємо якийсь набір чисел на виходах мережі. Таким чином, робота нейромережі складається в перетворенні вхідного вектора у вихідний вектор, причому це перетворення задається вагами мережі.

Практично будь-яку задачу можна звести до задачі, яка розв'язується нейромережою. У таблиці, яка приведена нижче, показано, яким чином варто сформулювати в термінах нейромережі задачу розпізнавання рукописних букв.

| Задача розпізнавання рукописних букв | |
|---|---|
| Дано: растрове чорно-біле зображення букви розміром 30х30 пікселів | Треба: визначити, яка це буква (в алфавіті 33 букви) |
| Формулювання для нейромережі | |
| Дано - вхідний вектор з 900 двоїстих символів 900=(30×30) | Треба: побудувати нейромережу з 900 входами і 33 виходами, що позначені буквами. Якщо на вході мережі - зображення букви "А", то максимальне значення вихідного сигналу досягається |

| | |
|---|---|
| | на виході "А". Аналогічно мережа працює для всіх 33 букв. |

Пояснимо, навіщо потрібно вибирати вихід з максимальним рівнем сигналу. Справа в тім, що рівень вихідного сигналу, як правило, може приймати будь-яке значення з якогось відрізка. Однак, у даній задачі нас цікавить не аналогова відповідь, а всього лише номер категорії (номер букви в алфавіті). Тому використовується наступний підхід - кожній категорії зіставляється свій вихід, а відповіддю мережі вважається та категорія, на чиїм виході рівень сигналу максимальний. У визначеному змісті рівень сигналу на виході "А" - це вірогідність того, що на вхід була подана рукописна буква "А". Задачі, у яких потрібно віднести вхідні дані до однієї з відомих категорій, називаються *задачами класифікації.* Викладений підхід - стандартний спосіб класифікації за допомогою нейроних мереж.

Існуючі на даний час нейромережі є групуванням штучних нейронів, у виді з'єднаних між собою шарів.

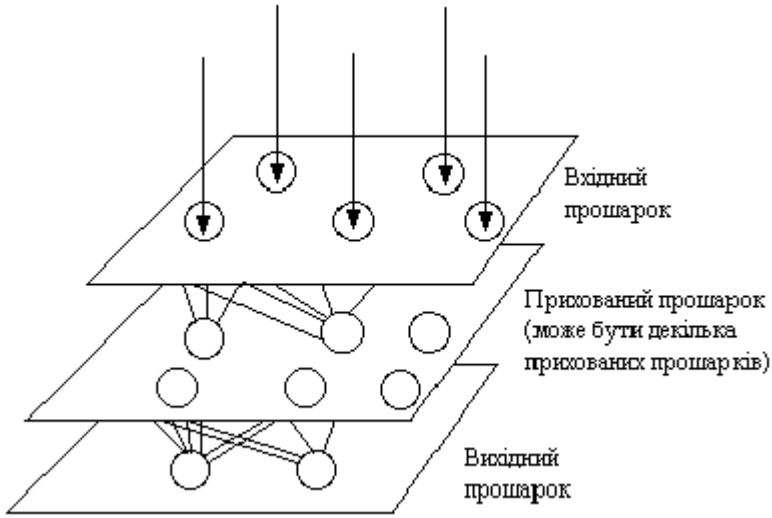



Рис. 2.8. Діаграма простої нейроної мережі

На рис. 2.8 показана типова структура штучних нейромереж. Хоча існують мережі, що містять лише один шар, або навіть один елемент, більшість реалізацій використовують мережі, що містять як мінімум три типи шарів - вхідний, схований і вихідний. Шар вхідних нейронів одержує дані або з вхідних файлів, або безпосередньо з електронних датчиків. Вихідний шар пересилає інформацію безпосередньо в зовнішнє середовище, до вторинного комп'ютерного процесу, або до іншого пристрою. Між цими двома шарами може бути кілька схованих шарів, що містять багато різноманітно зв'язаних нейронів. Входи і виходи кожного зі схованих нейронів з'єднані з іншими нейронами. Напрямок зв'язку від одного нейрона до іншого є важливим аспектом нейромереж. У більшості мереж кожен нейрон схованого шару одержує сигнали від усіх нейронів попереднього шару і звичайно від нейронів вхідного шару. Після виконання операцій над сигналами, нейрон передає свій вихід усім нейронам наступних шарів, забезпечуючи передачу сигналу вперед (feedforward) на вихід. При зворотному зв'язку, вихід

нейронів шару направляється до нейронів попереднього шару (рис. 2.9).

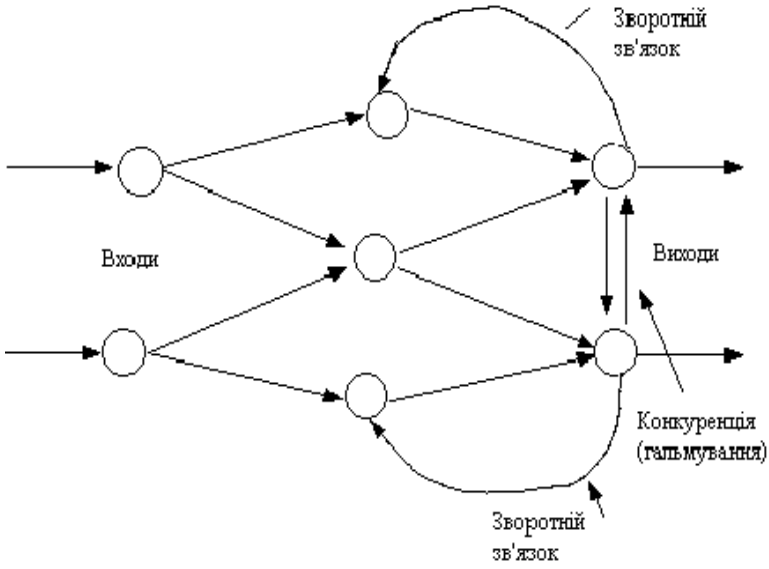



Рис. 2.9.

Вид з'єднання між нейронами має великий вплив на роботу мережі. Більшість пакетів програмних реалізацій нейроних мереж дозволяють користувачеві додавати, віднімати і керувати з'єднаннями як завгодно. Постійно корегуємі параметри зв'язку можна зробити як збудливими так і гальмуючими.

## 2.4.1. Найпростіші моделі штучних нейроних мереж

### 2.4.1.1.Персептрон

Модель МакКаллока-Піттса стала відправною точкою для побудови найпростішої односпрямованої нейроної мережі, названою *персептроном.* Таку мережу запропонував і

досліджував Розенблатт наприкінці п'ятидесятих - початку шістдесятих років XX століття. На рис. 2.10 представлена структура персептрона, іноді називаного *найпростішим персептроном.*

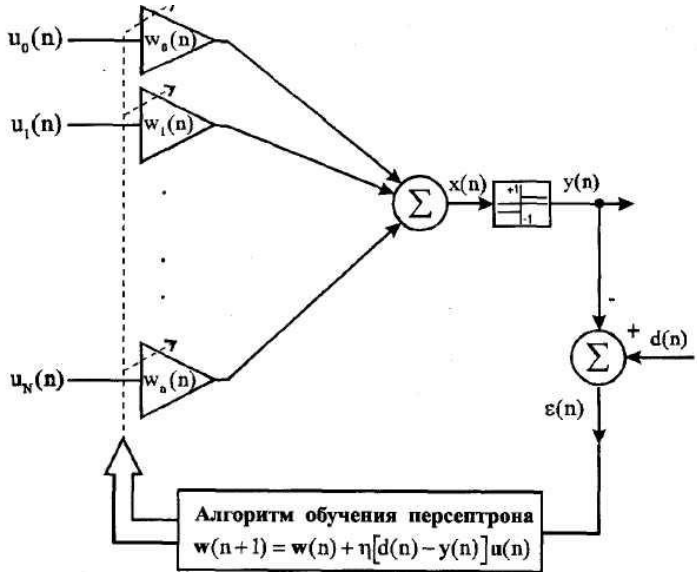



Рис. 2.10. Персептрон.

Як функцію *f* у моделі МакКаллока-Питтса (2.2) застосовувалася біполярна функція активації (2.4). Сигнал x на виході лінійної частини персептрона задається виразом

$$x = \sum_{i=1}^{N} w_i u_i - v = \sum_{i=0}^{N} w_i u_i , \qquad (2.8)$$

де $w_0 = v$, $u_0 = -1$.

Задача персептрона полягає в класифікації вектора

$i=[u_1.....u_N]^T$ у змісті віднесення його до одному з двох класів, що позначаються символами $L_1$ і $L_2$. Персептрон відносить вектор *i* до класу $L_1$, якщо вихідний сигнал *y* приймає значення 1, і до класу $L_2$, якщо вихідний сигнал *y* приймає значення -1. Після цього персептрон розділяє N-мірний простір вхідних векторів *i* на

два півпростори, поділювані (N-1)-мірною гіперплощиною, що задається рівнянням

$$\sum_{i=1}^{N} w_i u_i - v = \sum_{i=0}^{N} w_i u_i = 0, \qquad (2.9)$$

Гіперплощина (2.9) називається *вирішальною границею (decision boundary).* Якщо *N*= 2, то вирішальна границя - це пряма лінія, що задається рівнянням

$$w_1 u_1 + w_2 u_2 - v = 0. \qquad (2.10)$$

Точка *(i1,i2), що* лежить над цій прямій (рис. 2.11), відноситься до класу $L_1$, тоді як точка *(i1,i2), що* лежить під цією прямою, відноситься до класу $L_2$. Точки, що лежать на границі рішення, можна довільно віднести і до класу $L_1$ і до класу $L_2$.

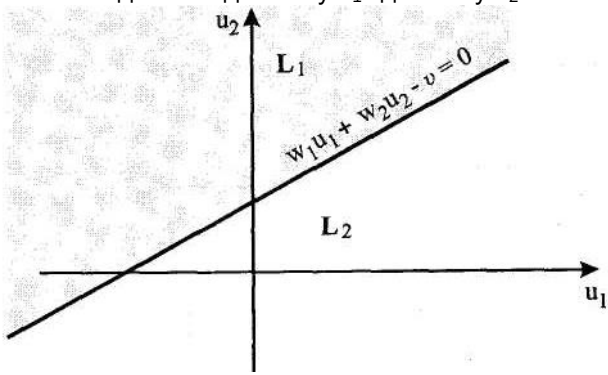



Рис. 2.11. Вирішальна границя для *N*=2.

Для подальших міркувань припустимо, що ваги $w_i$, $i$ = 0, 1, ..., $N$ у рівнянні гіперплощини (2.9) невідомі, тоді як на вхід персептрона послідовно подаються так називані навчальні сигнали $u(n)$, $n$ = 1, 2...., де

$$u(n) = [u_1(n), .... u_N]^T.$$

Невідомі значення ваг будуть визначатися в процесі навчання персептрона. Такий підхід одержав назву «навчання з учителем» або «навчання під наглядом». Роль «учителя» полягає в коректному віднесенні сигналів $u(n)$ до класів $L_1$ або $L_2$, незважаючи на невідомість ваг рівняння вирішальної границі (2.9). По завершенні процесу навчання персептрон повинний коректно класифікувати сигнали, які надходять на його вхід, у тому числі і ті, котрі були відсутні в навчальній послідовності $u(n)$, $n$=1, 2..... Крім того, приймемо, що множині векторів $u(n)$,

*n*=1, 2, ..., для яких вихід персептрона приймає відповідно значення 1 і -1, лінійно відділені, тобто лежать у двох різних півпросторах, розділених гіперплощиною (2.9). Іншими словами, допускається поділ навчальної послідовності {*u*(n)} на двох послідовностей {$u_1(n)$} і {$u_2$(n)} так, що {$u_1(n)$}$\in$ L$_1$ і {$u_2(n)$} $\in L_2$.

У *n*-й момент часу сигнал на виході лінійної частини персептрона визначається виразом

$$x(n)=\sum_{i=0}^{N} w_i(n)u_i(n)=w^T(n)u(n), \qquad (2.11)$$

де

$$u(n) = [-1,u_1(n),u_2(n).....u_N(n)]^T, \qquad (2.12)$$

$$w(n) = [v(n), w_1(n), w_2(n).....w_N(n)]^T. \qquad (2.13)$$

Навчання персептрона полягає в рекуррентній корекції вектора ваг *w(n)* відповідно до формул

$$w(n+1)=\begin{cases} w(n), & якщо\ w^T(n)u(n)\ge 0\ і\ u(n)\in L_1, \\ w(n), & якщо\ w^T(n)u(n)<0\ і\ u(n)\in L_2, \end{cases} \qquad (2.14)$$

і

$$w(n+1)=\begin{cases} w(n)-\eta u(n), & якщо\ w^T(n)u(n)\ge 0\ і\ u(n)\in L_1, \\ w(n)+\eta u(n), & якщо\ w^T(n)u(n)<0\ і\ u(n)\in L_2, \end{cases} \qquad (2.15)$$

де параметр η при 0<η<1 - крок корекції, тоді як початкові значення компонент вектора ваг установлюються рівними нулеві, тобто

$$w(0) = 0. \qquad (2.16)$$

Залежності (2.14) і (2.15) можна представити в більш стислому виді. Для цього визначимо так називаний еталонний (заданий) сигнал *d(n)* у формі

$$d(n)=\begin{cases} +1, & якщо\ \ u(n)\in L_1, \\ -1, & якщо\ \ u(n)\in L_2. \end{cases} \qquad (2.17)$$

Крім того, відзначимо, що вихідний сигнал персептрона може бути описаний виразом

$$y(n) = sgn(w(n)u(n)). \qquad (2.18)$$

З урахуванням уведених позначень рекурсії (2.14) і (2.15) приймають вид

$$w(n + 1) = w(n) + [d(n) – y(n)]\ u(n). \qquad (2.19)$$

Різницю *d(n)-y(n)* можна інтерпретувати як похибку між еталонним (заданим) сигналом *d(n)* і фактичним вихідним сигналом *y(n).*

Збіжність алгоритму (2.19) досліджував Розенблатт, а також інші автори. З урахуванням прийнятої вище умови лінійної сепарабельності вхідних сигналів алгоритм (2.19) сходиться, тобто

$$w(n_0) = w(n_0 + 1) = w(n_0 + 2) = ... \qquad (2.20)$$

По завершенні навчання вирішальна границя персептрона визначається виразом

$$\sum_{i=0}^{N} w_i(n_0)u_i = 0, \qquad (2.21)$$

а персептрон коректно класифікує як сигнали, що належать до навчальної вибірки {*u(n)*}, так і які не входять в цю множину, але виконують умову лінійної сепарабельності. Нагадаємо, що умові лінійної сепарабельності не відповідає логічна функція XOR, задана таблицею 2.1.

Таблиця 2.1. Логічна функція XOR

| $u_1$ | $u_2$ | d=XOR ($u_1$, $u_2$) |
|---|---|---|
| +1 | +1 | -1 |
| +1 | -1 | +1 |
| -1 | +1 | +1 |
| -1 | -1 | -1 |

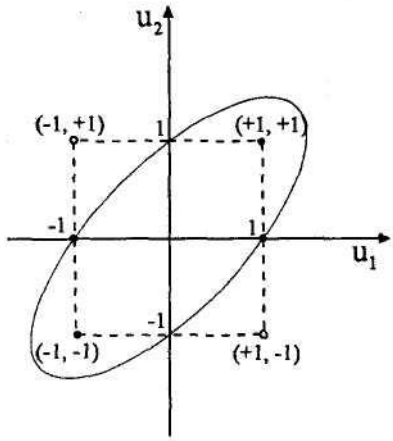



Рис**.** 2.12. Ілюстрація проблеми XOR.

З рис. 2.12 випливає, що не існує прямої, що відокремила б точки зі значеннями функції XOR, рівними - 1, від точок зі значеннями, рівними 1. У цьому випадку роль зразкової границі грає еліпс, і тому алгоритм (2.18) не був би таким,що сходиться. Проблему XOR можна розв'язати за допомогою двошарового персептрона.

### 2.4.1.2. Системи типу Адалайн

Системи типу Адалайн *(Adaptive Linear Neuron* - адаптивний лінійний нейрон) були запропоновані в 1960 р. Видроу і Хоффом. Видроу і Лер описали ціле сімейство систем типу Адалайн. Перед тим як перейти до детального обговорення систем типу Адалайн (пп. **2.4.1.2.2** і **2.4.1.2.3**) розглянемо модель так називаного лінійного зваженого суматора.

#### 2.4.1.2.1. Лінійний зважений суматор

На рис. 2.13. представлена структура лінійного зваженого суматора *(linear combiner).*

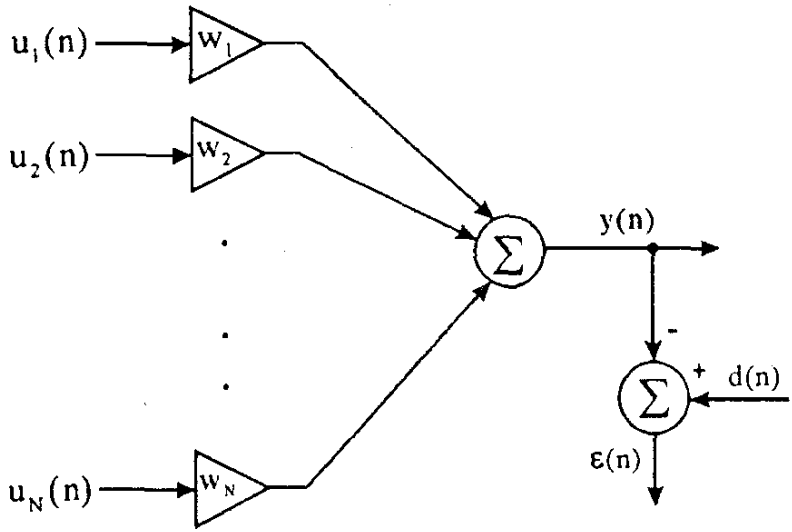



Рис. 2.13. Лінійний зважений суматор.

Його вихід утворюється сигналом y(*n*), що являє собою лінійну комбінацію усіх входів $u_1(n), u_2(n), \ldots, u(n)$, $n$=1,2.... Уведемо позначення

$$u(n) = [u,(n), u_2(n).....u(n)]^T. \qquad (2.22)$$

Конкретні компоненти вектора $u(n)$ збільшуються на компоненти вектора ваг

$$w=[w_1,w_2.....w]^T. \qquad (2.23)$$

У результаті вихідний сигнал лінійного зваженого суматора описується формулою

$$y(n)= \sum_{k=1}^{M} w_k u_k(n) = w^T u(n). \qquad (2.24)$$

Вихід лінійного зваженого суматора $y(n)$ буде використовуватися як реалізація деякого сигналу $d(n)$, називаного еталонним або заданим сигналом. У результаті порівняння реалізації $y(n)$ із сигналом $d(n)$ одержуємо похибку реалізації

$$\varepsilon(n) = d(n)-y(n). \qquad (2.25)$$

Ваги лінійного зваженого сумматора $w_1$, $w_2$.....$w$ будуть підбиратися так, щоб мінімізувати міру похибки

$$Q(w) = E[(n)^2] = E\left[(d(n) - \sum_{k=1}^{M} w_k u_k(n))^2\right] =$$

$$= E[(d(n) - w^T u(n))^2] \qquad (2.26)$$

Припустимо, що вхідний сигнал *u(n)* і еталонний сигнал *d(n)* - це реалізації дискретних стохастических процесів *{u(n)}* і *{d(n)},* спільно стаціонарних у широкому змісті, тобто

1) *{u(n)}* - стаціонарний у широкому змісті стохастический процес;

2) *{d(n)}* - стаціонарний у широкому змісті стохастический процес;

3) функція взаємної кореляції процесів *{u(n)}* і *{d(n - k)}* залежить тільки від значення *k.*

Міра похибки (2.26) називається *середньоквадратичною похибкою реалізації.* Позначимо $[w^*_1.....w^*_N]^T = w^*$ вектор ваг, які мінімізують похибку (2.26). Представлена на рис. 2.13 система, ваги якої приймають значення $w^*_1$, ..., $w^*_N$, називається *просторовим фільтром Вінера (spatial filter).*

Процес фільтрації полягає в множенні входів $u_1(n).....u_N(n)$ на відповідну їм множину ваг $w^*_1....w^*_N$ *з* наступним підсумовуванням окремих добутків для одержання реалізації *y(n)* еталонного сигналу *d(n).* Покажемо, що середньоквадратична похибка реалізації (2.26) - це функція другого порядку вектора ваг *w.*

Оскільки

$$[d(n) - w^T u(n)]^2 = d2(n) - 2d(n)w^T u(n) + w^T u(n)u(n)w\ , \qquad (2.27)$$

то формула (2.26) приймає вид

$$Q(w) = E[d_2(n)] - 2w^T E[d(n)u(n)] + w^T E[u(n)u(n)]\ w\ . \qquad (2.28)$$

В останньому доданку вираз (2.28) можна виділити матрицю автокореляції компонентів вхідного вектора

$$R = E[i(n)u^T(n)] =$$

$$= \begin{bmatrix} E[(u^2{}_1(n))] & E[(u_1(n)u_2(n)]...E[(u_1(n)u_N(n)] \\ E[(u_2(n)u_1(n)] & E[(u^2{}_2(n))] \quad ...E[(u_2(n)u_N(n)] \\ ..... \\ E[(u_N(n)u_1(n)] & E[(u_N(n)u_2(n)]...E[(u^2{}_N(n))] \end{bmatrix} \qquad (2.29)$$

а також вектор взаємної кореляції між сигналами *d(n)* і *y(n)*

$$p = E[d(n)u(n)] = \begin{bmatrix} E[d(n)(u_1(n))] \\ E[d(n)(u_2(n))] \\ \cdots\cdots\cdots\cdots \\ E[d(n)(u_N(n))] \end{bmatrix} \quad (2.30)$$

З використанням позначень (2.29) і (2.30) середньоквадратична похибка реалізації (2.28) може бути записана у виді

$$Q(w) - E[d^2(n)] - 2w^T p + w^T R w . \quad (2.31)$$

З виразу (2.31) випливає, що середньоквадратична похибка реалізації *Q(w)* - це функція другого порядку вектора ваг *w*. З геометричної точки зору *Q(w)* представляється гіперпараболоїдом, що має єдиний глобальний екстремум Q*. Цей гіперпараболоїд називається *поверхнею середньоквадратичної похибки.* Рис. 2.14 представляє фрагмент типової поверхні середньоквадратичної похибки для *N* = 2 (у цьому випадку це параболоїд).

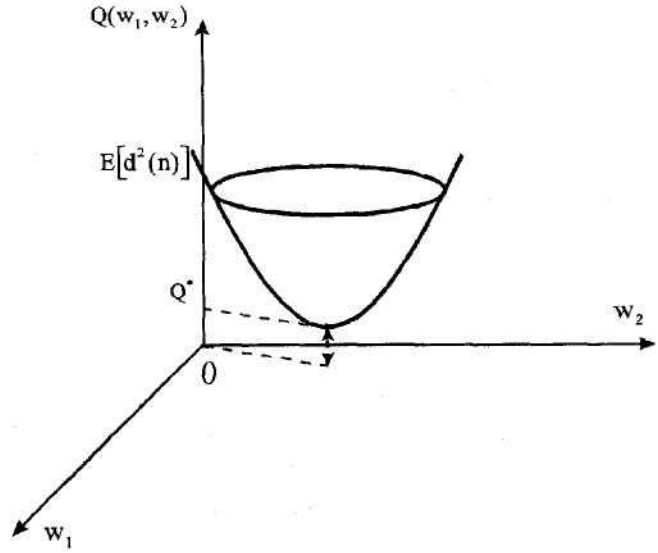



Рис. 2.14. Поверхня середньоквадратичної похибки.

Поверхня середньоквадратичної похибки, яка описується рівнянням (2.31), має єдиний глобальний екстремум Q*, що досягається при оптимальних значеннях ваг $w^*_1$.....$w^*_N$. Обчислення оптимальних значень ваг зводиться до визначення вектора градієнта $\nabla$ функції $Q(w)$ і прирівнюванню отриманого результату до нуля:

$$\nabla = \frac{\partial Q}{\partial w} = \left[ \frac{\partial Q}{\partial w_1}, \frac{\partial Q}{\partial w_2}, ..., \frac{\partial Q}{\partial w_N} \right] = 2Rw - 2p = 0, \qquad (2.32)$$

де 0 є N-мірний нульовий вектор. Припустимо, що гессіан

$$\nabla^2 = \frac{\partial^2 Q}{\partial w^2} = 2R \qquad (2.33)$$

це додатно визначена матриця. У цьому випадку вектор ваг $w^*$ мінімізує середньоквадратичну похибку реалізації (2.31). З рівняння (2.32) випливає, що для цього вектора справедлива рівність

$$Rw^* = p. \qquad (2.34)$$

Рівність (2.34) називається *нормальним рівнянням.* Якщо det R ≠0, то його розв'язком є вектор

$$w^* = R^{-1}p . \qquad (2.35)$$

При $w=w^*$ похибка приймає мінімальне значення, що позначається Q* і рівне

$$Q(w^*) = Q^* = E[d_2(n)] - 2w^{*T}p + w^{*T}Rw^*. \qquad (2.36)$$

Якщо підставити рівність (2.34) у вираз (2.36), то одержимо формулу для розрахунку мінімальної середньоквадратичної похибки реалізації

$$Q^* = E[d_2(n)] - w^{*T}p . \qquad (2.37)$$

Для знаходження оптимального вектора ваг w*, що задовольняє нормальному рівнянню (2.34), потрібно інвертувати матрицю автокореляції R. Замість цього можна використовувати метод найшвидшого спуску, що широко застосовується в теорії оптимізації. У цьому методі передбачається ітеративний розрахунок послідовних наближень оптимального вектора w*. Позначимо $w(n)$ наближення, розраховане на $n$-й ітерації

$$w(n)=[w_1(n).....w_N(n)]^T. \qquad (2.38)$$

Чергові корекції компонентів вектора ваг $w(n)$ повинні виконіватися в напрямку, протилежному знакові компонентів вектора градієнта

$$\frac{\partial Q(w(n))}{\partial (w(n))} = \left[ \frac{\partial Q(w(n))}{\partial w_1(n)}, \ldots, \frac{\partial Q(w(n))}{\partial w_N(n)} \right]^T \qquad (2.39)$$

Алгоритм найшвидшого спуску можна представити у виді

$$w(n+1) = w(n) - \frac{1}{2}\eta \frac{\partial Q(w(n))}{\partial w(n)} \ . \qquad (2.40)$$

При підстановці формули (2.32) у залежність (2.40) одержуємо рекурсію

$$W(n+1) = w(n) + [p - Rw(n)] \ , \qquad (2.41)$$

де константа η>0 визначає величину кроку корекції.

Можна показати , що алгоритм найшвидшого спуску (2.41) сходиться, тобто

$$\lim_{n\to\infty} w(n) = w^* . \qquad (2.42)$$

якщо крок корекції η лежить у межах

$$0 < \eta < \frac{1}{\lambda_{max}} \ , \qquad (2.43)$$

де $\lambda_{max}$ - це найбільше власне значення матриці автокореляції R. Крім того, доведено, що швидкість збіжності алгоритму найшвидшого спуску залежить від відношення найменших і найбільшого власних значень матриці R. Якщо

$$\frac{\lambda_{min}}{\lambda_{max}} \cong 1, \qquad (2.44)$$

то алгоритм найшвидшого спуску сходиться швидко. Якщо ж

$$\frac{\lambda_{min}}{\lambda_{max}} \cong 0 \qquad (2.45)$$

то алгоритм найшвидшого спуску сходиться повільно.

#### 2.4.1.2.2. Адаптивний лінійний зважений суматор

Застосування алгоритму (2.41) припускає знання матриці R і вектора $\partial$. У випадку, коли ці величини невідомі, варто замінити

градієнт (2.39) його наближенням. Запишемо рекурсивний вираз (2.40) у виді

$$w(n+1) = w(n) - \frac{1}{2}\eta \frac{\partial E[\varepsilon^2(n)]}{\partial w(n)} . \qquad (2.46)$$

Якщо в цій формулі замінити градієнт його наближеним локальним значенням *(instantaneous estimate),* тобто

$$\frac{\partial E[\varepsilon^2(n)]}{\partial w(n)} \to \frac{\partial \varepsilon^2(n)}{\partial w(n)} \qquad (2.47)$$

то одержимо рекурсію виду

$$\hat{w}(n+1) = \hat{w}(n) - \frac{1}{2}\eta \frac{\partial \varepsilon^2(n)}{\partial w(n)} . \qquad (2.48)$$

З виразів (2.24) і (2.25) випливає, що

$$\frac{\partial \varepsilon^2(n)}{\partial w(n)} = 2\varepsilon(n) \frac{\partial \varepsilon(n)}{\partial w(n)} - 2\varepsilon(n)u(n). \qquad (2.49)$$

При підстановці залежності (2.49) у формулу (2.48) одержуємо так називаний алгоритм LMS *(Least Mean Square)* у векторній формі

$$w(n+1) = w(n) + u(n)[d(n) - w(n)u(n)] \qquad (2.50)$$

або в скалярній формі

$$\hat{w}_k(n+1) = \hat{w}_k(n) - u_k(n)[d(n) - \sum_{k=1}^{N} w_k(n)u_k(n)] \qquad (2.51)$$

для $k = 1.....N$.

На рис. 2.15 представлено адаптивний лінійний зважений суматор, відомий у літературі за назвою Адалайн *(Adaptive Linear Neuron).* Він складається з двох основних частин:

1) лінійного зваженого суматора з адаптивно вагами, що коректуються

$$\hat{w}_1(n)..... \hat{w}_N(n),$$

*2)* підсистеми, призначеної для адаптивної корекції цих ваг і реалізуючої алгоритм LMS.

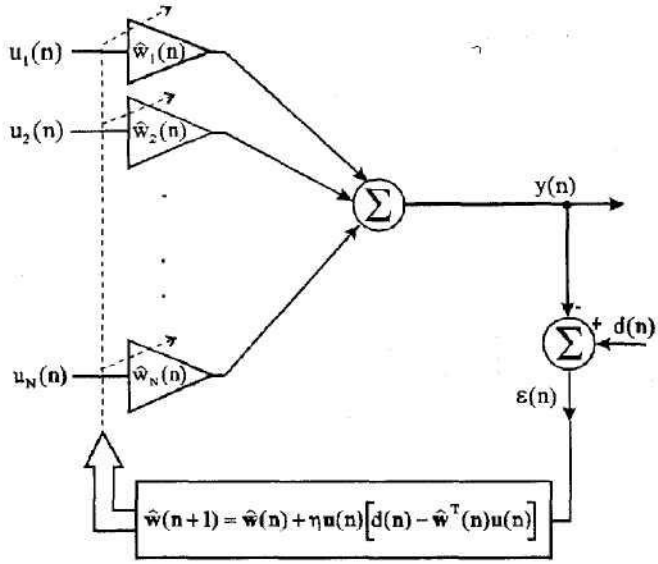



Рис. 2.15. Адаптивний лінійний зважений суматор.

Параметр $\eta$ в алгоритмі (2.50) підбирається так, щоб виконувалася умова

$$0 < \mu < \frac{2}{TrR} = \frac{2}{\sum_{k=1}^{N} E[(u_k(n))^2]} \qquad (2.52)$$

де *Tr* позначає слід матриці **R.**

### 2.4.1.2.3. Адаптивний лінійний зважений суматор із сігмоїдою на виході

Вихідний сигнал адаптивного лінійного сумматора із сігмоїдою на виході (рис. 2.16) можна описати виразом

$$y(n) = f(\sum_{k=1}^{N} w_k(n)u_k(n)), \qquad (2.53)$$

де функція *f* визначається формулою (2.6). Похибка реалізації (2.25) дорівнює

$$\varepsilon(n) = d(n) - f\left(\sum_{k=1}^{N} \hat{w}_k(n) u_k(n)\right), \qquad (2.54)$$

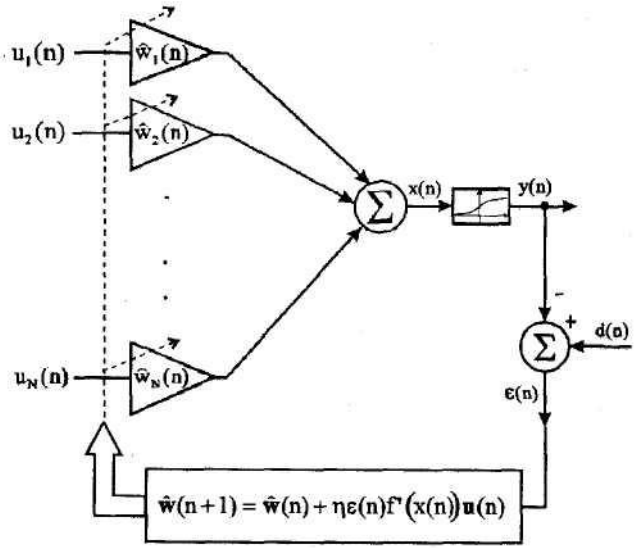



Рис. 2.16. Адаптивний лінійний зважений суматор із сігмоїдою на виході.

Для корекції ваг $\hat{w}_k(k)$, $k$ = 1..... $N$ застосуємо алгоритм LMS у рекурсивній формі (2.48). У цьому випадку очевидна рівність

$$\frac{\partial \varepsilon^2(n)}{\partial \hat{w}(n)} = 2\varepsilon(n)\frac{\partial \varepsilon(n)}{\partial \hat{w}(n)}, \qquad (2.55)$$

а також

$$\frac{\partial \varepsilon(n)}{\partial \hat{w}(n)} = -\frac{\partial f(x(n))}{\partial \hat{w}(n)} = -f(x(n))\frac{\partial x(n)}{\partial \hat{w}(n)}, \qquad (2.56)$$

де

$$x(n) = \sum_{k=1}^{N} \hat{w}_k(n)u_k(n)) = \hat{w}^T(n)u(n). \quad (2.57)$$

Оскільки

$$\frac{\partial x(n)}{\partial \hat{w}(n)} = u(n), \quad (2.58)$$

то

$$\frac{\partial \varepsilon(n)}{\partial \hat{w}(n)} = -f'(x(n))i(n). \quad (2.59)$$

При підстановці рівностей (2.55) і (2.59) у рекурсивний вираз (2.48) одержимо наступний алгоритм адаптивної корекції ваг:

$$\hat{w}(n+1) = \hat{w}(n) + \eta\varepsilon(n)f'(x(n))u(n), \quad (2.60)$$

або в скалярній формі

$$\hat{w}_k(n+1) = \hat{w}_k(n) + (n)f'(x(n))u_k(n), \quad (2.61)$$

для $k$= 1.....$N$. Якщо $\varepsilon(n)$=1, то функція (2.6) відповідає умові

$$f'(x) = f(x)(1 - f(x)). \quad (2.62)$$

Тому алгоритм (2.60) можна записати у формі

$$\hat{w}_k(n+1) = \hat{w}_k(n) + (n)f(x(n))(1 - f(x(n))u_k(n) \quad (2.63)$$

для $k$=1.....N, де похибка $\varepsilon(n)$ визначається виразом (2.54).

Алгоритми (2.60) і (2.63) покладені в основу методу зворотного поширення похибки, що докладно описується в далі.

## 2.5. Класифікація штучних нейроних мереж

**Класифікація по типу вхідної інформації**

- Аналогові нейроні мережі (використовують інформацію у формі дійсних чисел);
- Двоїсті нейроні мережі (оперують з інформацією, представленої в двоїстому виді).

**Класифікація по характеру навчання**

- З учителем (вихідний простір розв'язань нейроної мережі відомий);
- Без учителя (нейрона мережа формує вихідний простір розв'язань тільки на основі вхідних впливів). Такі мережі називають самоорганізуючими;
- З критиком (система призначення штрафів і заохочень).

**Класифікація по характеру настроювання синапсов**

- Мережі з фіксованими зв'язками (вагові коефіцієнти нейроної мережі вибираються відразу, виходячи з умов задачі, при цьому: $dW/dt=0$, де W — вагові коефіцієнти мережі);
- мережі з динамічними зв'язками (для них у процесі навчання відбувається настроювання синаптичних зв'язків, тобто $dW/dt\neq 0$, де W — вагові коефіцієнти мережі).

**Класифікація по характеру зв'язків**

**Мережі прямого поширення (Feedforward)**

Усі зв'язки спрямовані строго від вхідних нейронів до вихідного. Прикладами таких мереж є одношаровий і багатошаровий перцептрони, мережі Ворда.

**Рекурентні мережі**

Сигнал з вихідних нейронів або нейронів схованого шару частково передається назад на входи нейронів вхідного шару. Як будь-яка система, що має зворотний зв'язок, рекурентна мережа прагне до стійкого стану. Як відомо, найбільш стійкий стан забезпечується мінімалізацією енергії системи. Рекурентна мережа «фільтрує» вхідні дані, повертаючись до стійкого стану і, таким чином, дозволяє розв'язувати задачі компресії даних і

побудови асоціативної пам'яті. Яскравим прикладом таких мереж є мережа Хопфілда.

**Двонаправлені мережі**

У таких мережах між шарами існують зв'язки як у напрямку від вхідного шару до вихідного, так і в зворотному.

**Радіально-базисні функції**

Штучні нейроні мережі, що використовують у якості активаційних функцій радіально-базисні (такі мережі скорочено називаються RBF-мережами). Загальний вид радіально-базисної функції:

$$f(x) = e^{-\frac{x^2}{\sigma^2}}$$

де σ — ширина вікна функції.

Радіально-базисна мережа характеризується трьома особливостями:

1. Єдиний схований шар

2. Тільки нейрони схованого шару мають нелінійну активаційну функцію

3. Синаптичні ваги зв'язків вхідного і схованого шарів дорівнюють одиниці

**Карти, що самоорганізуються**

Такі мережі являють собою, як правило, двовимірну структуру нейронів. Перед навчанням структура випадкова, нейрони розподілені приблизно рівномірно. При навчанні, для

кожного навчального запису розраховується точка, що відповідає їй у структурі мережі. Нейрон, що знаходиться ближче усього до шуканої точки, називається *нейроном-переможцем*. Ваги зв'язків, що з'єднують цей нейрон з іншими, збільшуються, тим самим трохи упорядковуючи структуру. Ваги від нейронів, що є «сусідами» нейрона- переможця, до інших нейронів також збільшуються, але слабкіше і т.д. Таким чином, чим частіше нейрон «перемагає» при порівнянні з ознакою, тим «щільніше» до нього знаходяться інші нейрони. Наприкінці навчання мережа представляє із себе кілька зон зосередження нейронів, називаних *кластерами.*

## 2.6. Етапи побудови мереж

Тепер, коли стало ясно, що саме ми хочемо побудувати, ми можемо переходити до питання "як створити необхідну нейрону мережу". Це питання вирішується в два етапи:

1. На першому етапі здійснюють вибір типу (архітектури) мережі.

2. На другому етапі виконують "навчання" мережі.

На першому етапі варто вибрати наступне:

• які нейрони ми хочемо використовувати (число входів, передатні функції);

• яким чином варто з'єднати їх між собою;

• що взяти за входи і виходи мережі.

Ця задача на перший погляд здається неозорої, але, на щастя, нам необов'язково придумувати нейромережу "з нуля" - існує кілька десятків різних нейромережових архітектур, причому ефективність багатьох з них доведена математично. Найбільш популярні і вивчені архітектури - це багатошаровий перцептрон, нейромережа із загальною регресією, мережі Кохонена й інші.

На другому етапі нам необхідно "навчити" обрану мережу, тобто підібрати такі значення її ваг, щоб мережа працювала потрібним чином. Ненавчена мережа подібна дитині - її можна навчити чому завгодно. У використовуваних на практиці нейромережах кількість ваг може складати кілька десятків тисяч, тому навчання - доволі складний процес. Для багатьох архітектур розроблені спеціальні алгоритми навчання, що

дозволяють настроїти ваги мережі певним чином. Найбільш популярний з цих алгоритмів - метод зворотного поширення похибки (Error Back Propagation), використовуваний, наприклад, для навчання перцептрона.

### 2.6.1. Архітектура штучних нейроних мереж

У загальному випадку поняття “штучна нейрона мережа” охоплює ансамблі нейронів будь-якої структури, однак практичне застосування знайшли тільки деякі з них. Це пояснюється тим, що архітектура ШНМ безпосередньо зв'язана з методом її навчання. Навіть різні етапи розвитку ШНМ визначалися появою нових архітектур мереж і спеціально розроблених для них методів навчання. Можна виділити чотири основні різновиди архітектури ШНМ.

**1. Одношарові прямонаправлені мережі.** Шаровою називається ШНМ, що складається з груп нейронів, розділених по шарах. ШНМ, що містить *k* шарів, називається *k*-шаровою. Якщо сигнали в мережі поширюються тільки по напрямку з початку в кінець, то така ШНМ називається прямонаправленою. На рис. 2.17 зображена одношарова прямонаправлена ШНМ.

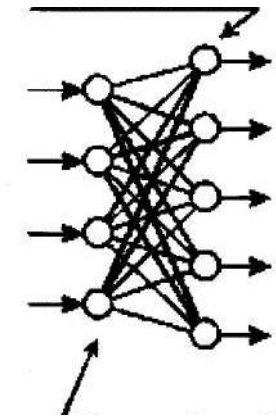
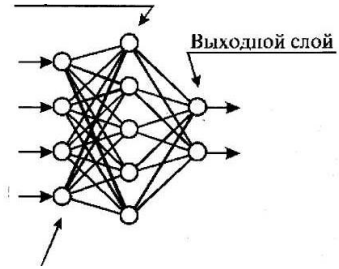



Рис. 2.17. Прямонаправлена мережа з одним шаром нейронів.

Вона містить у собі шар вхідних нейронів і шар вихідних. Нейрони вхідного шару просто ретранслюють сигнали на вихідний шар, не преутворюючи їх. У вихідному шарі відбувається перетворення сигналів і формування реакції мережі. Необхідно відзначити, що нейрони ШНМ також іноді називаються вузлами або обчислювальними модулями. Кількість нейронів в одному шарі визначає розмір шару.На відміну від прийнятої методики позначення, такі ШНМ називаються одношаровими, а не двошаровими. Цим підкреслюється, що обчислення виконуються лише одним шаром мережі. **2. Багатошарові прямонаправлені мережі**. Вони характеризуються наявністю одного або декількох схованих шарів, що здійснюють перетворення інформації. Нейрони схованого шару називаються схованими нейронами або схованими вузлами. Використання схованих шарів дозволяє ШНМ здійснювати нелінійні перетворення вхід-вихід будь-якої складності або витягати з вхідних даних статистичні показники високих порядків. Ці унікальні властивості багатошарових мереж особливо виявляються при високій розмірності простору вхідних сигналів. На рис. 2.18 представлена схема тришарової прямонаправленої ШНМ з одним схованим шаром. Для опису такої мережі використовується запис *NN*3-5-2. Тут 3 — розмір вхідного шару мережі, 5 — схованого, і 2 — вихідного.

Схований шар




Вхідний шар

Рис. 2.18. Цілком зв'язана тришарова прямонаправлена мережа.

У загальному випадку прямонаправлена ШНМ із $g$ вхідними нейронами, $q$ вихідними нейронами і $n$ схованими шарами розміру $h_i$ позначається

$$NNg\text{-}h_1\text{-}h_2\text{-}...\text{-}h_n\text{-}q.$$

Нейрони вхідного шару в таких мережах просто ретранслюють вхідні сигнали на перший схований шар, не преутворюючи їх. У схованих нейронах послідовно, шар за шаром, відбувається нелінійне перетворення сигналів. Сигнали з останнього схованого шару надходять на нейрони вихідного шару, що формують реакцію мережі.

Формально, не існує обмежень на типи активаційних функцій нейронів різних шарів ШНМ або навіть одного шару, однак зазвичай всі сховані нейрони вибираються одного типу. Вихідний шар ШНМ може складатися з нейронів з тим же типом активаційної функції, що й у нейронів схованого шару, але найбільш розповсюдженим є модель прямонаправленої мережі з лінійними вихідними нейронами. ШНМ цього типу з активаційними функціями нейронів схованого шару (1.7, 1.10) називаються багатошаровими перцептронами (БШП). БШП знайшли широке застосування при розв'язанні різних задач і є одним з головних об'єктів теоретичних досліджень.

Зображена на рис. 2.18 ШНМ називається цілком зв'язаною прямонаправленою ШНМ, тому що кожен нейрон з одного шару зв'язаний із усіма нейронами наступного шару. Широке застосування також знаходять частково зв'язані прямонаправлені ШНМ, у яких нейрони одного шару зв'язані тільки з визначеною частиною нейронів наступного шару. Така архітектура дозволяє закласти в ШНМ апріорні знання про бажаний закон обробки сигналів у мережі.

Тришарові прямонаправлені ШНМ широко використовуються для розв'язання задач класифікації, розпізнавання образів, апроксимації і керування.

**3. Рекурентні мережі**. Цей тип ШНМ відрізняється існуванням зворотних зв'язків і елементів тимчасової затримки сигналу. Найбільш простим випадком рекурентної мережі є один шар нейронів, охоплений зворотними зв'язками. При цьому кожен нейрон одержує затримані вихідні сигнали всіх інших нейронів. На рис. 2.19 представлена рекуррентна ШНМ, що містить схований шар нейронів. У цьому випадку кожен нейрон одержує, крім вхідних сигналів, ще і усі вихідні сигнали мережі. Частина ШНМ, яка охоплена зворотними зв'язками, може мати і більшу кількість схованих шарів.

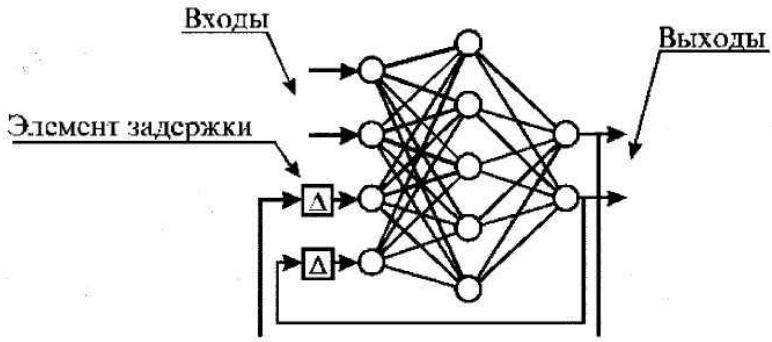



Рис. 2.19. Рекурентна мережа з одним схованим шаром

Наявність зворотних зв'язків і елементів тимчасової затримки сигналів додає рекурентним мережам власні нелінійні динамічні властивості. Це також позначається на їхній здатності до навчання. Тренування рекурентних мереж вимагають урахування їхніх динамічних властивостей. Одним з головних застосувань рекурентних ШНМ є нейроемулятори динамічних об'єктів, тобто їхні нейромережні моделі. Такі мережі можуть також використовуватися для розв’язання задач апроксимації тимчасових послідовностей, класифікації, розпізнавання образів і керування.

**4. Цілком зв'язані мережі.** Характерною ознакою ШНМ цього типу є наявність зв'язків між усіма нейронами. Найбільш відомим різновидом цілком зв'язаних мереж є мережі Хопфіилда (див. рис. 2.20). У них кожен нейрон має двосторонні зв'язки з всіма іншими нейронами мережі. У загальному випадку мережа Хопфілда має симетричну кільцеву структуру, у ній не можна виділити сховані нейрони і єдиний напрямок поширення сигналів. Робота цілком зв'язаної ШНМ і обмін даними контролюється одним головним нейроном.

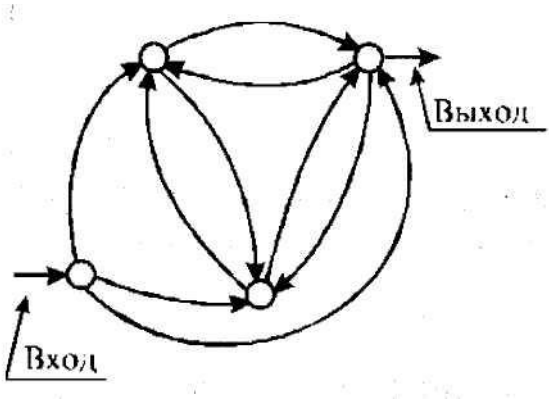



Рис. 2.20. Цілком зв'язана мережа Хопфілда

Мережа Хопфілда являє приклад цілком зв'язаної динамічної мережі, що ґрунтується на принципах самоорганізації, однак у ній не використовуються в явному виді елементи тимчасової затримки. Іншим прикладом служать решіткові мережі (див. рис. 2.21). Вони являють собою масив нейронів, кожний з яких зв'язаний із вхідними нейронами. Розмірність масиву нейронів визначає розмірність решіткової мережі. Така ШНМ є прямонаправленою, тому що в ній немає зворотних зв'язків, однак у ній не можна виділити скриті елементи або шари.

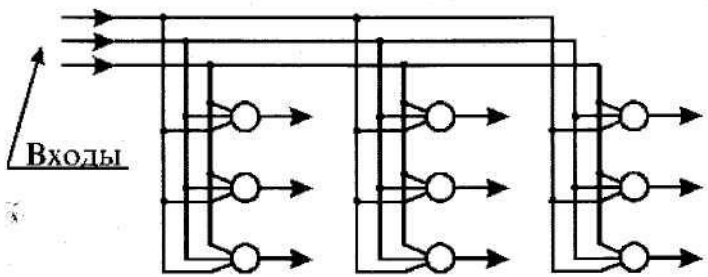



Рис. 2.21. Одномірна решіткова мережа з трьох нейронів

Цілком зв'язані мережі знаходять особливо широке застосування при розв'язанні задач класифікації і розпізнавання образів.

## 2.6.2. Навчання нейромереж

Як і їхні біологічні прообрази, ШНМ можуть навчатися, тобто поліпшувати свою роботу під впливом навколишнього середовища, що змінює її параметри. Існує безліч визначень терміна "навчання", однак стосовно до ШНМ найбільше підходить наступне, дане Менделем і Маклареном:

*Навчання — це процес, при якому вільні параметри нейроної мережі адаптуються в результаті її безперервної стимуляції зовнішнім оточенням. Тип навчання визначається тим способом, яким виробляються зміни параметрів.*

У сучасній літературі крім терміна "навчання" також використовуються рівноправні поняття "тренування мережі" і "настроювання параметрів мережі". У загальному можна виділити два основних види навчання : контрольоване навчання (supervised learning) і самонавчання (self-organized learning). Перший вид має на увазі наявність "учителя", що спостерігає реакцію мережі і направляє зміни її параметрів. В другому випадку мережа самоорганізується під дією зовнішнього середовища і вивчає його самостійно, без допомоги "учителя".

Самонавчання властиве задачам розпізнавання образів і класифікації. При розв’язанні задач керування зазвичай використовується контрольоване навчання ШНМ.

Існує два різновиди контрольованого навчання: пряме контрольоване навчання і стимулююче навчання (reinforcement learning). Так як перший вид з'явився раніш другого і більш розповсюджений, то зазвичай на нього посилаються просто як на контрольоване навчання.

Нейроні мережі не програмуються в звичному змісті цього слова, вони **навчаються**. Можливість навчання — одне з головних переваг нейроних мереж перед традиційними алгоритмами. Технічно навчання полягає в знаходжені коефіцієнтів зв'язків між нейронами. У процесі навчання нейрона мережа здатна виявляти складні залежності між вхідними даними і вихідними, а також виконувати узагальнення. Це значить, що, у випадку успішного навчання, мережа зможе повернути вірний результат на підставі даних, що були відсутні в навчальній вибірці. Навчити нейромережу - значить, повідомити їй, чого ми від неї домагаємося. Цей процес дуже схожий на навчання дитини алфавітові. Показавши дитині зображення букви "А", ми запитуємо її: "Яка це буква?" Якщо відповідь невірна, ми повідомляємо дитині ту відповідь, яку ми хотіли б від неї одержати: "Це буква А". Дитина запам'ятовує цей приклад разом з вірною відповіддю, тобто в її пам'яті відбуваються деякі зміни в потрібному напрямку. Ми будемо повторювати процес пред'явлення букв знову і знову доти, коли всієї 33 букви будуть твердо запам’ятовані. Такий процес називають "навчання з учителем".

При навчанні мережі ми діємо зовсім аналогічно. У нас є деяка база даних, що містить приклади (набір рукописних зображень букв). Пред'являючи зображення букви "А" на вхід мережі, ми одержуємо від неї деяку відповідь, не обов'язково вірну. Нам відома і вірна (бажана) відповідь - у даному випадку нам хотілося б, щоб на виході з міткою "А" рівень сигналу був максимальний. Зазвичай як бажаний вихід у задачі класифікації беруть набір (1, 0, 0,...), де 1 стїть на виході з міткою "А", а 0 - на всіх інших виходах. Обчислюючи різницю між бажаною

відповіддю і реальною відповіддю мережі, ми одержуємо 33 числа - *вектор похибки.* Алгоритм зворотного поширення похибки - це набір формул, що дозволяє по векторі похибки обчислити необхідні виправлення для ваг мережі. Ту саму букву (а також різні зображення однієї і тієї ж букви) ми можемо пред'являти мережі багато разів. У цьому змісті навчання скоріше нагадує повторення вправ у спорті - тренування. На рис. 2.22 приведена схема процесу навчання мережі.

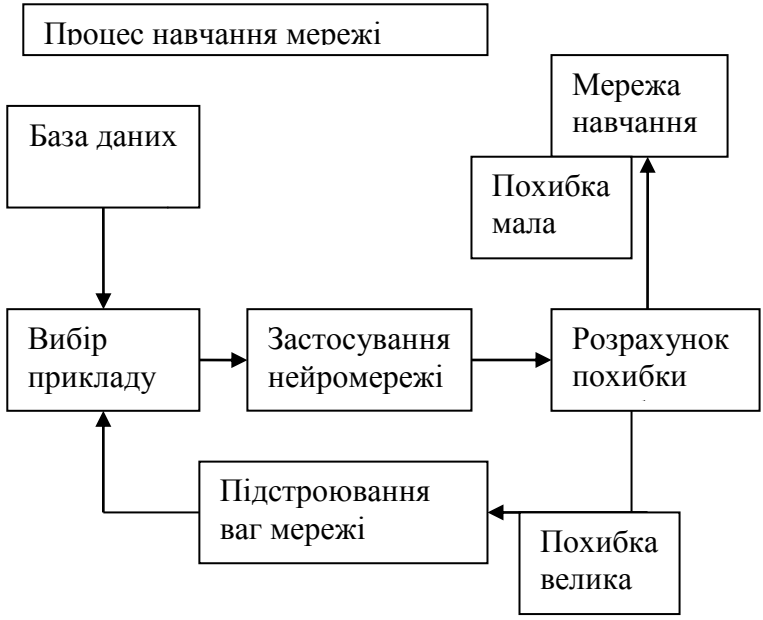



Рис. 2.22. Схема процесу навчання мережі

Після багаторазового пред'явлення прикладів ваги мережі стабілізуються, причому мережа дає правильні відповіді на всі (або майже усі) приклади з бази даних. У такому випадку говорять, що "мережа вивчила всі приклади", " мережа навчена", або "мережа натренована". У програмних реалізаціях можна

бачити, що в процесі навчання величина похибки (сума квадратів похибок по усіх виходах) поступово зменшується. Коли величина похибки досягає нуля або прийнятного малого рівня, тренування зупиняють, а отриману мережу вважають натренованою і готовою до застосування на нових даних.

Важливо відзначити, що вся інформація, яку мережа має про задачу, утримується в наборі прикладів. Тому якість навчання мережі прямо залежить від кількості прикладів у навчальній вибірці, а також від того, наскільки повно ці приклади описують дану задачу. Так, наприклад, безглуздо використовувати мережу для пророкування фінансової кризи, якщо в навчальній вибірці криз не представлено. Вважається, що для повноцінного тренування потрібно хоча б кілька десятків (а краще сотень) прикладів. Як ми вже говорили, що навчання мережі - складний і наукомісткий процес. Алгоритми навчання мають різні параметри і настроювання, для керування якими потрібне розуміння їхнього впливу.

### 2.6.2.1. Алгоритм зворотного поширення похибки

Обговоримо алгоритм зворотного поширення похибки, що дозволяє навчати багатошарові нейроні мережі. Цей алгоритм вважається найбільш відомим і найчастіше застосовуваним у штучних нейроних мережах.

На рис. 2.23. представлена багатошарова нейрона мережа, що складається з $L$ шарів.

У кожнім шарі розташовано $N_k$ елементів, $k$ = 1.....$L$, що позначаються $AD^k_i$, $j$= 1, ..., $N_k$. Елементи $AD^k_i$ будемо називати нейронами, причому кожний з них може бути системою типу Адалайн із нелінійною функцією (сігмоїдою або гіперболічним тангенсом) на виході.

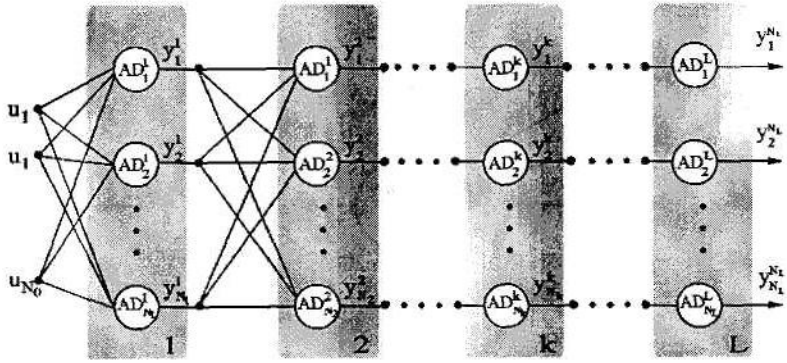


Рис. 2.23. Багатошарова нейрона мережа.

Обговорювана нейронная мережа має $N_0$ входів, на які подаються сигнали $u_1(n),...,u_{N_0}(n)$, записувані у векторній формі як

$$u = [u_1(n), ..., u_{N_0}(n)]^T, n= 1.2. ... \quad (2.64)$$

Вихідний сигнал *i-го* нейрона в *k-м* шарі позначається $y_i^{(k)}(n)$, $i= 1.....N_k$, $k = 1.....L$.

На рис. 2.24 показана детальна структура *i* -го нейрона в *k* -м шарі.

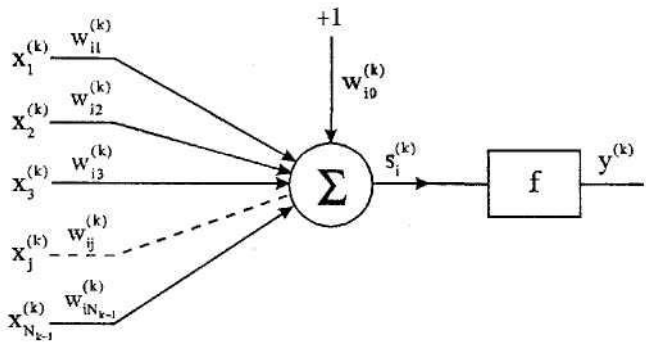


Рис. 2.24. Структура нейрона $AD^k_j$ .

Нейрон $AD^k_i$, має $N_k$ входів, що утворять вектор

$$\boldsymbol{x}^{(k)}(n)=[x_0^{(k)}(n),\ldots,x_{N_{k-1}}^{(k)}(n)]^T, \tag{2.65}$$

причому $x_j^{(k)}(n)$= +1 для $i$ = 0 і $k$ = 1.....L .

Звернемо увагу на факт, що вхідний сигнал нейрона $AD^k_i$ зв'язаний з вихідним сигналом $(k$ - 1)-шару в такий спосіб:

$$x_j^{(k)}(n)=\begin{cases} u_i(n) & \text{для } k=1, \\ y_i^{(k-1)} & \text{для } k=2,\ldots,L, \\ +1 & \text{для } i=0, k=1,\ldots,L. \end{cases} \tag{2.66}$$

На рис. 2.24 $w_{ij}^{(k)}(n)$ позначає вагу зв'язку $i$-го нейрона, $i$=1, ..., $N_k$, розташованого в $k$-у шарі, що з'єднує цей нейрон з $j$-м вхідним сигналом $x_j^{(k)}(n)$, $j$= 0, 1.....$N_{k1}$ Вектор ваг нейрона $AD^k_i$ позначимо

$$\boldsymbol{w}_i^{(k)}(n)=[w_{i,0}^{(k)}(n),\ldots,w_{i,N_{k-1}}^{(k)}(n)]^T,\ k=1,\ \ldots,\ L,\ i=1,\ \ldots,\ N_k. \tag{2.67}$$

Вихідний сигнал нейрона $AD^k_i$ у n-й момент часу, $n$ = 1, 2, ... визначається як

$$y_i^{(k)}(n) = f(s_i^{(k)}(n)), \tag{2.68}$$

причому

$$s_i^{(k)}(n)=\sum_{j=0}^{N_{k-1}} w_{ij}^{(k)}(n)x_j^{(k)}(n). \tag{2.69}$$

Відзначимо, що вихідні сигнали нейронів у L-м шарі

$$y_1^L(n), y_2^L(n),\ldots,y_{N_L}^L(n) \tag{2.70}$$

одночасно є вихідними сигналами всієї мережі. Вони порівнюються з так називаними еталонними сигналами мережі

$$d_1^L(n), d_2^L(n),\ldots,d_{N_L}^L(n), \tag{2.71}$$

у результаті чого одержуємо похибку

$$\varepsilon_i^{(L)}(n)=d_i^{(L)}(n)-y_i^{(L)}(n)\ ,\ i=1,\ \ldots,\ N_L. \tag{2.72}$$

Можна сформулювати міру похибки, засновану на порівнянні сигналів (2.70) і (2.71), у виді суми квадратів різностей (2.72), тобто

$$Q(n)=\sum_{i=1}^{N_L}\varepsilon_i^{(L)^2}(n)=\sum_{i=1}^{N_L}(d_i^{(L)}(n)-y_i^{(L)}(n))^2\ . \tag{2.73}$$

З виразів (2.68) і (2.69) випливає, що міра похибки (2.73) - це функція від ваг мережі. Навчання мережі засноване на адаптивній корекції усіх ваг $w_{ij}^{(k)}(n)$ таким чином, щоб мінімізувати її значення. Для корекції довільної ваги можна використовувати правило найшвидшого спуску, що приймає вид

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) - \eta \frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} \ , \tag{2.74}$$

де константа $\eta>0$ визначає величину кроку корекції. Звернемо увагу, що

$$\frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} = \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} \frac{\partial s_i^{(k)}(n)}{\partial w_{ij}^{(k)}(n)} = \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} x_j^{(k)}(n) \ . \tag{2.75}$$

Якщо ввести позначення

$$\delta_i^{(k)}(n) = -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} \ , \tag{2.76}$$

то одержимо рівність

$$\frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} = -2\delta_i^{(k)}(n) x_j^{(k)}(n) \ . \tag{2.77}$$

При цьому алгоритм (2.74) приймає вид

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta\delta_i^{(k)}(n) x_j^{(k)}(n) \ . \tag{2.78}$$

Спосіб розрахунку значення $\delta_i^{(k)}(n)$, заданого виразом (2.76), залежить від номера шару. Для останнього шару одержуємо

$$\begin{aligned} \delta_i^{(L)}(n) &= -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2} \frac{\partial \sum_{m=1}^{N_L} \varepsilon_m^{(L)^2}(n)}{\partial s_i^{(L)}(n)} = \\ &= -\frac{1}{2} \frac{\partial \varepsilon_i^{(L)^2}(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2} \frac{\partial (d_i^{(L)}(n) - y_i^{(L)}(n))^2}{\partial s_i^{(L)}(n)} = \\ &= \varepsilon_i^{(L)}(n) \frac{\partial y_i^{(L)}(n)}{\partial s_i^{(L)}(n)} = \varepsilon_i^{(L)}(n) f'(s_i^{(L)}(n)) \ . \end{aligned} \tag{2.79}$$

Для довільного шару $k \neq L$ одержуємо

$$\delta_i^{(k)}(n) = -\frac{1}{2}\frac{\partial Q(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2}\sum_{m=1}^{N_{k+1}}\frac{\partial Q(n)}{\partial s_m^{(k+1)}(n)}\frac{\partial s_m^{(k+1)}(n)}{\partial s_i^{(k)}(n)} =$$

$$= \sum_{m=1}^{N_{k+1}}\delta_m^{(k+1)}(n)w_{mi}^{(k+1)}(n)f'(s_i^{(k)}(n)) =$$

$$= f'(s_i^{(k)}(n))\sum_{m=1}^{N_{k+1}}\delta_m^{(k+1)}(n)w_{mi}^{(k+1)}(n) . \tag{2.80}$$

Визначимо похибку у *k*-у (не останньому) шарі для і-го нейрона у виді

$$\varepsilon_i^{(k)}(n) = \sum_{m=1}^{N_{k+1}}\delta_m^{(k+1)}(n)w_{mi}^{(k+1)}(n),\ k = 1, \ldots, L-1 . \tag{2.81}$$

Якщо підставити вираз (2.81) у формулу (2.80), то одержимо

$$\delta_i^{(k)}(n) = \varepsilon_i^{(k)}(n)f'(s_i^{(k)}(n)) . \tag{2.82}$$

У результаті алгоритм зворотного поширення похибки можна записати у виді

$$y_i^{(k)}(n) = f(s_i^{(k)}(n)), \qquad s_i^{(k)}(n) = \sum_{j=0}^{N_{k-1}}w_{ij}^{(k)}(n)x_j^{(k)}(n) , \tag{2.83}$$

$$\varepsilon_i^{(k)}(n) = \begin{cases} d_i^{(L)}(n) - y_i^{(L)}(n) & \text{для } k = L , \\ \sum_{m=1}^{N_{k+1}}\delta_m^{(k+1)}(n)w_{mi}^{(k+1)}(n) & \text{для } k = 1,\ldots,L-1 , \end{cases} \tag{2.84}$$

$$\delta_i^{(k)}(n) = \varepsilon_i^{(k)}(n)f'(s_i^{(k)}(n)) , \tag{2.85}$$

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta\delta_i^{(k)}(n)x_j^{(k)}(n) . \tag{2.86}$$

Назва алгоритму зв'язана зі способом розрахунку похибок у конкретних шарах. Спочатку розраховуються похибки в останньому шарі (на основі вихідних і еталонних сигналів), далі - у передостанньому і так аж до першого шару. Початкові значення ваг, що утворять мережу, вибираються випадковим чином і, як правило, установлюються близькими до нуля. Крок корекції $\eta$ найчастіше приймає великі значення (близькі одиниці) на початкових етапах процесу навчання, але згодом його варто зменшувати в міру того як ваги наближаються до деяких

заздалегідь визначених значенн. У літературі, присвяченій нейроним мережам, рекомендуються різні модифікації алгоритму зворотного поширення похибки. Одна з найбільш відомих модифікацій полягає у введенні в рекурсію (2.86) додаткового члена, називаного *моментом:*

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta\varepsilon_i^{(k)}(n)f'(s_i^{(k)}(n))x_j^{(k)}(n) + \\ + \alpha[w_{ij}^{(k)}(n) - w_{ij}^{(k)}(n-1)] , \tag{2.87}$$

у якому параметр $\alpha \in (0,1)$. Експериментальні дослідження показують, що введення *моменту* прискорює збіжність алгоритму зворотного поширення похибки.

### 2.6.2.2. Застосування рекурентного методу найменших квадратів для навчання нейроних мереж

Навчання мережі з використанням викладеного в п. 2.6.2.1 алгоритм зворотного поширення похибки вимагає великої кількості ітерацій. Тому в літературних джерелах приводяться відомості про різні спроби створення більш швидких алгоритмів для навчання нейроних мереж, одним із яких є рекурентний *метод найменших квадратів (recursive least squares* - RLS). Як міру похибки використовують вираз

$$Q(n) = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \varepsilon_j^{(L)^2}(t) = \\ = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} [d_j^{(L)}(t) - f(\boldsymbol{x}^{(L)^T}(t)\boldsymbol{w}_j^{(L)}(n))]^2 , \tag{2.88}$$

де λ - так називаний *коефіцієнт забування (forgetting factor),* значення якого вибирається з інтервалу [0, 1]. Звернемо увагу на те, що степінь впливу членів виразу (2.88) на його значення зростає зі збільшенням номера члена. У ході подальших міркувань будемо використовувати позначення, введені в п. 2.6.2.1, з урахуванням особливостей, показаних на рис. 2.25, тобто

$$\varepsilon_i^{(k)}(t) = d_i^{(k)}(t) - y_i^{(k)}(t) , \tag{2.89}$$

а також

$$b_i^{(k)}(t) = f^{-1}(d_i^{(k)}(t)), \tag{2.90}$$

де *f*- обратима функція, *t* =1.....*n*, *j*= 1, ..., $N_k$, *k*= 1, ..., *L*.

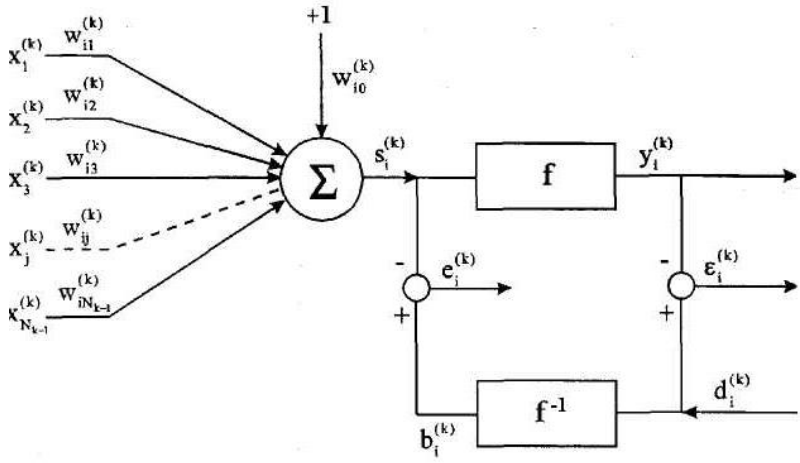



Рис. 2.25. Структура нейрона, застосовуваного для реалізації алгоритму RLS.

Якщо розрахувати градієнт міри похибки і прирівняти його до нуля, то одержимо рівняння

$$\frac{\partial Q(n)}{\partial \boldsymbol{w}_i^{(k)}(n)} = 2\sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \frac{\partial \varepsilon_j^{(L)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)} \varepsilon_j^{(L)}(t) =$$

$$= -2\sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \frac{\partial y_j^{(L)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)} \varepsilon_j^{(L)}(t) = \boldsymbol{0} \ . \tag{2.91}$$

При використанні залежностей (2.68) і (2.69) рівняння (2.91) приймає вид

$$\sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\frac{\partial y_j^{(L)}(t)}{\partial s_j^{(L)}(t)}\sum_{p=1}^{N_{L-1}}\frac{\partial s_t^{(L)}(t)}{\partial y_p^{(L-1)}(t)}\frac{\partial y_p^{(L-1)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)}\varepsilon_j^{(L)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{p=1}^{N_{L-1}}\frac{\partial y_p^{(L-1)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)}\sum_{j=1}^{N_L}\frac{\partial y_j^{(L-1)}(t)}{\partial s_j^{(L)}(t)}w_{jp}^{(L)}\varepsilon_j^{(L)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{p=1}^{N_{L-1}}\frac{\partial y_p^{(L-1)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)}\varepsilon_p^{(L-1)}(t)=\sum_{t=1}^{n}\lambda^{n-t}\sum_{q=1}^{N_k}\frac{\partial y_p^{(k)}(t)}{\partial w_i^{(k)}(n)}\varepsilon_q^{(k)}(t)=\mathbf{0}\ , \tag{2.92}$$

де

$$\varepsilon_p^{(k)}(t)=\sum_{j=1}^{N_{k+1}}\frac{\partial y_j^{(k+1)}(t)}{\partial s_j^{(k+1)}(t)}w_{jp}^{(k+1)}(n)\varepsilon_j^{(k+1)}(t)\ . \tag{2.93}$$

Вираз (2.93) задає спосіб послідовного визначення похибок у кожнім шарі, починаючи з останнього. При подальших перетвореннях одержуємо послідовність рівностей виду

$$\sum_{t=1}^{n}\lambda^{n-t}\sum_{q=1}^{N_k}\frac{\partial y_q^{(k)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)}\varepsilon_q^{(k)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{q=1}^{N_k}\frac{\partial y_q^{(k)}(t)}{\partial s_q^{(k)}(n)}\frac{\partial s_q^{(k)}(t)}{\partial \boldsymbol{w}_i^{(k)}(n)}\varepsilon_q^{(k)}(t)=$$

$$=\sum_{j=1}^{n}\lambda^{n-t}\frac{\partial y_i^{(k)}(t)}{\partial s_i^{(k)}(n)}\boldsymbol{y}^{(k-1)^T}(t)\varepsilon_i^{(k)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-1}\frac{\partial y_i^{(k)}(t)}{\partial s_i^{(k)}(n)}y^{(k-1)^T}(t)[d_i^{(k)}(t)-y_i^{(k)}(t)]=\mathbf{0}\ , \tag{2.94}$$

де

$$\boldsymbol{y}^{(k)}=[y_1^{(k)},\ldots,y_{N_k}^{(k)}]^T\ .$$

При використанні апроксимації

$$f(b_i^{(k)}(t))=f(s_i^{(k)}(t))+f'(s_i^{(k)}(t))(b_i^{(k)}(t)-s_i^{(k)}(t)) \tag{2.95}$$

одержуємо нормальне рівняння

$$\sum_{t=1}^{n} \lambda^{n-t} f'^2(s_i^{(k)}(t))[b_i^{(k)}(t) - \boldsymbol{x}^{(k)^T}(t)\boldsymbol{w}_i^{(k)}(n)]\boldsymbol{x}^{(k)^T}(t) = \mathbf{0} \ , \tag{2.96}$$

векторна форма якого має вигляд

$$\boldsymbol{r}_i^{(k)}(n) = \mathbf{R}_i^{(k)}(n)\boldsymbol{w}_i^{(k)}(n) \ , \tag{2.97}$$

де

$$\mathbf{R}_i^{(k)}(n) = \sum_{t=1}^{n} \lambda^{n-t} f'^2(s_i^{(k)}(t))\boldsymbol{x}^{(k)}(t)\boldsymbol{x}^{(k)^T}(t) \ , \tag{2.98}$$

$$\mathbf{r}_i^{(k)}(n) = \sum_{t=1}^{n} \lambda^{n-t} f'^2(s_i^{(k)}(t))b_i^{(k)}(t)\boldsymbol{x}^{(k)}(t) \ . \tag{2.99}$$

Рівняння (2.97) можна розв'язати рекурентним способом, без інвертування матриці $R_i^{(k)}(n)$. Це вимагає використання алгоритму RLS, відповідно до якого адаптивна корекція усіх ваг $w_i^{(k)}$ виробляється відповідно до правил

$$\varepsilon_i^{(k)}(n) = \begin{cases} d_i^{(L)}(n) - y_i^{(L)}(n) & \text{для } k = L \ , \\ \sum_{j=1}^{N_{k+1}} f'(s_j^{(k+1)}(n))w_{ji}^{(k+1)}(n)e_j^{(k+1)}(n) & \text{для } k = 1,\ldots,L-1 \ , \end{cases} \tag{2.100}$$

$$\boldsymbol{g}_i^{(k)}(n) = \frac{f'(s_i^{(k)}(n))\mathbf{P}_i^{(k)}(n-1)\boldsymbol{x}^{(k)}(n)}{\lambda + f'^2(s_i^{(k)}(n))\boldsymbol{x}^{(k)^T}(n)\mathbf{P}_i^{(k)}(n-1)\boldsymbol{x}^{(k)}(n)} \ , \tag{2.101}$$

$$\mathbf{P}_i^{(k)}(n) = \lambda^{-1}[\mathbf{I} - f'(s_i^{(k)}(n))\boldsymbol{g}_i^{(k)}(n)\mathrm{x}^{(k)^T}(n)]\mathbf{P}_i^{(k)}(n-1) \ , \tag{2.102}$$

$$\boldsymbol{w}_i^{(k)}(n) = \boldsymbol{w}_i^{(k)}(n-1) + \boldsymbol{g}_i^{(k)}(n)\varepsilon_i^{(k)}(n) \ , \tag{2.103}$$

де $i$=1, ..., $N_k$, $k$= 1, ..., $L$.

Початкові значення в алгоритмі RLS, як правило, встановлюються в такий спосіб:

$$\mathrm{P}^{(k)}(0) = \delta\mathrm{I}, \ \delta \gg 0 \ , \tag{2.104}$$

$$w_i^{(k)}(0) = 0. \tag{2.105}$$

Початкові значення ваг $w_i^{(k)}(0)$ нейроної мережі можуть також вибиратися випадковим чином із заздалегідь установленого діапазону.

**Приклад.** Порівняємо функціонування алгоритму зворотного поширення похибки (2.86), модифікованого алгоритму (2.87) і алгоритму RLS (2.103). Для цього двошарову нейрону мережу із

сігмоїдальними функціями будемо використовувати для імітації логічної системи XOR і декодера 4-2-4. Процес імітації повинний тривати досить довго для того, щоб значення похибки *Q(n)* стало менше заданого порога *γ,* тобто

$$Q(n) = \sum_{i=1}^{N_L} (d_i^{(L)}(n) - y_i^{(L)}(n))^2 \le \gamma\,, \tag{2.106}$$

де $N_L$=1 у випадку логічної системи XOR і $N_L$=4 у випадку декодера 4-2-4. На малюнках, що ілюструє результати моделювання, під *епохою* (ер) розуміється кількість ітерацій, яка рівна числу різних пар векторів вхідних і еталонних сигналів (один цикл пред'явлення навчальної вибірки). В обох прикладах кожна епоха складається з чотирьох ітерацій навчального алгоритму.

а) *Логічна система XOR*. Нейрона мережа має 2 входи, 2 нейрони в схованому шарі і 1 вихід. Заданий поріг *γ* дорівнює 0.02. В окремі епохи виділено наступні пари векторів вхідних і еталонних сигналів:

(0,0; 0), (0,1; 1), (1,0; 1), (1,1; 0).

Результати моделювання представлені на рис. 2.26.

б) *Декодер 4-2-4.* Нейрона мережа має 4 входи, 2 нейрони в схованому шарі і 4 нейрони у вихідному шарі. Заданий поріг *γ* дорівнює 0,02. В окремі епохи виділено наступні пари векторів вхідних і еталонних сигналів:

(1,0,0,0; 1,0,0,0) (0,1,0,0; 0,1,0,0), (0,0,1,0; 0,0,1,0) (0,0,0,1; 0,0,0,1).

Результати моделювання представлені на рис. 2.27.

Легко помітити, що модифікований (з урахуванням *моменту)* алгоритм зворотного поширення похибки працює в кілька разів швидше традиційного алгоритму, тоді як застосування алгоритму RLS дозволяє збільшити цю швидкість ще на порядок.

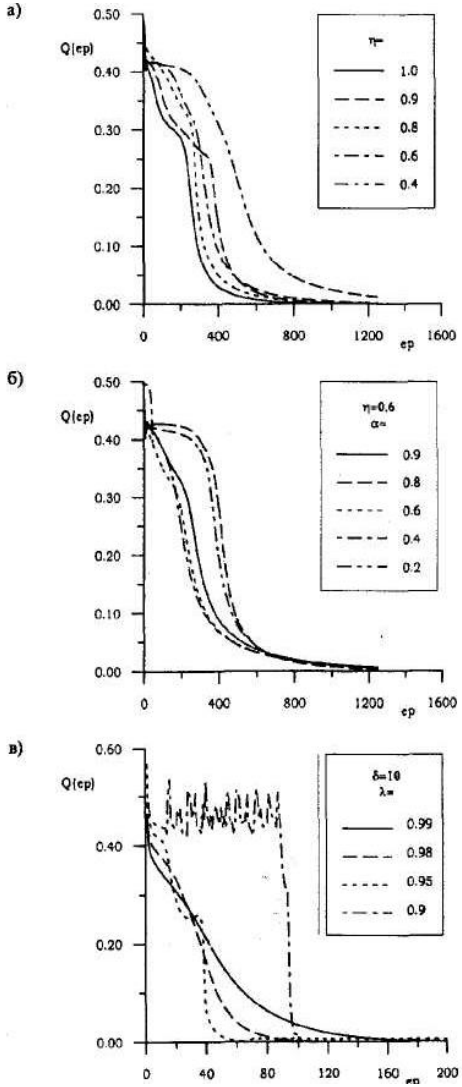



Рис. 2.26. Результати моделювання логічної системи XOR: а) алгоритм зворотного поширення похибки; б) модифікований алгоритм зворотного поширення похибки (з урахуванням *моменту);* в) алгоритм RLS.

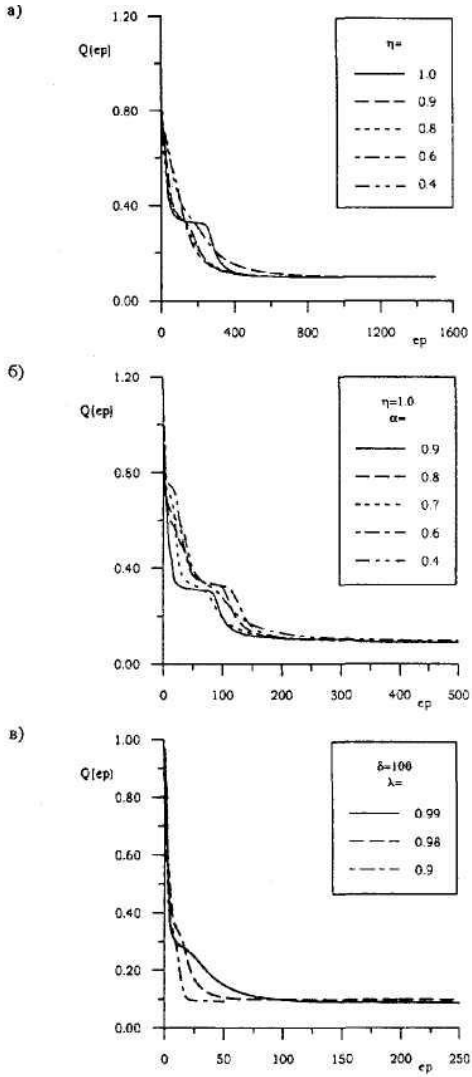



Рис. 2.27. Результати моделювання декодера 4-2-4: а) алгоритм зворотного поширення похибки; б) модифікований алгоритм зворотного поширення похибки (з урахуванням *моменту);* в) алгоритм RLS.

## 2.7. Застосування нейромереж

Нейромережі не можна вважати універсальними для розв'язання всіх обчислювальних проблем. Комп'ютери й обчислювальні методи є ідеальними для багатьох застосувань. Комп'ютери перевершують людину по здатності робити числові і символьні обчислення. Однак людина може без зусиль вирішувати складні задачі сприйняття зовнішніх даних (наприклад, пізнавання людини в юрбі по її особистості) з такою швидкістю і точністю, що наймогутніший комп'ютер у порівнянні з нею здається тугодумом. У таблиці приведено характеристики комп'ютера з архітектурою фон Неймана в порівнянні з нейроною мережею

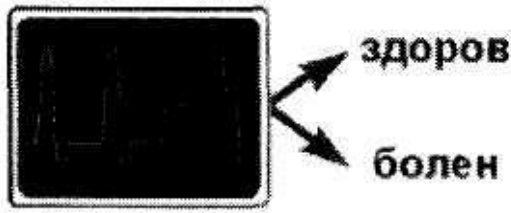
| ** | Комп'ютер | Нейрона мережа |
|---|---|---|
| Процесор | Складний<br>Високошвидкісний<br>Один або декілька | Простий<br>Низькошвидкісний<br>Велика кількість |
| Пам'ять | Окремо від процесора<br>Локалізована<br>Адресація за адресою | У процесорі<br>Паралельні<br>Адресація за змістом |
| Обчислення | Централізовані<br>Послідовні<br>Збережені програми | Паралельні<br>Паралельні<br>Самонавчання |
| Надійність | Уразливість | Живучість |
| Спеціалізація | Числові операції | Проблеми сприйняття |
| Середовище функціонування | Строго визначене<br>Строго обмежене | Погано визначене<br>Без обмежень |
| Функції | Логічно, через правила, концепції, обчислення | Через зображення, малюнки, керування |
| Метод навчання | За правилами | По прикладах |
| Застосування | Числова обробка | Розпізнавання мови, розпізнавання бразів, |

| | | |
|---|---|---|
| | інформації | розпізнавання текстів |

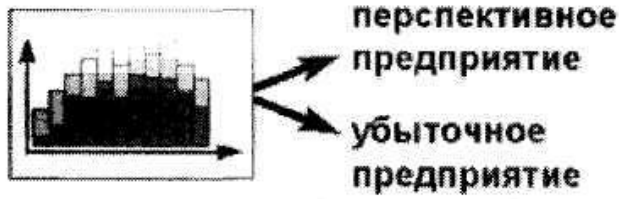
Розглянемо докладніше застосування нейромереж для розв’язання практичних задач.

**Распізнавання образів і класифікація**

Як образи можуть виступати різні по своїй природі об'єкти: символи тексту, зображення, зразки звуків і т.д. При навчанні мережі пропонуються різні зразки образів із указівкою того, до якого класу вони відносяться. Зразок, як правило, представляється як вектор із його ознак. При цьому сукупність всіх ознак повинна *однозначно визначати клас*, до якого відноситься зразок. У випадку, якщо ознак недостатньо, мережа може співвіднести той самий зразок з декількома класами, що невірно. По закінченні навчання мережі можна пред'являти невідомі їй раніше образи й одержувати від неї відповідь про приналежність до визначеного класу. Топологія такої мережі характеризується тим, що кількість нейронів у вихідному шарі, як правило, дорівнює кількості обумовлених класів. При цьому установлюється відповідність між виходом нейроної мережі і класом, що він представляє. Коли мережі пред'являється якийсь образ, на одному з її виходів повинна з'явитися ознака того, що образ належить цьому класові. У той же час на інших виходах повинна бути ознака того, що образ даному класові не належить. Якщо на двох або більш виходах є ознака приналежності до класу, вважається що мережа «не упевнена» у своїй відповіді. Відзначимо, що задачі класифікації (типу розпізнавання букв) дуже погано алгоритмизуются. Якщо у випадку розпізнавання букв вірна відповідь очевидна для нас заздалегідь, то в більш складних практичних задачах навчена нейромережа виступає як експерт, що володіє великим досвідом і здатний дати відповідь на важке запитання. Прикладом такої задачі служить медична діагностика, де мережа може враховувати велику кількість числових параметрів (енцефалограма, тиск, вага і т.д.). Звичайно, "думку" мережі в цьому випадку не можна вважати остаточним.




Класифікація підприємств по ступені їхньої перспективності - це вже звичний спосіб використання нейромереж у практиці різних компаній. При цьому мережа також використовує множину економічних показників, складним чином зв'язаних між собою.




Нейромережний підхід особливо ефективний у задачах експертної оцінки з тієї причини, що він сполучає у собі здатність комп'ютера до обробки чисел і здатність мозку до узагальнення і розпізнавання. Говорять, що в гарного лікаря здатність до розпізнавання у своїй області настільки велика, що він може провести приблизну діагностику вже по зовнішньому вигляді пацієнта. Можна погодитися також, що досвідчений трейдер почуває напрямок руху ринку по виду графіка. Однак у першому випадку усі фактори наочні, тобто характеристики пацієнта миттєво сприймаються мозком як "бліда особа", "блиск в очах" і т.д. У другому ж випадку враховується тільки один фактор, показаний на графіку - курс за визначений період часу. Нейромережа дозволяє обробляти величезну кількість факторів (до декількох тисяч), незалежно від їхньої наочності - це універсальний "гарний лікар", що може поставити свій діагноз у будь-якій області.

**Кластеризація**

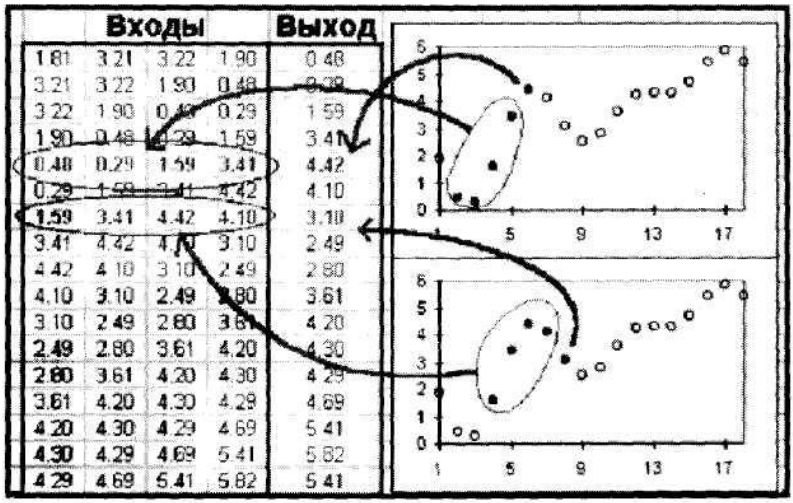
Крім задач класифікації, нейромережі широко використовуються для пошуку залежностей у даних і кластеризації.

Наприклад, нейромережа на основі методики МГОА (метод групового обліку аргументів) дозволяє на основі навчальної вибірки побудувати залежність одного параметра від інших у виді полінома $y= x_1^3 - 4x_3\ ^2x_8^5 + x_3^2$. Така мережа може не тільки миттєво вивчити таблицю множення, але і знайти складні сховані залежності в даних (наприклад, фінансових), що не виявляються стандартними статистичними методами. Під *кластеризацією* розуміється розбивка множини вхідних сигналів на класи, при тім, що ні кількість, ні ознаки класів заздалегідь невідомі. Після навчання така мережа здатна визначати, до якого класу відноситься вхідний сигнал. Мережа також може сигналізувати про те, що вхідний сигнал не відноситься до жодного з виділених класів — це є ознакою нових, відсутніх у навчальній вибірці, даних. Таким чином, подібна мережа *може виявляти нові, невідомі раніше класи сигналів*. Відповідність між класами, виділеними мережею, і класами, що існують у предметній області, установлюється людиною. При розв'язанні задачі кластеризації, множина, що навчає, не має міток класів. Алгоритм кластеризації заснований на подобі образів і поміщає схожі образи в один кластер. Відомі випадки застосування кластеризації для добутку знань, стиску даних і дослідження властивостей даних.

**Прогнозування і апроксимація**

Здатності нейроної мережі до прогнозування прямо випливають з її здатності до узагальнення і виділення схованих залежностей між вхідними і вихідними даними. Після навчання мережа здатна пророчити майбутнє значення якоїсь послідовності на основі декількох попередніх значень і/або якихось існуючих у даний момент факторів. Слід зазначити, що прогнозування можливе тільки тоді, коли *попередні зміни дійсно в якомусь степені визначають майбутні*. Наприклад, прогнозування котирувань акцій на основі котирувань за минулий тиждень може виявитися успішним (а може і не виявитися), тоді як прогнозування результатів завтрашньої лотереї на основі даних за останні 50 років майже напевно не дасть ніяких результатів. Задачі

прогнозування особливо важливі для практики, зокрема, для фінансових додатків, тому пояснимо способи застосування нейромереж у цій області більш докладно. Розглянемо практичну задачу, відповідь у якій неочевидна - задачу прогнозування курсу акцій на 1 день уперед. Нехай у нас є база даних, що містить значення курсу за останні 300 днів. Найпростіший варіант у даному випадку - спробувати побудувати прогноз завтрашньої ієни на основі курсів за останні кілька днів. Зрозуміло, що прогнозуюча мережа повинна мати всього один вихід і стільки входів, скільки попередніх значень ми хочемо використовувати для прогнозу - наприклад, 4 останні значення. Скласти навчальний приклад дуже просто - вхідними значеннями будуть курси за 4 послідовних дні, а бажаним виходом - відомий нам курс у наступний день за цими чотирма.Якщо нейросеть сумісна з якою-небудь системою обробки електронних таблиць (наприклад, Excel), то підготовка навчальної вибірки складається з наступних операцій:
1. Скопіювати стовпець даних значень котирувань у 4 сусідніх стовпця.
2. Зрушити другий стовпець на 1 очко нагору, третій стовпець - на 2 очки нагору і т.д.




Зміст цієї підготовки легко побачити на рисунку - тепер кожен рядок таблиці являє собою навчальний приклад, де перші 4

числа - вхідні значення мережі, а п'яте число - бажане значення виходу. Виключення складають останні 4 рядка, де даних недостатньо - ці рядки не враховуються при тренуванні. Помітимо, що в четвертому знизу рядку задані всі 4 вхідні значення, але невідоме значення виходу. Саме до цього рядка ми застосуємо навчену мережу й одержимо прогноз наступного дня.

Як видно з цього приклада, обсяг навчальної вибірки залежить від обраної нами кількості входів. Якщо зробити 299 входів, то така мережа потенційно могла б будувати кращий прогноз, чим мережа з 4 входами, однак у цьому випадку ми маємо всього 1 навчальний приклад, і навчання безглузде. При виборі числа входів варто враховувати це, вибираючи розумний компроміс між глибиною пророкування (число входів) і якістю навчання (обсяг тренувального набору).

*Апроксимація функцій*. Припустимо, що є навчальна вибірка $((x_1,y_1), (x_2,y_2)..., (x_n,y_n))$ (пари даних вхід-вихід), що генерується невідомою функцією F, перекрученої шумом. Завдання апроксимації полягає в знаходжені невідомої функції F. Апроксимація функцій необхідна при розв'язанні численних інженерних і наукових задач моделювання.

**Прийняття рішень і керування**

Ця задача близька до задачі класифікації. Класифікації підлягають ситуації, характеристики яких надходять на вхід нейроної мережі. На виході мережі при цьому повинна з'явиться ознака рішення, яку вона прийняла. При цьому як вхідні сигнали використовуються різні критерії опису стану керованої системи. Розглянемо динамічну систему, задану сукупністю {u(t), y(t)}, де u(t) - вхідний керуючий вплив, а y(t) - вихід системи в момент часу t. У системах керування з еталонною моделлю метою керування є розрахунок такого вхідного впливу u(t), при якому система діє по бажаній траєкторії, яка задана еталонною моделлю. Прикладом є оптимальне керування двигуном.

**Стиск даних і асоціативна память**

Здатність нейромереж до виявлення взаємозв'язків між різними параметрами дає можливість виразити дані великої розмірності більш компактно, якщо дані тісно взаємозалежні одні з одними. Зворотний процес — відновлення вихідного набору даних з частини інформації — називається (авто)*асоціативною пам'яттю*. Асоціативна пам'ять дозволяє також відновлювати вихідний сигнал/образ із зашумлених/ушкоджених вхідних даних. Розв'язання задачі гетероасоціативної пам'яті дозволяє реалізувати пам'ять, яка адресована по вмісту.

У традиційних комп'ютерах звертання до пам'яті доступно тільки за допомогою адреси, що не залежить від змісту пам'яті. Більш того, якщо допущено помилку в обчисленні адреси, то може бути знайдена зовсім інша інформація. Асоціативна пам'ять або пам'ять, що адресується за змістом, доступна за вказівкою заданого змісту. Уміст пам'яті може бути викликано навіть по частковому вході або ушкодженому змісті. Асоціативна пам'ять може бути використана в мультимедійних інформаційних базах даних.

**Оптимізація.**

Численні проблеми в математиці, статистиці, техніці, науці, медицині й економіці можуть розглядатися як проблеми оптимізації. Задачею алгоритму оптимізації є знаходження такого рішення, що задовольняє системі обмежень і максимізує або мінімізує цільову функцію.

Незважаючи на переваги нейроних мереж в окремих областях над традиційними обчисленнями, що існують, нейромережі не є досконалими рішеннями. Вони навчаються і можуть робити "помилки". Крім того, не можна гарантувати, що розроблена мережа буде оптимальною мережею. Застосування нейромереж жадає від розроблювача виконання ряду умов:

- множину даних, що містять інформацію, яка характеризує проблему;

- відповідно встановлена по розмірах множина даних для навчання і тестування мережі;

- розуміння базової природи розв'язуваної проблеми;

- вибір функції суматора, передатної функції і методів навчання;

- розуміння інструментальних засобів розроблювача;

- відповідна потужність обробки.

Нові можливості обчислень вимагає умінь розроблювача поза границями традиційних обчислень. Спочатку обчислення були лише апаратними й інженери зробили їх працюючими. Потім, були фахівці з програмного забезпечення: програмісти, системні інженери, фахівці з баз даних і проектувальники. Тепер з'явилися нейроні архітектори. Новий професіонал повинний мати кваліфікацію, вищу чим у попередників. Наприклад, він повинний знати статистику для вибору й оцінювання навчальних і тестових множин.

При створенні ефективних нейромереж, важливим для сучасних інженерів програмного забезпечення є логічне мислення, емпіричне уміння й інтуїція.

### 2.8. Основні етапи розв'язання задач

- Збір даних для навчання;
- Підготовка і нормалізація даних;
- Вибір топології мережі;
- Експериментальний підбір характеристик мережі;
- Експериментальний підбір параметрів навчання;
- Власне навчання;
- Перевірка адекватності навчання;
- Коректування параметрів, остаточне навчання;
- Вербалізація мережі з метою подальшого використання.

Розглянемо докладніше деякі з цих етапів.

**Збір даних для навчання**

Вибір даних для навчання мережі і їхня обробка є самим складним етапом розв'язання задачі. Набір даних для навчання повинний задовольняти декільком критеріям:

- Репрезентативність — дані повинні ілюструвати дійсне положення речей у предметній області;
- Несуперечність — суперечливі дані в навчальній вибірці призведуть до поганої якості навчання мережі;
- Обсяг — як правило, число записів у вибірці повинне на кілька порядків перевершувати кількість зв'язків між нейронами в мережі. У противному випадку мережа просто «запам'ятає» усю навчальну вибірку і не зможе виконати узагальнення.

**Підготовка і нормалізація даних**

Вихідні дані перетворяться до виду, у якому їх можна подати на входи мережі. Кожен запис у файлі даних називається *навчальною парою* або *навчальним вектором*. Навчальний вектор містить по одному значенню на кожен вхід мережі і, у залежнсті від типу навчання (із вчителем або без), по одному значенню для кожного виходу мережі. Навчання мережі на «сирому» наборі, як правило, не дає якісних результатів. Існує ряд способів поліпшити «сприйняття» мережі.

- *Нормування* виконується, коли на різні входи подаються дані різної розмірності. Наприклад, на перший вхід мережі подаються величини зі значеннями від нуля до одиниці, а на другий — від ста до тисячі. При відсутності нормування значення на другому вході будуть завжди робити істотно більший вплив на вихід мережі, чим значення на першому вході. При нормуванні розмірності усіх вхідних і вихідних даних зводяться воєдино;

- *Квантування* виконується над безперервними величинами, для яких виділяється кінцевий набір дискретних значень. Наприклад, квантування використовують для завдання частот звукових сигналів при розпізнаванні мови;
- *Фільтрація* виконується для «зашумлених» даних.

Крім того, велику роль грає саме представлення як вхідних, так і вихідних даних. Припустимо, мережа навчається розпізнаванню букв на зображеннях і має один числовий вихід — номер букви в алфавіті. У цьому випадку мережа одержить неправильне уявлення про те, що букви з номерами 1 і 2 більш схожі, чим букви з номерами 1 і 3, що, загалом, невірно. Для того, щоб уникнути такої ситуації, використовують топологію мережі з великим числом виходів, коли кожен вихід має свій зміст. Чим більше виходів у мережі, тим більша відстань між класами і тим складніше їх поплутати.

**Вибір топології мережі**

Вибирати тип мережі необхідно виходячи з постановки задачі і наявних даних для навчання. Для навчання з учителем потрібна наявність для кожного елемента вибірки «експертної» оцінки. Іноді одержання такої оцінки для великого масиву даних просто неможливо. У цих випадках природним вибором є мережа, що навчається без учителя, наприклад, така як *самоорганізуюча карта Кохонена* або *нейрона мережа Хопфілда*. При розв'язанні інших задач, таких як прогнозування тимчасових рядів, експертна оцінка вже утримується у вихідних даних і може бути виділена при їхній обробці. У цьому випадку можна використовувати *многошаровий перцептрон* або *мережу Ворда*.

**Експериментальний підбір характеристик мережі**

Після вибору загальної структури потрібно експериментально підібрати параметри мережі. Для мереж, подібних перцептрону, це буде число шарів, число блоків у

схованих шарах (для мереж Ворда), наявність або відсутність обхідних з'єднань, передатні функції нейронів. При виборі кількості шарів і нейронів у них варто виходити з того, що *здатності мережі до узагальнення тим вище, чим більше сумарне число зв'язків між нейронами*. З іншого боку, число зв'язків обмежене зверху кількістю записів у навчальних даних.

**Експериментальний підбір параметрів навчання**

Після вибору конкретної топології, необхідно вибрати параметри навчання нейроної мережі. Цей етап особливо важливий для мереж, які *навчаються с учителем*. Від правильного вибору параметрів залежать не тільки те, наскільки швидко відповіді мережі будуть сходитися до правильних відповідей. Наприклад, вибір низької швидкості навчання збільшить час сходження, однак іноді дозволяє уникнути *паралічу* мережі. Збільшення моменту навчання може привести як до збільшення, так і до зменшення часу збіжності, у залежності від форми *поверхні похибки*. Виходячи з такого суперечливого впливу параметрів, можна зробити висновок, що їх значення потрібно вибирати експериментально, керуючись при цьому критерієм завершення навчання (наприклад, мінімізація похибки або обмеження за часом навчання).

**Власне навчання мережі**

У процесі навчання мережа у визначеному порядку переглядає навчальну вибірку. Порядок перегляду може бути послідовним, випадковим і т.д. Мережі, які *навчаються без учителя*, переглядають вибірку тільки один раз. При навчанні з учителем мережа переглядає вибірку множину разів, при цьому один повний прохід по вибірці називається *епохою навчання*. Зазвичай набір вихідних даних поділяють на дві частини — власне навчальну вибірку і тестові дані; принцип поділу може бути довільним. Навчальні дані подаються мережі для навчання, а перевірочні використовуються для розрахунку похибки мережі (перевірочні дані ніколи для навчання мережі не застосовуються). Таким чином, якщо на перевірочних даних похибка зменшується, то мережа дійсно виконує узагальнення.

Якщо похилка на навчальних даних продовжує зменшуватися, а похибка на тестових даних збільшується, виходить, мережа перестала виконувати узагальнення і просто «запам'ятовує» навчальні дані. Це явище називається перенавчанням мережі або *оверфіттінгом*. У таких випадках навчання зазвичай припиняють. У процесі навчання можуть проявитися інші проблеми, такі як *параліч* або влучення мережі в локальний мінімум поверхні похибок. Неможливо заздалегідь пророчити прояв тієї або іншої проблеми, так само як і дати однозначні рекомендації до їх розв'язанню.

**Перевірка адекватності навчання**

Навіть у випадку успішного, на перший погляд, навчання мережа не завжди навчається саме тому, чого від неї хотів творець. Відомий випадок, коли мережа навчалася розпізнаванню зображень танків по фотографіях, однак пізніше з'ясувалося, що всі танки були сфотографовані на тому самому тлі. У результаті мережа «навчилася» розпізнавати цей тип ландшафту, замість того, щоб «навчитися» розпізнавати танки. Таким чином, мережа «розуміє» не те, що від неї було потрібно, а те, що найпростіше узагальнити.

### 2.9. Компоненти і принципи роботи нейроних мереж

Результати сучасних досліджень фізіології мозку відкривають лише обмежене розуміння роботи нейронів і процесу мислення. Дослідники працюють як у біологічній, так і в інженерній областях для розшифровки ключових механізмів нейроної обробки, що допомагає створювати могутні і компактні нейромережі. Приведемо опис деяких компонентів нейроних мереж і їхню роботу.

## 2.9.1. Розширена модель штучного нейрона

Заглиблені представлення щодо будівлі біологічного нейрона, дозволяють представити модель штучного нейрона в

розширеному виді, деталізована структура якого приведена на рис. 2.28, де

1. Суматор, що моделює функції тіла біологічного нейрона

2. Функціональний перетворювач, що виконує роль аксонного горбка

3. Збудливий синапс

4. Гальмуючий синапс

5. Вхідний сигнал

6. Дихотомічне розгалуження вхідного сигналу

7. Вихідний сигнал

8. Дихотомічне розгалуження вихідного сигналу

9. Прямий зв'язок, який відповідає аксодендритному зв'язку між біологічними нейронами

10. Зворотний (аксосоматичний) зв'язок.

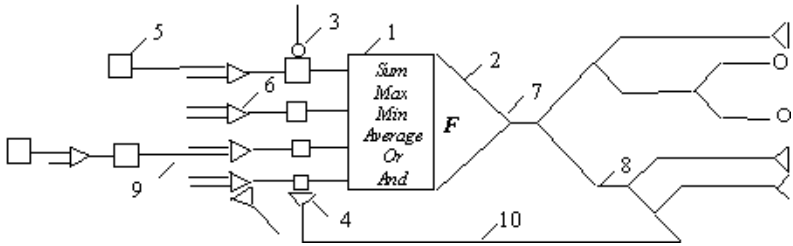



Рис. 2.28. Розширена модель нейроного елемента

Підставою для деталізації моделі нейроного елемента можна вважати встановлення нових фактів в області нейрофізіології, зокрема:

1. Наявність декількох місць синаптичного контакту.
2. Дихотомічне розгалуження дендритів різних порядків, що відповідають у технічних аналогах логічним операціям "І", "АБО", " Виключаюче АБО", виділення максимального або мінімального сигналу .
3. Різні діаметри стовбурних дендритів, галузей, що безпосередньо прилягають до тіла нейрона, причому величина діаметра визначає степінь важливості інформації, що проходить через дендрит.
4. Наявність "доріжок" на поверхні соми, які проходять від головних стовбурних дендритів до аксона, обумовлює наявність паралельних шляхів обробки інформації і надає можливість застосування логічних операцій над сигналами, що надходять від різних стволових дендритів.
5. Особливості функціонування аксонного горбка; саме аксоний горбок установлює передатну функцію нейрона, що має більш складну форму, чим прийняті в нейромережних технологіях сігмоїдальні або лінійні передатні функцій.
6. Наявність дихотомічного розгалуження аксона; у вузлах розгалуження відбувається керування проходженням сигналу, що залежить від співвідношення діаметрів різних галузей аксона; при математичному моделюванні ці особливості можна реалізувати за допомогою логічних операцій.
7. Наявність зворотного аксосоматичного зв'язку, що вже знайшло свою реалізацію при побудові рекурентных нейромереж.

Заглиблені знання щодо будівлі біологічного нейрона, як ефективного перетворюючого інструмента, можна розглядати як джерело базових ідей і концепцій по створенню нових парадигм нейромереж не тільки в сучасному часі, але і на віддалену перспективу.

## 2.9.2. Компоненти штучного нейрона

Незалежно від розташування і функціонального призначення, усі штучні нейроні елементи мають загальні компоненти. Розглянемо **сім** основних компонентів штучного нейрона.

**Компонент 1. Вагові коефіцієнти**

При функціонуванні нейрон одержує безліч вхідних сигналів одночасно. Кожен вхід має свою власну синаптичну вагу, що впливає на нього і необхідний для функції суматора. Вага є мірою важливості вхідних зв'язків і моделює поводження синапсів біологічних нейронів. Ваги впливового входу підсилюються і, навпаки, вага несуттєвого входу примусово зменшується, що визначає інтенсивність вхідного сигналу. Ваги можуть змінюватися відповідно до навчальних прикладів, архітектурою мережі і правилами навчання.

**Компонент 2. Функція суматора**

Першою дією нейрона є обчислення зваженої суми усіх входів. Математично, вхідні сигнали і відповідні їм ваги представляються векторами ($x_{10}$, $x_{20}$ ... $x_{n0}$) і ($w_{10}$, $w_{20}$ . . . $w_{n0}$). Добуток цих векторів відповідає узагальненому вхідному сигналові. Спрощеною функцією суматора є множення кожного значення компонента вектора x на відповідне значення компонента вектора w: вхід1 = $x_{10} * w_{10}$, вхід2 = $x_{20} * w_{20}$, і знаходження суми всіх добутків: вхід 1 + вхід 2 + . . . + вхід n. Результатом буде одне число, а не многоэлементний вектор. Функція суматора може бути складніша, наприклад, вибір мінімуму, максимуму, середнього арифметичного, добуток або інший алгоритм. Вхідні сигнали і вагові коефіцієнти перед надходженням у передатну функцію можуть комбінуватися багатьма способами. Алгоритми для комбінування входів нейронів визначаються в залежності від архітектури і правил навчання. У деяких нейромережах до функції суматора додають функцію активації, що зміщає вихід функції суматора щодо часу. Поки функції активації використовуються обмежено і більшість сучасних нейроних реалізацій використовують функцію активації "тотожності", що еквівалентна її відсутності. Її краще використовувати як компонент мережі в цілому, чим як компонент окремого нейрона.

**Компонент 3. Передатна функція**

Результат функції суматора проходить через передатну функцію і перетворюється у вихідний сигнал. У передатній функції для визначення виходу нейрона загальна сума порівнюється з деяким порогом. Якщо сума більше значення порога, нейрон генерує сигнал, у противному випадку сигнал буде нульовим або гальмуючим. Переважно застосовують нелінійну передатну функцію, оскільки лінійні (прямолінійні) функції обмежені і їхній вихід пропорційний входові. На рис. 2.29 зображені найбільш розповсюджені передатні функції

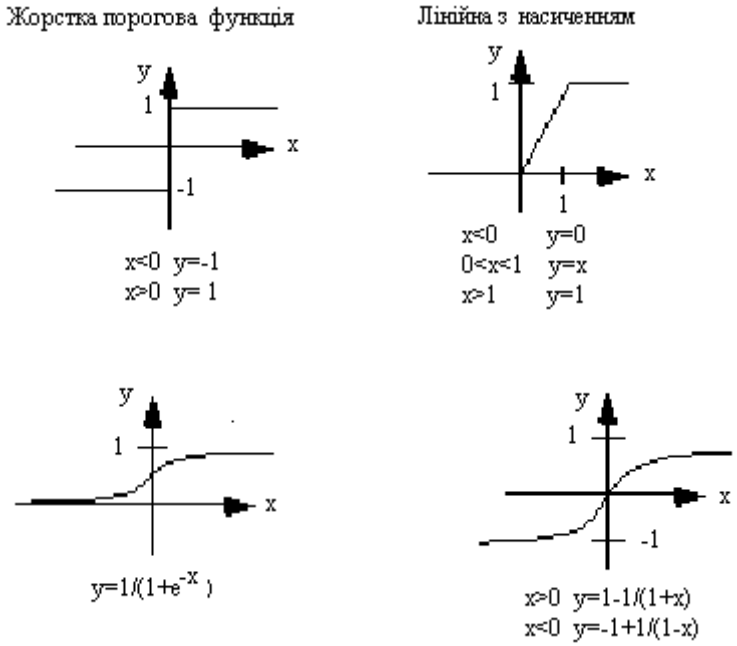



Рис. 2.29. Найбільш розповсюджені передатні функції.

Для простої передатної функції нейромережа може видавати 0 і 1, 1 і -1 або інші числові комбінації. Передатна функція в таких випадках є "твердим обмежником" або граничною функцією (рмс. 2.29,а). Інший тип передатної функції - лінійна з насиченням віддзеркалює вхід усередині заданого діапазону і діє як твердий обмежник за межами цього діапазону. Це лінійна функція, що відтинається мінімальним і максимальним значеннями, роблячи її нелінійною (рис. 2.29,б).

Наступною є S-подібна крива, що наближає мінімальне і максимальне значення в асимптотах і називається сігмоїдою (рис. 2.29,в), коли її діапазон [0, 1], або гіперболічним тангенсом (рис. 2.29,г), при діапазоні [-1, 1]. Важливою рисою цих кривих є безперервність функцій і їхніх похідних. Застосування S-функцій дає гарні результати і має широке застосування.

Для різних нейромереж можуть вибиратися різні передатні функції. Перед надходженням у передатну функцію до вхідного сигналу часом додають рівномірно розподілений випадковий шум, джерело і кількість якого визначається режимом навчання. У літературі цей шум трактується як "температура" штучних нейронів, що додає математичній моделі елемент реальності.

**Компонент 4. Масштабування**

Після передатної функції вихідний сигнал проходить додаткову обробку масштабування, тобто результат передатної функції збільшується на масштабний коефіцієнт і додається зсув.

**Компонент 5. Вихідна функція (змагання)**

За аналогією з біологічним нейроном, кожен штучний нейрон має один вихідний сигнал, що передається сотням інших нейронів. Переважно, вихід прямо пропорційний результату передатної функції. У деяких мережних архитектурах результати передатної функції змінюються для створення змагання між сусідніми нейронами. Нейронам дозволяється змагатися між собою, блокуючи дії нейронів, що мають слабкий сигнал.

Змагання (конкуренція) може відбуватися між нейронами, що знаходяться на одному або різних шарах. По-перше, конкуренція визначає, який штучний нейрон буде активним і забезпечить вихідний сигнал. По-друге, виходи, що конкурують, допомагають визначити, який нейрон візьме участь у процесі навчання.

**Компонент 6. Функція похибки і розповсюджуване обернене значення**

У більшості мереж, що застосовують контрольоване навчання обчислюється різниця між отриманим і бажаним виходом. Похибка відхилення (потокова похибка) перетворюється функцією похибки відповідно до заданої архітектури. У базових архитектурах похибка відхилення використовується безпосередньо, у деяких парадигмах використовується квадрат або куб похибки і зі збереженням знака. Після проходження всіх шарів потокова похибка поширюється назад до попереднього шару і може бути безпосередньо похибкою або похибкою, масштабованною певним чином у залежності від типу мережі (наприклад, похідній від передатної функції). Це розповсюджене назад значення враховується в наступному циклі навчання.

**Компонент 7. Функція навчання**

Метою навчання є настроювання ваг з'єднань на входах кожного нейрона відповідно до визначеного алгоритму навчання для досягнення бажаного результату. Існує два типи навчання: контрольоване і неконтрольоване. Контрольоване навчання вимагає навчальної множини даних або спостерігача, що відслідковує ефективність результатів мережі. У випадку неконтрольованого навчання система самоорганізовується за внутрішнім критерієм, який закладено в алгоритм навчання.

## 2.9.3. Архітектура з'єднань штучних нейронів

Об’єднуючись в мережі, нейрони утворюють системи обробки інформації, що забезпечують ефективну адаптацію

моделі до постійних змін з боку зовнішнього середовища. У процесі функціонування мережі відбувається перетворення вхідного вектора сигналів у вихідний. Конкретний вид перетворення визначається як архітектурою нейромережі, так і характеристиками нейроних елементів, засобами керування і синхронізацією інформаційних потоків між нейронами. Важливим фактором ефективності мережі є встановлення оптимальної кількості нейронів і типів зв'язків між ними.

При описі нейромереж використовують кілька термінів, що у різних джерелах можуть мати різне трактування, зокрема:

- структура нейромережі - спосіб зв'язків нейронів у нейромережі;
- архітектура нейромережі - структура нейромережі і типи нейронів;
- парадігма нейромережі - спосіб навчання і використання; іноді містить і поняття архітектури.

На основі однієї архітектури можуть бути реалізовані різні парадигми нейромереж і навпаки. Серед відомих архітектурних рішень виділяють групу слабкозв’язаних нейроних мереж, коли кожен нейрон мережі зв'язаний лише із сусідніми (рис.2.30).

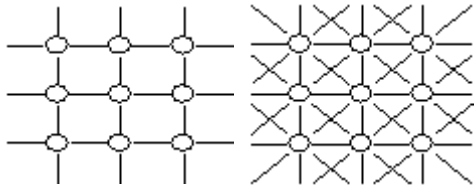


Рис. 2.30. Слабкозв’язані нейромережі

Навпаки, якщо входи кожного нейрона зв'язані з виходами всіх інших нейронів, тоді мова йде про повнозв’язані нейромережі (рис. 2.31).

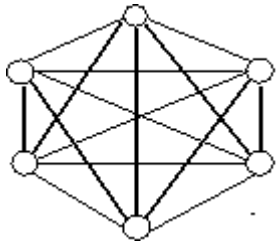


Рис. 2.31. Повнозв’язана мережа

Зрозуміло, що такий розподіл носить умовний характер. Аналізуючи найбільш відомі на даний час розробки нейромереж, слід зазначити, що найпоширенішим варіантом архітектури є багатошарові мережі (рис.2.32). Нейрони в даному випадку поєднуються в шари з єдиним вектором вхідних сигналів. Зовнішній вхідний вектор сигналів подається на вхідний шар нейроної мережі (рецептори). Виходами нейроної мережі є вихідні сигнали останнього шару (ефектори). Крім вхідного і вихідного шарів, нейромережа має один або кілька схованих шарів нейронів, що не мають контактів із зовнішнім середовищем.

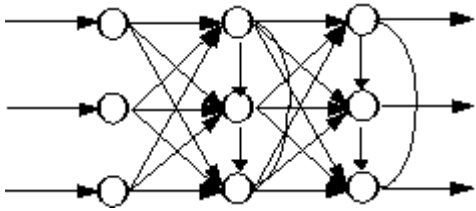


Рис. 2.32. Багатошаровий тип з'єднання нейронів

Типи зв'язків:

- Зв'язки між нейронами різних шарів називають проективними.
- Зв'язки, що спрямовані від вхідних шарів до вихідного називаються аферентними.

- При зворотному напрямку зв'язків - еферентними.
- Зв'язки між нейронами одного шару - бічними (латеральними).

По архітектурі зв'язків, більшість відомих нейромереж можна згрупувати в два великих класи:

1. Мережі прямого поширення (з односпрямованими послідовними зв'язками).
2. Мережі зворотного поширення (з рекурентними зв'язками).

Нижче приводяться назви мереж типових архитектур, що мають безліч модифікацій і можуть бути складовими частинами в інших мережах.

**Нейроні мережі**

Мережі прямого поширення
Рекурентні мережі
Перцептрони
Мережа Хопфілда
Мережа Back Propagation
Мережа адаптивної резонансної теорії
Мережа зустрічного поширення
Двонаправленная мережа
Карта Кохонена

*Мережі прямого поширення* відносять до статичних, тому що на задані входи нейронів надходить не залежний від попереднього стану мережі вектор вхідних сигналів. Рекурентні мережі вважаються динамічними, тому що за рахунок зворотних зв'язків (петель) входи нейронів модифікуються в часі, що приводить до зміни стану мережі.

Оригінальність нейромереж, як аналога біологічного мозку, складається в здатності до навчання по прикладах, що складають навчальну множину. Процес навчання нейромережі розглядається як настроювання архітектури і вагових коефіцієнтів синаптичних зв'язків відповідно до даних навчальної чножини так, щоб ефективно вирішувати поставлену задачу. Розрізняють варіанти контрольованого і неконтрольованого навчання.

## 2.9.4. Навчання штучної нейронной мережі

### Контрольоване навчання

Більшість рішень виходить від нейромереж з контрольованим навчанням, де поточний вихід постійно порівнюється з бажаним виходом. Ваги спочатку установлюються випадково, але під час наступних ітерацій коректуються для досягнення відповідності між бажаним і поточним виходом. Створені методи навчання націлені на мінімізацію поточних похибок усіх нейронів, створюваних безперервною зміною ваг для досягнення прийнятної точності мережі. Перед використанням, нейромережа з контрольованим навчанням повинна бути навчена. Фаза навчання може бути тривалою, зокрема, у системах, з невідповідною процесорною потужністю, займати кілька годин. Навчання вважається закінченим при досягненні нейромережою визначеного рівня точності, коли видаються бажані значення виходів для заданої послідовності вхідних сигналів. Після навчання ваги з'єднань фіксуються для подальшого застосування. Деякі типи мереж дозволяють використовувати безперервне навчання для адаптації до умов, що змінюються. Навчальні множини повинні бути досить великими і містити необхідну інформацію для виявлення важливих особливостей і зв'язків. Але і навчальні приклади повинні мати широке різноманіття даних. Якщо мережа учиться лише для одного приклада, ваги, установлені для даного приклада, радикально змінюються при навчанні для наступного приклада. Попередні приклади при навчанні наступних просто забуваються. У результаті система повинна навчатися всьому разом, знаходячи найкращі вагові коефіцієнти для загальної

множини прикладів. Наприклад, у навчанні системи розпізнавання пиксельних образів десяти цифр, які представлені двадцятьма прикладами кожної цифри, усі приклади цифри "сім" не варто представляти послідовно. Краще надати мережі спочатку один тип представлення всіх цифр, потім другий тип і так далі. Головним компонентом для успішної роботи мережі є представлення і кодування вхідних і вихідних даних. Нейроні мережі працюють лише з числовими вхідними даними, отже, нечислові дані, що надходять із зовнішнього середовища повинні перетворюватися. Додатково необхідно здійснювати масштабування, тобто нормалізацію даних відповідно діапазонові всіх значень. Нормалізація виконується шляхом розподілу кожного компонента вхідного вектора на довжину вектора, що перетворює вхідний вектор в одиничний. Попередня обробка зовнішньої даних, отриманих за допомогою сенсорів, у машинний формат, загальна для стандартних комп'ютерів і легко доступна. Якщо після контрольованого навчання нейромережа ефективно обробляє дані навчальної множини, важливим стає ефективність при роботі з даними, що не використовувалися для навчання. У випадку одержання незадовільних результатів для тестової множини, навчання продовжується. Тестування необхідне для забезпечення запам'ятовування не тільки даних заданого навчальної множини, але і створення загальних образів, що можуть утримуватися в даних.

**Неконтрольоване навчання**

Неконтрольоване навчання може бути великим надбанням майбутнього, коли комп'ютери зможуть самонавчатися. У наш час, неконтрольоване навчання використовується в мережах відомих, як самоорганізовані карти (self organizing maps), що знаходяться в досить обмеженому користуванні, але доводять перспективність самоконтрольованого навчання. Мережі не використовують зовнішнього впливу для коректування своїх ваг і внутрішньо контролюють свою точність, шукають регулярність або тенденції у вхідних сигналах і адаптуються відповідно до навчальної функції. Навіть без повідомлення правильності або неправильності, мережа повинна мати інформацію щодо власної організації, що закладено в архітектуру мережі і навчальні

правила. Алгоритм неконтрольованого навчання спрямований на знаходження близькості між групами нейронів, що працюють разом. Якщо зовнішній сигнал активує будь-який вузол у групі нейронів, дія всієї групи в цілому збільшується. Аналогічно, якщо зовнішній сигнал у групі зменшується, це приводить до гальмуючого ефекту на всю групу. Конкуренція між нейронами формує основу для навчання. Навчання конкуруючих нейронів підсилює відгуки визначених груп на визначені сигнали. Це зв'язує групи між собою і відгуком. При конкуренції змінюються ваги лише нейрона-переможця.

**Оцінки навчання**

Оцінка ефективності навчання нейромережі залежить від декількох керованих факторів. Теорія навчання розглядає три фундаментальних властивості, зв'язані з навчанням: *ємність, складність зразків і обчислювальна складність*. Під *ємністю* розуміють, скільки зразків може запам'ятати мережа, і які границі прийняття рішень можуть бути на ній сформовані. *Складність* зразків визначає число навчальних прикладів, необхідних для досягнення здатності мережі до узагальнення. *Обчислювальна* складність зв'язана з потужністю процесора ЕОМ.

**Правила навчання**

Є безліч правил навчання, але більшість з цих правил є деякою зміною відомого і найстаршого правила навчання Хебба. Дослідження різних правил навчання продовжуються, і нові ідеї регулярно публікуються в наукових і комерційних виданнях. Представимо кілька основних правил навчання.

*Правило Хебба*

Опис правила з'явилося в його книзі "Організація поводження" у 1949 р. "Якщо нейрон одержує вхідний сигнал від іншого нейрона й обоє є високо активними (математично мають той же знак), вага між нейронами повина бути посилена". При збудженні одночасно двох нейронів з виходами ($x_j$, $y_i$) на t-тім

кроці навчання вага синаптического з'єднання між ними зростає, в іншому випадку - зменшується, тобто

$$\Delta W_{ij}(k)=r\, x_j(k)\, y_i(k),$$

де $r$ - коефіцієнт швидкості навчання.

Може застосовуватися при навчанні "із учителем" і "без учителя".

*Правило Хопфілда*

Схоже на правило Хебба за винятком того, що воно визначає величину посилення або ослаблення. "Якщо одночасно вихідний і вхідний сигнал нейрона активні або неактивні, збільшуємо вагу з'єднання оцінкою навчання, інакше зменшуємо вагу оцінкою навчання".

*Правило "дельта"*

Це правило є зміною правила Хебба і є одним з використовуваних. Воно базується на простій ідеї безперервної зміни синаптичних ваг для зменшення різниці ("дельта") між значенням бажаного і поточного вихідного сигналу нейрона.

$$\Delta W_{ij}= x_j\,(d_i - y_i).$$

За цим правилом мінімізується середньоквадратична похибка мережі. Це правило також згадується як правило навчання Відрова-Хоффа і правило навчання найменших середніх квадратів.

У правилі "дельта" похибка, яка отримана у вихідному шарі, перетворюється похідною передатною функцією і послідовно послойно поширюється назад на попередні шари для корекції синаптичних ваг. Процес зворотного поширення похибок мережі рухається до першого шару. Від цього методу обчислення

похибки успадкувала своє ім'я відома парадигма Feed Forward BackPropagation.

При використанні правила "дельта" важливим є невпорядкованість множини вхідних даних. При добре упорядкованій або структурованій навчальній множині результат мережі може не досягти бажаної точності і мережа буде вважатися нездатної до навчання.

**Правило градієнтного спуску**

Це правило схоже на правило "дельта" використанням похідної від передатної функції для зміни похибки "дельта" перед застосуванням її до ваг з'єднань. До кінцевого коефіцієнта зміни додається пропорційна константа, яка зв'язана з оцінкою навчання. І хоча процес навчання прагне до точки стабільності дуже повільно, це правило поширене і часте використовується.

Доведено, що різні оцінки навчання для різних шарів мережі допомагає процесові навчання сходиться швидше. Оцінки навчання для шарів, близьких до виходу, установлюються меншими, чим для шарів, близьких до входу.

**Навчання методом змагання**

На відміну від навчання Хебба, у якому множина вихідних нейронів може збуджуватися одночасно, при навчанні методом змагання вихідні нейрони змагаються між собою за активізацію. Це явище, відомо як правило "переможець одержує все". Подібне навчання має місце в біологічних нейронних мережах. Навчання за допомогою змагання дозволяє кластеризувати вхідні дані: схожі приклади групуються мережею відповідно до кореляції і представляються одним елементом.

При навчанні модифікуються синаптичні ваги нейрона-переможця. Ефект цього правила досягається за рахунок такої зміни збереженого в мережі зразка (вектора синаптичних ваг нейрона-переможця), при якому він стає ближче до вхідного

приклада. Нейрон з найбільшим вихідним сигналом з'являється переможцем і має можливість гальмувати своїх конкурентів і збуджувати сусідів. Використовується вихідний сигнал нейрона-переможця і тільки йому і його сусідам дозволяється коректувати свої ваги з'єднань.

$$W_{ij}(k+1) = W_{ij}(k) + r\,[x_j - W_{ij}(k)]$$

Розмір області сусідства може змінюватися під час періоду навчання. Звичайна парадигма повинна починатися з великої області визначення сусідства і зменшуватися під час процесу навчання. Оскільки елемент-переможець визначається по високій відповідності вхідному зразкові, мережі Кохонена моделюють розподіл входів. Це правило використовується в самоорганізованих картах.

## 2.10. Основні типи нейроних мереж

Оскільки всі штучні нейроні мережі базуються на концепції нейронів, з'єднань і передатних функцій, існує подібність між різними структурами або архітектурами нейроних мереж. Більшість відмінностей залежить від різних правил навчання. Розглянемо деякі відомі моделі штучних нейромереж.

### 2.10.1. Перцептрон Розенблатта

Першою моделлю нейромереж вважають перцептрон Розенблатта. Теорія перцептронів є основою для багатьох типів штучних нейромереж прямого поширення і класикою для вивчення. Одношаровий перцептрон здатний розпізнавати найпростіші образи. Окремий нейрон обчислює зважену суму елементів вхідного сигналу, віднімає значення зрушення і пропускає результат через тверду граничну функцію, вихід якої дорівнює +1 або -1. У залежності від значення вихідного сигналу приймається рішення:

+1 - вхідний сигнал належить класові А,

-1 - вхідний сигнал належить класові В.

На рис. 2.33 показана схема нейронів, використовуваних в одношарових перцептронах, графік передатної функції і схема вирішальних областей, створених у багатомірному просторі вхідних сигналів. Вирішальні області визначають, які вхідні образи будуть віднесені до класу А, які - до класу В. Перцептрон, що складається з одного нейрона, формує дві вирішальні області, які розділені гіперплощиною. На рисунку показано випадок, коли розмірність вхідного сигналу дорівнює 2. При цьому поділяюча поверхня являє собою пряму лінію на площині. Рівняння, що задає поділяючу пряму, залежить від значень синаптичних ваг і зрушення.

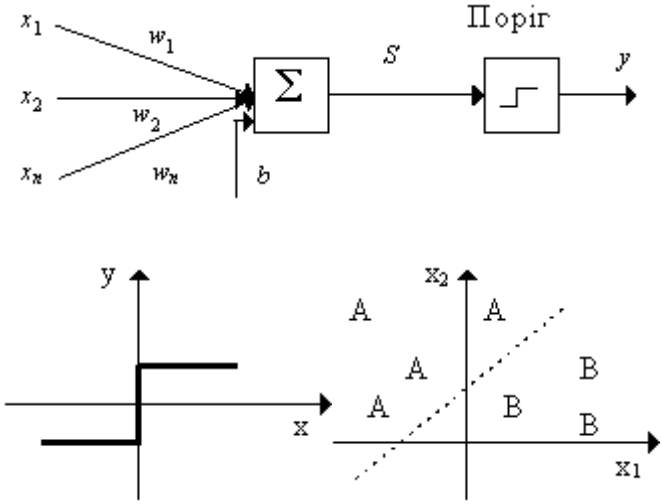



Рис. 2.33. Схема нейрона, графік передатної функції і поділяюча поверхня

Алгоритм навчання одношарового перцептрона

1. Ініціалізація синаптичних ваг і зрушення: синаптичні ваги приймають невеликі випадкові значення.
2. Пред'явлення мережі нових вхідних і бажаного вихідного сигналів:

Вхідний сигнал $x=(x_1, x_2, ..., x_n)$ пред'являється нейронові разом з бажаним вихідним сигналом $d$.

3. Обчислення вихідного сигналу нейрона:

$$y(t) = f\left(\sum_{i=1}^{N} w_i(t)x_i(t) - b\right)$$

4. Настроювання значень ваг:

$$w_i(t+1)=w_i(t)+r[d(t)-y(t)]x_i(t),\ i=1, ..., N$$

$$d(t) = \begin{cases} +1, \text{вихідний клас A} \\ -1, \text{вихідний клас B} \end{cases}$$

де $w_i(t)$ - вага зв'язку від *i-го* елемента вхідного сигналу до нейрона в момент часу $t$, $r$ - швидкість навчання (менше 1); $d(t)$ - бажаний вихідний сигнал.

Якщо мережа приймає правильне рішення, синаптичні ваги не модифікуються.

5. Перехід до кроку 2.

Тип вхідних сигналів: бінарні або аналогові.

Розмірності входу і виходу обмежені при програмній реалізації тільки можливостями обчислювальної системи, на якій моделюється нейрона мережа, при апаратній реалізації - технологічними можливостями.

Області застосування: розпізнавання образів, класифікація.

*Недоліки.* Примітивні поділяючі поверхні (гіперплощини) дають можливість розв’язувати лише найпростіші задачі розпізнавання.

*Переваги*. Програмні й апаратні реалізації моделі дуже прості. Простий і швидкий алгоритм навчання.

*Модифікації*. Багатошарові перцептрони дають можливість будувати більш складні поділяючі поверхні і тому більш поширені.

На рис. 2.34 показана схема перцептрона з декількома виходами.

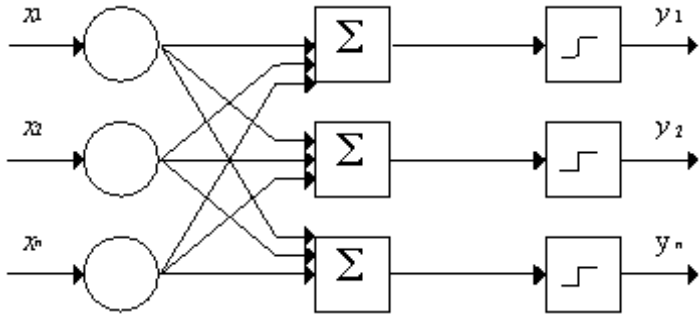



Рис. 2. Перцептрон з декількома виходами

### 2.10.2. Нейромережа зворотного поширення похибки (Back Propagation)

Архітектура FeedForward BackPropagation була розроблена на початку 1970-х років декількома незалежними авторами: Вербор (Werbor); Паркер (Parker); Румельгарт (Rumelhart), Хинтон (Hinton) і Вільямс (Williams). Зараз, парадигма Backpropagation найбільш популярна, ефективна і легка модель навчання для складних, багатошарових мереж. Вона використовується в різних типах додатків і породила великий клас нейромереж з різними структурами і методами навчання.

Типова мережа Backpropagation має вхідний шар, вихідний шар і принаймні один схований шар. Теоретично, обмежень щодо числа схованих шарів не існує, але практично

застосовують один або два. На рис. 2.35 представлена схема багатошарового перцептрона.

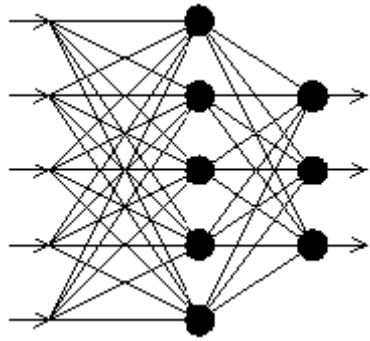


Рис. 2.35. Багатошаровий перцептрон

Нейрони організовані в пошарову структуру з прямою передачею сигналу. Кожен нейрон мережі продукує зважену суму своїх входів, пропускає цю величину через передатну функцію і видає вихідне значення. Мережа може моделювати функцію практично будь-якої складності, причому число шарів і число нейронів у кожнім шарі визначають складність функції. Визначення числа проміжних шарів і числа нейронів у них є важливим при моделюванні мережі. Більшість дослідників і інженерів, застосовуючи архітектуру до визначених проблем використовують загальні правила, зокрема:

1. Кількість входів і виходів мережі визначаються кількістю вхідних і вихідних параметрів досліджуваного об'єкта, явища, процесу, і т.п.. На відміну від зовнішніх шарів, число нейронів схованого шару вибирається емпіричним шляхом. У більшості випадків достатня кількість нейронів складає $n_{ск} \le n_{вх} + n_{вих}$, де $n_{вх}$, $n_{вих}$ - кількість нейронів у вхідному і, відповідно, у вихідному шарах.

2. Якщо складність у відношенні між отриманими і бажаними даними на виході збільшується, кількість нейронів схованого шару повинна також збільшитися.

3. Якщо моделюємий процес може розділятися на багато етапів, потрібний додатковий схований шар (шари). Якщо процес не розділяється на етапи, тоді додаткові шари можуть допустити перезапам'ятовування і, відповідно, невірне загальне рішення.

Після того, як визначене число шарів і число нейронів у кожнім з них, потрібно знайти значення для синаптичних ваг і порогів мережі, здатних мінімізувати похибку спродукованного результату. Саме для цього існують алгоритми навчання, де відбувається припасування моделі мережі до наявних навчальних даних. Похибка для конкретної моделі мережі визначається шляхом проходження через мережу всіх навчальних прикладів і порівняння спродукованних вихідних значень з бажаними значеннями. Множина похибок створює функцію похибок, значення якої можна розглядати, як похибка мережі. Як функцію похибок частіше використовують суму квадратів похибок.

Для кращого розуміння алгоритму навчання мережі Back Propagation потрібно роз'яснити поняття поверхні станів. Кожному значенню синаптичних ваг і порогів мережі (вільних параметрів моделі кількістю N) відповідає один вимір у багатомірному просторі. N+1-й вимір відповідає похибці мережі. Для різних з'єднань ваг відповідну похибку мережі можна зобразити точкою в N+1-вимірному просторі, усі ці точки утворюють деяку поверхню - поверхня станів. Ціль навчання нейромережі складається в перебуванні на багатомірній поверхні найбільш низької точки. Поверхня станів має складну будову і доволі неприємні властивості, зокрема, наявність локальних мінімумів (точки, найбільш низькі у своєму визначеному околі, але вище глобального мінімуму), плоскі ділянки, сідлові точки і довгі вузькі яри. Аналітичними засобами неможливо визначити розташування глобального мінімуму на поверхні станів, тому навчання нейромережі по суті складається в дослідженні цієї поверхні. Відштовхуючись від початкової конфігурації ваг і порогів (від випадково обраної точки на

поверхні), алгоритм навчання поступово відшукує глобальний мінімум. Обчислюється вектор градієнта поверхні похибок, що вказує напрямок найкоротшого спуску по поверхні з заданої точки. Якщо трошки просунутися по ньому, похибка зменшиться. Зрештою, алгоритм зупиняється в нижній точці, що може виявитися лише локальним мінімумом (в ідеальному випадку - глобальним мінімумом). Складність тут складається у виборі довжини кроків. При великій довжині кроку збіжність буде швидша, але є небезпека перестрибнути розв'язання, або піти в неправильному напрямку. При маленькому кроці, правильний напрямок буде виявлено, але зростає кількість ітерацій. На практиці розмір кроку береться пропорційним крутості схилу з деякою константою - швидкістю навчання. Правильний вибір швидкості навчання залежить від конкретної задачі і визначається дослідницьким шляхом. Ця константа може також залежати від часу, зменшуючись у міру просування алгоритму. Алгоритм діє ітеративно, його кроки називаються епохами. На кожній епосі на вхід мережі по черзі подаються всі навчальні приклади, вихідні значення мережі порівнюються з бажаними значеннями й обчислюється похибка. Значення похибки, а також градієнта поверхні станів використовують для корекції ваг, і дії повторюються. Процес навчання припиняється або якщо пройдено визначену кількість епох, або якщо похибка досягає визначеного рівня малості, або якщо похибка перестає зменшуватися (користувач переважно сам вибирає потрібний критерій останова).

*Алгоритм навчання мережі*

1. Ініціалізація мережі: вагові коефіцієнти і зрушення мережі приймають малі випадкові значення.
2. 2. Визначення елемента навчальної множини: (вхід - вихід). Входи ($x_1$, $x_2$... $x$), повинні розрізнятися для всіх прикладів навчальної множини.
3. Обчислення вихідного сигналу:

$$S_{i_m} = \sum_{i_{m-1}}^{N_{m-1}} w_{i_m j_{m-1}} y_{i_{m-1}} - b_{i_m}$$

$$y_{im} = f(S_{jm})$$

$$i_m = 1, 2, ..., N_m, m = 1, 2, ..., L$$

де $S$ - вихід суматора, $w$ - вага зв'язку, $y$ - вихід нейрона, $b$ - зсув, $i$ - номер нейрона, $N$ - число нейронів у шарі, $m$ - номер шару, $L$ - число шарів, $f$ - передатна функцяя.

4. Настроювання синаптичних ваг:

$$w_{ij}(t+1) = w_{ij}(t) + r g_j x'_i$$

де $w_{ij}$ - вага від нейрона $i$ або від елемента вхідного сигналу $i$ до нейрона $j$ у момент часу $t$, $x_i'$ - вихід нейрона $i$, $r$ - швидкість навчання, $g_j$ - значення похибки для нейрона $j$.

Якщо нейрон з номером j належить останньому шарові, тоді

$$g_j = y_j(1-y_j)(d_j-y_j)$$

де $d_j$ - бажаний вихід нейрона $j$, $y_j$ - потоковий вихід нейрона $j$.

Якщо нейрон з номером j належить одному з шарів з першого по передостанній, тоді

$$g_j = x'_j(1 - x'_j)\sum_k g_k w_{jk}$$

де k пробігають усі нейрони шару з номером на одиницю більше, ніж у того, котрому належить нейрон j.

Зовнішні зрушення нейронів b набудовуються аналогічним чином.

*Тип вхідних сигналів*: цілі або дійсні.

*Тип вихідних сигналів*: дійсні з інтервалу, заданого передатною функцією нейронів.

*Тип передатної функції*: сігмоїдальна. У нейроих мережах застосовуються кілька варіантів сіигмоїдальних передатних функцій.

*Функція Фермі* (експонентна сігмоїда):

$$f(S) = \frac{1}{1 + e^{-2\alpha S}}$$

де s - вихід суматора нейрона, α - деякий параметр.

*Раціональна сігмоїда:*

$$f(S) = \frac{S}{|S| + \alpha}$$

*Гіперболічний тангенс:*

$$f(S) = th\frac{S}{\alpha} = \frac{e^{-\frac{S}{\alpha}} - e^{-\frac{S}{\alpha}}}{e^{\frac{S}{\alpha}} + e^{-\frac{S}{\alpha}}}$$

Згадані функції відносяться до однопараметричних. Значення функції залежить від аргументу й одного параметра. Також використовуються багатопараметричні передатні функції, наприклад:

$$f(S) = p_1 \frac{S}{|S| + p_2} + p_3$$

Сігмоїдальні функції є монотонно зростаючими і мають відмінні від нуля похідні по всій області визначення. Ці характеристики забезпечують правильне функціонування і навчання мережі.

*Області застосування*. Розпізнавання образів, класифікація, прогнозування.

*Недоліки.* Багатокритеріиальна задача оптимізації в методі зворотного поширення розглядається як набір однокритеріальних задач - на кожній ітерації відбуваються зміни значень параметрів мережі, що поліпшують роботу лише з одним прикладом навчальної вибірки. Такий підхід істотно зменшує швидкість навчання.

*Переваги.* Зворотне поширення - ефективний і популярний алгоритм навчання багатошарових нейроних мереж, з його допомогою розв'язуються численні практичні задачі.

*Модифікації*. Модифікації алгоритму зворотного поширення зв'язані з використанням різних функцій похибок, різних процедур визначення напрямку і величини кроку.

### 2.10.3. Мережа Delta Bar Delta

Мережа Delta bar Delta була розроблена Робертом Джекобсом (Robert Jacobs), для поліпшення оцінки навчання стандартних мереж Feed Forward і є модифікацією мережі Back Propagation

Процедура Back Propagation базується на підході крутого спуску, що мінімізує похибку мережі під час процесу зміни синаптичних ваг. Стандартні оцінки навчання застосовуються на базисі "шар за шаром" і значення моменту призначаються глобально. Моментом вважається фактор, що використовується для згладжування оцінки навчання. Момент додається до стандартної зміни ваги і пропорційний до попередньої зміни ваги.

Хоча цей метод успішний у розв’язанні багатьох задач, збіжність процедури занадто повільна для використання.

Delta bar Delta має "неформальний" підхід до навчання штучних мереж, при якому кожна вага має свій власний самоадаптований фактор навчання і минулі значення похибки використовуються для обчислення майбутніх значень. Знання ймовірних похибок дозволяє мережі робити інтелектуальні кроки при зміні ваг, але процес ускладнюється тим, що кожна вага може мати зовсім різний вплив на загальну похибку. Джекобс запропонував поняття "здорового глузду", коли кожна вага з'єднання мережі повинна мати власну оцінку навчання, а розмір кроку, що призначений одній вазі з'єднання не застосовується для усіх ваг у шарі.

Оцінка навчання будь-якої ваги з'єднання змінюється на основі інформації про поточну похибку, знайденої зі стандартної Васкргораgation. Якщо локальна похибка має однаковий знак для декількох послідовних тимчасових кроків, оцінка навчання для цього з'єднання лінійно збільшується. Якщо локальна похибка часто змінює знак, оцінка навчання зменшується геометрично і це гарантує, що оцінки навчання з'єднання будуть завжди додатними. Оцінкам навчання дозволено мінятися в часі. Призначення оцінки навчання до кожного з'єднання, дозвіл цій оцінці навчання неперервно мінятися з часом обумовлюють зменшення часу збіжності.

З розрішенням різних оцінок навчання для будь-якої ваги з'єднання в мережі, пошук крутого спуску може не виконуватися. Замість цього, ваги з'єднань змінюються на основі часток похідних похибок щодо самої ваги й оцінки "кривизни поверхні похибки" поблизу поточної точки. Зміни ваг відповідають обмеженню місцевості і вимагають інформацію від нейронів, з якими вони з'єднані.

*Переваги*. Парадигма Delta Bar Delta є спробою прискорити процес збіжності алгоритму зворотного поширення за рахунок використання додаткової інформації про зміну параметрів і ваг під час навчання.

*Недоліки*

- Навіть невелике лінійне збільшення коефіцієнта може привести до значного росту швидкості навчання, що викликає стрибки в просторі ваг.
- Геометричне зменшення коефіцієнта іноді буває недостатньо швидким.

### 2.10.4. Мережа Extended Delta Bar Delta

Елі Минай (Ali Minai) і Рон Вільямс (Ron Williams) розробили алгоритм роботи мережі Eextended Delta bar Delta, як природне продовження роботи Джекобса. В алгоритм убудовується пам'ять з особливістю відновлення. Після кожного представлення навчальні дані епохи, оцінюється накопичена похибка. Якщо похибка менша за попередню мінімальну похибку, ваги зберігаються в пам'яті, як найкращі на цей час. Параметр допуску керує фазою відновлення. У випадку, якщо поточна похибка перевищує мінімальну попередню похибку, модифіковану параметром допуску, усі значення ваг з'єднань стохастично повертаються до збереженого в пам'яті найкращої множини ваг.

### 2.10.5. Мережа спрямованого випадкового пошуку

Мережа спрямованого випадкового пошуку (Directed Random Search), використовує стандартну архітектуру FeedForward, що не базується на алгоритмі BackProragation і коректує ваги випадковим чином. Для забезпечення порядку в такому процесі, до випадкового кроку додається компонент напрямку, що гарантує напрямок ваг до попередньо успішного напрямку пошуку. Вплив на нейрони здійснюється окремо. Для збереження ваг усередині компактної області, де алгоритм працює добре, установлюють верхню границю величини ваг. Установлюючи границі ваг великими, мережа може продовжувати працювати, оскільки дійсний глобальний оптимум залишається невідомим. Іншою особливістю правила навчання є початкова відмінність у випадковому розподілі ваг. У більшості комерційних пакетів існує

рекомендоване розроблювачем число для параметра початкової відмінності. Парадигма випадкового пошуку має кілька важливих рис. Вона швидка і легка у використанні, найкращі результати виходять, коли початкові ваги знаходяться близько до найкращих ваг. Швидкою парадигма є завдяки тому, що для проміжних нейронів похибки не обчислюються, а обчислюється лише вихідна похибка. Алгоритм дає ефект лише в невеликій мережі, оскільки при збільшенні числа з'єднань, процес навчання стає довгим і важким. Існує чотири ключових компоненти мережі з випадковим пошуком. Це - випадковий крок, крок реверсування, спрямований компонент і самокоригувальна відмінність.

*Випадковий крок*. До будь-якої ваги додається випадкова величина. Уся навчальна множина пропускається через мережу, створюючи " похибку пророкування". Якщо нова загальна похибка навчальної множини менша ніж попередня найкраща похибка пророкування, текучі значення ваг, що включають випадковий крок, стають новою множиною "найкращих" ваг. Поточна похибка пророкування зберігається як нова, найкраща похибка пророкування.

*Крок реверсування*. Якщо результати випадкового кроку гірші попередньої найкращої, випадкова величина віднімається від початкового значення ваги. Це створює множину ваг, що знаходяться в протилежному напрямку до попереднього випадкового кроку. Якщо загальна " похибка пророкування" менше попередньої найкращої похибки, текуче значення ваг і поточна похибка пророкування зберігаються як найкращі. Якщо і прямій і зворотний кроки не поліпшують результат, до найкращих ваг додається цілком нова множина випадкових значень і процес починається спочатку.

*Спрямований компонент*. Для збіжності мережі створюється множина спрямованих компонентів, отриманих за результатами прямого і зворотного кроків. Спрямовані компоненти, що відображають ланцюг успіхів або невдач попередніх випадкових кроків, додаються до випадкових компонентів на кожнім кроці процедури і забезпечують елемент "здорового глузду" у пошуку.

Доведено, що додавання спрямованих компонентів забезпечує різке підвищення ефективності алгоритму.

*Самокоригувальна відмінність*. Визначається параметр початкової відмінності для керування початковим розміром випадкових кроків, що додається до ваг. Адаптивний механізм змінює параметр відмінності, що базується на поточній оцінці успіху або невдачі. Правило навчання припускає, що поточний розмір кроків для ваг у правильному напрямку збільшується для випадку декількох послідовних успіхів. Навпаки, якщо відбувається кілька послідовних невдач, відмінність зменшується для зменшення розміру кроку.

*Переваги.* Для невеликих і середніх нейромереж, спрямований випадковий пошук дає гарні результати за короткий час. Навчання автоматичне, вимагає невеликої взаємодії з користувачем.

*Недоліки*. Кількість ваг з'єднань накладає практичні обмеження на розмір задачі. Якщо мережа має більше чим 200 ваг з'єднань, спрямований випадковий пошук може вимагати збільшення часу навчання, але продукувати прийнятні рішення.

### 2.10.6. Нейрона мережа вищого порядку або функціонально - зв'язана нейрона мережа

Функціонально-зв’язані мережі були розроблені Йох-Хан Пао (Yoh-Han Pao) і детально описані в його книзі "Адаптивне розпізнавання образів і нейроні мережі" (Adaptive Pattern Recognition and Neural Networks). Нейромережа розширює стандартну архітектуру FeedForward BackPropagation модифікацією вузлів на вхідному шарі. Входи комбінуються математичним шляхом за допомогою функцій вищого порядку, таких як квадрати, куби або синуси і розширюють сприйняття мережею заданої проблеми. З назв цих функцій вищого порядку або функціонально-зв’язаних входів і випливає назва нейромереж.

Існує два основних способи додавання вхідних вузлів. У першому, у модель можуть додаватися перехресні добутки вхідних елементів, це називається вхідним добутком або тензорною моделлю, де кожен компонент вхідного образа перемножується з усіма компонентами вхідного вектора. Наприклад, для мережі Backpropagation із трьома входами (A, B і C), перехресними добутками будуть AA, BB, CC, AB, AC і BC (елементи другого порядку). Також можуть додаватися елементи третього порядку, такі як ABC.

Другим методом для додавання вхідних вузлів є функціональне розширення базових входів. У випадку моделі з входами A, B і C, її можна перетворити в модель нейроної мережі вищого порядку з входами: A, B, C, SIN(A), COS(B), LOG(C), MAX(A,B,C) і ін. Повний ефект повинний забезпечити мережа з об'єднанням моделі тензорного і функціонального розширення. Ніякої нової інформації не додається, але розширене представлення входів робить мережу простішою для навчання. Існують обмеження для цієї моделі. Для перетворення початкових входів необхідна обробка більшої кількості вхідних вузлів, що впливають на швидкодію мережі, тому при розширенні входів повинно бути враховане об'єднання точного розв’язання і порівняно невеликого часу навчання.

### 2.10.7. Мережа Кохонена

Мережа розроблена Тойво Кохоненом на початку 1980-х рр. і принципово відрізняється від розглянутих вище мереж, оскільки використовує неконтрольоване навчання і навчальна множина складається лише зі значень вхідних змінних. Мережа розпізнає кластери в навчальних даних і розподіляє дані по відповідних кластерах. Якщо далі мережа зустрічається з набором даних, несхожим ні на один з відомих зразків, вона відносить його до нового кластера. Якщо в даних утримуються мітки класів, то мережа здатна розв’язувати задачі класифікації. Мережа Кохонена має всього два шари (рис. 2.36): вхідний і вихідний, її називають самоорганізованою картою. Елементи карти розташовуються в деякому просторі - як правило двовимірному.

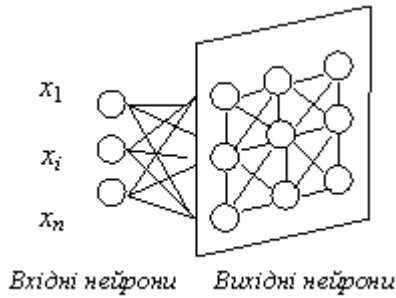



Рис. 2.36. Мережа Кохонена

Мережа Кохонена навчається методом послідовних наближень. Починаючи з випадковим чином обраного вихідного розташування центрів, алгоритм поступово поліпшується для кластеризації навчальних даних. Основний ітераційний алгоритм Кохонена послідовно проходить ряд епох, на кожній епосі обробляється один навчальний приклад. Вхідні сигнали (вектори дійсних чисел) послідовно пред'являються мережі, бажані вихідні сигнали не визначаються. Після пред'явлення достатнього числа вхідних векторів, синаптичні ваги мережі визначають кластери. Ваги організуються так, що топологічно близькі вузли чуттєві до схожих вхідних сигналів.

Для реалізації алгоритму необхідно визначити міру сусідства нейронів (околиця нейрона-переможця). На рис. 2.37 показані зони топологічного сусідства нейронів на карті ознак у різні моменти часу. NEj(t) - множина нейронів, що вважаються сусідами нейрона j у момент часу t. Зони сусідства зменшуються з часом.

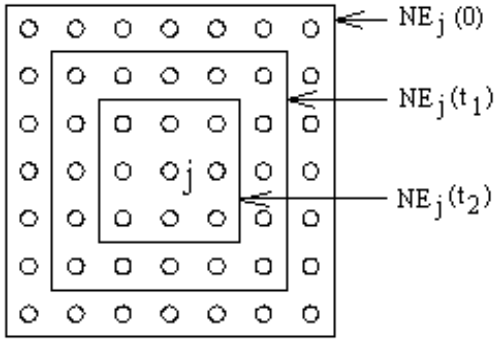



Рис. 2.37. Зони топологічного сусідства на карті ознак у різні моменти часу

*Алгоритм функціонування мережі Кохонена*

1. *Ініціалізація мережі*. Ваговим коефіцієнтам мережі даються невеликі випадкові значення. Загальне число синаптичних ваг - M * N (див. рис. 2.36). Початкова зона сусідства показана на рис. 2.37.
2. Пред'явлення мережі нового вхідного сигналу.
3. Обчислення відстані до всіх нейронів мережі:

Відстані $d_j$ від вхідного сигналу до кожного нейрона j визначаються по формулі:

$$d_j = \sum_{i=1}^{N} \left(x_i(t) \cdot w_{ij}(t)\right)^2$$

де $x_i$ - *i-ий* елемент вхідного сигналу в момент часу $t$, $w_{ij}(t)$ - вага зв'язку від *i-го* елемента вхідного сигналу до нейрона *j* у момент времени$t$.

4. Вибір нейрона з найменшою відстанню:

Вибирається нейрон-переможець $j^*$, для якого відстань $d_j$ найменша.

5. Настроювання ваг нейрона $j^*$ і його сусідів:

Виконується настроювання ваг для нейрона $j^*$ і всіх нейронів з його околиці NE. Нові значення ваг:

$$w_{ij}(t+1)=w_{ij}(t)+r(t)(x_i(t)-w_{ij}(t))$$

де $r(t)$ - швидкість навчання, що зменшується з часом (додатне число, яке менше одиниці).

6. Повернення до кроку 2.

В алгоритмі використовується коефіцієнт швидкості навчання, що поступово зменшується, для тонкої корекції на новій епосі. У результаті центр встановлюється у визначеній позиції, що задовільним чином кластеризує приклади, для яких даний нейрон є переможцем. Властивість топологічної упорядкованості досягається в алгоритмі за допомогою використання поняття околиці. Околиця - це кілька нейронів, що оточують нейрон-переможець. Відповідно швидкості навчання, розмір околиці поступово зменшується, так, що спочатку до нього належить досить велике число нейронів (можливо вся карта), на самих останніх етапах околиця стає нульовою і складається лише з нейрона-переможця. В алгоритмі навчання корекція застосовується не тільки до нейрона-переможця, але і до всіх нейронів з його поточної околиці. У результаті такої зміни околиці, початкові досить великі ділянки мережі іммігрують убік навчальних прикладів. Мережа формує грубу структуру топологічного порядку, при якій схожі приклади активують групи нейронів, що близько знаходяться на топологічній карті. З кожною новою епохою швидкість навчання і розмір околиці зменшуються, і усередині ділянок карти виявляються більш тонкі розбіжності, що приводить до точного настроювання кожного нейрона. Часто навчання навмисно розбивають на дві фази: більш коротку, з великою швидкістю навчання і великих околиць, і більш тривалу з маленькою швидкістю навчання і нульовими

або майже нульовими околицями. Після того, як мережа навчена розпізнаванню структури даних, її можна використовувати як засіб візуалізації при аналізі даних.

*Області застосування*. Кластерний аналіз, розпізнавання образів, класифікація.

*Недоліки*. Мережа може бути використана для кластерного аналізу тільки в тому випадку, якщо заздалегідь відоме число кластерів.

*Переваги*. Мережа Кохонена здатна функціонувати в умовах перешкод, тому що число кластерів фіксоване, ваги модифікуються повільно, настроювання ваг закінчується після навчання.

*Модифікації*. Одна з модифікацій полягає в тому, що до мережі Кохонена додається мережа MAXNET, що визначає нейрон з найменшою відстанню до вхідного сигналу.

### 2.10.8. Мережа квантування навчального вектора (Learning VectorQuantization)

Мережа була запропонована Тойво Кохоненом у середині 80-х рр., пізніше, ніж його робота із самоорганізованих карт. Мережа базується на шарі Кохонена, здатному до сортування прикладів у відповідні кластери і використовується як для проблем класифікації, так і для кластеризації зображень. Мережа містить вхідний шар, самоорганізовану карту Кохонена і вихідний шар. Приклад мережі зображено на рис. 2.38. Вихідний шар має стільки нейронів, скільки є відмінних категорій або класів. Карта Кохонена має ряд нейронів, згрупованих для кожного з цих класів, кількість яких залежить від складності відношення "вхід-вихід". Звичайно, кожен клас буде мати однакову кількість елементів по всьому шарі. Шар Кохонена навчається класифікації за допомогою навчальної множини. Мережа використовує правила контрольованого навчання. Вхідний шар містить стільки нейронів, скільки є окремих вхідних

параметрів. Квантування навчального вектора класифікує свої вхідні дані у визначені групування, тобто відображає n-мірний простір у m-мірний простір (бере n входів і створює m виходів). Карти зберігають відношення між близькими сусідами в навчальній множині так, що вхідні образи, що не були попередньо вивчені, будуть розподілені по категоріях їхніх найближчих сусідів у навчальних даних.

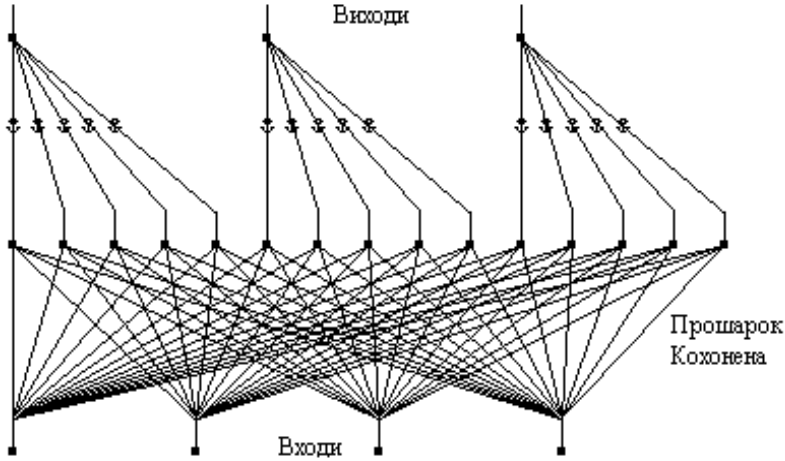



Рис. 2.38. Приклад мережі з квантуванням навчального вектора

У режимі навчання, контрольована мережа використовує шар Кохонена, де обчислюється відстань від навчального вектора до будь-якого нейрона і найближчий нейрон з'являється переможцем. Існує лише один переможець на весь шар. Переможцеві дозволено збуджувати лише один вихідний нейрон, повідомляючи клас або кластер до якого належить вхідний вектор. Якщо нейрон-переможець знаходиться в очікуваному класі навчального вектора, його ваги підсилюються в напрямку навчального вектора. Якщо нейрон-переможець не знаходиться в класі навчального вектора, ваги з'єднань зменшуються. Ця остання операція згадується як відштовхування (repulsion). Під

час навчання окремі нейрони, що приписані до часткового класу мігрують до області, зв'язаної з їхнім специфічним класом. Під час режиму функціонування, обчислюється відстань від вхідного вектора до будь-якого нейрона і знову найближчий нейрон з'являється переможцем. Це у свою чергу генерує один вихід, визначаючи частковий клас, знайдений мережею.

*Недоліки*. Для складної класифікації схожих вхідних прикладів, мережа вимагає великої карти Кохонена з великою кількістю нейронів на клас. Це може бути переборено доцільним вибором навчальних прикладів або розширенням вхідного шару. Деякі нейрони мають тенденцію до перемоги занадто часто, тобто набудовують свої ваги дуже швидко, у той час як інші постійно залишаються незадіяними. Це часто трапляється, якщо їх ваги мають значення далекі від навчальних прикладів. Для усунення цього, нейрон, що перемагає занадто часто штрафується, тобто зменшуються ваги його зв'язків з кожним вхідним нейроном. Це зменшення ваг пропорційно до різниці між частотою перемог нейрона і частотою перемог середнього нейрона.

*Переваги*. Алгоритм граничної корекції використовується для удосконалення рішення навіть якщо було знайдено відносно гарне рішення. Алгоритм здатний діяти, якщо нейрон-переможець знаходиться в неправильному класі, а другий найкращий нейрон у правильному класі. Навчальний вектор повинний бути близько від середньої точки простору, що з'єднує ці два нейрони. Неправильний нейрон-переможець зміщається з навчального вектора, а нейрон з іншого місця просувається до навчального вектора. Ця процедура робить чіткої границю між областями, де можлива невірна класифікація. На початку навчання бажано відключити відштовхування. Нейрон-переможець просувається до навчального вектора лише тоді, коли навчальний вектор і нейрон-переможець знаходяться в одному класі. Такий підхід доцільний, якщо нейрон повинний обійти область, яка має відмінний клас для досягнення необхідної області.

## 2.10.9. Мережа зустрічного поширення (CounterPropagation)

Роберт Хехт-Нильсен (Robert Hecht-Nielsen) розробив мережу Counterpropagation як засіб для об'єднання неконтрольованого шару Кохонена з контрольованим вихідним шаром. Мережа призначена для розв'язання складних класифікацій, при мінімізації числа нейронів і часу навчання. Навчання для мережі Counterpropagation схоже на мережі з квантуванням навчального вектора. Приклад мережі зображено на рис. 2.39. Односпрямована мережа CounterPropagation має три шари: вхідний шар, самоорганізована карта Кохонена і вихідний шар, що використовує правило "дельта" для зміни вхідних ваг з'єднань. Цей шар називають шаром Гроссберга.

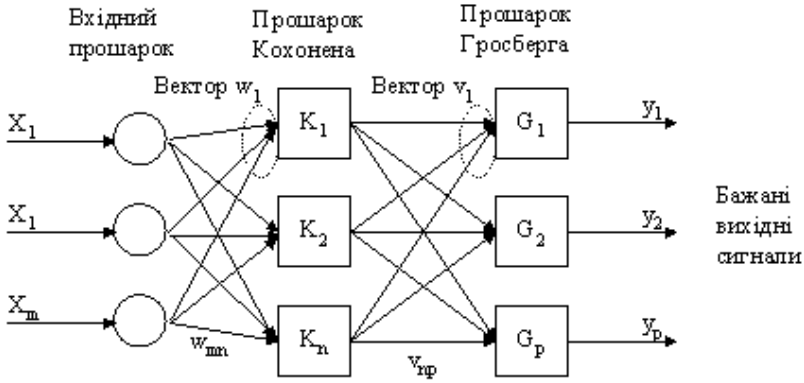



Рис. 2.39. Мережа зустрічного поширення без зворотних зв'язків

Перша мережа Counterpropagation складалася з двонаправленого відображення між вхідним і вихідним шарами. Дані надходять на вхідний шар для генерації класифікації на вихідному шарі, вихідний шар по черзі приймає додатковий вхідний вектор і генерує вихідну класифікацію на вхідному шарі мережі. Через такий зустрічно-расповсюджений потік інформації мережа одержала свою назву. Багато розроблювачів

використовують односпрямований варіант Counterpropagation, коли існує лише один шлях прямого поширення від вхідного до вихідного шару. У мережі зустрічного поширення об'єднані два алгоритми: самоорганізована карта Кохонена і зірка Гроссберга (Grossberg Outstar). Нейроні мережі, що поєднують різні нейропарадигми як будівельні блоки, більш близькі до мозку по архітектурі, чим однорідні структури. Вважається, що в мозку саме каскадні з'єднання модулів різної спеціалізації дозволяють виконувати необхідні обчислення. Кожен елемент вхідного сигналу подається на всі нейрони шару Кохонена. Ваги зв'язків ($w_{mn}$) утворюють матрицю W. Кожен нейрон шару Кохонена з'єднаний із усіма нейронами шару Гроссберга. Ваги зв'язків ($v_{np}$) утворюють матрицю ваг V. У процесі навчання мережі зустрічного поширення вхідні вектори асоціюються з відповідними вихідними векторами (двоїстими або аналоговими). Після навчання мережа формує вихідні сигнали, що відповідають вхідним сигналам. Узагальнююча здатність мережі дає можливість одержувати правильний вихід, коли вхідний вектор неповний або перекручений.

***Навчання мережі***

У результаті самонавчання шар здобуває здатність розділяти несхожі вхідні вектори. Який саме нейрон буде активуватися при пред'явленні конкретного вхідного сигналу, заздалегідь важко передбачити. При навчанні шару Кохонена на вхід подається вхідний вектор і обчислюються його скалярні добутки з векторами ваг усіх нейронів. Скалярний добуток є мірою подібності між вхідним вектором і вектором ваг. Нейрон з максимальним значенням скалярного добутку вважається "переможцем" і його ваги підсилюються (ваговий вектор наближається до вхідного).

$$w_н = w_c + r(x - w_c)$$

де $w_н$ - нове значення ваги, що з'єднує вхідний компонент $x$ з нейроном, що виграв, $w_c$ - попереднє значення ваги, r - коефіцієнт швидкості навчання, що спочатку зазвичай дорівнює 0,7 і може поступово зменшуватися в процесі навчання. Це

дозволяє робити великі початкові кроки для швидкого грубого навчання і менші кроки при підході до остаточної величини. Кожна вага, зв'язана з нейроном-переможцем Кохонена, змінюється пропорційно різниці між його величиною і величиною входу, до якого він приєднаний. Напрямок зміни мінімізує різниця між вагою і відповідним елементом вхідного шару. Навчальна множина може містити багато схожих між собою вхідних векторів, і мережа повинна бути навчено активувати один нейрон Кохонена для кожного з них. Ваги цього нейрона є усередненням вхідних векторів, що його активують. Виходи шару Кохонена подаються на входи нейронів шару Гроссберга. Входи нейронів обчислюються як зважена сума виходів шару Кохонена. Кожна вага коректується лише в тому випадку, якщо він з'єднаний з нейроном Кохонена, що має ненульовий вихід. Величина корекції ваг пропорційна різниці між вагою і необхідним виходом нейрона Гроссберга. Навчання шару Гроссберга - це навчання "із учителем", алгоритм використовує задані бажані виходи.

***Функціонування мережі.***

У своїй найпростішій формі шар Кохонена функціонує за правилом "переможець одержує все". Для даного вхідного вектора один і тільки один нейрон Кохонена видає логічну одиницю, всі інші видають нуль. Шар Гроссберга функціонує в схожій манері. Його вихід є зваженою сумою виходів шару Кохонена. Якщо шар Кохонена функціонує так, що лише один вихід дорівнює одиниці, а інші дорівнюють нулеві, то кожен нейрон шару Гроссберга видає величину ваги, що зв'язує цей нейрон з єдиним нейроном Кохонена, чий вихід відмінний від нуля. У повній моделі мережі зустрічного поширення є можливість одержувати вихідні сигнали по вхідним і навпаки. Цим двом діям відповідають пряме і зворотне поширення сигналів.

*Області застосування*. Розпізнавання образів, відновлення образів (асоціативна пам'ять), стиск даних (із утратами).

*Недоліки*. Мережа не дає можливості будувати точні апроксимації (точні відображення). У цьому мережа значно уступає мережам зі зворотним поширенням похибки. До недоліків моделі також варто віднести слабкий теоретичний базис модифікацій мережі зустрічного поширення.

*Переваги*

- Мережа зустрічного поширення проста. Вона дає можливість одержувати статистичні властивості з множини вхідних сигналів. Кохонен довів, що для навченої мережі ймовірність того, що випадково обраний вхідний вектор буде найближчим до будь-якого заданого вагового вектора, дорівнює 1/k, k - число нейронів Кохонена.
- Мережа піддається швидкому навчанню. Час навчання в порівнянні зі зворотним поширенням може бути в 100 разів менше.
- По своїх можливостях будувати відображення мережа зустрічного поширення значно перевершує одношарові перцептрони.
- Мережа корисна для додатків, де потрібна швидка початкова апроксимація.
- Мережа дає можливість будувати функцію і зворотну до неї, що знаходить застосування при рішенні практичних задач.

*Модифікації.* Мережі зустрічного поширення можуть розрізнятися способами визначення початкових значень синаптичних ваг.

- Для підвищення ефективності навчання застосовується додавання шуму до вхідних векторів.
- Ще один метод підвищення ефективності навчання - надання кожному нейронові "почуття справедливості". Якщо нейрон стає переможцем частіше, ніж 1/k (k - число нейронів Кохонена), то йому тимчасово збільшують поріг, даючи тим самим учитися й іншим нейронам.
- Крім "методу акредитації", при якому для кожного вхідного вектора активується лише один нейрон Кохонена, може

бути використаний "метод інтерполяції", при використанні якого ціла група нейронів Кохонена, що мають найбільші виходи, може передавати свої вихідні сигнали в шар Гроссберга. Цей метод підвищує точність відображень, реалізованих мережею.

### 2.10.10. Ймовірнісна нейрона мережа.

Ймовірнісна нейрона мережа була розроблена Дональдом Спехтом (Donald Specht). Ця мережна архітектура була вперше представлена в двох статтях : "Ймовірнісні нейроні мережі для класифікації" (Probabilistic Neural Networks for Classification) 1988, "Відображення або асоціативна пам'ять і ймовірнісні нейроні мережі" (Mapping or Associative Memory and Probabilistic Neural Networks) 1990 р.

Виходи мережі можна інтерпретувати, як оцінки ймовірності приналежності елемента до визначеного класу. Ймовірнісна мережа учиться оцінювати функцію щільності ймовірності, її вихід розглядається як очікуване значення моделі в даній точці простору входів. Це значення зв'язане з щільністю ймовірності загального розподілу вхідних і вихідних даних. Задача оцінки щільності ймовірності відноситься до області байєсівської статистики. Звичайна статистика по заданій моделі показує, яка ймовірність того або іншого виходу (наприклад, на гральній кісті 6 очок буде випадати в середньому в одному випадку із шести). Байєсівська статистика інтерпретує по іншому: правильність моделі оцінюється по наявним достовірним даним, тобто дає можливість оцінювати щільність ймовірності розподілу параметрів моделі по наявним даним. При рішенні задач класифікації можна оцінити щільність ймовірності для кожного класу, порівняти між собою ймовірності приналежності до різних класів і вибрати модель з параметрами, при яких щільність ймовірності буде більшою. Оцінка щільності ймовірності в мережі заснована на ядерних оцінках. Якщо приклад розташований у даній точці простору, тоді в цій точці є визначена щільність ймовірності. Кластери з поруч розташованих точок, свідчать, що в цьому місці щільність ймовірності велика. Біля спостереження є більша довіра до рівня щільності, а в міру віддалення від нього довіра зменшується і прагне до нуля. У методі ядерних оцінок у

точці, що відповідає кожному прикладові, міститься деяка проста функція, потім вони усі додаються й у результаті виходить оцінка для загальної щільності ймовірності. Частіше як ядерні функції беруть колоколоподібні функції (гауссовскі). Якщо є достатня кількість навчальних прикладів, такий метод дає гарні наближення до дійсної щільності ймовірності. Ймовірнісна мережа має три шари: вхідний, радіальний і вихідний. Радіальні елементи беруться по одному на кожен приклад. Кожний з них має гауссову функцію з центром у цьому прикладі. Кожному класові відповідає один вихідний елемент. Вихідний елемент з'єднаний лише з радіальними елементами, що відносяться до його класу і підсумовує виходи всіх елементів, що належать до його класу. Значення вихідних сигналів виходять пропорційно ядерним оцінкам ймовірності приналежності відповідним класам.

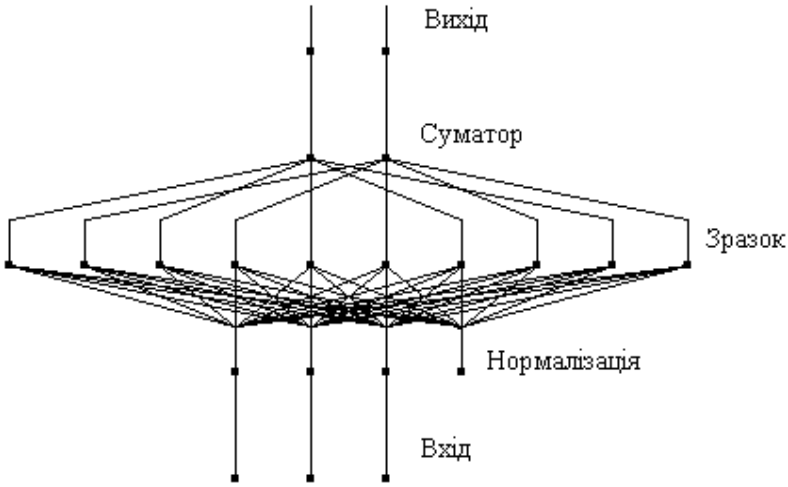



Рис. 2.40. Схема ймовірнисної нейронної мережі.

*Переваги*. Навчання ймовірнисної нейронної мережі набагато простіше, ніж BackPropagation.

*Недоліки.* Недоліком мережі є її розмір, оскільки вона фактично містить у собі всі навчальні дані, вимагає багато пам'яті і може повільно працювати.

*Модифікації*. Базова модель ймовірнисної нейроної мережі має модифікації. Припустимо, що пропорції класів у навчальній множині відповідають їх пропорціям у всій досліджуваній множині (апріорна ймовірність). Наприклад, якщо серед усіх людей хворими є 2%, то в навчальній множині для мережі, що діагностує захворювання, хворих також повинне бути 2%. Якщо ж апріорні ймовірності відрізняються від пропорції в навчальній вибірці, мережа буде видавати невірний результат. Це можна врахувати, уводячи коригувальні коефіцієнти для різних класів.

## 2.10.11. Мережа Хопфілда

Джон Хопфілд уперше представив свою асоціативну мережу в 1982 р. у Національній Академії Наук. На честь Хопфілда і нового підходу до моделювання, ця мережна парадигма згадується як мережа Хопфілда. Мережа базується на аналогії фізики динамічних систем. Початкові застосування для цього виду мережі включали асоціативну, або адресовану за змістом, пам'ять і розв’язували задачі оптимізації. Мережа Хопфілда використовує три шари: вхідний, шар Хопфілда і вихідний шар. Кожен шар має однакова кількість нейронів. Входи шару Хопфілда приєднані до виходів відповідних нейронів вхідного шару через ваги з'єднань, що змінюються. Виходи шару Хопфілда приєднуються до входів усіх нейронів шару Хопфілда, за винятком самого себе, а також до відповідних елементів у вихідному шарі. У режимі функціонування, мережа направляє дані з вхідного шару через фіксовані ваги з'єднань до шару Хопфілда. Шар Хопфілда коливається, поки не буде довершена визначена кількість циклів, і поточний стан шару передається на вихідний шар. Цей стан відповідає образові, уже запрограмованому в мережу. Навчання мережі Хопфілда вимагає, щоб навчальний образ був представлений на вхідному і вихідному шарах одночасно. Рекурсивний характер шару Хопфілда забезпечує засоби корекції усіх ваг з'єднань. Недвійкова реалізація мережі повинна мати граничний механізм

у передатній функції. Для правильного навчання мережі відповідні пари "вхід-вихід" повинні відрізнятися між собою. Якщо мережа Хопфілда використовується як пам'ять, адресуєма за змістом, вона має два головних обмеження. По-перше, число образів, що можуть бути збережені і точно відтворені є строго обмеженим. Якщо зберігається занадто багато параметрів, мережа може сходиться до нового неіснуючого образа, відмінному від усіх запрограмованих образів, або не сходиться взагалі. Границя ємності пам'яті для мережі приблизно 15% від числа нейронів у шарі Хопфілда. Другим обмеженням парадигми є те, що шар Хопфілда може стати нестабільним, якщо навчальні приклади є занадто схожими. Зразок образа вважається нестабільним, якщо він застосовується за нульовий час і мережа сходиться до деякого іншого образа з навчальної множини. Ця проблема може бути розв'язана вибором навчальних прикладів більш ортогональних між собою.

Структурна схема мережі Хопфилда приведена на рис. 2.41.

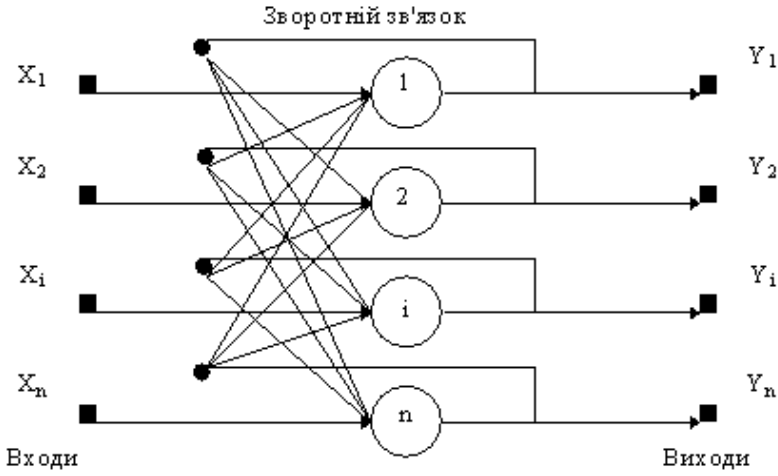



Рис. 2.41. Структурна схема мережі Хопфілда

Задача, розв'язувана даною мережею як асоціативна пам'ять, як правило, формулюється так. Відомий деякий зразковий набір двоїстих сигналів (зображень, звукових оцифровок, інших даних, що описують визначені об'єкти або характеристики процесів). Мережа повинна вміти з зашумленого сигналу, представленого на її вхід, виділити ("згадати" за частковою інформацією) відповідний зразок або "дати висновок" про те, що вхідні дані не відповідають жодному зі зразків. У загальному випадку, будь-який сигнал може бути описаний вектором $x_1$, $x_i$, $x_n$..., $n$ - число нейронів у мережі і величина вхідних і вихідних векторів. Кожен елемент $x_i$ дорівнює або +1, або -1. Позначимо вектор, що описує k-й зразок, через $X_k$, а його компоненти, відповідно, - $x_{ik}$, $k$=0, ..., $m$-1, $m$ - число зразків. Якщо мережа розпізнає (або "згадує") визначений зразок на основі пред'явлених їй даних, її виходи будуть містити саме його, тобто $Y=X_k$, де $Y$ - вектор вихідних значень мережі: $y_1$, $y_i$, $y_n$. У противному випадку, вихідний вектор не збіжиться з жодним зразком. Якщо, наприклад, сигнали являють собою якесь зображення, то, відобразивши в графічному виді дані з виходу мережі, можна буде побачити картинку, що цілком збігається з однією зі зразкових (у випадку успіху) або ж "вільну імпровізацію" мережі (у випадку невдачі).

*Алгоритм функціонування мережі*

1. На стадії ініціалізації мережі синаптичні коефіцієнти встановлюються в такий спосіб:

$$W_{ij} = \begin{cases} \sum_{k=1}^{m} x_i^k x_j^k, & i \neq j \\ 0, & i = j \end{cases}$$

Тут $i$ і $j$ - індекси, відповідно, передсинаптичного і постсинаптичного нейронів; $x_{ik}$, $x_{jk}$ - *i-ий* і *j-ий* елементи вектора *k*-ого зразка.

2. На входи мережі подається невідомий сигнал. Його поширення безпосередньо встановлює значення виходів: $y_i(0) = x_i$ , $i = 0...n-1$, тому позначення на схемі мережі вхідних сигналів у явному виді носить чисто умовний характер. Нуль у дужці $y_i$ означає нульову ітерацію в циклі роботи мережі.

3. Розраховується новий стан нейронів

$$S(t+1) = \sum_{i=1}^{n} w_{ij} y_j(t), \quad j=0...n-1$$

і нові значення виходів

$$y_j(t+1) = f\left[S_j(t+1)\right]$$

де $f$ - передатна функція у виді граничної, приведена на рис. 2.42.

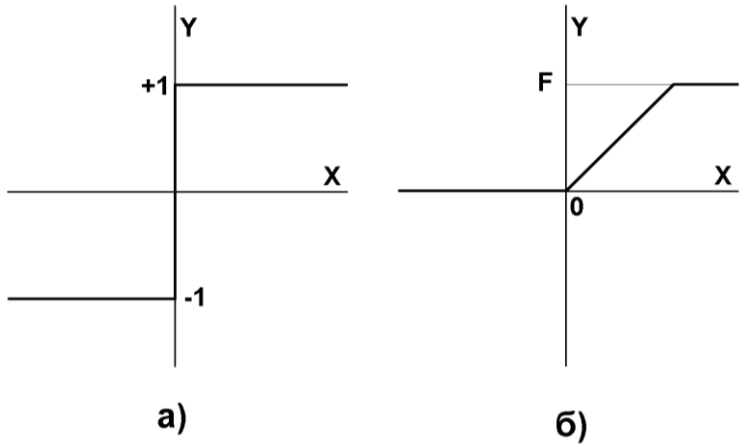



Рис. 2.42. Передатні функції

4. Перевіряємо чи змінилися вихідні значення виходів за останню ітерацію. Якщо так - перехід до пункту 2, інакше (якщо виходи стабілізувалися) - кінець. При цьому вихідний вектор являє собою зразок, що найкраще відповідає вхідним даним.

Іноді мережа не може провести розпізнавання і видає на виході неіснуючий образ. Це зв'язано з проблемою обмеженості можливостей мережі. Крім того, якщо два образи А і Б сильно схожі, вони, можливо, будуть викликати в мережі перехресні асоціації, тобто пред'явлення на входи мережі вектора А приведе до появи на її виходах вектора Б и навпаки. Завдяки ітераційному алгоритмові, машина просувається до найкращого рішення

## 2.10.12. Мережа «Машина Больцмана»

Мережа «Машина Больцмана» (Boltzmann mashine) схожа по функції і дії на мережу Хопфілда і включає поняття "модельованого віджигу" для пошуку в просторі станів шару образів глобального мінімуму.

Экли (Ackley), Хинтон (Hinton) і Сейновски (Sejnowski) розробили правило больцмановского навчання в 1985 р. Подібно мережі Хопфілда, машина Больцмана має простір станів, що базується на вагах з'єднань у шарі образів. Процеси навчання мережі, наповненої образами, включає згладжування рельєфу простору станів. Машина Больцмана моделює віджиг металу, що додається до процесу навчання мережі. Як і при фізичному віджигу, температура починається з великих значень і зменшується з часом. Збільшена температура додає збільшений шумовий коефіцієнт до будь-якого нейрона в шарі образів. Переважно, кінцевою температурою буде нуль. Для досягнення оптимального рішення доцільно на нижчих температурах додавати більше ітерацій. Машина Больцмана, навчаючись на високій температурі, поводиться як випадкова модель, а на низьких температурах вона як детермінована модель. Через випадковий компонент в віджиговому навчанні, нейрон може прийняти нове значення стану, що збільшується швидше, ніж зменшується загальний простір станів. Імітація фізичного віджигу

дозволяє просуватися до глобального мінімуму, уникаючи локальний. Як і в мережі Хопфілда, мережі може бути представлений частковий образ для відновлення відсутньої інформації.

*Алгоритм функціонування мережі*

1. Визначити змінну T, що представляє штучну температуру.
2. Пред'явити мережі множину входів і обчислити виходи і цільову функцію.
3. Дати випадкову зміну вагам і перерахувати вихід мережі і зміну цільової функції відповідно до зміни ваг.
4. Якщо цільова функція зменшилася, тоді нова множина ваг зберігається. Якщо зміна ваг приводить до збільшення цільової функції, то ймовірність збереження цієї зміни обчислюється за допомогою розподілу Больцмана:

$$P(c) = \exp\left(-\frac{c}{kT}\right)$$

де P(c) - ймовірність зміни c у цільовій функції; k - константа, аналогічна константі Больцмана, вибирається в залежності від задачі; T - штучна температура.

Вибирається випадкове число r з рівномірного розподілу від нуля до одиниці. Якщо P(c) більше, ніж r, то зміна зберігається, у противному випадку величина ваги повертається до попереднього значення. Ця процедура дає можливість системі робити випадковий крок у напрямку, що псує цільову функцію, дозволяючи їй вириватися з локальних мінімумів. Кроки 3 і 4 повторюються для усіх ваг мережі, поступово зменшуючи температуру T, поки не буде досягнуто припустиме низьке значення цільової функції. У цей момент пред'являється інший вхідний вектор і процес навчання повторюється. Мережа учиться на усіх векторах навчальної множини, поки цільова функція не стане припустимої для усіх з них. Швидкість зменшення

температури повинна бути обернено пропорційна логарифмові часу. При цьому мережа сходиться до глобального мінімуму.

*Області застосування*.

Розпізнавання образів, класифікація.

*Недоліки*. Повільний алгоритм навчання.

*Переваги.* Алгоритм дає можливість мережі вибиратися з локальних мінімумів адаптивного рельєфу простору станів.

*Модифікації*. Випадкові зміни можуть проводитися не тільки для окремих ваг, але і для всіх нейронів шарів у багатошарових мережах або для всіх нейронів мережі одночасно. Ці модифікації алгоритму дають можливість скоротити загальне число ітерацій навчання.

### 2.10.13. Мережа Хеммінга

Мережа Хеммінга (Hamming) - розширення мережі Хопфілда. Ця мережа була розроблена Річардом Ліппманом (Richard Lippman) у середині 80-х рр. Мережа Хеммінга реалізує класифікатор, що базується на найменшій похибці для векторів двоїстих входів, де похибка визначається відстанню Хеммінга. Відстань Хеммінга визначається як число біт, що відрізняються між двома відповідними вхідними векторами фіксованої довжини. Один вхідний вектор є незашумленим прикладом образа, інший - зіпсованим образом. Вектор виходів навчальної множини є вектором класів, до яких належать образи. У режимі навчання вхідні вектори розподіляються по категоріях, для яких відстань між зразковими вхідними векторами і поточним вхідним вектором є мінімальним. Мережа Хеммінга має три шари: вхідний шар з такою кількістю вузлів, скільки є окремих двоїстих ознак; шар категорій (шар Хопфілда), з кількістю вузлів, скільки є категорій або класів; вихідний шар, що відповідає числу вузлів у шарі категорій. Мережа є простою архітектурою прямого поширення з вхідним рівнем, цілком приєднаним до шару

категорій. Кожен елемент обробки в шарі категорій є зворотно приєднаним до кожного нейрона в тім же самому шарі і прямо приєднаним до вихідного нейрона. Вихід із шару категорій до вихідного шару формується через конкуренцію. Навчання мережі Хеммінга схоже на методологію Хопфілда. На вхідний шар надходить бажаний навчальний образ, а на вихід вихідного шару надходить значення бажаного класу, до якого належить вектор. Вихід містить лишь значення класу до якого належить вхідний вектор. Рекурсивний характер шару Хопфілда забезпечує засоби корекції усіх ваг з'єднань.

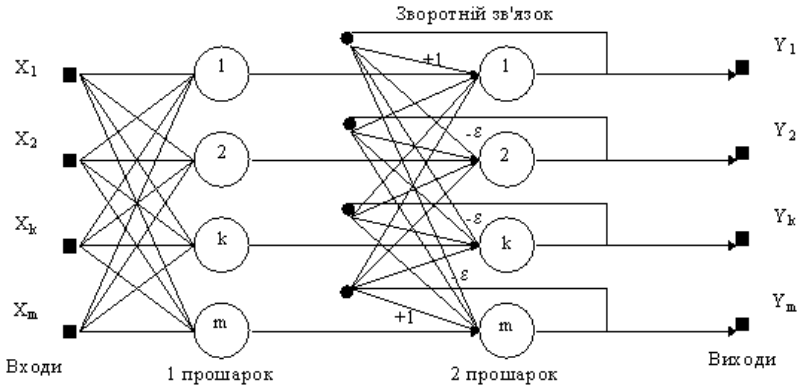



Рис. 2.43. Структурна схема мережі Хеммінга

*Алгоритм функціонування мережі Хеммінга*

1. На стадії ініціалізації ваговим коефіцієнтам першого шару і порогові передатної функції привласнюються такі значення:

$$W_{ik}=x_i^k/2,\ i=0...n-1,\ k=0...m-1$$

$$b_k = n / 2,\ k = 0...m-1$$

Тут $x_i^k$ - *i*-ий елемент *k*-ого зразка.

Вагові коефіцієнти гальмуючих синапсів у другому шарі беруть рівними деякій величині 0<v<1/m. Синапс нейрона, зв'язаний з його ж виходом має вагу +1.

2. На входи мережі подається невідомий вектор $x_1$, $x_i$, $x_n$ ... Розраховуються стани нейронів першого шару (верхній індекс у дужках указує номер шару):

$$y_j^{(1)} = S_j^{(1)} = \sum_{i=1}^{n} w_{ij} x_i + b_j \quad , j=0...m-1$$

Після цього отримані значення ініціюють значення виходів другого шару:

$$y_j^{(2)} = y_j^{(1)}, j = 0...m-1$$

3. Обчислюються нові стани нейронів другого шару:

$$S_j^{(2)}(t+1) = y_j(t) - \varepsilon \sum_{k=1}^{m} y_k^{(2)}(t), k \neq j, j = 1..m$$

і значення їхніх виходів:

$$y_j^{(1)}(t+1) = f\left[S_j^{(2)}(t+1)\right]$$

Передатна функція *f* має вигляд порога, причому величина b повинна бути досить великою, щоб будь-які можливі значення аргументу не приводили до насичення.

4. Перевіряється, чи змінилися виходи нейронів другого шару за останню ітерацію. Якщо так - перейти до кроку 3. Інакше -кінець.

Роль першого шару є умовною: скориставшись один раз на першому кроці значеннями його вагових коефіцієнтів, мережа

більше не повертається до нього, тому перший шар може бути узагалі виключений з мережі. Мережа Хеммінга має ряд переваг над мережею Хопфілда. Вона здатна знайти мінімальну похибку, якщо похибки вхідних біт є випадковими і незалежними. Для функціонування мережі Хеммінга потрібна менша кількість нейронів, оскільки середній шар вимагає лише один нейрон на клас, замість нейрона на кожен вхідний вузол. І, зрештою, мережа Хеммінга не страждає від неправильних класифікацій, що можуть статися в мережі Хопфілда. У цілому, мережа Хеммінга швидша і точніша, ніж мережа Хопфілда.

### 2.10.14. Мережа мережної моделі з двонаправленою асоціативною пам'ятю

Ця мережна моделі була розроблена Бартом Козко (Bart Kosko) і розширює модель Хопфілда. Множина парних образів учиться по образах, що представлені як біполярні вектори. Подібно мережі Хопфілда, коли представляється зашумлена версія одного образа, визначається найближчий образ, асоційований з ним. На рис. 2.44 показаний приклад двонаправленої асоціативної пам'яті.

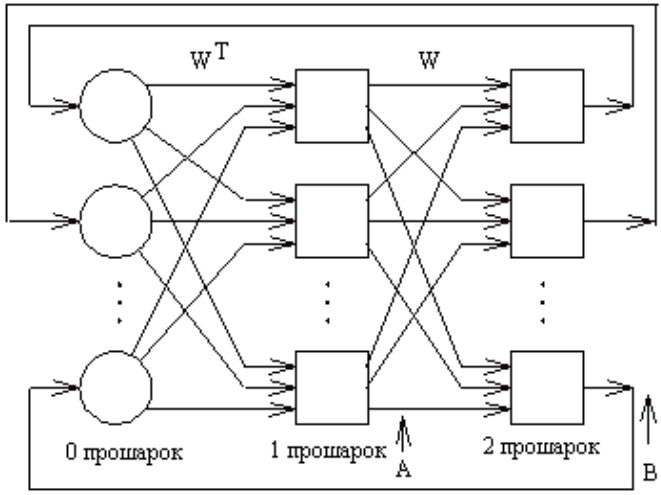



Рис. 2.44. Двонаправлена асоціативна пам'ять

Вона має стільки входів, скільки є виходів. Два схованих шари містяться на двох окремих асоціативних елементах пам'яті і представляють подвоєний розмір вхідних векторів. Середні шари цілком з'єднані між собою. Вхідний і вихідний шари потрібні для реалізації засобів введення і відтворення інформації з мережі. Середні шари розроблені для збереження асоційованих пар векторів. Якщо на вхід мережі надходить зашумлений вектор образа, середні шари коливаються до досягнення стабільного стану рівноваги, що відповідає найближчій навченій асоціації і на виході генерується образ з навчальної множини. Подібно мережі Хопфілда, двонаправлена асоціативна пам'ять схильна до неправильного відшукування навченого образа, якщо надходить невідомий вхідний вектор, що не був у складі навчальної множини.

Двонаправлена асоціативна пам'ять відноситься до гетероасоціативної пам'яті. Вхідний вектор надходить на один набір нейронів, а відповідний вихідний вектор продукується на іншому наборі нейронів. Вхідні образи асоціюються з вихідними.

Для порівняння: мережа Хопфілда автоасоціативна. Вхідний образ може бути відновлений або виправлений мережею, але не може бути асоційований з іншим образом. У мережі Хопфілда використовується одношарова структура асоціативної пам'яті, у якій вихідний вектор з'являється на виході тих же нейронів, на які надходить вхідний вектор. Двонаправлена асоціативна пам'ять, як і мережа Хопфілда, здатна до узагальнення, виробляючи правильні вихідні сигнали, незважаючи на ушкоджені входи.

Розглянемо схему двонаправленої асоціативної пам'яті. Вхідний вектор A обробляється матрицею ваг W мережі, у результаті чого продукується вектор вихідних сигналів мережі B. Вектор B обробляється транспонованою матрицею WT ваг мережі, що продукує сигнали, які представляють новий вхідний вектор A. Цей процес повторюється доти, поки мережа не досягне стабільного стану, у якому ні вектор A, ні вектор B не змінюються.

Нейрони в шарах 1 і 2 функціонують, як і в інших парадигмах, обчислюючи суму зважених входів і значення передатної функції F:

$$b_j = F\sum_j a_j w_{ij}$$

або у векторній формі:

$$B=F(AW)$$

де *B*- вектор вихідних сигналів нейронів шару 2, A - вектор вихідних сигналів нейронів шару 1, W - матриця ваг зв'язків між шарами 1 і 2, F - передатна функція.

Аналогічно

$$A=F(BW^T),$$

де $W^T$ є транспозицією матриці W.

Як передатну функцію використовується експонентна сігмоїда. Шар 0 не виконує обчислень і не має пам'яті. Він служить лише засобом розподілу вихідних сигналів шару 2 до елементів матриці $W^T$. Формула для обчислення значень синаптичних ваг:

$$W = \sum_j A_j^T B_j$$

де $A_j$ і $B_j$ - вхідні і вихідні сигнали навчальної множини.

Вагова матриця обчислюється як сума добутків усіх векторних пар навчальної множини. Системи зі зворотним зв'язком мають тенденцію до коливань. Вони можуть переходити від стану до стану, ніколи не досягаючи стабільності. Доведено,

що двонаправлена асоціативна пам'ять безумовно стабільна при будь-яких значеннях ваг мережі.

*Області застосування*.

Асоціативна пам'ять, розпізнавання образів.

*Недоліки*. Ємкість двонаправленої асоціативної пам'яті жорстко обмежена. Якщо n - кількість нейронів у вхідному шарі, то число векторів, що можуть бути запом'ятовані в мережі не перевищує $L=n/2log_2n$. Так, якщо n=1024, то мережа здатна запам'ятати не більш 25 образів. Двоунаправлена асоціативна пам'ять має деяку непередбачуваність у процесі функціонування, можливі помилкові відповіді.

*Переваги*

- У порівнянні з автосоціативною пам'яттю (наприклад, мережею Хопфілда), двоунаправлена асоціативна пам'ять дає можливість будувати асоціації між векторами A і B, що у загальному випадку мають різні розмірності. За рахунок таких можливостей гетероасоціиативна пам'ять має більш широкий спектр застосувань, чим автоасоціативна пам'ять.
- Двонаправлена асоціативна пам'ять - проста мережа, що може бути реалізована у виді окремої СБІС або оптоелектроним способом.
- Процес формування синаптичних ваг простий і швидкий. Мережа швидко сходиться в процесі функціонування.
- Сигнали в мережі можуть бути як дискретними, так і аналоговими. Для обох випадків доведена стабільність мережі.

### 2.10.15. Мережа адаптивної резонансної теорії (ART)

Розроблена Стивеном Гроссбергом і Карпентером у середині 80-х рр. Парадигма використовує неконтрольоване навчання, аналізує значимі вхідні дані, виявляє можливі ознаки і класифікує образи у вхідному векторі.

Мережа адаптивної резонансної теорії складається з двох взаємозалежних шарів нейронів, розташованих між вхідним і вихідним шарами. Кожен вхідний образ нижчого шару резонансу стимулює очікуваний образ на вищому шарі, що пересилається до нижчого шару, щоб впливати на наступний вхід. Це створює "резонанс" між нижчим і вищим шарами для полегшення мережної адаптації образів. Мережа переважно використовується в біологічному моделюванні, проте існують деякі технічні застосування. Головним обмеженням мережної архітектури є її шумова чутливість. Навіть невелика кількість шуму на вхідному векторі плутає узагальнюючі можливості навченої мережі. Мережа ART-1 реалізує алгоритм кластерізації, дуже схожий на алгоритм "послідовного лідера". Дотримуючись цього алгоритму перший вхідний сигнал вважається зразком першого кластера. Наступний вхідний сигнал порівнюється зі зразком першого кластера. Говорять, що вхідний сигнал "направляється за лідером" і належить першому кластерові, якщо відстань до зразка першого кластера менша порога. У противному випадку другий вхідний сигнал - зразок другого кластера. Цей процес повторюється для всіх наступних вхідних сигналів. Таким чином, число кластерів росте з часом і залежить як від значення порога, так і від метрики відстані, використовуванної для порівняння вхідних сигналів і зразків класів. Основна частина мережі ART-1 схожа на мережу Хеммінга. За допомогою послідовних зв'язків обчислюється відповідність вхідних сигналів і зразків кластерів. Максимальне значення відповідності підсилюється за допомогою латеральних зв'язків вихідних нейронів.

Мережа ART-1 відрізняється від мережі Хеммінга зворотними зв'язками від вихідних нейронів до вхідних, крім того є можливість виключати вихідний нейрон з максимальним значенням відповідності і проводити тестування відповідності вхідного сигналу і зразків кластерів, як того вимагає алгоритм "послідовного лідера".

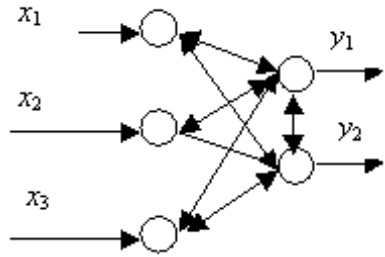



Рис. 2.45. Основні компоненти класифікатора Карпентер/Гроссберга

Алгоритм функціонування мережі

1. Ініціалізація мережі:

$$v_{ij}(0)=1; W_{ij}(0)=1/(1+N)$$

$$0 \le i \le N-1;\ 0 \le j \le M-1;\ 0 \le b \le 1;$$

де $w_{ij}(t)$ - синаптичні вага зв'язку від і-го нейрона першого шару до j-го нейрона другого шару в момент часу t, $v_{ij}(t)$ - синаптична вага зв'язку від j-го нейрона другого шару до і-го нейрона першого шару в момент часу t, b - значення порога. Ваги $v_{ij}(t)$ і $w_{ij}(t)$ визначають зразок, що відповідає нейронові $j$. Поріг b показує, наскільки повинен вхідний сигнал збігатися з одним із запам’ятованих зразків, щоб вони вважалися схожими. Близьке до одиниці значення порога вимагає майже повного збігу. При невеликих значеннях порога вхідний сигнал і зразок, навіть, якщо вони сильно розрізняються, вважаються приналежними до одного кластера.

2. Пред'явлення мережі нового бінарного вхідного сигналу:

Вхідні сигнали пред'являються вихідному шарові нейронів аналогічно мережі Хеммінга.

3. Обчислення значень відповідності:

$$y_j = \sum_{i=1}^{N} w_{ij}(t) x_i$$

Значення відповідності обчислюються паралельно для всіх зразків, запам'ятованих у мережі, аналогічно мережі Хеммінга.

4. Вибір зразка з найбільшою відповідністю:

$$y_j = max(y_j)$$

Ця операція виконується за допомогою латерального гальмування.

5. Порівняння з порогом:

$$\|X\| = \sum_{j=1}^{N} x_j \quad \|V \cdot X\| = \sum_{j=1}^{N} v_{ij} \times x_j$$

;

якщо $\frac{\|V \cdot X\|}{\|X\|} > b$ перехід до кроку 7, інакше до кроку 6.

На цьому кроці обчислюється відношення скалярного добутку вхідного сигналу і зразка з найбільшим значенням відповідності до числа одиничних біт вхідного сигналу. Значення відношення порівнюється з порогом, уведеному на першому кроці. Якщо значення відношення більше порога, то вхідний сигнал вважається схожим на зразок найбільшому значенню відповідності. У цьому випадку зразок модифікується шляхом виконання операції AND (логічне "І"). Новий зразок є зразком на попередньому кроці + вхідний сигнал. Якщо значення відношення менше порога, то вважається, що вхідний сигнал відрізняється від усіх зразків і розглядається як новий зразок. У

мережу вводиться нейрон, що відповідає новому зразкові, і обчислюються значення синаптичних ваг.

6. Виключення приклада з найбільшим значенням відповідності:

Вихід нейрона з найбільшим значенням відповідності тимчасово встановлюється рівним нулеві і більш не приймає участі в кроці 4.

7. Адаптація приклада з найбільшим значенням відповідності:

$$v_{ij}(t+1)=v_{ij}(t)x_j$$

$$w_{ij}(t+1)=\frac{v_{ij}(t)x_j}{0,5+\sum_{j=1}^{N}v_{ij}(t)x_j}$$

8. Включення усіх виключених на кроці 6 зразків. Повернення до кроку 2.

Вхідні сигнали в цій моделі бінарні.

Розмірності входу і виходу обмежені при програмній реалізації тільки можливостями обчислювальної системи, на якій моделюється нейрона мережа, при апаратній реалізації - технологічними можливостями. Ємкість мережі збігається з числом нейронів другого шару і може збільшуватися в процесі функціонування мережі.

*Області застосування*. Розпізнавання образів, кластерний аналіз.

*Недоліки*. Необмежене збільшення числа нейронів у процесі функціонування мережі. З появою шуму виникають значні проблеми, зв'язані з неконтрольованим ростом числа зразків.

*Переваги.* Навчання без учителя.

*Модифікації.* Існує модель ART-2 з аналоговими значеннями вхідними сигналами.

## 2.11. Нейромержі в задачах відображення

### 2.11.1. Типи задач відображення і підходи до їхнього розв’язання

Відображення розглядають як функцію f, яка визначена на множині X, що приймає свої значення серед елементів множини Y і кожному елементові з X може відповідати один і лише один елемент із Y. Пари векторів X і Y називають прикладом або реалізацією.

У задачах відображення нейромережі здійснюють оцінювання і прогнозування поводження об'єктів, у тому числі систем і процесів, заданих визначеними законами і сукупністю своїх реалізацій. Кожна реалізація повинна містити набір ознак, що визначають основний зміст об'єкта. Якщо одним з ознак об'єкта дослідження є час, тоді реалізації можуть бути представлені у виді годинних рядів.

У більшості реальних об'єктів дослідження можна виділити їх основні складові:

- детермінована складова, котра в принципі підлягає точному прогнозуванню;
- ймовірнісна складова, котру можна прогнозувати з заданим степенем ймовірності;

- чисто випадкова складова, котру неможливо ні врахувати, ні передбачити.

У залежності від степеня впливу тієї або інший складової, можна говорити про визначений тип множин даних, використовуваних для навчання нейромереж:

- детермінована множина даних, із усіма врахованими основними параметрами, викликаними дією відомих причин. Вона характеризується малим рівнем шумів;
- множина даних з наявністю ймовірнісної складової, що випливає з експериментальної постановки задачі, з різним степенем урахуванням діючих факторів і з впливом похибок оцінювання;
- множина даних з наявністю випадкової складової, унаслідок неврахованих ознак явища.

Такий розподіл варто вважати приблизним, але при оцінюванні об'єкта дослідження потрібно виділяти такі ознаки, для яких можливе зменшення чисто випадкової складової, оцінювання ймовірностей складових і максимальне збільшення детермінованої частини.

Задачі відображення можна розбити на два основних класи:

- класифікація;
- регресія.

У задачах класифікації потрібно визначити, до якого з заданих класів належить вхідний набір. Прикладами можуть служити надання кредиту, діагностика захворювань, розпізнавання образів.

У задачах регресії передбачають значення змінної, яка приймає безперервні числові значення: ціна акцій, витрата палива в автомобілі, прибуток кампанії і т.п.

Передбачення явищ можна розділити на:

- передбачення виходів для множини дискретних вхідних даних, не зв'язаних часом (економічні, соціологічні оцінки й ін.); дані представляються таблично;
- прогнозування явищ, що безупинно змінюються в часі (фізичні процеси, природні явища, і т.п.); дані представлляются у виді тимчасових рядів.

Для розв'язання задачі за допомогою нейроної мережі, необхідно зібрати дані для навчання. Навчальна множина даних являє собою набір прикладів, для яких відомо значення вхідних і вихідних параметрів. Перше, що потрібно вирішити, - які параметри використовувати і скільки прикладів вибрати.

Спочатку вибір параметрів здійснюється інтуїтивно. Досвід роботи в предметній області допомагає визначити, які змінні є важливими. Для початку має сенс включити всі змінні, котрі можуть впливати на результат - на наступних етапах цю множину можна скоротити. Для забезпечення обґрунтованого вибору і видалення несуттєвих ознак, що вносять додаткові перекручування при навчанні, можливе застосування методів математичної статистики.

**Факторний аналіз**

Внесок кожної вхідної ознаки можна оцінити по її впливі на середнє значення вихідної величини. Нехай зовнішній вихід моделі нейромережі залежить від декількох факторів

$$y=f(a_1x_1, a_2x_2, ..., a_ix_i...)$$

Виберемо деякий фактор $a_ix_i$. Для всіх реалізацій навчальної множини визначимо значення вихідної величини при наявності і відсутності цього фактора. Обчислимо дисперсію, викликану відсутністю фактора $a_ix_i$.

$$S^2_{a_i} = \frac{1}{N} S^2_y = \frac{1}{N} \sum_{i=1}^{N} \left( y^*_{x_i} - y_{x_i} \right)^2$$

де $y_{xi}^*$, $y_{xi}$ - відповідно значення середньої величини при відсутності і наявності фактора $a_i x_i$.

Визначаємо інтервал $\Delta a = \pm 2S_{ai}$, куди не повинна попадати оцінка коефіцієнтів $a_i$. При малих коефіцієнтах даний фактор вилучається.

**Кореляційний аналіз**

Деякі з параметрів, прийняті в увагу, мають незначний вплив на формування виходів і можуть бути відкинуті. Як показник взаємозалежності між системою вхідних величин $X=(X_1, X_2, ..., X_n)$ і вихідних величин $Y$ , можна вибрати коэфициент парної кореляції (наприклад вхідний змінної $X_1$ і вихідного значення $Y$)

$$k = \frac{\sum_{i=1}^{N}(X_{1i} - \overline{X}_{1i})(Y_i - \overline{Y}_i)}{\sqrt{\sum_{i=1}^{N}(X_{1i} - \overline{X}_{1i})^2 \sum_{i=1}^{N}(Y_i - \overline{Y}_i)^2}},$$

де $\overline{X}_{1i} = \frac{1}{N}\sum_{i=1}^{N} X_{1i}$; $\overline{Y}_i = \frac{1}{N}\sum_{i=1}^{N} Y_i$; $N$ - число реалізацій.

Значення $k$<0,6 вважають граничним. Наприклад, при знаходженні коефіцієнтів кореляції між виходом і входами можна визначити стеупінь впливу кожного вхідного параметра на вихід, і використовувати даний показник для ранжирування входів.

**Ранжирування входів**

При прогнозуванні, істотним для якості є облік реального впливу кожного параметра входу $x(x_1, ..., x_n)$ на вихідний вектор

*y*. За допомогою кореляційного аналізу заздалегідь обчислюються коефіцієнти парної кореляції між виходом y і кожним з параметрів входу $x_1$, ..., $x_j$, ..., $x_n$, що дозволяє сформувати вхідну матрицю відповідно до степеня впливу кожного параметра і застосувати принцип ранжирування входів, що узгоджується з будівлею біологічного нейрона. У нейромережу уводиться єдиний параметр для усіх входів мережі - коефіцієнт зважування $K_f$, що приймає значення в діапазоні від 0 до 1.

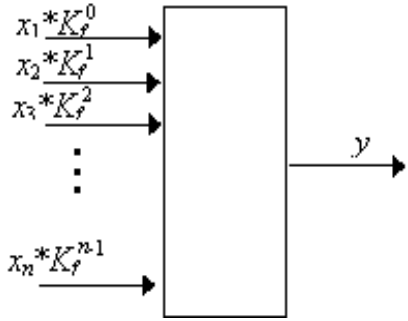



Рис. 2.46. Вплив коефіцієнта зважування входів

Для вході 1 усі значення параметра $x_1$ не змінюються, для входу 2 зменшуються в $K_f^1$ раз, а для останнього n-го входу вага параметра $x_n$ зменшується в $K_f^{n-1}$ раз (рис. 2.46). При $K_f$=1 усі входи рівнозначні, при $K_f$=0 враховується лише перший вхід, інші входи ігнорується, при $0<K_f<1$ зменшується вплив несуттєвих параметрів на вихідну величину y. Такий підхід вимагає проведення попереднього аналізу інформації, але значне поліпшення точності прогнозу підтверджує його ефективність.

## 2.11.2. Вибір даних для обробки

Будь-яка нейрона мережа приймає на вході числові значення і видає на виході також числові значення. Передатна функція для кожного елемента мережі зазвичай вибирається

таким чином, щоб її вхідний аргумент міг приймати довільні значення, а вихідні значення лежали б у строго обмеженому діапазоні. При будь-яких вхідних значеннях виникає ефект насичення, коли елемент чуттєвий лише до вхідних значень, які лежать в обмеженій області (наприклад, S - функції). У цьому випадку вихідне значення завжди буде лежати в інтервалі (0,1), а область чутливості для входів ледь ширше інтервалу (-1,+1). Дана функція є гладкою, а її похідна легко обчислюється - ця обставина важлива для роботи алгоритму навчання мережі (у цьому також криється причина того, що гранична функція для цієї мети практично не використовується).

При використанні нейроних мереж можуть виникати деякі проблеми, зокрема:

- дані мають нестандартний масштаб,
- дані є нечисловими.
- у даних є пропущене або недостовірне значення.

Числові дані масштабуються в прийнятий для мережі діапазон. Зазвичай дані масштабуються по лінійній шкалі. У пакетах програмних нейромереж реалізовані алгоритми, що автоматично знаходять масштабуючі параметри для перетворення числових значень у потрібний діапазон.

Більш складною задачею є робота з даними нечислового характеру. Нехай потрібно навчити нейромережу оцінювати вартість об'єктів нерухомості. Ціна будинку залежить від того, у якому районі міста він розташований. Місто може бути розділене на кілька десятків районів, що мають власні назви, і природно ввести для позначення району змінну з номінальними значеннями. На жаль, у цьому випадку навчити нейрону мережу буде важко, і замість цього краще привласнити кожному районові визначений ранг (базуючись на експертних оцінках).

Найчастіше нечислові дані представляють у виді номінальних змінних. Номінальні змінні можуть бути двозначними (наприклад, Стать={Чоловік, Жінка}) або

багатозначними (тобто приймати більше двох значень). Двозначну номінальну змінну легко перетворити в числову (наприклад, Чоловік=0, Жінка=1). З багатозначними номінальними змінними складніше. Їх теж можна представити одним числовим значенням (наприклад, Собака=0, Миша=1, Кішка = 2), однак при цьому можливе помилкове присвоєння значень номінальної змінної: у розглянутому прикладі Миша виявиться чимось середнім між Собакою і Кішкою. Існує більш точний спосіб, відомий як кодування 1-із-N, у якому одна номінальна змінна представляється декількома числовими перемінними. Кількість числових змінних дорівнює числу можливих значень номінальної змінної; при цьому всякий раз тільки одна з N змінних приймає ненульове значення (наприклад, Собака={1,0,0}, Миша={0,1,0}, Кішка = {0,0,1}). На жаль, номінальна змінна з великим числом можливих станів вимагає при кодуванні методом 1-із-N дуже великої кількості числових змінних, а це приводить до росту розмірів мережі і створює труднощі при її навчанні. У таких ситуаціях краще спробувати знайти інший спосіб представлення даних.

Нечислові дані інших типів можна або перетворити в числову форму, або оголосити незначними. Значення дат і часу, якщо вони потрібні, можна перетворити в числові, віднімаючи з них початкову дату (час). Значення грошових сум перетворити зовсім нескладно. З довільними текстовими полями (наприклад, прізвищами людей) працювати не можна і їхній потрібно зробити незначними. У багатьох реальних задачах приходиться мати справу з не зовсім достовірними даними. Значення деяких змінних можуть бути зіпсовані шумом або бути частково відсутніми. Існують спеціальні засоби роботи з пропущеними значеннями (вони можуть бути замінені на середнє значення цієї змінної або на інші її статистики), так що якщо даних небагато, можна включити в розгляд випадки з пропущеними значеннями. Нейроні мережі в основному стійкі до шумів. Однак у цієї стійкості є границя. Наприклад, викиди, тобто значення, що лежать дуже далеко від області нормальних значень змінної, можуть спотворити результат навчання. У таких випадках найкраще постаратися знайти ці викиди (вилучити відповідні приклади або перетворити викиди

в пропущені значення). Якщо викиди знайти важко, то можна скористатися можливостями зробити процес навчання стійким до викидів, однак таке навчання, як правило, менш ефективне.

**Згладжування даних**

Позитивний ефект досягається при використанні додаткової нейромережі, що функціонує в режимі згладжування вхідних даних навчальної множини. У режимі навчання додаткової мережі кожна реалізація навчальної множини здобуває вид: вектор вхідних значень®вектор вхідних значень (рис. 2.42).

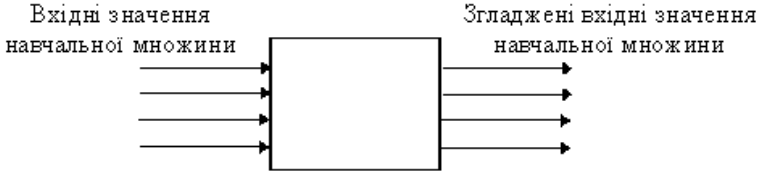



Рис. 2.42. Приклад застосування нейромережі для згладжування даних

У режимі функціонування на входи подаються вхідні значення навчальної множини, на виході одержуємо згладжені значення, без наявних викидів, що надалі можна використовувати для обробки. Можна дати наступне пояснення ефекту згладжування даних. Залежність вихідних значень нейромережі від вхідних може бути представлена сумарним степеневим поліномом, так як передатні функції нейронів схованого шару - поліноміальні. При незначному числі нейронів схованого шару і невисоких степенів поліномів сумарний поліном буде невисокого степеня, приводить до згладжування. Питання про те, скільки прикладів потрібно мати для навчання мережі, доволі непростий. Відомо ряд правил, що погоджують число необхідних прикладів з розмірами мережі (найпростіше з них говорить, що число прикладів повинне бути в десять разів більше числа зв'язків у мережі). Насправді, це число залежить також від складності відображення, що нейрона мережа прагне відтворити. З ростом кількості параметрів кількість необхідних

прикладів росте нелінійно, так що вже при досить невеликому числі параметрів може знадобитися величезне число прикладів. Для більшості реальних задач буває досить декількох сотень або тисяч прикладів. Для складних задач може знадобитися ще більша кількість, однак рідко зустрічається задача, де вистачило б менш сотні прикладів. Якщо даних менше, то інформації для навчання мережі недостатньо.

### 2.11.3. Задачі прогнозування

Особливе значення мають задачі прогнозування і прогнозування тимчасових рядів, серед яких виділяються задачі з набором визначених специфічних ознак, тому варто провести їхню класифікацію. Задачі дослідження явищ, розвиток яких згодом зв'язано, можна поділити на кілька класів:

*По характеру основних ознак об'єкта*:

- прогнозування явищ, реалізації яких представлені у виді детермінованих тимчасових рядів. Такі задачі, зокрема, можна розв’язувати шляхом застосування методів математичного аналізу;
- прогнозування явищ, реалізації яких представлені у виді індетермінованих тимчасових рядів. Розв’язання цих задач традиційно здійснюється шляхом застосування методів теорії ɑмовірностей і математичної статистики. Зокрема, реалізації таких явищ, можуть мати вигляд:

а) стаціонарного тимчасового ряду, що характеризується однорідністю в часі, без істотних змін характеру коливань і їхньої середньої амплітуди; вибір проміжку для формування навчальної множини довільний; як приклад такого ряду на рис. 2.48 приведений графік сумарного річного стоку Дніпра за період з 1810 до 1964 року;

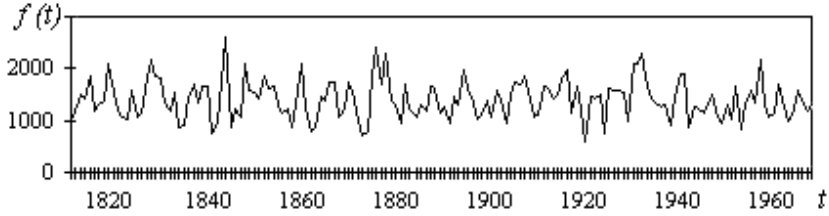



Рис. 2.48. Розподіл річного стоку Дніпра в часі

б) нестаціонарного тимчасового ряду, що характеризується визначеною тенденцією розвитку в часі ; при дослідженні нестаціонарних процесів можна виділити ділянки, на яких процес можна вважати стаціонарним; вибір проміжку для формування навчальної множини в такому випадку обирається згідно задачі прогнозування. На рис. 2.49 приведені щоденні нормовані дані мікросейсмічних коливань Землі за визначений період часу.

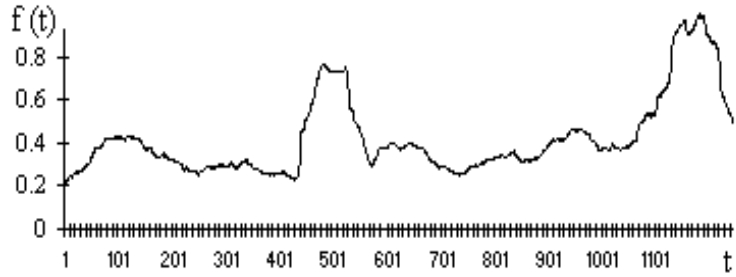



Рис. 2.49. Розподіл мікросейсмічних коливань Землі в часі

*По числу ознак об'єкта досліджень*: одномірна задача; явище представлене лише одною ознакою, зміни якої відбуваються в часі; на рис. 2.50 зображені даніспостережень відносних чисел Вольфа, усереднені за місяць, за період з 1900 по 1924 рік;

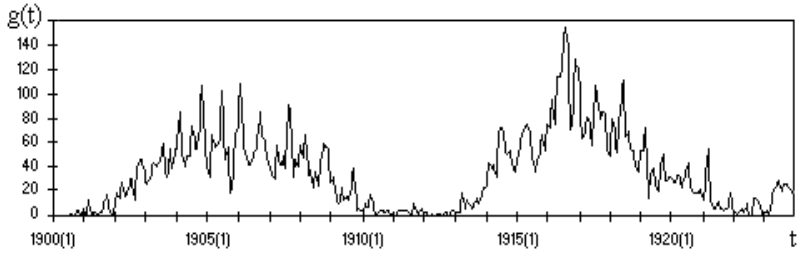



Рис. 2.50. Розподіл чисел Вольфа в часі

• багатомірна задача; об'єкт або явище представлені декількома ознаками (рис. 2.51); задача прогнозування може бути розширена завдяки представленню даних у просторі (рис. 2.52).

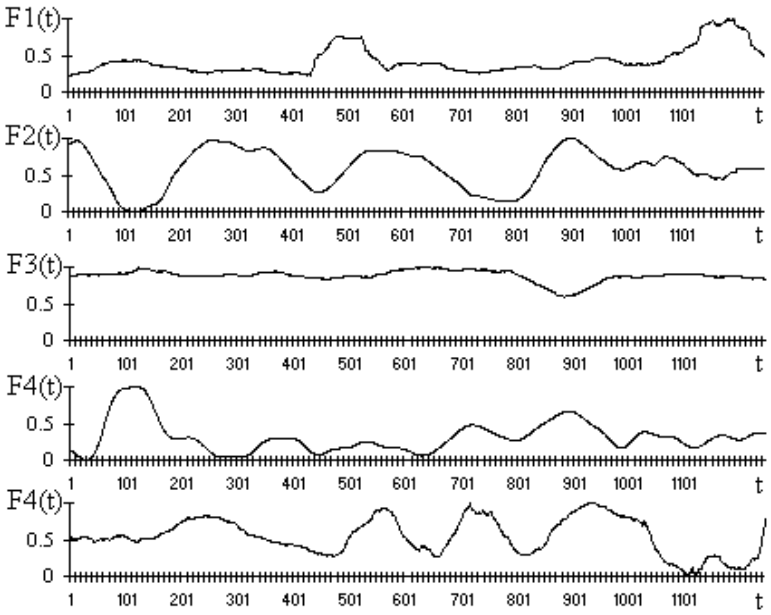



Рис. 2.51. Основні провісники сейсмічної активності

На рис. 2.51 представлені щоденні дані деяких з основних ознак, що визначають землетрус. Приведені дані, зокрема:

- F1 - акоїстика,
- F2 - деформації земної поверхні,
- F3 - мікросейсмічні коливання Землі,
- F4 - сумарна енергія землетрусу,
- F5 - температура Землі,

складають ознаки, обмірювані в однакові відрізки часу за визначений період і надалі використовувалися для експериментів.

Як приклад багатомірної задачі покажемо оцінювання числа груп сонячних плям, усереднених за рік, що визначаються двовимірним розподілом в області двох аргументів - широта-час (рис. 2.52).

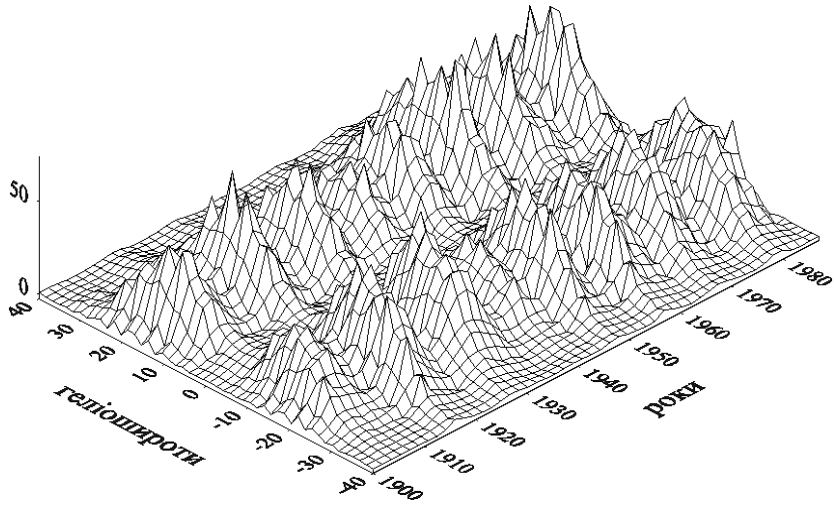



Рис. 2.52. Просторове відображення кількості сонячних плям у координатах геліоширота-час

З огляду на специфічний характер прогнозування тимчасових рядів і визначений різнобій у термінології, будемо дотримуватись ряду означень. *Передісторією ряду* назвемо набір елементів тимчасового ряду, що враховується для одного кроку прогнозування наступних елементів тимчасового ряду. Однокрокове прогнозування зводиться до задач відображення у випадку, коли значення елементів передісторії можуть визначати лише один дискретний відлік вихідних величин. Багатокрокове прогнозування характеризується збільшенням дискретних відрахувань вихідної величини і, відповідно, збільшенням часу, на який здійснюється прогноз (час випередження *Топ*). При багатокроковому прогнозуванні *Топ=a*R*, де *R* - кількість кроків обчислення прогнозування; а - крок дискретизації вихідного параметра (наприклад, рік, місяць, день, і т.п.).

*За часом випередження розрізняють види прогнозів*:

- згладжування, R= 0;
- короткостроковий прогноз, R= 1 - 2;
- середньосроковий прогноз, R= 3 - 7;
- довгостроковий прогноз, R= 10 - 15.

Очевидно, що вид прогнозу істотно впливає на вибір засобів і методику його реалізації.

## 2.11.4. Адаптація нейромереж у режимах прогнозування до даних навчальних множин

Дані про поводження об'єкта, ознаки якого зв'язані з часом, представлені як результати спостережень у рівномірні моменти часу. Для моментів часу t=1, 2, ..., n дані спостережень здобувають вид тимчасового ряду $x(t_1)$, $x(t_2)$, ..., $x(t_n)$. Інформація про значення тимчасового ряду до моменту n дозволяє давати оцінки параметрам $x(t_{n+1})$, $x(t_{n+2})$, ..., $x(t_{n+m})$. Для прогнозування елементів тимчасових рядів широко використовують так називаний метод "тимчасових вікон".

У залежності від кількості ознак, що представляють значення рядів при формуванні множин даних, виділимо задачі двох типів.

**Однопараметрична задача прогнозування**

Нехай часовий ряд x(t) заданий моментами процесу $x(t_1)$, $x(t_2)$,..., $x(t_i)$ у дискретні моменти часу t. Задамо ширину (кількість дискретних моментів) вхідного тимчасового вікна m, ширину вихідного вікна p. Вхідне і вихідне вікна накладаються на дані ряду, починаючи з першого елемента (рис. 2.53).


Рис. 2.53. Формування множин даних для однопараметричної задачі прогнозування по методу "тимчасових вікон"

Вхідне вікно формує дані для входів нейроної мережі, а вихідне, відповідно, для виходів. Подібна пара вхідного і вихідного векторів приймається за одну реалізацію тимчасового ряду. При зрушенні тимчасових вікон по тимчасовому ряду з кроком s, одержуємо другу і наступні реалізації. Значення ширини вікон і кроку зсуву повинні узгоджуватися з особливостями тимчасового ряду, що забезпечується шляхом проведення експериментів. Нехай вхідне вікно має ширину m, вихідне вікно p=1, крок зсуву s=1. Тоді сформована множина значень для однопараметричної задачі буде мати вигляд, приведений нижче:

Таблиця 2.2. Множина даних для однопараметричної задачі

| Входи | | | | Вихо ди |
|---|---|---|---|---|
| $x(t_1)$ | $x(t_2)$ | ... | $x(t_m)$ | $x(t_{m+1})$ |
| $x(t_2)$ | $x(t_3)$ | ... | $x(t_{m+1})$ | $x(t_{m+2})$ |
| $x(t_3)$ | $x(t_4)$ | ... | $x(t_{m+2})$ | $x(t_{m+3})$ |
| ... | ... | ... | ... | ... |
| $x(t_i)$ | $x(t_{i+1})$ | ... | $x(t_{i+m-1})$ | $x(t_{i+m})$ |

**Багатопараметрична задача прогнозування**

У багатомірних (багатопараметричних) задачах прогнозування підходи до рішення проблеми подібні (рис. 2.54).

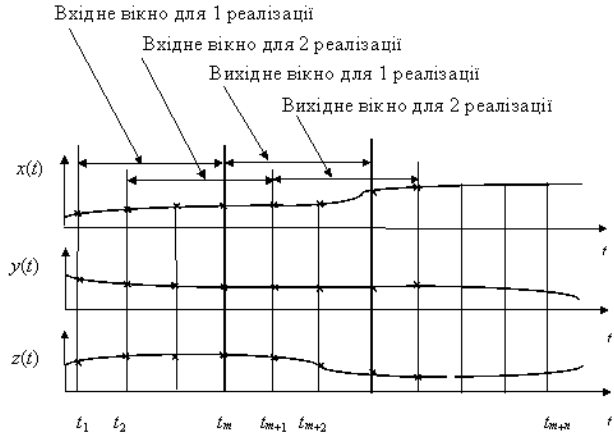



Рис. 2.54. Формування множин даних для багатопараметричної задачі

Нехай необхідно спрогнозувати взаємозалежні величини x(t), y(t), ..., z(t). Якщо прийняти ширину вхідного вікна m, вихідного p=1, кроку зсуву s=1, можна сформувати множину даних у такий спосіб:

Таблиця 2.3

Множина даних для багатопараметричної задачі

| Входи | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x(t_1)$ | $x(t_2)$ | $x(t_m)$ | $y(t_1)$ | $y(t_2)$ | $y(t_m)$ | $z(t_1)$ | $z(t_2)$ | $z(t_m)$ |
| $x(t_2)$ | $x(t_3)$ | $x(t_{m+1})$ | $y(t_2)$ | $y(t_3)$ | $y(t_{m+1})$ | $z(t_2)$ | $z(t_3)$ | $z(t_{m+1})$ |
| $x(t_3)$ | $x(t_4)$ | $x(t_{m+2})$ | $y(t_3)$ | $y(t_4)$ | $y(t_{m+2})$ | $z(t_3)$ | $z(t_4)$ | $z(t_{m+2})$ |
| $x(t_i)$ | $x(t_{i+1})$ | $x(t_{i+m-1})$ | $y(t_i)$ | $y(t_{i+1})$ | $y(t_{i+m-1})$ | $z(t_i)$ | $z(t_{i+1})$ | $z(t_{i+m-1})$ |

| Виходи | | |
|---|---|---|
| $x(t_{m+1})$ | $y(t_{m+1})$ | $z(t_{m+1})$ |
| $x(t_{m+2})$ | $y(t_{m+2})$ | $z(t_{m+2})$ |
| $x(t_{m+3})$ | $y(t_{m+3})$ | $z(t_{m+3})$ |
| $x(t_{i+m-1})$ | $y(t_{i+m-1})$ | $z(t_{i+m-1})$ |

Функціонування нейромережі здійснюється відповідно до показаного методу тимчасових вікон, зберігаючи значення ширини вікон і кроку зсуву. Конкретизація підходів до реалізації прогнозування значною мірою залежить від особливостей досліджуваного явища.

**Однокрокове прогнозування (передбачення)**

Задача однокрокового прогнозування зводиться до задачі відображення, коли один вхідний вектор відображається у вихідний (рис. 2.55).

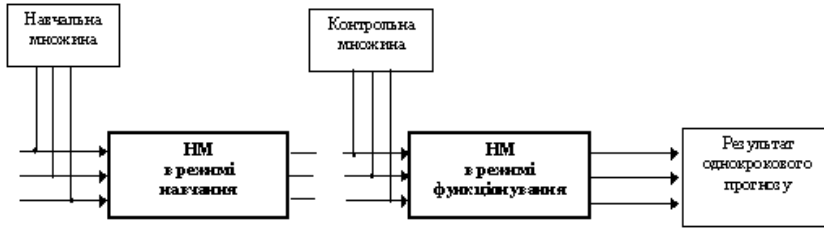


Рис. 2.55. Послідовність використання нейромережі для задач передбачення

У випадку однопараметричної задачі передбачення навчальна множина до моменту n, за умови m=3, p=1, s=1, буде мати вигляд приведеный у табл. 2.4.

Таблиця 2.4

| Входи | | | Вихід |
|---|---|---|---|
| $x(t_1)$ | $x(t_2)$ | $x(t_3)$ | $x(t_4)$ |
| $x(t_2)$ | $x(t_3)$ | $x(t_4)$ | $x(t_5)$ |
| ... | ... | ... | ... |
| $x(t_{n-3})$ | $x(t_{n-2})$ | $x(t_{n-1})$ | $x(t_n)$ |

У режимі навчання встановлюються коефіцієнти ваг зв'язків, після чого стає можливим перехід до режиму функціонування. Для передбачення на входи нейромережі надходять значення останньої реалізації навчальної множини $x(t_{n-2})$, $x(t_{n-1})$, $x(t_n)$. На виході формується прогнозована величина $x^*(t_{n+1})$.

Для багатопараметричної задачі передбачення на входи навченої нейромережі подаються вектори $x(t_{n-2})$, $y(t_{n-2})$, $z(t_{n-2})$, $x(t_{n-1})$, $y(t_{n-1})$, $z(t_{n-1})$, $x(t_n)$, $y(t_n)$, $z(t_n)$. На виходи нейромережі надходять передбачені величини $x^*(t_{n+1})$, $y^*(t_{n+1})$, $z^*(t_{n+1})$, що відкладаються у вихідний вектор передбачених даних.

Показаний режим є однокроковим, що працює в режимі відображення (реальний вхід - прогнозований вихід). Передбачення застосовують для моделювання дискретних послідовностей, що не зв'язані часом. З огляду на специфіку тимчасових рядів, такий тип прогнозу не завжди доцільний, але у визначених випадках короткострокових прогнозів їм можна скористатися.

**Багатокрокове прогнозування**

Багатокрокове прогнозування застосовують лише для явищ, ознаки яких представлені у виді тимчасових рядів. Для однопараметричної задачі прогнозування навчальна множина буде мати вигляд приведений у табл. 2.3. Під час навчання мережа налаштовує коефіцієнти ваг зв'язків і поліномів передатних функцій, що надалі і визначає режим функціонування. Багатокрокове прогнозування тимчасового ряду здійснюється в такий спосіб (рис. 2.56). На входи нейромережі подається вектор відомих значень $x(t_{n-2})$, $x(t_{n-1})$, $x(t_n)$. На виході формується прогнозована величина $x^*(t_{n+1})$, що визначає вектор прогнозованих виходів і одночасно додається до значень навчальної множини, тобто, приймається як достовірна. Далі на входи подається вектор $x(t_{n-1})$, $x(t_n)$, $x^*(t_{n+1})$, а на виході виходить $x^*(t_{n+2})$ і наступні прогнозовані значення.

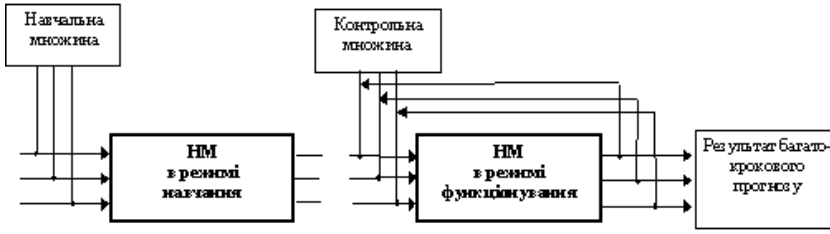



Рис. 2.56. Послідовність використання нейромереж для задач багатокрокового прогнозування

Для багатомірної задачі прогнозування на входи навченої нейромережі подаються вектори $x(t_{n-2})$, $y(t_{n-2})$, $z(t_{n-2})$, $x(t_{n-1})$, $y(t_{n-1})$,

$z(t_{n-1})$, $x(t_n)$, $y(t_n)$, $z(t_n)$. На виході продукуються величини $x^*(t_{n+1})$, $y^*(t_{n+1})$, $z^*(t_{n+1})$, що формують вектор вихідних значень і послідовно додаються до значень навчальної множини. При зрушенні вікна на крок, вихідні дані, які спродуковані мережею, сприймаються як реальні і беруть участь у прогнозуванні наступного значення виходу, тобто на входи подаємо вектор $x(t_{n-1})$, $y(t_{n-1})$, $z(t_{n-1})$, $x(t_n)$, $y(t_n)$, $z(t_n)$, $x^*(t_{n+1})$, $y^*(t_{n+1})$, $z^*(t_{n+1})$, а на виході одержуємо $x^*(t_{n+2})$, $y^*(t_{n+2})$, $z^*(t_{n+2})$ і наступні прогнозовані значення.

Багатокрокове прогнозування дозволяє робити коротко- і середньострокові прогнози, оскільки істотний вплив на точність має нагромадження похибки на кожнім кроці прогнозування. При застосуванні довгострокового багатокрокового прогнозування спостерігається характерне для багатьох прогнозуючих систем поступове загасання процесу, фазові зрушення й інші перекручування картини прогнозу. Такий тип прогнозування підходить для стаціонарних тимчасових рядів з невеликою випадковою складовою.

**Багатокрокове прогнозування з перенавчанням нейромережі на кожнім кроці прогнозу**

Швидкі неітераційні алгоритми навчання дозволяють запропонувати новий тип багатокрокового прогнозу, що може бути застосований при довгострокових прогнозах зі збереженням задовільної точності прогнозування.

Аналогічно з попереднім алгоритмом прогнозування на входи мережі в режимі функціонування надходить остання реалізація навчальної множини $x(t_{n-2})$,$x(t_{n-1})$, $x(t_n)$. Прогнозоване значення виходу $x^*(t_{n+1})$ відкладається у векторі прогнозованих вихідних значень і в якості достовірного додається до реальних значень навчальної множини. Навчальна множина збільшується на одне тимчасове вікно. Відбувається процес перенавчання мережі на збільшеній навчальній множині, під час якого визначаються нові вагові коефіцієнти k синаптичних зв'язків і поліномів передатних функцій нейронів (рис. 2.57).

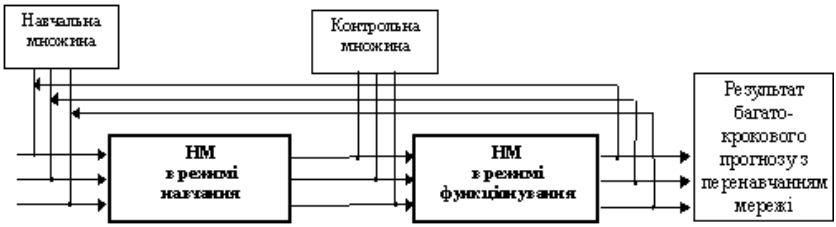



Рис. 2.57. Послідовність використання нейромережі для задач багатокрокового прогнозування з перенавчанням

Реалізація $x(t_{n-1})$, $x(t_n)$, $x^*(t_{n+1})$, як значення наступного вхідного вікна подається на входи мережі в режимі функціонування. Мережа продукує нове вихідне значення $x^*(t_{n+2})$, що також відкладається у вектор спродукованних виходів і додається до реальних значень навчальної множини, з метою подальшого перенавчання мережі і установлення відновлених коефіцієнтів поліномів передатних функцій і синаптичних зв'язків. Ітераційна процедура перенавчання поширюється до прогнозованого значення $x^*(t)$. Такий підхід дозволяє при великих інтервалах випередження усунути загасання прогностичних властивостей мережі за рахунок постійного коректування вагових коефіцієнтів синаптических зв'язків. Відзначимо, що алгоритм багатокрокового прогнозування з перенавчанням мережі для традиційних мереж прямого поширення з ітераційним навчанням є практично нездійсненним через великі тимчасові затримки, необхідні на перелагоджування коефіцієнтів мережі.

### 2.11.5 Критерії оцінки якості функціонування мережі

Критерії оцінки можна розділити на внутрішні і зовнішні. Внутрішні критерії формуються на основі інформації множини даних, використаної для навчання, тоді як зовнішні критерії використовують нову інформацію тестової множини, елементи яких не використовувалися при навчанні. Оптимальна

складність моделі мережі встановлюється на основі зовнішніх і внутрішніх критеріїв.

До зовнішніх критеріїв вибору моделі можна віднести:

*Критерій регулярності* - зовнішній критерій, для обчислення величини якого потрібна тестова множина даних.

$$\Delta(1) = \frac{\sum_{i=1}^{Nnp}\left(\Phi_i^* - \Phi_i\right)^2}{\sum_{i=1}^{Nnp}\left(\Phi^*\right)^2} \to \min$$

де $\Delta(1)$ - величина критерію; $i$ - номер виходу; $\Phi_i$ - значення виходів мережі; $\Phi_i^*$ - точні значення виходів.

Фізичний зміст застосування критерію регулярності складається у виборі моделі, що буде максимально точною на елементах тестової множини, що не входить до складу навчальної множини. Критерій мінімального зсуву вимагає максимального збігу значень вихідної величини для двох моделей, де як навчальні приклади були використані дані різних підмножин навчальної множини. Критерій мінімального зсуву дозволяє вибрати модель, що "слабко реагує" на зміну навчальної множини і зашумленні тестові дані. Критерій зміщенності показників у часі - допомагає оцінити рівень взаємозв'язку змінних. Окремі показники можуть мати різні післядії, тому роздільне прогнозування кожного з них може забезпечити кращий результат. Відповідно, при наявності тісного взаємозв'язку між показниками, прогнозування їхньої сукупності поліпшує результати прогнозу кожного з них. Можливо також включення нового параметра, що може бути або додатковою ознакою або лінійною комбінацією вже наявних ознак. Застосування цього критерію допомагає в оптимальному підборі ознак явища, що забезпечують високу точність прогнозування.

*Критерій фізичної вірогідності* - вимагає виключення моделей, які під час проведення експерименту, можуть продукувати нереальні результати (великі викиди для прогнозованої множини).

**Оцінювання точності прогнозів**

Як правило, після навчання нейромережі здійснюють контрольне відтворення даних, що складають навчальну множину. Якщо точність відтворення задовільна і відхилення знаходяться в припустимих границях, вважають, що побудовано задовільну модель і варто очікувати достатню якість відображення. Якщо при відтворенні мережею даних навчальної множини спостерігаються великі розбіжності, можна припустити що це викликано:

- наявністю неточних даних з великою випадковою складовою. Для усунення цього підвищують вимоги до точності вимірів; у випадку тимчасового ряду, можливе зменшення кроку дискретизації, наприклад використання щомісячних значень замість річних;
- неврахованими ознаками, що значною мірою визначають закономірність; ця проблема може бути вирішена розширенням набору ознак, прийнятих в увагу.

Після одержання передбачених значень при наявності правильних можна одержати абсолютні і відносні відхилення на всій контрольній множині, для кожного кроку прогнозування. При наявності задовільних результатів прогнозування на контрольній множині, можна вважати, що налаштована мережа для даної задачі має оптимальну складність і готова до відтворення даних, для яких немає відомих виходів.

## 2.12. Сучасні напрямки розвитку нейрокомп’ютерних технологій

Детальний аналіз розробок нейрокомп’ютеріов дозволяє виділити основні перспективні напрямки сучасного розвитку

нейрокомп’ютерних технологій: нейропакети, нейромережні експертні системи, СУБД із включенням нейромережних алгоритмів, обробка зображень, керування динамічними системами й обробка сигналів, керування фінансовою діяльністю, оптичні нейрокомп’ютери, віртуальна реальність. Розробками в цій області займається більш 300 закордонних компаній, причому число їх постійно збільшується. Серед них такі гіганти, як Intel, IBM і Motorolla. Сьогодні спостерігається тенденція переходу від програмних реалізацій до програмно-апаратної реалізації нейромережних алгоритмів з різким збільшенням числа розробок нейрочипів з нейромережною архітектурою. Різко виросла кількість військових розробок, в основному спрямованих на створення надшвидкісних, "розумних" суперобчислювачів.

Якщо говорити про основний напрямок - інтелектуалізацію обчислювальних систем, придання їм властивостей людського мислення і сприйняття, то тут нейрокомп’ютери - практично єдиний шлях розвитку обчислювальної техніки. Більшість невдач на шляху удосконалення штучного інтелекту протягом останніх 30 років зв'язане з тим, що для рішення важливій і складних по постановці задач вибиралися обчислювальні засоби, не адекватні по можливостях розв'язуваній задачі, в основному з числа традиційних комп'ютерів. При цьому, як правило, не вирішувалася задача, а показувалася принципова можливість її рішення. Сьогодні активний розвиток комп'ютерних технологій створюють об'єктивні умови для побудови обчислювальних систем, адекватних по можливостях і архітектурі практично будь-яким задачам штучного інтелекту.

У Японії з 1993 року прийнята програма "Real world computing program". Її основна мета - створення адаптивної ЕОМ, яка еволюціонує. Проект розрахований на 30 років. Основою розробки є нейротехнологія, що використовується для розпізнавання образів, обробки семантичної інформації, керування інформаційними потоками і роботами, здатних адаптуватися до навколишнього середовища. Тільки в 1996 році було проведено біля сотні міжнародних конференцій по нейрокомп’ютерам і суміжним проблемах. Розробки

нейрокомп’ютерів ведуться в багатьох країнах світу, зокрема, в Австралшї створено зразок комерційного супернейрокомп’ютера.

Для якого класу задач ефективне застосування обчислювального пристрою, побудованого за новою технологією? Відносно нейрокомп’ютерів відповідь на це питання постійно міняється протягом 50 років.

В історії обчислювальної техніки завжди були задачі, не розв'язувані традиційними комп'ютерами з архітектурою фон Неймана і для них перехід до нейромережних технологій закономірний у випадку збільшення розмірності простору або скорочення часу обробки. Можна виділити три ділянки застосування нейромережних технологій: загальний, прикладн і спеціальний.

**Загальні задачі**

Ці задачі зводяться до обробки нейроною мережею багатомірних масивів змінних, наприклад:

- контроль кредитних карток. Сьогодні 80% кредитних карток у США обробляються за допомогою нейромережних технологій;
- система виявлення схованих речовин за допомогою системи на базі теплових нейронів і за допомогою нейрокомп’ютера на замовлених цифрових нейрочіпах. Подібна система фірми SAIC експлуатується в багатьох аеропортах США при огляді багажу для виявлення наркотиків, вибухових речовин, ядерних і інших матеріалів;
- система автоматизованого контролю безпечного збереження ядерних виробів.

**Прикладні задачі**

*Обробка зображень*

Перспективними задачами обробки зображень нейрокомп'ютерів є обробка аерокосмічних зображень (стиск із відновленням, сегментація, обробка зображень), пошук, виділення і розпізнавання на зображенні рухливих об'єктів заданої форми, обробка потоків зображень, обробка інформації у високопродуктивних сканерах.

*Обробка сигналів*

У першу чергу це клас задач, зв'язаний із прогнозуванням тимчасових залежностей:

- прогнозування фінансових показників;
- прогнозування надійності елементів і систем;
- передбачення потужності АЕС і прогнозування надійності систем електроживлення на літаках;

При розв'язанні цих задач спостерігається перехід від найпростіших регресійних і інших статистичних моделей прогнозу до нелінійних адаптивних екстраполюючих фільтрів, реалізованих у виді складних нейроних мереж.

*Системи керування динамічними об'єктами*

Це одна із самих перспективних областей застосування нейрокомп'ютерів. У США і Фінляндії ведуться роботи з використання нейрокомп'ютерів для керування хімічними реакторами.

Перспективною вважається розробка нейрокомп'ютера для керування рухливою установкою гіперзвукового літака. Актуальною для розв'язання за допомогою нейрокомп'ютера є задача навчання нейроної мережі виготовленню точного маневру винищувача, задача керування роботами: пряма, зворотна кінематична і динамічна задачі, планування маршруту руху робота. Перехід до нейрокомп'ютерів зв'язаний у першу чергу з обмеженістю обсягів розміщення обчислювальних

систем, а також з необхідністю реалізації ефективного керування в реальному масштабі часу.

**Нейромережні експертні системи**

Необхідність реалізації експертних систем з алгоритмом нейромереж виникає при значному збільшенні числа правил і висновків. Прикладами реалізації конкретних нейромережних експертних систем можуть служити система вибору повітряних маневрів у ході повітряного бою і медична діагностична експертна система для оцінки стану льотчика.

**Нейрочіпи і нейрокомп'ютери**

Головним результатом розробки нейромережних алгоритмів розв'язання задачі є можливість створення архітектури нейрочіпа, адекватного розв'язуваній задачі. Для реалізації нейромережних алгоритмів з використанням універсальних мікропроцесорних засобів эфективній створити архітектуру, орієнтовану на виконання нейромережни операцій, чим використовувати стандартні алгоритми, орієнтовані на модифікацію рішення задачі.

На відміну від інших напрямків розвитку зверхпродуктивної обчислювальної техніки, нейрокомп'ютери дають можливість вести розробки з використанням наявного потенціалу електронної промисловості. Необхідно відзначити ряд важливих особливостей даних робіт:

- цей напрямок дозволяє створювати унікальні суперкомп'ютери на наявній елементній базі;
- розробки нейрочіпів і нейрокомп'ютерів характеризуються переходом від цифрової обробки до аналого-цифрової й аналогової;
- нейромережні архітектури в порівнянні з іншими приводять до активізації використання нових технологічних напрямків реалізації: нейросистеми на пластмасі, оптоелектронні й оптичні нейрокомп'ютери, молекулярні нейрокомп'ютери і

нанонейроелементи; виникає потреба в універсалізації САПР нейрочіпів.

- створення технології систем на пластмасі і нанотехнології може привести до появи нових зверхпаралельних архитектур. Починаючи з нанонейроелементів, ми впритул підходимо до принципово нових архітектурних елементів, що утворять зверхпаралельні високопродуктивні обчислювальні системи.

**Деякі висновки**

Нейрокомп'ютери є перспективним напрямком розвитку сучасної високопродуктивної обчислювальної техніки, а теорія нейроних мереж і нейроматематика являють собою пріоритетні напрямки обчислювальної науки, і при відповідній підтримці інтенсивно розвиваються. Основою активного розвитку нейрокомп'ютерів є принципова відмінність нейромережних алгоритмів рішення задач від однопроцесорних і малопроцесорних. Нейрокомп'ютери є предметом досліджень відразу декількох дисциплін, тому єдине визначення нейрокомп'ютера можна дати тільки з урахуванням різних точок зору, адекватних різним напрямкам науки.

*Математична статистика*

Нейрокомп'ютери - це системи, що дозволяють сформувати опис характеристик випадкових процесів і сукупності випадкових процесів, що мають складні, багатомодальні або невідомі функції розподілу.

*Математична логіка і теорія автоматів*

Нейрокомп'ютери - це системи, у яких алгоритм розв'язання задачі представлений логічною мережею елементів окремого виду - нейронів, з повним відмовленням від булевских елементів типу І, АБО, НІ, унаслідок цього введені специфічні зв'язки між елементами, що є предметом окремого розгляду.

*Теорія керування*

Як об'єкт керування вибирається добре формалізований об'єкт - багатошарова нейрона мережа, а динамічний процес її настроювання представляє процес рішення задачі. При цьому практично весь апарат синтезу адаптивних систем керування переноситься на нейрону мережу, як окремий вид об'єкта керування.

*Обчислювальна математика*

Нейрокомп'ютери реалізують алгоритми розв'язання задач, представлені у виді нейроних мереж. Це обмеження дозволяє розробляти алгоритми, потенційно більш паралельні, чим інша їхня фізична реалізація. Множина нейромережних алгоритмів розв'язання задач складає перспективний розділ обчислювальної математики, умовно названий нейроматематикою.

*Обчислювальна техніка*

Нейрокомп'ютер - це обчислювальна система, у якій реалізовані два принципових технічних рішення:

- спрощений до рівня нейрона процесорний елемент однорідної структури і складні зв'язки між елементами;
- програмування обчислювальної структури перенесено на зміну вагових коефіцієнтів зв'язків між процесорними елементами.

Загальне визначення нейрокомп'ютера може бути представлене в наступному виді.

Нейрокомп'ютер - це обчислювальна система з архітектурою апаратного і програмного забезпечення, адекватного виконанню алгоритмів, представлених у нейромережному логічному базисі.

- Нейрокомп’ютери дають стандартний спосіб розв’язання багатьох нестандартних задач. І неважливо, що спеціалізована машина краще розв’язує один клас задач. Важливіше, що один нейрокомп’ютер розв’яже і цю задачу, і другу, і третю і не треба щораз проектувати спеціалізовану ЕОМ, нейрокомп’ютер зробить усе сам і майже не гірше.
- Замість програмування навчання. Нейрокомп’ютер учиться, потрібно лише формувати навчальні множини. Робота програміста заміняється новою роботою вчителя. Краще це чи гірше? Ні те, ні інше. Програміст указує машині всі деталі роботи, учитель створює навчальне середовище, до якого пристосовується нейрокомп’ютер. З'являються нові можливості для роботи.
- Нейрокомп’ютери ефективні там, де потрібний аналог людської інтуїції, зокрема, для розпізнавання образів, читання рукописних текстів, підготовки аналітичних прогнозів, перекладу з однієї мови на інший і т.п. Саме для таких задач зазвичай важко скласти явний алгоритм.

# 3. Генетичні алгоритми

## 3.1. Природний відбір у природі

Еволюційна теорія стверджує, що кожен біологічний вид цілеспрямовано розвивається і змінюється для того, щоб щонайкраще пристосуватися до навколишнього середовища. У процесі еволюції багато видів комах і риб придбали захисне

фарбування, їжак став невразливим завдяки голкам, людина стала власником дуже складної нервової системи. Можна сказати, що еволюція - це процес оптимізації всіх живих організмів. Розглянемо, якими ж засобами природа вирішує цю задачу оптимізації.

Основний механізм еволюції - це *природний відбір*. Його суть полягає в тому, що більш пристосовані особи мають більше можливостей для виживання і розмноження і, отже, приносять більше нащадків, ніж погано пристосовані особи. При цьому завдяки передачі генетичної інформації *(генетичному спадкуванню)* нащадки успадковують від батьків основні їхні якості. Таким чином, нащадки сильних індивідуумів також будуть відносно добре пристосованими, а їхня частка в загальній масі особей буде зростати. Після зміни декількох десятків або сотень поколінь середня пристосованість особей даного виду помітно зростає.

Щоб зробити зрозумілими принципи роботи генетичних алгоритмів, пояснимо також, як улаштовані механізми генетичного спадкування в природі. У кожній клітці будь-якої тварини утримується вся генетична інформація цієї особи. Ця інформація записана у виді набору дуже довгих молекул ДНК (Дезоксирибо Нуклеїнова Кислота). Кожна молекула ДНК - це ланцюжок, що складається з молекул *нуклеотидів* чотирьох типів, що позначаються A, T, C і G. Власне, інформацію несе порядок проходження нуклеотидів у ДНК. Таким чином, генетичний код індивідума - це просто дуже довгий рядок символів, де використовуються всього 4 букви. У тваринній клітці кожна молекула ДНК оточена оболонкою - таке утворення називається *хромосомою.*

Кожна уроджена якість особи (колір ока, спадкоємні хвороби, тип волосся і т.д.) кодується визначеною частиною хромосоми, що називається *геном* цієї властивості. Наприклад, ген кольору ока містить інформацію, що кодує визначений колір очей. Різні значення гена називаються його *аллелями.*

При розмноженні тварин відбувається злиття двох батьківських полових кліток і їхні ДНК взаємодіють, утворюючи ДНК нащадка. Основний спосіб взаємодії - *кроссовер (cross-over,*

*схрещування).* При кроссовері ДНК предків розділяються на дві частини, а потім обмінюються своїми половинками.

При спадкуванні можливі мутації через радіоактивність або інші впливи, у результаті яких можуть змінитися деякі гени в полових клітках одного з батьків. Змінені гени передаються нащадкові і додають йому нові властивості. Якщо ці нові властивості корисні, вони, швидше за все, збережуться в даному виді - при цьому відбудеться стрибкоподібне підвищення пристосованості виду. Ключову роль в еволюційній теорії грає природний відбір. Його суть полягає в тому, що найбільш пристосовані особи краще виживають і приносять більше нащадків, чим менш пристосовані. Помітимо, що сам по собі природний відбір ще не забезпечує розвиток біологічного виду. Тому дуже важливо зрозуміти, яким чином відбувається спадкування, тобто як властивості нащадка залежать від властивостей батьків. Основний закон спадкування інтуїтивно зрозумілий кожному - він полягає в тому, що нащадки схожі на батьків. Зокрема, нащадки більш пристосованих батьків будуть, швидше за все, одними з найбільш пристосованих у своєму поколінні. Щоб зрозуміти, на чому заснована ця подібність, потрібно трохи поглибитися в побудову природної клітки - у світ генів і хромосом. Майже в кожній клітці будь-якої особи є набір хромосом, що несуть інформацію про цю особу. Основна частина хромосоми - нитка ДНК, що визначає, які хімічні реакції будуть відбуватися в даній клітці, як вона буде розвиватися і які функції виконувати.

***Ген** - це відрізок ланцюга ДНК, відповідальний за визначену властивість особи, наприклад за колір очей, тип волосся, колір шкіри і т.д.*

***Уся сукупність генетичних ознак людини кодується за допомогою приблизно 60 тис. генів, довжина яких складає більш 90 млн. нуклеотидів.***

Розрізняють два види кліток: *полові* (такі, як сперматозоїд і яйцеклітина) і *соматичні*. У кожної соматичній клітці людини утримується 46 хромосом. Ці 46 хромосом - насправді 23 пари, причому в кожній парі одна з хромосом отримана від батька, а друга - від матері. Парні хромосоми відповідають за однакові ознаки - наприклад, батьківська хромосома може містити ген чорного кольору ока, а парна їй материнська - ген блакитного

кольору. Існують ***визначені закони, що керують участю тих або інших генів у розвитку особи***. Зокрема, у нашому прикладі нащадок буде чорнооким, оскільки ген блакитних очей є "слабким" і придушується геном будь-якого іншого кольору.

У полових клітках хромосом тільки 23, і вони непарні. При запліднені відбувається злиття чоловічих і жіночої полових кліток і виходить клітка зародка, що містить 46 хромосом. Які властивості нащадок одержить від батька, а які - від матері? Це залежить від того, які саме полові клітки брали участь у запліднені. *Процес вироблення полових кліток (так званий **мейоз**) в організмі піддається ймовірностям, завдяки яким нащадки багато в чому відрізняються від своїх батьків*. При мейозі, відбуваються наступне: парні хромосоми соматичної клітки зближаються впритул, потім їхні нитки ДНК розриваються в декількох випадкових місцях і хромосоми обмінюються своїми частинами (рис. 3.1).

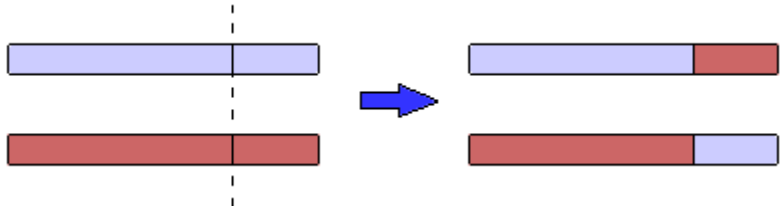


Рис. 3.1. Умовна схема кросінговера

Цей процес забезпечує появу нових варіантів хромосом і називається "кросінговер". Кожна із знову з'явившихся хромосом виявиться потім усередині однієї з полових кліток, і її генетична інформація може реалізуватися в нащадках даної особи.

Другий важливий фактор, що впливає на спадковість, - це мутації, що виражаються в зміні деяких ділянок ДНК. Мутації також випадкові і можуть бути викликані різними зовнішніми факторами, такими, як радіоактивне опромінення. Якщо мутація відбулася в половій клітці, то змінений ген може передатися нащадкові і проявитися у виді спадкоємної хвороби або в інших

нових властивостях нащадка. Вважається, що саме мутації є причиною появи нових біологічних видів, а кросінговер визначає вже мінливість усередині виду (наприклад, генетичні розходження між людьми).

## 3.2. Що таке генетичний алгоритм

Нехай дана деяка складна функція *(цільова функція),* що залежить від декількох змінних, і потрібно знайти такі значення змінних, при яких значення функції максимальне. Задачі такого роду називаються *задачами оптимізації* і вони зустрічаються на практиці дуже часто.

Один з найбільш наочних прикладів - задача розподілу інвестицій. У цій задачі змінними є обсяги інвестицій у кожен проект (10 змінних), а функцією, яку потрібно максимізувати - сумарний дохід інвестора. Також приведені значення мінімального і максимального об'єму вкладення в кожний із проектів, що задають область зміни кожної із змінних.

Спробуємо розв'язати цю задачу, застосовуючи відомі нам природні способи оптимізації. Будемо розглядати кожен варіант інвестування (набір значень змінних) як індивідум, а прибутковість цього варіанта - як пристосованість цього індивідума. Тоді в процесі еволюції (якщо ми зуміємо його організувати) пристосованість індивідумів буде зростати, а виходить, будуть з'являтися усе більш і більш дохідні варіанти інвестування. Зупинивши еволюцію в деякий момент і вибравши найкращого індивідума, ми одержимо досить гарне рішення задачі.

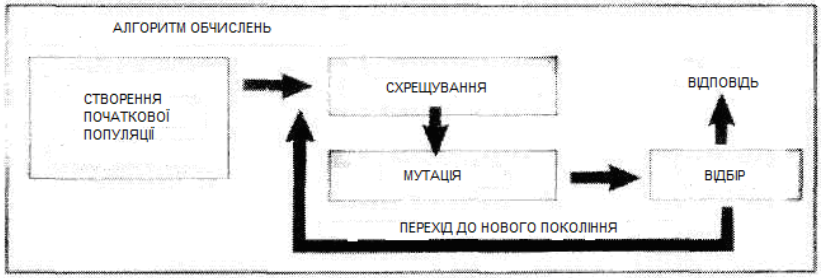
***Генетичний алгоритм*** *- це проста модель еволюції в природі, яка реалізована у виді комп'ютерної програми*.

У ньому використовуються як аналог механізму генетичного спадкування, так і аналог природного відбору. При цьому зберігається біологічна термінологія в спрощеному виді. От як моделюється генетичне спадкування

| | |
|---|---|
| Хромосома | Вектор (послідовність) з нулів і одиниць.<br>Кожна позиція (біт) називається геном. |
| Ідивідум= | Набір хромосом= варіант рішення |

| генетичний код | задачі |
|---|---|
| Кросовер | Операція, при якій дві хромосоми обмінюються своїми частинами |
| Мутація | Випадкова зміна однієї або декількох позицій в хромосомі |

Щоб змоделювати еволюційний процес, згенеруємо спочатку випадкову популяцію - кілька індивідумів з випадковим набором хромосом (числових векторів). Генетичний алгоритм імітує еволюцію цієї популяції як циклічний процес схрещування індивідумів і зміни поколінь.




Життєвий цикл популяції - це кілька випадкових схрещувань (за допомогою кроссовера) і мутацій, у результаті яких до популяції додається якась кількість нових індивідумів. Відбір у генетичному алгоритмі - це процес формування нової популяції зі старої, після чого стара популяція гине. Після відбору до нової популяції знову застосовуються операції кросовера і мутації, потім знову відбувається відбір, і так далі.

Відбір у генетичному алгоритмі тісно зв'язаний із принципами природного відбору в природі в такий спосіб:

| Пристосування індивідума | Значення цільової функції на цьому індивідумі |
|---|---|
| Виживання найбільш пристосованих | Популяція наступного покоління формується у відповідності з цільовою функцією. Чим пристосованіший індивідум, тим більша ймовірність його участі у кросовері, тобто розмноженні |

Таким чином, модель відбору визначає, яким чином варто будувати популяцію наступного покоління. Як правило, ймовірність участі індивідума в схрещуванні береться пропорційно його пристосуванню. Часто використовується так називана *стратегія елітизма,* при якій кілька кращих індивідумів переходять у наступне покоління без змін, не беручи участі у кросовері і доборі. У будь-якому випадку кожне наступне покоління буде в середньому краще попереднього. Коли пристосованість індивідумів перестає помітно збільшуватися, процес зупиняють і як розв'язання задачі оптимізації беруть найкращого зі знайдених індивідумів.

Повертаючись до задачі оптимального розподілу інвестицій, пояснимо особливості реалізації генетичного алгоритму в цьому випадку.

1. Індивідум = варіант рішення задачі = набір з 10 хромосом $X_j$
2. Хромосома $X_j$= обсяг вкладення в проект $j$ = 16-розрядний запис цього числа
3. Так як обсяги вкладень обмежені, не всі значення хромосом є припустимими. Це враховується при генерації популяцій
4. Так як сумарний обсяг інвестицій фіксований, то реально варіюються тільки 9 хромосом, а значення 10-ої визначається по них однозначно.

Нижче приведені результати роботи генетичного алгоритму для трьох різних значень сумарного обсягу інвестицій $K$ (рис. 3.2). Квадратами на графіках прибутків відзначено, який обсяг вкладення в даний проект рекомендовано генетичним алгоритмом. Видно, що при малому значенні $K$ інвестуються тільки ті проекти, які прибуткові при мінімальних вкладеннях.


Рис. 3.2.

Якщо збільшити сумарний обсяг інвестицій, стає прибутковим вкладати гроші й у більш дорогі проекти (рис.3.3).

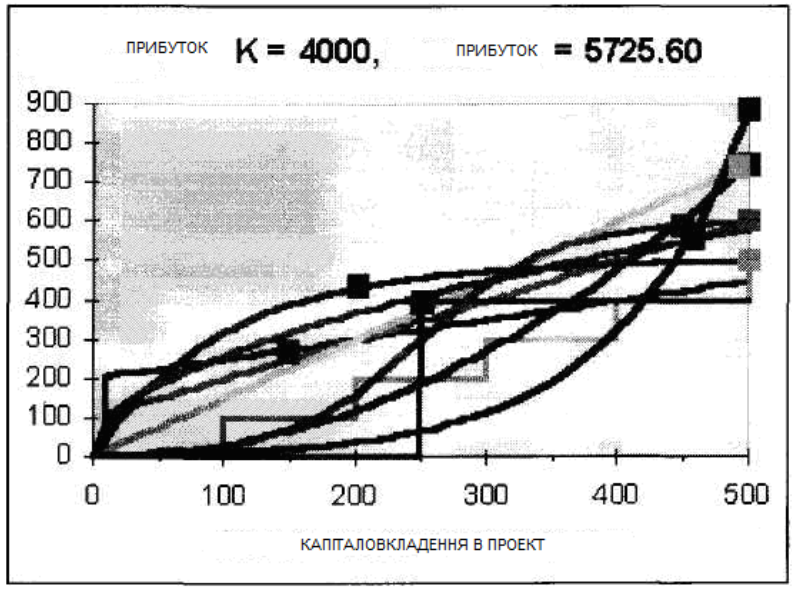



Рис. 3.3.

При подальшому збільшенні *K* досягається поріг максимального вкладення в прибуткові проекти, і інвестування в малоприбуткові проекти знову набуває сенсу.

## 3.3. Особливості генетичних алгоритмів

Генетичний алгоритм - новітній, але не єдино можливий спосіб розв'язання задач оптимізації. Віддавна відомі два основних шляхи розв'язання таких задач - *переборний* і *локально-градієнтний*. У цих методів свої достоїнства і недоліки, і в кожному конкретному випадку варто подумати, який з них вибрати. Розглянемо достоїнства і недоліки стандартних і генетичних методів на прикладі класичної *задачі комівояжера* (TSP - travelling salesman problem). Суть задачі полягає в тому, щоб знайти найкоротший замкнений шлях обходу декількох міст, заданих своїми координатами. Виявляється, що вже для 30 міст пошук оптимального шляху являє собою складну задачу, що

спонукала розвиток різних нових методів (у тому числі нейромереж і генетичних алгоритмів).

Кожен варіант розв'язання (для 30 міст) - це числовий ряд, де на *j*-ому місці стоїть номер *j*-ого по порядку обходу міста. Таким чином, у цій задачі 30 параметрів, причому не всі комбінації значень припустимі. Природно, першою ідеєю є повний перебір усіх варіантів обходу.

Переборний метод найбільш простий по своїй суті і тривіальний у програмуванні. Для пошуку оптимального розв'язання (точки максимуму цільової функції) потрібно послідовно обчислити значення цільової функції у всіх можливих точках, запам'ятовуючи максимальне з них (рис.3.4).

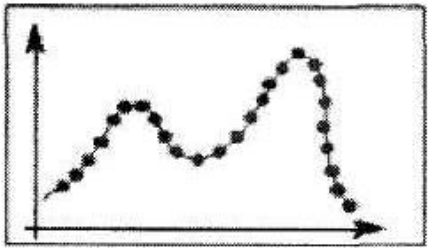


Рис. 3.4.

Недоліком цього методу є велика обчислювальна вартість. Зокрема, у задачі комівояжера буде потрібно прорахувати довжини більш $10^{30}$ варіантів шляхів, що зовсім нереально. Однак, якщо перебір усіх варіантів за розумний час можливий, то можна бути абсолютно упевненим у тім, що знайдене розв'язання дійсно оптимальне.

Другий популярний спосіб заснований на методі градієнтного спуска. При цьому спочатку вибираються деякі випадкові значення параметрів, а потім ці значення поступово змінюють, домагаючись найбільшої швидкості росту цільової функції (рис.3.5).

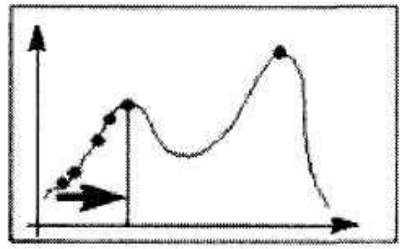


Рис. 3.5.

Досягши локального максимуму, такий алгоритм зупиняється, тому для пошуку глобального оптимуму будуть потрібні додаткові зусилля.

Градієнтні методи працюють дуже швидко, але не гарантують оптимальності знайденого розв'язання.

Вони ідеальні для застосування в так званих *унімодальних* задачах, де цільова функція має єдиний локальний максимум (він же - глобальний). Легко бачити, що задача комівояжера унімодальною не є (рис. 3.6).

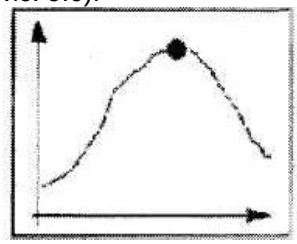


Рис. 3.6.

Типова практична задача, як правило, *мультимодальна* і багатомірна, тобто містить багато параметрів. Для таких задач не існує жодного універсального методу, що дозволяв би досить швидко знайти абсолютно точне розв'язання (рис.3.7).


Рис. 3.7.

Однак, комбінуючи переборний і градієнтний методи, можна сподіватися отримати хоча б наближене розв'язання, точність якого буде зростати при збільшенні часу розрахунку (рис.3.8).

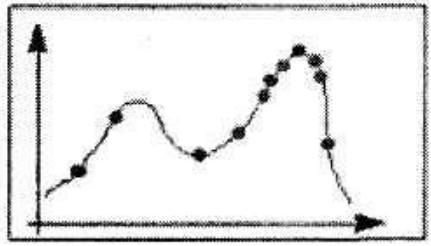


Рис. 3.8.

Генетичний алгоритм являє собою саме такий комбінований метод. Механізми схрещування і мутації в якомусь змісті реалізують переборну частину методу, а добір кращих розв'язаннь - градієнтний спуск. На рисунку показано, що така комбінація дозволяє забезпечити стійко гарну ефективність генетичного пошуку для будь-яких типів задач (рис. 3.9).

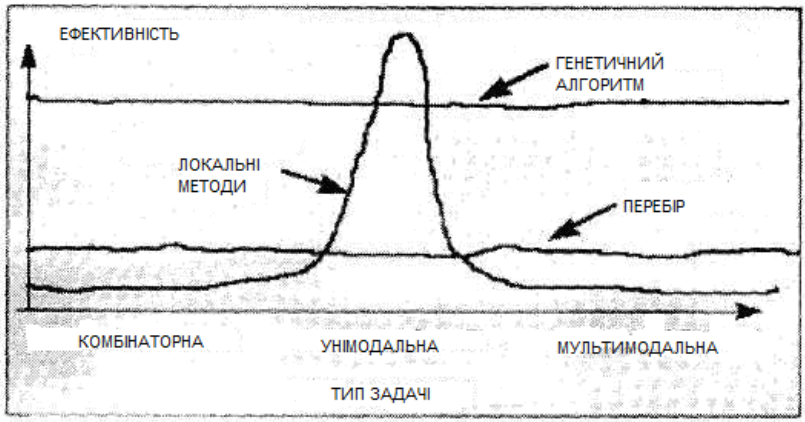



Рис. 3.9.

Отже, якщо на деякій множині задана складна функція від декількох змінних, то генетичний алгоритм - це програма, що за розумний час знаходить точку, де значення функції досить близьке до максимально можливого. Вибираючи прийнятний час розрахунку, ми одержимо одне з кращих розв'язаннь, що узагалі можливо одержати за цей час.

## 3.4. Задачі оптимізації і застосування алгоритмів

Генетичні алгоритми - це дуже популярні в даний час способи розв'язання задач оптимізації. У їхній основі лежить використання еволюційних принципів для пошуку оптимального розв'язання. Уже сама ідея виглядає досить інтригуючою і цікавою, щоб перетворити її в життя, а численні позитивні результати тільки розпалюють інтерес з боку дослідників. Тут не буде розглядатися історія становлення і визнання еволюційних обчислень взагалі і генетичних алгоритмів зокрема. Замість цього ми відразу перейду до розгляду самих алгоритмів. Як ми вже говорили, генетичні алгоритми в основному застосовуються для розв'язання задач оптимізації, тобто задач, у яких є деяка функція декількох змінних $F(x_1, x_2, \ldots, x_n)$ і необхідно знайти або її максимум, або її мінімум. Функція $F$ називається

*цільовою функцією*, а змінні - *параметрами функції*. Генетичні алгоритми "пришиваються" до даної задачі в такий спосіб. Параметри задачі є генетичним матеріалом - генами. Сукупність генів складає хромосому. Кожна особа має свою хромосому, а, отже, свої набори параметрів. Підставивши параметри в цільову функцію, можна одержати якесь значення. Те, наскільки це значення задовольняє поставленим умовам, визначає характеристику особи, що називається *пристосованістю (fitness)*. Функція, що визначає пристосованість повинна задовольняти наступній умові: чим "краща" особа, тим вище пристосованість. Генетичні алгоритми працюють з популяцією як правило фіксованого розміру, що складається з особей, заданих за допомогою способу, описаного вище. Особи "схрещуються" між собою за допомогою *генетичних операторів* (про те, як відбувається цей процес – буде описано окремо), і в такий спосіб виходять нащадкти, причому частина нащадків замінюють представників більш старого покоління у відповідності зі *стратегією формування нового покоління*. Вибір особей для схрещування проводиться відповідно до *селективної стратегії (selection strategy)*. Заново сформована популяція знову оцінюється, потім вибираються найбільш гідні для схрещування особи, що схрещуються між собою, виходять «діти» і займають місце «старих» індивідумів і т.д. Усі це продовжується доти поки не знайдеться особа, гени якої представляють оптимальний набір параметрів, при яких значення цільової функції близьке до максимуму або мінімуму, або дорівнює йому. Зупинка роботи ГА може відбутися також у випадку, якщо популяція вироджується, тобто якщо практично немає розмаїтості в генах особей популяції, або якщо просто вийшов ліміт часу. Виродження популяції називають *передчасною збіжністю (premature convergence)*.

Може створитися враження, що ГА є просто перекрученим варіантом випадкового пошуку. Але пристосованість була введена зовсім не даремно. Справа в тім, що вона безпосередньо впливає на шанс особи взяти участь у схрещуванні з наступним «народженням дітей». Вибираючи щораз для схрещування найбільш пристосованих особей, можна з визначеним степенем упевненості стверджувати, що нащадки будуть або не набагато гірші, ніж батьки, або кращі їх. Приблизно

цю величину впевненості можна оцінити за допомогою теореми шаблонів (теореми шим).

Теоретичні аспекти ГА, що випливають:

- Представлення данних в генах
- Генетичні оператори
- Моделі ГА
- Функції
- Стратегії відбору і формування нового покоління

Де ж застосовуються ГА? Усього застосувань дуже багато, тому приведений список не є вичерпним.

- Екстремальні задачі (пошук точок мінімуму і мінімуму);
- Задачі про найкоротший шлях;
- Задачі компонування;
- Складання розкладів;
- Апроксимація функцій;
- Добір (фільтрація) вхідних даних;
- Настроювання штучної нейронної мережі;
- Моделювання штучного життя (Artificial life systems) ;
- Біоінформатика (згортання білків і РНК);
- Ігрові стратегії;
- Нелінійна фільтрація;
- Агенти, що розвиваються/машини (Evolvable agents/machines);
- Оптимизация запросів в базах данних
- Різноманітні задачі на графах (задача комівояжера, розфарбування, знахождення паросполученнь)
- Навчання штучної нейронної мережі
- Штучне життя

Деякі розділи можуть містити підпункти. Так, наприклад, екстремальні задачі включають у мебя цілий клас задач лінійного і нелінійного програмування.

## 3.5. Історія появи еволюційних алгоритмів

Генетичні алгоритми є частиною більш загальної групи методів, які називають *еволюційними обчисленнями*, що поєднують різні варіанти використання еволюційних принципів для досягнення поставленої мети. Стимулом виникнення цих методів стали кілька відкриттів у біології. Чарльз Дарвін опублікував у 1859р. свою знамениту роботу "Походження видів", де були проголошені основні принципи еволюційної теорії: *спадковість, мінливість і природний відбір*. Тим самим було показано, що використання даних механізмів здатно привести до визначеного результату під впливом навколишнього середовища. Однак довгий час залишалося відкритим питання про те, як же все-таки передається генетична інформація від батьків нащадкам. У 1944 році Эйвері, Маклеод і Маккарті опублікували результати своїх досліджень, що доводили, що за спадкоємні процеси відповідальна "кислота дезоксирибозного типу". Однак про те, як працює ДНК, стало відомо 27 квітня 1953 року, коли в номері журналу "Nature" вийшла стаття Уотсона і Лементу, які вперше запропонували модель дволанцюгової спіралі ДНК. У такий спосіб стали відомі всі необхідні компоненти для реалізації моделі еволюції на комп'ютері. Тут варто трохи зупинитися на наступному. Справа в тім, що, напевно, єдина мета існування природного комплексу - виживання в постійно змінюванних умовах навколишнього середовища. І хоча сенс існування - споконвічне питання для філософів, але якщо трохи абстрагуватися від глобальних задумів начебто збагнення сутності Всесвіту, досягнення трансцендентності, злиття з космічним Я і пошуку відповіді на питання "Що було раніш, курка або яйце?", то можна зробити припущення, що ***сенс життя - у самому житті***. Загалом, можна сказати, що *у всіх штучних системах* є таке поняття, як *призначення*, інакше кажучи, *зміст, мета*. Ці параметри є й в еволюційних алгоритмів, за винятком, може, систем штучного життя (Artificial Life (A-Life) Systems). . Перша публікація, яку можна віднести до генетичних алгоритмів з'явилася в 1957м, автором був Барічеллі Н.А. (Barricelli N.A.), і називалася вона "Symbiogenetic evolution processes realised by artificial methods". У 1962 році з'явилася ще одна його робота "Numerical testing of evolution theories". Приблизно в той же час

ще один дослідник Фрейзер А.С. (Fraser A.S.) також опублікував дві статі: "Simulation of genetic systems by automatic digital computers: S-linkage, dominance, and epistasis" (1960) і "Simulation of genetic systems" (1962). Незважаючи на те, що роботи обох авторів були спрямовані насамперед на розуміння природного феномена спадковості, робота Фрейзера має багато загального із сьогоднішнім баченням генетичних алгоритмів. Він моделював еволюцію 15-бітних рядків і підраховував процентний вміст особей із вдалим фенотипом в успішних поколіннях. Його роботи нагадують оптимізацію функцій, однак у роботах Фрейзера немає жодного згадування про можливості використовувати ГА для штучних задач. У багатьох джерелах саме Холланда називають батьком сучасної теорії ГА. Однак, Холланд займався ними не із самого початку. Його цікавила, насамперед, здатність природних систем до адаптації, а його мрією було створення такої системи, що могла б пристосуватися до будь-яких умов навколишнього середовища. Заслуга Холланда в тім, що він усвідомив значення еволюційних принципів в адаптації і розвив свої припущення. Разом зі своїми студентами, що слухали курси по адаптивних системах в Університеті штату Мічиган, він розробляє те, що згодом назве "генетичним планом", а нам це відомо як "генетичний алгоритм". Про те, наскільки серйозно велися роботи в цьому напрямку говорить уже те, що один зі студентів Холланда - Кеннет Де Йонг (Kenneth De Jong) захищав дисертацію на степінь Доктора Філософії (Doctor of Philosophy) у 1975 році! Дисертація називається "An Analysis of the Behavior of a Class of Genetic Adaptive Systems" і може стати першим путівником для тих, хто хоче займатися ГА. У тому ж році виходить знаменита книга Холланда "Адаптація в природних і штучних системах" ( "Adaptation in Natural and Artificial Systems", 1975). Після цього ГА починають привертати до себе усе більше уваги, ними займається усе більше дослідників, що знаходять їм нові сфери застосування. У 1992 році, за даними бібліографії Ярмо Аландера (Jarmo T. Alander, 1994), число публікацій перевалило за 500, що, узагалі ж, небагато, однак у 1982 році їх було всього 15! На закінчення можна сказати, що *ГА разом зі штучними нейронними мережами є напрямками сучасного штучного інтелекту.* Справа в тім, що є ще класичний штучний інтелект, на якому побудовані більшість експертних систем і баз

знань, його відмінна риса в тім, що він, в основному, використовує логіку для побудови висновків.

## Основні поняття генетичних алгоритмів

**Генети́чний алгори́тм** (англ. *Genetic algorithm*)*—це евристичний алгоритм пошуку, використовуваний для рішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.* Є різновидом еволюціних числень (англ. *evolutionary computation*). Відмінною рисою генетичного алгоритму є акцент на використання оператора «схрещування», що робить операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещування в живій природі. «Батьком-засновником» генетичних алгоритмів, як ми вже казали, вважається Джон Холланд (англ. *John Holland*), книга якого «Адаптація в природних і штучних системах» (англ.*AdaptationinNaturalandArtificialSystems*) є основною працею в цій області досліджень.

При описі генетичних алгоритмів використовуються означення, запозичені з генетики. Наприклад, мова йде про *популяції особей,* а як базові поняття застосовуються *ген, хромосома, генотип, фенотип, аллель.* Також використовуються відповідним цим термінам означення з технічного лексикона, зокрема, *ланцюг, двоїчна послідовність, структура.*

***Популяція*** - це кінцева множина *особей.*

***Особи,*** що входять у популяцію, у генетичних алгоритмах представляються хромосомами з закодованим у них множинами *параметрів задачі,* тобто рішень, які інакше називаються *точками в просторі пошуку (search points).* У деяких роботах особи називаються *організмами.*

***Хромосоми*** (інші назви - *ланцюжки* або *кодові послідовності)* - це упорядковані *послідовності генів.*

***Ген*** (також називаний *властивістю, знаком* або *детектором)* - це атомарний елемент *генотипу,* зокрема, хромосоми.

***Генотип*** або ***структура*** - це набір хромосом даної особи. Отже, особами популяції можуть бути генотипи або одиничні

хромосоми (у досить розповсюдженому випадку, коли генотип складається з однієї хромосоми).

***Фенотип*** - це набір значень, що відповідають даному генотипові, тобто *декодована структура* або множина *параметрів задачі (рішення, точка простору пошуку).*

***Аллель*** - це значення конкретного гена, також обумовлене як *значення властивості* або *варіант властивості.*

***Локус*** або ***позиція*** вказує місце розміщення даного гена в хромосомі (ланцюжку). Множина позицій генів - це ***локи***.

Дуже важливим поняттям у генетичних алгоритмах вважається ***функція пристосованості*** *(fitness function),* інакше називана ***функцією оцінки***. Вона представляє міру пристосованості даної особи в популяції. Ця функція відіграє найважливішу роль, оскільки дозволяє оцінити степінь пристосованості конкретних особей у популяції і вибрати з них найбільш пристосовані (тобто найбільші значення функції, що мають пристосованості) відповідно до еволюційного принципу виживання «найсильніших» (найкраще пристосувалися). Функція пристосованості також одержала свою назву безпосередньо з генетики. Вона впливає на функціонування генетичних алгоритмів і повинна мати точне і коректне означення. У задачах оптимізації функція пристосованості, як правило, оптимізується (точніше кажучи, максимізується) і називається *цільовою функцією.* У задачах мінімізації цільова функція перетворюється, і проблема зводиться до максимізації. У теорії керування функція пристосованості може приймати вид *функції похибки,* а в теорії ігор - *вартісної функції.* На кожній ітерації генетичного алгоритму пристосованість кожної особи даної популяції оцінюється за допомогою функції пристосованості, і на цій основі створюється наступна популяція особей, що складають множину потенційних рішень проблеми, наприклад, задачі оптимізації.

Чергова популяція в генетичному алгоритмі називається ***поколінням,*** а до знову створюваної популяції особей застосовується термін «нове покоління» або «покоління нащадків».

**Приклад 3.1**

Розглянемо функцію

$$f(x) = 2x^2+1 \qquad (3.1)$$

і припустимо, що $x$ приймає цілі значення з інтервалу від 0 до 15. Задача оптимізації цієї функції полягає в переміщенні по

простору, що складається з 16 точок зі значеннями 0, 1.....15 для виявлення тієї точки, у якій функція приймає максимальне (або мінімальне) значення.

У цьому випадку як *параметр задачі* виступає змінна *x.* Множина {0, 1.....15} складає *простір пошуку* й одночасно - множину потенційних рішень задачі. Кожне з 16 чисел, що належать до цієї множини, називається *точкою простору пошуку, рішенням, значенням параметра, фенотипом.* Слід зазначити, що рішення, яке оптимізує функцію, називається *найкращим* або *оптимальним* рішенням. Значення параметра *x* від 0 до 15 можна закодувати в такий спосіб:

0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111

Це широко відомий спосіб двоїчного кодування, зв'язаний із записом десяткових цифр у двоїчній системі. Представлені *кодові послідовності* також називаються *ланцюгами* або *хромосомами.* У розглянутому прикладі вони виступають і в ролі *генотипів.* Кожна з хромосом складається з 4 генів (інакше можна сказати, що двоїчні послідовності складаються з 4 бітів). Значення гена в конкретній позиції називається аллелью, що приймає в даному випадку значення 0 або 1. *Популяція* складається з *особей,* які обираються серед цих 16 *хромосом.* Прикладом популяції з чисельністю, рівною 6, може бути, наприклад, множина хромосом {0010, 0101, 0111, 1001, 1100, 1110}, що представляють собою закодовану форму наступних фенотипів: {2, 5, 7, 9, 12, 14}. *Функція пристосованості* в цьому прикладі задається виразом (3.1). Пристосованість окремих хромосом у популяції визначається значенням цієї функції для значень *x*, що відповідають цим хромосомам, тобто для *фенотипів,* що відповідають визначеним *генотипам.*

**Приклад 3.2**

Розглянемо наступний приклад постановки оптимізаційної задачі. Для системи, зображеної на рис. 3.10, варто знайти

$$\min_{k_1,k_2} J = \int_0^T f(y_1, y_2, u)dt$$

де $k_1, k_2 \in [k_{\min}, k_{\max}]$.

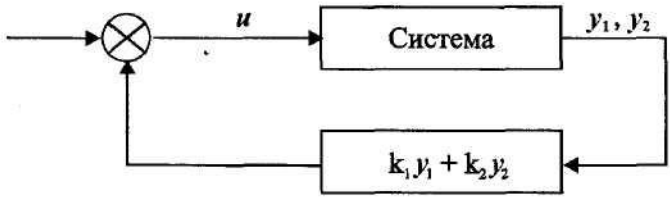



Рис. 3.10. Схема оптимизаційної двопараметричної системи.

У ролі параметрів цієї задачі виступають $k_1$ і $k_2$. Простір пошуку повинен містити кінцеву кількість точок, який можна закодувати у виді хромосом. Параметри $k_1$ і $k_2$ дискретизовані; множина їхніх значень у всьому діапазоні від мінімального $k_{min}$ до максимального $k_{max}$ відображаються на відповідні двоїчні кодові послідовності. При цьому значенню $k_{min}$ зіставлена кодова послідовність, що складається з одних нулів, а значенню $k_{max}$ - кодова послідовність, що складається з одних одиниць. Довжина цих кодових послідовностей залежить від значень $k_1$ і $k_2$, а також від частоти дискретизації інтервалу [$k_{min}$, $k_{max}$].

Припустимо, що $k_{min}$ = -25, а $k_{max}$ = 25 і для кожного з параметрів $k_1$ і $k_2$ застосовуються кодові послідовності довжиною 10. Приклад популяції, що складається з 10 особей, приведено у таблиці 3.1.

Перші 10 генів кожного генотипу відповідають параметрові $k_1$, а останні 10 генів - параметрові $k_2$. Таким чином, довжина хромосом дорівнює 20.

Спосіб кодування значень параметрів $k_1$ і $k_2$ у виді хромосом буде докладно викладений далі.

**Таблиця 3.1.** Популяція особей (для приклада 3.2)

| Генотипы | Фенотипы | |
|---|---|---|
| 0000000000000000000 | –25,00 | –25,00 |
| 10100010010011001011 | 6,72 | –15,08 |
| 01101000101111010010 | –4,57 | 22,8. |
| 11011010011110000111 | 17,67 | 19,13 |
| 00011011000000010001 | –19,72 | –24,17 |
| 00110000101011111010 | –15,52 | 12,24 |
| 11111111111111111111 | 25,00 | 25,00 |

**Приклад 3.3**

Розглянемо нейронну мережу, представлену на рис. 3.11, для якої необхідно підібрати оптимальні ваги $w_{11}$, $w_{12}$, $w_{21}$, $w_{22}$, $w_{31}$, $w_{32}$ і $w_{10}$, $w_{20}$ і $w_{30}$, що мінімізують значення похибки

$$Q = \frac{1}{4}\sum_{i=1}^{4}(d_i - y_i)^2 .$$

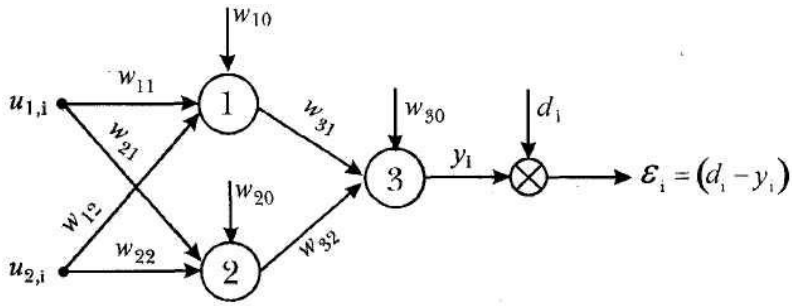



Рис. 3.11. Нейронна мережа, що реалізує операцію XOR

Це мережа, що реалізує систему XOR, тому значення $u_{1,i}$ і $u_{2,i}$, а також $d_i$ для $i$ = 1.....4 приймають значення, що приведені у таблиці.

| $u_1$ | $u_2$ | $d = \text{XOR}(u_1, u_2)$ |
|---|---|---|
| +1 | +1 | –1 |
| +1 | –1 | +1 |
| –1 | +1 | +1 |
| –1 | –1 | –1 |

В якості параметрів розглянутої задачі виступають перераховані вище ваги, тобто задача має 9 параметрів. У даному прикладі набір з цих 9 параметрів визначає точку простору пошуку і, отже, являє собою можливе рішення. Припустимо, що ваги можуть приймати значення з інтервалу [-10, 10]. Тоді кожна хромосома буде комбінацією з 9 двоїчних послідовностей, що кодують конкретні ваги. Це, мабуть, і є генотипи. Відповідні ним фенотипи представлені значеннями окремих ваг, тобто множинами відповідних дійсних чисел з інтервалу [-10, 10].

У приведених прикладах (3.1 - 3.3) хромосоми і генотипи позначають те саме - фенотипи особей популяції, закодовані у формі упорядкованих послідовностей генів зі значеннями (аллелями), рівними 0 або 1.

У генетиці генотип задає генетичну структуру особи, що може включати більш однієї хромосоми. Наприклад, клітки людини містять 46 хромосом. У генетичних алгоритмах генотип визначається аналогічним чином, однак найчастіше він складається усього з однієї хромосоми, що і виступає в ролі особи популяції.

Довжина хромосом залежить від умов задачі. Варто помітити, що в природних організмах хромосома може складатися з декількох сотень і навіть тисяч генів. У людини є близько 100 000 генів, хоча їхня точна кількість дотепер невідома.

## 3.6. Опис алгоритму

Задача кодується таким чином, щоб її рішення могло бути представлене у виді вектора («хромосома»). Випадковим чином створюється деяка кількість початкових векторів («початкова популяція»). Вони оцінюються з використанням «функції

пристосованості», у результаті чого кожному векторові привласнюється визначене значення («пристосованість»), що визначає ймовірність виживання організму, представленого даним вектором. Після цього з використанням отриманих значень пристосованості вибираються вектори (*селекції*), допущені до «схрещування». До цих векторів застосовуються «генетичні оператори» (у більшості випадків «схрещування»-crossoveri «мутація»-mutation), створюючи в такий спосіб наступне «покоління». Особи наступного покоління також оцінюються, потім виробляється селекція, застосовуються генетичні оператори і т.д. Так моделюється «еволюційний процес», що продовжується кілька життєвих циклів (*поколінь*), поки не буде виконаний критерій зупинки алгоритму. Таким критерієм може бути:

- отримання глобального, або субоптимального рішення;
- вичерпання числа поколінь, відпущених на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

Генетичні алгоритми служать, головним чином, для пошуку рішень у дуже великих, складних просторах пошуку.

Таким чином, можна виділити наступні етапи генетичного алгоритму:

1. Створення початкової популяції
2. Обчислення функцій пристосованості для особей популяції (оцінювання)

- (Початок циклу)

    1. Вибір індивидів з поточної популяції (селекція)
    2. Схрещування і\або мутація
    3. Обчислення функцій пристосованості для всіх особей
    4. Формування нового покоління
    5. Якщо виконуються умови іншого, то (кінець циклу), інакше (початок циклу).

## Створення початкової популяції

Перед першим кроком потрібно випадковим чином створити якусь початкову популяцію; навіть якщо вона виявиться зовсім неконкурентоспроможною, генетичний алгоритм усе рівно досить швидко переведе її в життєздатну популяцію. Таким чином, на першому кроці можна особливо не намагатися зробити занадто дуже пристосованих особей, досить, щоб вони відповідали форматові особей популяції, і на них можна було підрахувати функцію пристосованості (Fitness). Підсумком першого кроку є популяція H, що складається з N особей.

## Відбір

На етапі відбору потрібно з усієї популяції вибрати визначену її частку, що залишиться "у живих" на цьому етапі еволюції. Є різні способи проводити відбір. Ймовірність виживання особи *h* повинна залежати від значення функції пристосованості Fitness (*h*). Сама частка що вижила *s* зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. За підсумками відбору з *N* особей популяції *H* повинні залишитися *s* особей, що ввійдуть у підсумкову популяцію *H*'. Інші особи гинуть.

## Розмноження

Розмноження в генетичних алгоритмах зазвичай полове - щоб зробити нащадка, потрібно декілька батьків; зазвичай, звичайно, потрібно рівно два. Розмноження в різних алгоритмах визначається по-різному - воно, зазвичай, залежить від представлення даних. Головна вимога до розмноження - щоб нащадок або нащадки мали можливість успадкувати риси обох батьків, "змішавши" їх яким-небудь досить розумним способом. Узагалі говорячи, для того щоб провести операцію розмноження, потрібно вибрати *(1-s)p/2* пар гіпотез з *H* і провести з ними розмноження, одержавши по два нащадка від кожної пари (якщо розмноження визначене так, щоб давати одного нащадка, потрібно вибрати *(1 - s)p* пар), і додати цих нащадків у *H*'. У результаті *H*' буде складатися з *N* особей. Чому особи для

розмноження звичайно вибираються з усієї популяції *H*, а не з елементів, що вижили на першому кроці, $H_0$ (хоча останній варіант теж має право на існування)? Справа в тім, що головна критика багатьох генетичних алгоритмів - недолік розмаїтості (diversity) в особях. Досить швидко виділяється один-єдиний генотип, що являє собою локальний максимум, а потім всі елементи популяції програють йому вибір, і вся популяція "забивається" копіями цієї особи. Є різні способи боротьби з таким небажаним ефектом; один з них - вибір для розмноження не самих пристосованих, але узагалі всіх особей.

### Мутації

До мутацій відноситься все те ж саме, що і до розмноження: є деяка частка мутантів *m*, що є параметром генетичного алгоритму, і на кроці мутацій потрібно вибрати *m* особей, а потім змінити їх відповідно до заздалегідь визначених операцій мутації.

### Приклад реалізації генетичного алгоритму мовою C++

Пошук в одномірному просторі, без схрещування.

```
#include <iostream.h>
#include <algorithm.h>
#include <numeric.h>
using namespace std;
int main()
{
    const int N = 1000;
    int a[N];
    //заповнюємо нулями
    fill(a, a+N, 0);
    for (;;)
    {
        //мутація у випадкову сторону кожного
елемента:
        for (int i = 0; i < N; ++i)
            if (rand()%2 == 1)
                a[i] += 1;
            else
                a[i] -= 1;
```

```
        //тепер вибираємо кращих, відсортувавши по
зростанню...
        sort(a, a+N);
        //... і тоді кращі виявляться в другій
половині масиву.
        //скопіюємо кращих у першу половину, куди
вони залишили нащадків, а перші вмерли:
        copy(a+N/2, a+N,    a /*куди*/);
        //тепер подивимося на середній стан
популяції. Як бачимо, вона усе краще і краще.
        cout << accumulate(a, a+N, 0) / N << endl;
    }
}
```

## 3.7. Класичний генетичний алгоритм

Основний (класичний) генетичний алгоритм (також називаний елементарним або простим генетичним алгоритмом) складається з наступних кроків:

1) ініціалізація, або вибір вихідної популяції хромосом;
2) оцінка пристосованості хромосом у популяції;
3) перевірка умови зупинки алгоритму;
4) селекція хромосом;
5) застосування генетичних операторів;
6) формування нової популяції;
7) вибір «найкращої» хромосоми.

Блок-схема основного генетичного алгоритму зображена на рис. 3.12.

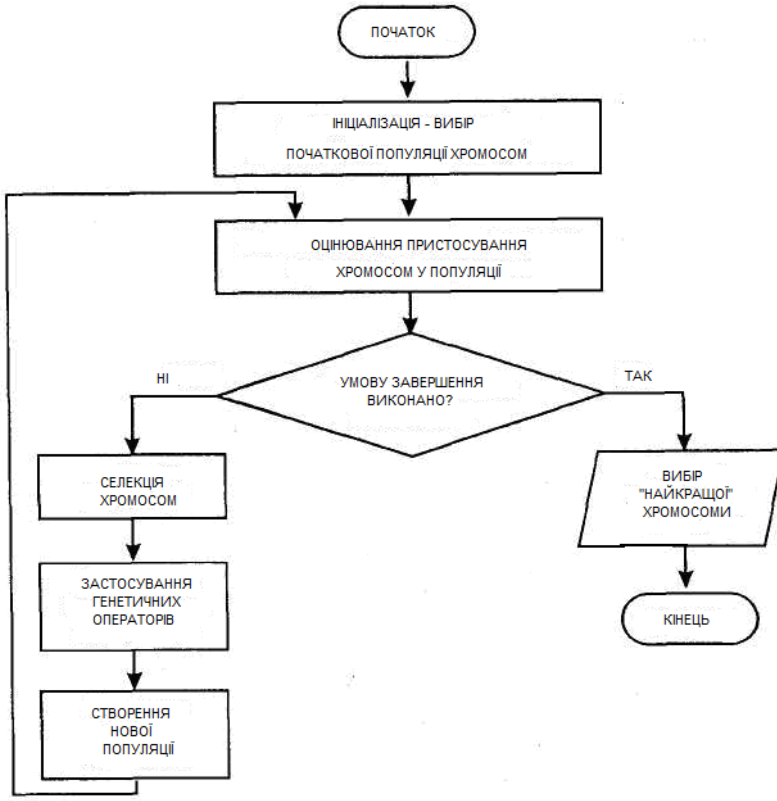



Рис. 3.12. Блок-схема генетичного алгоритму.

Розглянемо конкретні етапи цього алгоритму більш докладно з використанням додаткових подробиць, представлених на рис. 3.13.

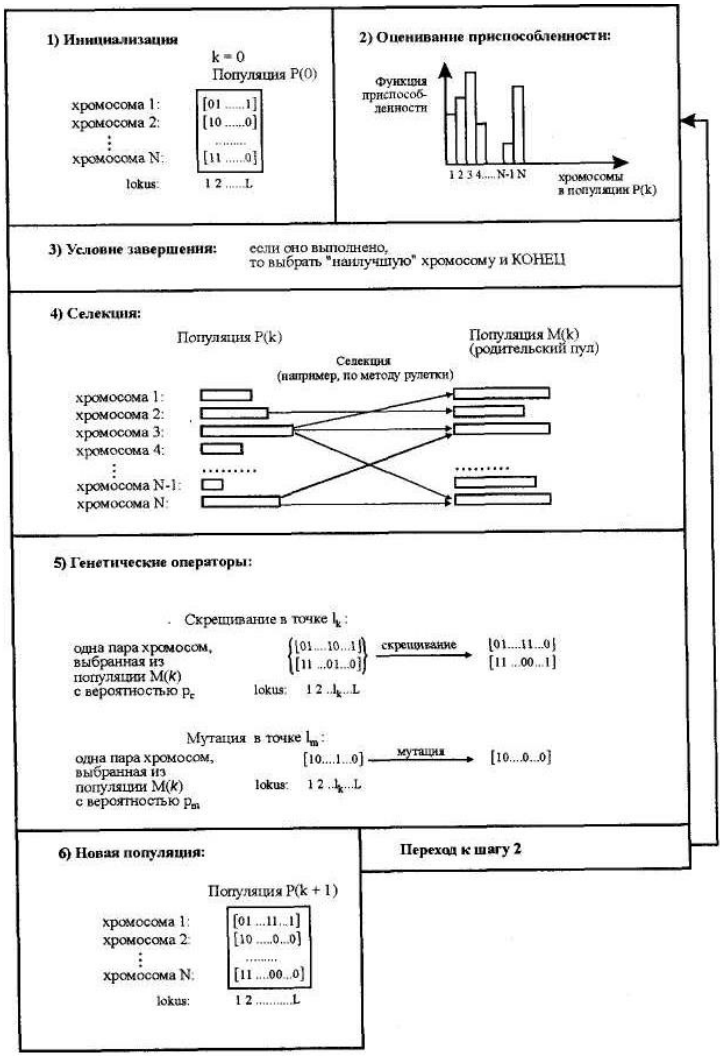



Рис. 3.13. Схема виконання генетичного алгоритму.

**Ініціалізація,** тобто формування вихідної популяції, полягає у випадковому виборі заданої кількості хромосом (особей), що

представляються двоїчними послідовностями фіксованої довжини.

**Оцінювання пристосованості** хромосом у популяції складається в розрахунку функції пристосованості для кожної хромосоми цієї популяції. Чим більше значення цієї функції, тим вище «якість» хромосоми. Форма функції пристосованості залежить від характеру розв'язуваної задачі. Передбачається, що функція пристосованості завжди приймає ненегативні значення і, крім того, що для рішення оптимізаційної задачі потрібно максимізувати цю функцію. Якщо вихідна форма функції пристосованості не задовольняє цим умовам, то виконується відповідне перетворення (наприклад, задачу мінімізації функції можна легко звести до задачі максимізації).

**Перевірка умови зупинки алгоритму.** Визначення умови зупинки генетичного алгоритму залежить від його конкретного застосування. В оптимізаційних задачах, якщо відомо максимальне (або мінімальне) значення функції пристосованості, то зупинка алгоритму може відбутися після досягнення очікуваного оптимального значення, можливо - із заданою точністю. Зупинка алгоритму також може відбутися у випадку, коли його виконання не приводить до поліпшення вже досягнутого значення. Алгоритм може бути зупинений після закінчення визначеного часу виконання або після виконання заданої кількості ітерацій. Якщо умова зупинки виконана, то виконується перехід до завершального етапу вибору «найкращої» хромосоми. У противному випадку на наступному кроці виконується селекція.

**Селекція хромосом** полягає у відборі (по розрахованним на другому етапі значенням функції пристосованості) тих хромосом, що будуть брати участь у створенні нащадків для наступної популяції, тобто для чергового покоління. Такий відбір виконується відповідно до принципу природного відбору, по якому найбільші шанси на участь у створенні нових особей мають хромосоми з найбільшими значеннями функції пристосованості. Існують різні методи селекції. Найбільш популярним вважається так називаний *метод рулетки (roulette wheel selection),* що свою назву одержав за аналогією з відомою азартною грою. Кожній хромосомі може бути зіставлений сектор

колеса рулетки, величина якого встановлюється пропорційно значенню функції пристосованості даної хромосоми. Тому чим більше значення функції пристосованості, тим більше сектор на колесі рулетки. Усе колесо рулетки відповідає сумі значень функції пристосованості всіх хромосом розглянутої популяції. Кожній хромосомі, що позначається $ch_i$ для $i$=1,2,..., $N$ (де $N$ позначає чисельність популяції) відповідає сектор колеса $v(ch_i)$, виражений у відсотках відповідно до формули

$$v(ch_i) = p_s(ch_i)100\% , \qquad (3.2)$$

де

$$p_s(ch_i)=\frac{F(ch_i)}{\sum_{i=1}^{N} F(ch_i)} \qquad (3.3)$$

причому $F(ch_i)$ - значення функції пристосованості хромосоми $ch_i$, а $p_s(ch_i)$ - *ймовірність селекції* хромосоми $ch_i$. Селекція хромосоми може бути представлена як результат повороту колеса рулетки, оскільки «вигравша» (тобто обрана) хромосома відноситься до того сектора колеса, що випав. Очевидно, що чим більше сектор, тим більше ймовірність «перемоги» відповідної хромосоми. Тому ймовірність вибору даної хромосоми виявляється пропорційною значенню її функції пристосованості. Якщо все коло колеса рулетки представити у виді цифрового інтервалу [0, 100], то вибір хромосоми можна ототожнити з вибором числа з інтервалу [*a*, *b*], де *a* і *b* позначають відповідно початок і закінчення фрагмента кола, що відповідає цьому секторові колеса; очевидно, що $0 \le a < b \le 100$. У цьому випадку вибір за допомогою колеса рулетки зводиться до вибору числа з інтервалу [0, 100], що відповідає конкретній точці на колі колеса. Інші методи селекції будуть розглядатися далі.

У результаті процесу селекції створюється *батьківська популяція,* яка також називана *батьківським пулом (mating pool)* з чисельністю *N,* рівною чисельності поточної популяції.

**Застосування генетичних операторів** до хромосом, відібраним за допомогою селекції, приводить до формування нової популяції нащадків від створеної на попередньому кроці батьківської популяції.

У класичному генетичному алгоритмі застосовуються два основних генетичних оператори: *оператор схрещування (crossover)* і *оператор мутації (mutation).* Однак слід зазначити,

що оператор мутації грає явно другорядну роль у порівнянні з оператором схрещування. Це означає, що схрещування в класичному генетичному алгоритмі виконується практично завжди, тоді як мутація - досить рідко. Ймовірність схрещування, як правило, досить велика (зазвичай $0{,}5 \le p_c \le 1$), тоді як ймовірність мутації встановлюється досить малою (найчастіше $0 \le p_m \le 0{,}1$). Це випливає з аналогії зі світом живих організмів, де мутації відбуваються надзвичайно рідко. У генетичному алгоритмі мутація хромосом може виконуватися на популяції батьків перед схрещуванням або на популяції нащадків, утворених у результаті схрещування.

**Оператор схрещування.** На першому етапі схрещування вибираються пари хромосом з батьківської популяції (батьківського пула). Це тимчасова популяція, що складається з хромосом, відібраних у результаті селекції і призначених для подальших перетворень операторами схрещування і мутації з метою формування нової популяції нащадків. На даному етапі хромосоми з батьківської популяції поєднуються в пари. Це виконується випадковим способом відповідно до ймовірності схрещування $p_c$. Далі для кожної пари відібраних у такий спосіб батьків розігрується позиція гена *(локус)* у хромосомі, що визначає так називану *точку схрещування.* Якщо хромосома кожного з батьків складається з $L$ генів, то очевидно, що точка схрещування $k$ являє собою натуральне число, менше $L$. Тому фіксація точки схрещування зводиться до випадкового вибору числа з інтервалу [1, -1]. У результаті схрещування пари батьківських хромосом виходить наступна пара нащадків:

1) нащадок, хромосома якого на позиціях від 1 до $k$ складається з генів першого батька, а на позиціях від $k$+1 до $L$ - з генів другого батька;

2) нащадок, хромосома якого на позиціях від 1 до $k$ складається з генів другого батька, а на позиціях від $k$+1 до $L$ - з генів першого батька.

Дія оператора схрещування буде проілюстрована прикладами 3.4 і 3.5.

**Оператор мутації** з ймовірністю $p_m$ змінює значення гена в хромосомі на протилежне (тобто з 0 на 1 або назад). Наприклад, якщо в хромосомі [100110101010] мутації піддається ген на позиції 7, то його значення, рівне 1, змінюється на 0, що призводить до утворення хромосоми [100110001010]. Як уже

згадувалося вище, ймовірність мутації зазвичай дуже мала, і саме від неї залежить, буде даний ген мутировати чи ні. Ймовірність $p_m$ мутації може емулюватися, наприклад, випадковим вибором числа з інтервалу [0, 1] для кожного гена і добором для виконання цієї операції тих генів, для яких розігране число виявляється меншим або рівним значенню $p_m$.

**Формування нової популяції.** Хромосоми, отримані в результаті застосування генетичних операторів до хромосом тимчасової батьківської популяції, включаються до складу нової популяції. Вона стає так називаною поточною популяцією для даної ітерації генетичного алгоритму. На кожній черговій ітерації розраховуються значення функції пристосованості для всіх хромосом цієї популяції, після чого перевіряється умова зупинки алгоритму й або фіксується результат у виді хромосоми з найбільшим значенням функції пристосованості, або здійснюється перехід до наступного кроку генетичного алгоритму, тобто до селекції. У класичному генетичному алгоритмі вся попередня популяція хромосом заміщується новою популяцією нащадків, що має ту ж чисельність.

**Вибір «найкращої» хромосоми.** Якщо умова зупинки алгоритму виконана, то варто вивести результат роботи, тобто представити шукане рішення задачі. Кращим рішенням вважається хромосома з найбільшим значенням функції пристосованості.

На завершення варто визнати, що генетичні алгоритми успадкували властивості природного еволюційного процесу, що складаються в генетичних змінах популяцій організмів з часом. Головний фактор еволюції - це природний відбір (тобто природна селекція), що приводить до того, що серед генетично розрізняючихся особей однієї і тієї ж популяції виживають і залишають нащадство тільки найбільш пристосовані до навколишнього середовища. У генетичних алгоритмах також виділяється етап селекції, на якому з поточної популяції відбираються і включаються в батьківську популяцію особи, що мають найбільші значення функції пристосованості. На наступному етапі, що іноді називається *еволюцією,* застосовуються генетичні оператори схрещування і мутації, що виконують рекомбінацію генів у хромосомах.

Операція схрещування полягає в обміні фрагментами ланцюжків між двома батьківськими хромосомами. Пари батьків

для схрещування вибираються з батьківського пула випадковим чином так, щоб ймовірність вибору конкретної хромосоми для схрещування дорівнювала б ймовірності $p_c$. Наприклад, якщо в якості батьків випадковим чином вибираються дві хромосоми з батьківської популяції чисельністю $N$, то $p_c = 2N$. Аналогічно, якщо з батьківської популяції чисельністю $N$ вибирається $2z$ хромосом ($z \le N/2$), які утворять $z$ пар батьків, то $p_c=2z/N$. Звертаємо увагу, що якщо всі хромосоми поточної популяції об'єднані в пари до схрещування, то $p_c=1$. Після операції схрещування батьки в батьківській популяції заміщаються їхніми нащадками.

Операція мутації змінює значення генів у хромосомах із заданою ймовірністю $p_m$ способом, представленим при описі відповідного оператора. Це приводить до інвертування значень відібраних генів з 0 на 1 і зворотно. Значення $p_m$, як правило, дуже мале, тому мутації піддається лише невелика кількість генів. Схрещування - це ключовий оператор генетичних алгоритмів, що визначає їхню можливості й ефективність. Мутація грає більш обмежену роль. Вона вводить у популяцію деяку розмаїтість і попереджає втрати, що могли б відбутися унаслідок виключення якого-небудь значного гена в результаті схрещування.

Основний (класичний) генетичний алгоритм, як ми вже відмічали, відомий у літературі як інструмент, у якому виділяються три види операцій: *репродукції, схрещування* і *мутації.* Терміни *селекція* і *репродукція* в даному контексті використовуються як синоніми. При цьому репродукція в даному випадку зв'язується скоріше зі створенням копій хромосом батьківського пула, тоді як більш розповсюджений зміст цього поняття означає процес формування нових особей, що відбуваються від конкретних батьків . Якщо ми приймаємо таке тлумачення, то оператори схрещування і мутації можуть вважатися операторами репродукції, а селекція - добором особей (хромосом) для репродукції.

## Генетичні оператори

Частково про генетичні оператори ми вже говорили. Зупинимося на їхньому описі більш докладно. Генетичні оператори необхідні, щоб застосувати принципи спадковості і мінливості до віртуальної популяції. Крім відмінних рис, про які буде розказано нижче, у них є така властивість як *ймовірність*. Тобто описувані оператори не обов'язково застосовуються до всіх особів, які схрещуються, що вносить додатковий елемент невизначеності в процес пошуку рішення. У даному випадку, невизначеність не має на увазі негативний фактор, а є таким собі "степенем волі" роботи генетичного алгоритму. Тут описуються тільки два найпоширеніші і необхідних оператори. Існують і інші генетичні оператори (наприклад, *інверсія*), але вони застосовуються дуже рідко і тому ми про них говорити не будемо.

## Оператор кросінговера

*Оператор кросінговера (crossover operator)*, також називаний кросовером, є основним генетичним оператором, за рахунок якого виконується обмін генетичним матеріалом між особами. Оператор моделює процес схрещування особей. Нехай є дві батьківські особи з хромосомами $X=\{x_i,\ i=1,L\}$ і $Y=\{y_i,\ i=1,L\}$. Випадковим чином визначається точка усередині хромосоми, у якій обидві хромосоми діляться на дві частини і обмінюються ними. Назвемо цю точку точкою розриву. Узагалі говорячи, в англомовній літературі вона називається *точкою кросінговера* (crossover point), просто, точка розриву більш образна назва і до того ж дозволяє в деяких випадках уникнути тавтології. Описаний процес зображено на рис.3.14.

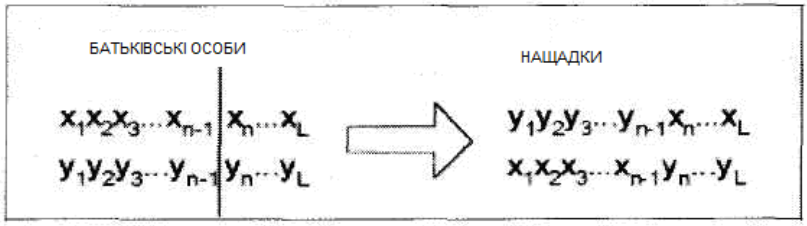



Рис.3.14. Кросінговер

Даний тип кросінговера називається *одноточковим*, так як при ньому батьківські хромосоми розрізаються тільки в одній випадковій точці. Також існують *2*-х і *n-точковий* оператори кросінговера. У 2-х точковому кросінговері точок розриву 2, а *n*-точковий кросінговер є своєрідним узагальненням 1- і 2-точкового кросінговерів для $n > 2$.

Крім описаних типів кросінговерів є ще *однорідний кросінговер*. Його особливість полягає в тім, що значення кожного біта в хромосомі нащадка визначається випадковим чином з відповідних бітів батьків. Для цього вводиться деяка величина $0<p_0<1$, і якщо випадкове число більше $p_0$, то на *n*-у позицію першого нащадка потрапляє *n*-й біт першого батька, а на *n*-у позицію другого - *n*-й біт другого батька. У противному випадку до першого нащадка потрапляє біт другого батька, а до другого - першого. Така операція проводиться для всіх бітів хромосоми.

Ймовірність кросінговера найвища серед генетичних операторів і дорівнює зазвичай 60% і більше.

## Оператор мутації

Оператор мутації (mutation operator) необхідний для "вибивання" популяції з локального экстремума і сприяє захистові від передчасної збіжності. Досягаються це за рахунок того, що інвертується випадково обраний біт у хромосомі, що і показано на рис. 3.15.

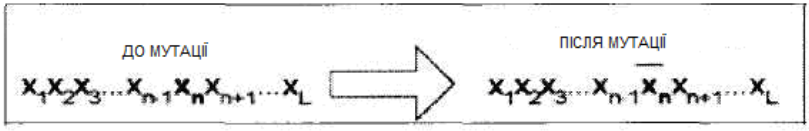



Рис. 3.15. Мутація

Так само як і кросінговер, мутація проводиться не тільки по одній випадковій точці. Можна вибирати деяку кількість точок у хромосомі для інверсії, причому їхнє число також може бути випадковим. Також можна інвертувати відразу деяку групу підряд

ідучих точок. Ймовірність мутації значно менша ймовірності кросінговера і рідко перевищує 1%. Серед рекомендацій з вибору ймовірності мутації нерідко можна зустріти варіанти 1/*L* або 1/*N*, де *L* - довжина хромосоми, *N* - розмір популяції. Необхідно також відзначити, що деякі автори вважають, що оператор мутації є основним пошуковим оператором і відомі алгоритми, що не використовують інших операторів (кросінговер, інверсія і т.д.) крім мутації.

## Приклад реалізації

Зазвичай при реалізації ГА до особів, що схрещуються, спочатку застосовують оператор кросінговера, а потім оператор мутації, хоча можливі варіанти. Наприклад, оператор мутації можна застосовувати тільки якщо до даної пари батьківських особей не був застосований оператор кросінговера. У принципі, ніхто не заважає взагалі не використовувати ймовірність проведення даного генетичного оператора і застосовувати кросінговер до усіх відібраних особам, а мутацію до кожному 100 нащадкові.

Нижче приведений приклад підпрограм, що реалізують оператори кросінговера і мутації мовою C++ для популяції, що містить 16-розрядні хромосоми. Розмір популяції: 50 особей. Популяція тут представлена умовно .

```
#include <stdlib.h>
unsigned short MutationMask[] = {0x1, 0x2, 0x4, 0x8,
                                 0x10, 0x20, 0x40, 0x80,
                                 0x100, 0x200, 0x400, 0x800,
                                 0x1000, 0x2000, 0x4000, 0x8000};

unsigned short Inds[50],     // Популяція
               SelInds[50];  // Особи відібрані для схрещування

// оператор 1-точкового кросінговера
void Cross (int p1, p2, c1, c2) {
  unsigned short left, right;
```

```
left =(unsigned short)(15.0 * (double)rand()/(double)RAND_MAX+ 1);
  left = ((unsigned short)0xffff>>left)<<left;
  right = (unsigned short)0xffff ^ left;

  Inds[c1] = (SelInds[p1] & left) | (SelInds[p2] & right);
  Inds[c2] = (SelInds[p2] & left) | (SelInds[p1] & right);
}

// оператор точкової мутації
void Mutate (int j) {
  int i, L = Inds[j].GetChromosomeLength();
  double rnd;

  for (i=0; i&ltL; i++) {
    rnd = (double)rand()/(double)RAND_MAX + 1);
    if (mutationRate > rnd) {  // mutationRate - ймовірність мутації
      Inds [j] = Inds [j] ^ MutationMask [i];
    }
  }
}
```

### Застосування оператора кросінговера для кодування

Припустимо, що $C1=\{c1_1,..,c1_n\}$ і $C2=\{c2_1,..,c2_n\}$ - дві хромосоми до яких застосовується оператор кросінговера. Будемо позначати нащадків як *H1* і *H2*. Нижче коротко описані деякі оператори схрещування для генетичних алгоритмів з кодуванням.

- **Двоточковий кросінговер (Two-point crossover).** Випадковим чином вибираються дві точки розриву *i* і *j* з діапазону [1, *n*-1]. Нехай при цьому *i<j*. Тоді нащадки визначаються шляхом обміну відповідних частин батьківських хромосом: $H1 = \{c1_1, c1_2,.., c2_i, c2_{i+1},.., c2_j, c1_{j+1},.., c1_n\}$, $H2 = \{c2_1, c2_2,.., c1_i, c1_{i+1},.., c1_j, c2_{j+1},.., c2_n\}$.

- **Однорідний кросінговер (Uniform crossover).** Працює як однорідний кросінговер для бінарного кодування з тією лише різницею, що ділянка хромосоми, для якого розігрується ймовірність потрапити до першого або до другого нащадка не окремий розряд, а значення параметра цілком. Тобто розігрується не біт, а весь ген. Іншими словами, для кожного гена, якщо випадкова бінарна змінна *Rnd*=0, то ген від 1-го батька попадає до першого нащадка, а ген другого батька - до другого. Якщо ж *Rnd*=1, то навпаки.

Використовуються також кросінговери:

- **Арифметичний кросінговер (Arithmetical crossover).**
- **Геометричний кросінговер (Geometrical crossover,).**
- **BLX-a кросінговер (BLX-a crossover).**
- **Імітований бінарний кросінговер (Simulated binary crossover).**
- **Нечітка рекомбінація (Fuzzy recombination).**

## Стратегії формування нового покоління

Після схрещування особей необхідно вирішити проблему про те, які з нових особей ввійдуть у наступне покоління, а які - ні, і що робити з їх предками. Є два способи:

1. Нові особи (нащадки) займають місця своїх батьків. Після чого настає наступний етап, у якому нащадки оцінюються, відбираються, дають потомство і поступаються місцем своїм "дітям".
2. Створюється проміжна популяція, що містить у собі як батьків, так і їхніх нащадків. Члени цієї популяції оцінюються, а потім з них вибираються *N* найкращих, котрі і ввійдуть у наступне покоління.

Ті хто знайомий з еволюційними стратегіями, то з цими способами ви вже зустрічалися. Який з цих двох варіантів краще – відповісти однозначно важко. Вочевидь другий варіант

практичніший (так як не дозволяє заміняти пристосованих батьків на "невідомо кого"), але тут може бути більше проблем з передчасною збіжністю, чим у першому варіанті. До того ж він вимагає сортування масиву розміром (як мінімум) 2*N. Узагалі говорячи, можна комбінувати стратегії відбору і формування наступного покоління як завгодно - обмежень немає ніяких.

### *Принцип "елітизма"*

Суть цього принципу полягає в тім, що в нове покоління включаються кращі батьківські особи. Їхнє число може бути від 1 і більше. Використання "елітизму" дозволяє не втратити гарне проміжне рішення, але, у той же час, через це алгоритм може "застрягти" у локальному экстремумі. Однак, досвід використання принципу "елітизму" , дозволяє зробити висновок, що в більшості випадків "елітизм" анітрохи не шкодить пошукові рішення, і головне - це надати алгоритмові можливість аналізувати *різні* рядки з простору пошуку.

У класичному описі генетичного алгоритму мається на увазі створення популяції нащадків і заміщення батьківських особей їх "дітьми". Такий підхід досить гарний, але не особливо ефективний, тому що нащадки, отримані в результаті генетичних перетворень, можуть бути гірші батьківських особей. У результаті з'явилося кілька підходів, "виправляючих" дане явище, які можна об'єднати, використовуючи, як ми казали, поняття "елітизм" (elitism) (іноді "стратегія елітизма" (elitist strategy). Нижче буде приведено короткий опис цих підходів. Відзначимо, що короткий опис елітизма вже було зроблено раніш (див. Стратегії відбору і формування нових нащадків), однак вважаємо за необхідне приділити йому окрему увагу.

Умовно елітизм можна розділити на два класи-підходи, назвемо їх *конкурентним і неконкурентним*. Коротко їхня суть у наступному:

1. *Конкурентний підхід*. Батьківські особи "змагаються" з нащадками і переможці (або переможець) переходять у наступне покоління.

2. *Неконкурентний підхід*. У цьому випадку частина батьківської підпопуляції (випадкова або визначена за заданим правилом) переходить у нове покоління без яких-небудь заперечень з боку електорату.

Іншими словами, у першому класі нащадки після створення мають, загалом, рівні права з батьками на те, щоб перейти в нове покоління і визначальну роль тут грає пристосованість особи, а не її положення на генеалогічному дереві. В другому класі старі індивіди мають визначений пріоритет і навіть якщо всі нащадки будуть краще будь-якого батька, якась частина батьківської підпопуляції неминуче буде присутня у новому поколінні.

Розглянемо кожен клас більш докладно. При цьому будемо по можливості брати до уваги напрацювання в області еволюційних стратегій (evolutionary strategies). Свою історію еволюційні стратегії ведуть з, як мінімум, середини 60-х років і прийняті в них позначення досить зручні для показу різних підходів до елітизму.

**1. Конкурентний підхід**

Виділимо в класі конкурентних підходів 2 підкласи (у дужках приведені жаргонні назви, що рекомендуються для вживання в псевдо-, квазі- і мета-науковій літературі):

- глобальне змагання (масове побоїще, жорстоке і нещадне)
- локальне змагання (бої помісних князьків на кулачках)

При глобальному змаганні спочатку створюються усі нащадки (ключове слово "усі"), які потім змагаються на загальних підставах з усіма батьками (ключове слово "усіма") і ті, хто виявляться кращими, незалежно від віку, переходять у нове покоління. Тобто після створення нащадків визначається їхня пристосованість і, знаючи пристосованість у поточній популяції (тобто в популяції, з якої вибиралися батьківські особи), можна,

відсортувавши особей з поточної популяції і нащадків, сформувати популяцію для наступного покоління. У даному випадку оптиміст скаже, що вибрали кращу частину, а песиміст скаже, що відкинули гіршу частину, і обидва будуть праві, точно також як і у випадку зі склянкою з водою. Нащадки не обов'язково змагаються з усіма особами з поточної популяції, можна улаштувати чемпіонат узявши тільки батьківських особей і їхніх нащадків. Головне в глобальних змаганнях, щоб особи змагалися разом.

Трохи інший підхід використовується в локальних змаганнях. Розглянемо досить розповсюджений випадок, коли дві батьківські особи використовуються для створення двох нащадків. Тоді, безпосередньо після створення, виконується оцінка нащадків, а потім нащадки змагаються тільки зі своїми батьками. Тут проблеми родини вирішуються усередині родини. У такий спосіб популяція нового покоління формується з переможців численних локальних змагань.

З погляду еволюційних стратегій конкурентний підхід відповідає (mu + lambda) стратегії (яка називається "плюс-стратегія" (plus strategy)), де mu -- кількість батьківських особей, а lambda -- кількість створених нащадків.

**2. Неконкурентний підхід**

У неконкурентному підході все значно простіше. Після оцінки поточної популяції вибираються кілька найкращих особей (тобто особей які моють максимальну пристосованість) і вони "автоматично" попадають у наступне покоління. При цьому елітні особи можуть брати участь у схрещуванні нарівні з іншими особами. У силу того, що, таким чином, батьківські особи не змагаються в тій або іншій формі з нащадками, такий підхід і одержав назву неконкурентний. В еволюційних стратегіях неконкурентний підхід відповідає (mu, lambda) стратегії (дослівний переклад виглядає як "кома-стратегія" (comma strategy)), зміст змінних mu і lambda залишається колишнім.

Тут ми проаналізуємо наслідки використання, або не використання того або іншого підходу до елітизму. Що ж тягне використання елітизму? Очевидно, що при елітизмі повільніше обновляється генетична інформація в популяції, так як частина особей, точніше їх геноми, залишаються незмінними при зміні поколінь. І хоча таке можливо і без елітизма (наприклад, схрещування двох "схожих" батьківських особей може привести до створення ідентичних з ними нащадків) у випадку з елітизмом ймовірність цієї події істотно збільшується, насамперед, у силу специфіки самого підходу до елітизму. Добре це чи погано? Спробуємо проаналізувати.

Для цього необхідно зрозуміти, коли (не)вигідне інтенсивне відновлення геномів популяції. Очевидно, що якщо популяція "наближається" до глобального оптимуму (тобто знаходиться на порівняно невеликому від нього видаленні), то різкі зміни особей (більшість з яких мають високу пристосованість) небажані, тому що можуть погіршити відповідні цим особям рішення. Тому, у даному випадку, елітизм, як спосіб збереження "гарних" особей, необхідний. Потрібно відзначити, що описана ситуація, як правило, не спостерігається на початку еволюції, за винятком випадків, коли вирішується дуже проста задача, але для них ГА, по великому рахунку, не потрібні. Якщо ж еволюційний пошук знаходиться на самому початку, то, мабуть, необхідно активно досліджувати простір пошуку, щоб виділити в ньому потенційні області, при влученні в які, популяція буде мати високу пристосованість. Однак, якщо в результаті генетичних перетворень виходить особа, що знаходиться в одній з таких областей (і допустимо має максимальну пристосованість у порівнянні з іншими особами в популяції) то необхідно, щоб характерні для цієї особи генотипічні (і, відповідно, фенотипічні) ознаки закріпилися в популяції, інакше дана область на невизначений час буде "загублена".

***Невеликий відступ**. Генотип особи - це, як правило, її генетична інформація, а фенотип особи - відповідне цієї особи рішення поставленої задачі, тобто, у найпростішому випадку, фенотип - декодований з генетичного представлення набір параметрів, які оптимізуються. Іншими словами, генотип - це*

*ДНК. Зміна генотипу, одержувана в результаті схрещування і мутації, тягне зміну відповідного фенотипу і для випадку прямого кодування тут все очевидно, так як генотип однозначно відповідає фенотипові (і зворотно, для кожного фенотипу можна знайти єдиний генотип). Для кодування Грея відображення генотип-фенотип також однозначне, хоча і не так наочне. Однак при неоднозначній відповідності генотипу і фенотипу можуть виникати проблеми в адекватній оцінці особи.*

Справа в тім, що і пристосована особа може зникнути, наприклад, тому що генератор випадкових чисел видав "невдалу" для цієї особи послідовність, у результаті чого вона або взагалі не прийняла участі в схрещуванні (зокрема, можуть "задавити числом" непристосовані особи), або її нащадки на неї не "схожі", або ж вони сильно змутовані. При використанні елітизма шанси, що генетичні властивості розглянутої (потенційно багатостраждальної) особи збережуться і поширяться в популяції, виглядають обнадійливо. Виходить, що і на початкових етапах елітизм потрібний.

Більш складна картина для неоднозначного відображення генотип-фенотип. Наприклад, коли в генотипі закодована структура штучної нейронної мережі (ШНС). Тоді одній і тій ж особи (структурі ШНС) можуть відповідати різні по характеристиках ШНС у наслідок різних значень ваг зв'язків. У цьому випадку, навіть якщо пристосованості деякої особи високі, це не дає достатніх умов для переваги перед менш пристосованими об’єктами, тому що всі міркування про пристосованість ведуться в умовах неповної інформації. До деякої міри ситуацію можна виправити, якщо для кожної особи досліджувати трохи різних відповідних їй фенотипів (у даному прикладі ШНС із різними вагами, але однаковою структурою). Проте, невизначеність, хоч і менша, залишається. У силу цих міркувань необхідність у елітизмі в даному випадку залишається під сумнівом. Цілком можливо, що краще взагалі заборонити конкуренцію між особами, що мають істотно різні по складу генотипи (яким відповідають структури ШНС, що сильно відрізняються). Зробити це можна, наприклад, за допомогою

"нішінга (niching)". У будь-якому випадку, не варто забувати про "формулу успіху" у генетичних алгоритмах, яку можна сформулювати в такий спосіб: "*Необхідно дотримувати баланс між дослідженням простору пошуку (exploration) і використанням знайдених гарних рішень (exploitation)".* Тобто якщо використовується елітизм, іншими словами, більш активно використовуються вже знайдені гарні особи, то необхідні міри, що будуть компенсувати вплив елітизма, наприклад, збільшення ймовірності мутації, або збільшення розміру популяції, або заборона на існування "особей-близнюків" у популяції. У противному випадку істотно збільшується ризик передчасної збіжності. Використання елітизма, у цілому, дозволяє істотно поліпшити результати роботи ГА. Проте, використовувати елітизм необхідно з визначеною обережністю, щоб уникнути передчасної збіжності. У результаті класифікації відомих методів елітизма пропонується виділяти наступні підходи:

1.1. Конкурентний підхід з локальним змаганням.

1.2. Конкурентний підхід із глобальним змаганням.

1.3. Неконкурентний підхід.

## 3.8. Приклад виконання класичного генетичного алгоритму

Розглянемо виконання описаного раніше класичного генетичного алгоритму на як можна більш простому прикладі. Простежимо послідовність виконання його етапів, що відповідають блок-схемі на рис. 3.12.

**Приклад 3.4**

Розглянемо сильно спрощений і досить штучний приклад, що складається із знаходження хромосоми з максимальною кількістю одиниць. Припустимо, що хромосоми складаються з 12 генів, а популяція нараховує 8 хромосом. Зрозуміло, що найкращою буде хромосома, що складається з 12 одиниць. Подивимося, як протікає процес рішення цієї досить тривіальної задачі за допомогою генетичного алгоритму. Змістовну

інтерпретацію поставленої в такий спосіб задачі можна знайти, зокрема, у прикладі 3.29.

**Ініціалізація, або вибір вихідної популяції хромосом.** Необхідно випадковим чином згенерувати 8 двоїчних послідовностей довжиною 12 бітів. Це можна досягти, наприклад, підкиданням монети (96 разів, при випаданні «орла» приписується значення 1, а у випадку «решки» - 0). У такий спосіб можна сформувати вихідну популяцію

$ch_1$, = [111001100101]
$ch_2$ = [001100111010]
$ch_3$ = [011101110011]
$ch_4$ = [001000101000]
$ch_5$ = [010001100100]
$ch_6$ = [010011000101]
$ch_7$ = [101011011011]
$ch_8$ = [000010111100]

**Оцінка пристосованості хромосом у популяції.** У розглянутому спрощеному прикладі вирішується задача перебування такої хромосоми, яка містить найбільшу кількість одиниць. Тому функція пристосованості визначає кількість одиниць у хромосомі. Позначимо функцію пристосованості символом *F.* Тоді її значення для кожної хромосоми з вихідної популяції будуть такі:

$F(ch_1) = 7$
$F(ch_2) = 6$
$F(ch_3) = 8$
$F(ch_4) = 3$
$F(ch_5) = 4$
$F(ch_6) = 5$
$F(ch_7) = 8$
$F(ch_8) = 5$

Хромосоми $ch_3$ і $ch_7$ характеризуються найбільшими значеннями функції приналежності. У цій популяції вони вважаються найкращими кандидатами на рішення задачі. Якщо відповідно до блок-схеми генетичного алгоритму (рис. 3.12) умова зупинки алгоритму не виконується, то на наступному кроці виконується селекція хромосом з поточної популяції.

**Селекція хромосом.** Селекція виконується методом рулетки. На підставі формул (3.2) і (3.3) для кожної з 8 хромосом поточної популяції (у нашому випадку - вихідної популяції, для якої $N = 8$)

одержуємо сектори колеса рулетки, виражені у відсотках (рис. 3.16)

$v(ch_1)= 15,22$
$v(ch_2) = 13,04$
$v(ch_3)= 17,39$
$v(ch_4)= 6,52$
$v(ch_5)= 8,70$
$v(ch_6) = 10,87$
$v(ch_7)= 17,39$
$v(ch_8)= 10,87$

Розігриш за допомогою колеса рулетки зводиться до випадкового вибору числа з інтервалу [0, 100], що вказує на відповідний сектор на колесі, тобто на конкретну хромосому. Припустимо, що розіграно наступних 8 чисел:

79 44 9 74 44 86 48 23

Це означає вибір хромосом

$ch_7$ $ch_3$ $ch_1$ $ch_7$ $ch_3$ $ch_7$ $ch_4$ $ch_2$

Як видно, хромосома $ch_7$ була обрана тричі, а хромосома $ch_3$ - двічі. Помітимо, що саме ці хромосоми мають найбільше значення функції пристосованості. Однак обрана і хромосома $ch_4$ з найменшим значенням функції пристосованості. Всі обрані в такий спосіб хромосоми включаються в так називаний батьківський пул.

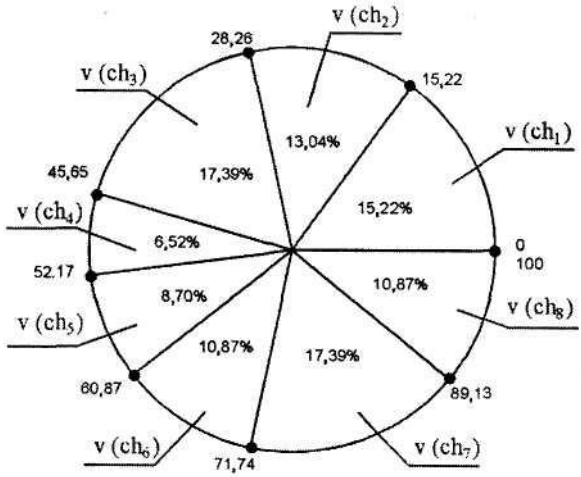


Рис. 3.16. Колесо рулетки для селекції в прикладі 3.4.

**Застосування генетичних операторів.** Припустимо, що жодна з відібраних у процесі селекції хромосом не піддається мутації, і усі вони складають популяцію хромосом, призначених для схрещування. Це означає, що ймовірність схрещування $p_c$=1, а ймовірність мутації $p_m$=0. Припустимо, що з цих хромосом випадковим чином сформовані пари батьків

$$ch_2 \text{ і } ch_7 \quad ch_1 \text{ і } ch_7 \quad ch_3 \text{ і } ch_4 \quad ch_3 \text{ і } ch_7$$

Для першої пари випадковим чином обрана точка схрещування $k$ = 4, для другої $k$ = 3, для третьої $k$ = 11, для четвертої $k$ = 5. При цьому процес схрещування протікає так, як показано на рис. 3.17. У результаті виконання оператора схрещування виходять 4 пари нащадків.

Як би при випадковому підборі пар хромосом для схрещування були об'єднані, наприклад, $ch_3$ з $ch_3$ і $ch_4$ з $ch_7$ замість $ch_3$ з $ch_4$ і $ch_3$ з $ch_7$, а інші пари залишилися без зміни, то схрещування $ch_3$ з $ch_3$ дало б дві такі ж хромосоми незалежно від розіграної точки схрещування. Це означало б одержання двох нащадків, ідентичних своїм батькам. Помітимо, що така ситуація найбільш ймовірна для хромосом з найбільшим значенням функції пристосованості, тобто саме такі хромосоми одержують найбільші шанси на перехід у нову популяцію.

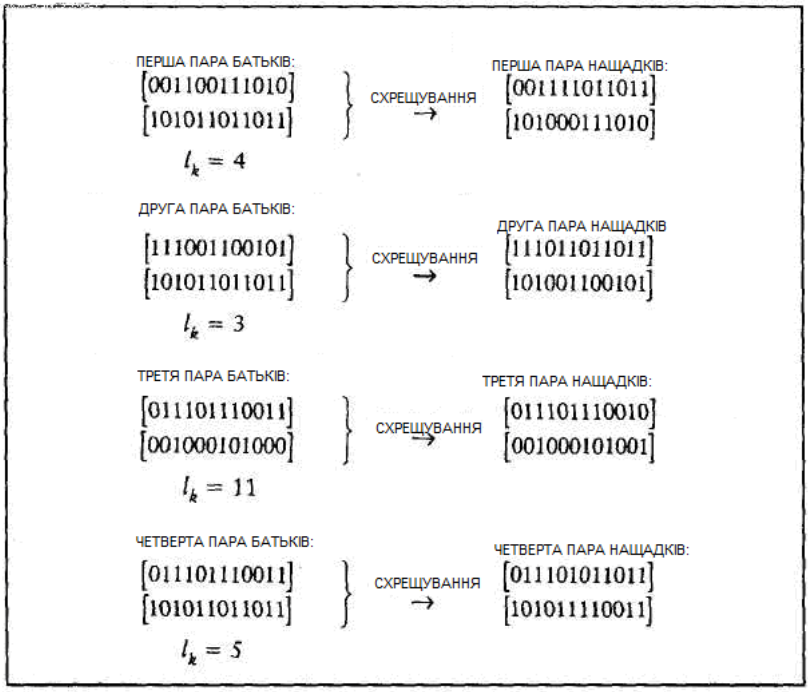



Рис. 3.17. Процес схрещування хромосом у прикладі 3.4.

**Формування нової популяції.** Після виконання операції схрещування ми одержуємо (згідно рис. 3.17) наступну популяцію нащадків:

$Ch_1$ = [001111011011]
$Ch_2$ = [101000111010]
$Ch_3$ = [111011011011]
$Ch_4$ = [101001100101]
$Ch_5$ = [011101110010]
$Ch_6$ = [001000101001]
$Ch_7$ = [011101011011]
$Ch_8$ = [101011110011]

Для відмінності від хромосом попередньої популяції позначення знову сформованих хромосом починаються з заголовної букви С.

Відповідно до блок-схеми генетичного алгоритму (рис. 3.12) виконується повернення до другого етапу, тобто до оцінки пристосованості хромосом зі знову сформованої популяції, що стає поточною. Значення функцій пристосованості хромосом цієї популяції складають

$F(Ch_1) = 8$
$F(Ch_2) = 6$
$F(Ch_3) = 9$
$F(Ch_4) = 6$
$F(Ch_5) = 7$
$F(Ch_6) = 4$
$F(Ch_7) = 8$
$F(Ch_8) = 8$

## 3.9. Представлення даних у генах

Для рішення деякої задачі за допомогою ГА її дані необхідно представити у виді генів особи. Для цього насамперед необхідно визначити які параметри задачі необхідно настроїти. Наприклад, якщо ми намагаємося апроксимувати за допомогою ГА набір точок функцією виду *f(x)=A*exp(k*x),* де *A, k* - константи, *x* - незалежна змінна, то як параметри будуть виступати *A* і *k*, тому що від їхнього значення залежить вид і поводження функції. Отже, маємо два параметри і, отже, два гени.

Наступний крок - це вибір числа розрядів у кожнім гені. Тут необхідно враховувати, що з одного боку, чим більше розрядів, тим краще, тому що вище точність і т.д., але з іншого боку - велике число розрядів тягне збільшення часу пошуку рішення (збіжності). Варто відзначити, що з ростом числа параметрів (у деяких задачах вони обчислюються сотнями) "зайві" розряди позначаються усе сильніше і сильніше. Тому необхідно знайти компроміс між точністю і швидкістю. Візьмемо для задачі апроксимації 16-розрядні гени, при цьому параметри будуть змінюватися від -32,768 до 32,767 із кроком у 0,001. Дані числа отримані, виходячи з того, що 16-розрядне число може приймати одне з $2^{16}$= 65536 значень від 0 до 65535. Якщо припустити, що 0 буде в середині цього інтервалу, причому кожне значення

розділиться потім на 1000, то одержимо інтервал зміни параметрів і крок зміни. Після того, як обрано параметри, їхнє число і розрядність, необхідно вирішити, як безпосередньо записувати дані. Можна використовувати звичайне кодування, коли $1011=1011_2=11_{10}$, або коди Грея, коли $1011=1110_2=14_{10}$. Незважаючи на те, що коди Грея тягнуть неминуче кодування/декодування даних, вони дозволяють уникнути деяких проблем, що з'являються в результаті звичайного кодування. Можна лише сказати, що перевага коду Грея в тім, що якщо два числа розрізняються на 1, то і їхні двоїчні коди розрізняються тільки на один розряд, а в двоїчних кодах не все так просто. Варто відзначити, що кодувати і декодувати у коди Грея досить зручно, описати це можна так: спочатку копіюється самий старший розряд, потім:

З двоїчного коду в код Грея: G[*i*] = XOR(B[*i*+1], B[*i*])
З коду Грея в двоїчний код: B[*i*] = XOR(B[*i*+1], G[*i*])

Тут, G[*i*] - *i*-й розряд коду Грея, а B[*i*] - *i*-й розряд бінарного коду. Наприклад, послідовність чисел від 0 до 7 у двоїчном коді: {000, 001, 010, 011, 100, 101, 110, 111}, а в кодах Грея: {000, 001, 011, 010, 110, 111, 101, 100}.

Отже, тепер задано всі необхідні характеристики генів, залишається тільки представити все це на якій-небудь мові програмування. Наприклад, на C++ одна особа із хромосомою з двох 16-розрядних генів може "виглядати" так:

```
short int Individual [2];
```

```
1.    class Individual {
2.      short int Genes [2];
3.    };
4.  unsigned char Individual [32];
```

Останній приклад на перший погляд здається як мінімум дивним: навіщо для завдання одного біта використовувати цілий байт? Однак бувають випадки, коли використовуються небінарні алфавіти, тобто кожен розряд може приймати не 2 різні значення

"0" або "1", а більше, наприклад, "1", "2", "4", "8". Саме для таких ситуацій останній приклад і зручний. Надалі мова буде йти тільки про бінарний алфавіт. Крім того, в останньому способі легше використовувати коди Грея. Для того щоб задати популяцію з 50 особей, можна поступити одним з наступних способів:

```
1. short int Population[50][2];

2.     class Population {
3.       Individual Inds[50];
4.     };
5. unsigned char Population[50][32];
```

Слід також зазначити, що в задачі апроксимації, узятої як приклад, у процесі "еволюції" будуть приймати участь усі гени особи, тобто генетичні оператори будуть застосовуватися до кожного гена. Однак, це не завжди необхідно, наприклад, розглянемо задачу компонування, тобто коли на деякій відомій поверхні потрібно розмістити *N* елементів різної площі так, щоб вони не накладалися один на одний. Для простоти будемо вважати, що всі елементи мають прямокутну форму. Параметрами, що настроюються (оптимізуються), будуть координати кожного елемента, тобто кожна хромосома буде містити два гени, що відповідають координатам лівого верхнього кута елемента. Але крім цього необхідно знати, який елемент представляє дана особу. Для цього необхідно додати в набір генів особи ще один ген, що містить номер елемента. При цьому нам потрібно зберегти значення цього гена незмінним на протязі всього процесу пошуку рішення. Таким чином, одержуємо особу із трьома генами, два з яких обробляються генетичними операторами, а один ні.

Крім розглянутих задач апроксимації і компонування можна привести варіанти кодування для деяких інших задач:

- Оптимізація функцій: гени - незалежні змінні;
- Настроювання ваг штучної нейронної мережі: гени - синаптичні ваги нейронів;

- Штучне життя (Artificial Life): гени - характеристики особи (сила, швидкість, і т.д.), також повинні бути незмінні гени, що позначають тип особи (рослина або тварина);
- Задача про найкоротший шлях: гени - пункти пересування. Уся хромосома цілком представляє із себе маршрут з початкової точки в кінцеву, причому не завжди існуючий.

І останнє, розрядність генів необов'язково повинна бути фіксованою, багато сучасних алгоритмів самостійно підбудовують число розрядів для кожного гена в залежності від проміжних результатів пошуку рішення.

## 3.10. Приклади кодування параметрів задачі в генетичному алгоритмі

Вибір вихідної популяції зв'язаний із представленням параметрів задачі у формі хромосом, тобто з так називаним хромосомним представленням. Це представлення визначається способом кодування. У класичному генетичному алгоритмі застосовується двоїчне кодування, тобто аллели всіх генів у хромосомі рівні 0 або 1. Довжина хромосом залежить від умов задачі, точніше кажучи - від кількості точок у просторі пошуку.

Генетичні алгоритми знаходять застосування головним чином у задачах оптимізації. Приклад 3.5 демонструє виконання класичного генетичного алгоритму, аналогічного розглянутому в прикладі 3.4, але для випадку оптимізації функції. Для простоти приймемо, що це функція однієї змінної. У новому прикладі хромосоми виступають у ролі закодованої форми відповідних фенотипів, а оптимізується сама функція пристосованості. У прикладі 3.6 оптимізується та ж функція, однак увага читача акцентується на іншому способі кодування хромосом для іншої області визначення змінної *x*.

**Приклад 3.5**

Розглянемо дуже простий приклад - задачу знаходження максимуму функції, заданої виразом (3.1) для цілочислової змінної *x*, що приймає значення від 0 до 31.

Для застосування генетичного алгоритму необхідно насамперед закодувати значення змінної *x* у виді двоїчних

послідовностей. Очевидно, що цілі числа з інтервалу [0, 31] можна представити послідовностями нулів і одиниць, використовуючи їхнє представлення в двоїчній системі числення. Число 0 при цьому записується як 00000, а число 31 - як 11111. У даному випадку хромосоми здобувають вид двоїчних послідовностей, що складаються з 5 бітів, тобто ланцюжками довжиною 5 (аналогічно прикладові 3.1).

Також очевидно, що в ролі функції пристосованості буде виступати цільова функція *f(x),* задана виразом (3.1). Тоді пристосованість хромосоми $ch_i$, $i$ = 1,2.....N, буде визначатися значенням функції *f(x)* для *x*, рівного фенотипові, що відповідає генотипові $ch_i$. Позначимо ці фенотипи $ch_i^*$. У такому випадку значення функції пристосованості хромосоми $ch_i$ (тобто $F(ch_i)$) буде дорівнювати $f(ch_i^*)$. Виберемо випадковим чином вихідну популяцію, що складається з 6 кодових послідовностей (наприклад, можна 30 разів підкинути монету); при цьому $N$=6. Припустимо, що обрано хромосоми

$ch_1$ = [10011]
$ch_2$ = [00011]
$ch_3$ = [00111]
$ch_4$ = [10101]
$ch_5$ = [01000]
$ch_6$ = [11101]

Відповідні ним фенотипи - це представлені нижче числа з інтервалу від 0 до 31:

$ch_1^*$ = 19
$ch_2^*$ =3
$ch_3^*$= 7
$ch_4^*$ = 21
$ch_5^*$, = 8
$ch_6^*$ = 29

По формулі (3.1) розраховуємо значення функції пристосованості для кожної хромосоми в популяції й одержуємо

$F(ch_1)$ = 723
$F(ch_2)$= 19
$F(ch_3)$ = 99
$F(ch_4)$ = 883
$F(ch_5)$ = 129
$F(ch_6)$=1683

**Селекція хромосом.** Методом рулетки (також, як і в прикладі 3.4), вибираємо 6 хромосом для репродукції. Колесо рулетки представлено на рис. 3.18.

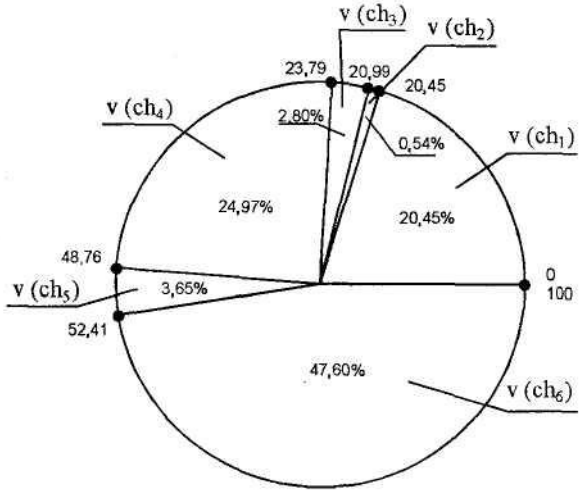



Рис. 3.18. Колесо рулетки для селекції в прикладі 3.5.

Припустимо, що обрано числа

97 26 54 13 31 88.

Це означає вибір хромосом

ch6 $ch_4$ $ch_6$ $ch_1$ $ch_4$ $ch_6$

Нехай схрещування виконується з ймовірністю $p_c$=1. Припустимо, що для схрещування сформовані пари

$ch_1$ і $ch_4$ $ch_4$ і $ch_6$ $ch_6$ і $ch_6$

Крім того, приустимо, що випадковим чином обрана точка схрещування, рівна 3 для хромосом $ch_1$, і $ch_4$, а також точка схрещування, рівна 2 для хромосом $ch_4$ і $ch_6$ (рис.3.19).

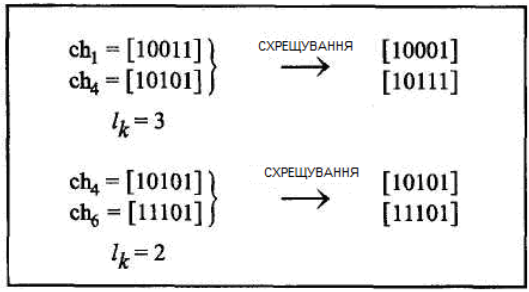



Рис. 3.19. Процес схрещування хромосом у прикладі 3.5.

За умови, що ймовірність мутації $p_m$=0, у нову популяцію включаються хромосоми

$Ch_1$ = [10001]
$Ch_2$ = [10111]
$Ch_3$ = [10101]
$Ch_4$ = [11101]
$Ch_5$ = [11101]
$Ch_6$ = [11101]

Для розрахунку значень функції пристосованості цих хромосом необхідно декодувати їхні двоїчні послідовності, що представляють, і одержати відповідні їм фенотипи. Позначимо їх $Ch_i$*. У результаті декодування одержуємо числа (з інтервалу від 0 до 31)

$Ch_1$*= 17
$Ch_2$* = 23
$Ch_3$* = 21
$Ch_4$* = 29
$Ch_5$* = 29
$Ch_6$* = 29

Відповідно, значення функції пристосованості хромосом нової популяції, розраховані по формулі (3.1), складуть

F($Ch_1$)= 579
F($Ch_2$)=1059
F($Ch_3$)= 883
F($Ch_4$)=1683
F($Ch_5$) = 1683
F($Ch_6$)=1683

Легко помітити, що в цьому випадку середнє значення пристосованості зросло з 589 до 1262.

Звертаємо увагу, що якщо на наступній ітерації будуть сформовані для схрещування пари хромосом, наприклад, $Ch_4$ і $Ch_2$, $Ch_5$ і $Ch_2$ або $Ch_6$ і $Ch_2$ із точкою схрещування 2 або 3, то серед інших буде отримана хромосома [11111] з фенотипом, рівним числу 31, при якому оптимізуєма функція досягає свого максимуму. Значення функції пристосованості для цієї хромосоми виявляється найбільшим і складає 1923. Якщо таке сполучення пар у даній ітерації не відбудеться, то можна буде очікувати утворення хромосоми з найбільшим значенням функції пристосованості на наступних ітераціях. Хромосома [11111] могла бути отримана і на поточній ітерації у випадку формування для схрещування пари $Ch_1$ і $Ch_6$ із точкою схрещування 3.

Відзначимо, що при довжині хромосом, рівній 5 бітам, простір пошуку дуже малий і нараховує всього $2^5=32$ точки. Представлений приклад має винятково демонстраційний характер. Застосування генетичного алгоритму для такого простого прикладу практично недоцільне, оскільки його оптимальне рішення може бути отримане миттєво. Однак цей приклад придатний для вивчення функціонування генетичного алгоритму.

Також варто згадати, що в малих популяціях часто зустрічаються ситуації, коли на початковому етапі кілька особей мають значно більші значення функції приналежності, чим інші особи даної популяції. Застосування методу селекції на основі «колеса рулетки» дозволяє в цьому випадку дуже швидко вибрати «найкращі» особи, іноді - протягом «життя» одного покоління. Однак такий розвиток подій вважається небажаним, оскільки він стає головною причиною передчасної збіжності алгоритму, називаною збіжністю до неоптимального рішення. З цієї причини використовуються й інші методи селекції, що відрізняються від колеса рулетки, або застосовується масштабування функції пристосованості.

**Приклад 3.6**

Розглянемо задачу, аналогічну задачі з приклада 3.5, тобто будемо шукати максимум функції, заданою формулою (3.1), але для змінної *x*, що приймає дійсні значення з інтервалу [*a, b*], де

$a$ = 0, $b$ = 3,1. Припустимо, що нас цікавить рішення з точністю до одного знака після коми.

Пошук рішення зводиться до перегляду простору, що складається з 32 точок 0,0;0,1;...;2,9;3,0;3,1. Ці точки (фенотипи) можна представити у виді хромосом (генотипів), якщо використовувати бінарні п'ятиланкові ланцюжки, оскільки за допомогою 5 бітів можна одержати $2^5$=32 різних кодові комбінації. Отже, можна використовувати таку ж множину кодових послідовностей, як і в прикладі 3.5, причому хромосома [00000] буде відповідати числу 0,0, хромосома [00001] - числу 0,1 і т.д., аж до хромосоми [11111], що відповідає числу 3,1.

Таким чином, ми можемо відтворити послідовність етапів генетичного алгоритму (так само, як у прикладі 3.5), не забуваючи, що конкретним хромосомам (генотипам) у даному прикладі відповідають інші фенотипи. Ті кодові послідовності, що у прикладі 3.5 представляли фенотипи 0, 1, ..., 29, 30, 31, у розглянутій ситуації позначають значення $x$, рівні 0,0; 0,1;...;2,9; 3,0; 3,1. У зв'язку з тим, що генетичний алгоритм заснований на випадковому виборі вихідної популяції і хромосом для наступного перетворення методом колеса рулетки, а також батьківських пар для схрещування і точки схрещування, то генетичний алгоритм у поточному прикладі буде виконуватися аналогічно, але не ідентично попередньому прикладові.

У результаті виконання цього алгоритму буде обрано найкраще рішення, що представляється хромосомою [11111] зі значенням фенотипу 3,1. Функція пристосованості цієї хромосоми дорівнює 20,22; це максимально можливе значення.
Помітимо, що якби в прикладі 3.6 нас цікавило рішення з точністю, що перевищує один знак після коми, то інтервал [0; 3,1] необхідно було б розбити на більшу кількість підінтервалів, і для кодування відповідно більшої кількості чисел потрібні були більш довгі хромосоми (з довжиною, що перевищує 5 бітів). Аналогічно, розширення області визначення змінної $x$ також зажадає застосування більш довгих хромосом. З цих спостережень можна зробити висновок, що довжина хромосом залежить від ширини області визначення $x$ і від необхідної точності рішення.

Представимо тепер задачу з приклада 3.6 у більш загальному виді. Припустимо, що шукається максимум функції $f(x_1,x_2.....x_n)>0$ для $x_i \in [a_i, b_i] \subset R$; $i$ = 1, 2, ..., $n$ і потрібно знайти рішення з точністю до $q$ знаків після коми для кожної змінної $x_i$. У такій

ситуації необхідно розбити інтервал [$a_i$, $b_i$] на ($bi$-$ai$)•$10^q$ однакових підінтервалів. Це означає застосування дискретизації з кроком $r$= $10^{-q}$. Найменше натуральне число $m_i$ що задовольняє нерівності

$$(bi\text{-}ai)\bullet 10^q \le 2^{m_i} - 1 , \qquad (3.4)$$

визначає необхідну і достатню довжину двоїчної послідовності, необхідної для кодування числа з інтервалу [$a_i$, $b_i$] із кроком $r$. Кожної такої двоїчної послідовності відповідає десяткове значення числа, що представляється даним кодом (з урахуванням правил перекладу десяткових чисел у двоїчну форму). Нехай $y_i$ - позначає десяткове значення двоїчної послідовності, що кодує число $x_i$. Значення $x_i$ можна представити виразом

$$x_i = a_i + y_i \frac{b_i - a_i}{2^{m_i} - 1} . \qquad (3.5)$$

Таким способом задаються фенотипи, що відповідають кодовим послідовностям з довжиною $m_i$. Приклад 3.6 - це окремий випадок задачі в даній постановці за умови, що $i$= 1 і $q$=1. Вираз (3.5) - це наслідок із простого лінійного відображення інтервалу [$a_i$, $b_i$] на інтервал [0, $2^{m_i}$ -1], де $2^{m_i}$ - десяткове число, закодоване двоїчною послідовністю довжиною $m_i$, і складене винятково з одиниць, а 0 - це, мабуть, десяткове значення двоїчної послідовності довжиною $m_i$, складеної тільки з нулів. Звертаємо увагу, що якщо $a_i$ = - 25, $b_i$ = 25 і застосовується крок $r$=0,05, то відповідно до формули (3.4) одержуємо $m_i$=10, а за допомогою формули (3.5) можна перевірити значення фенотипів для генотипів, представлених у табл. 3.1.

## 3.11. Основна теорема про генетичні алгоритми

Для того щоб краще зрозуміти функціонування генетичного алгоритму, будемо використовувати поняття *схема* і сформулюємо основну теорему, що відноситься до генетичних алгоритмів і називається теоремою про схеми. Поняття схема було введено Холландом і використовується для аналізу роботи ГА. Зокрема розглядаються процеси конструювання і руйнування

визначеної схеми протягом розвитку популяції (schema dynamics).

*Схемою* називається рядок виду

$$(a_1, a_2, ..., a_i, ..., a_l),\ a_i \in \{0, 1, *\}.$$

Символом "*" у деякому розряді позначається те, що там може бути як 1, так і 0. Наприклад, для двох бінарних рядків "111000111000" і "110011001100" схема буде виглядати в такий спосіб: "11*0****1*00". Тобто за допомогою схем можна як би виділяти загальні ділянки двоїчних рядків і маскувати розходження. Маючи в складі схем *m* символів "*" можна закодувати (узагальнити) $2^m$ двоїчних рядків. Так, наприклад, схема "01*0*1" описує набір рядків

{"010001", "010011", "011001", "011011"}.

*Визначальною довжиною схеми (schema defining length)* називається відстань між двома крайніми символами "0" і/або "1". Для схеми "01*0*1" визначальна довжина дорівнює 5, а для схеми "**0**1*" визначальна довжина дорівнює 3. *Порядок схеми (schema order)* - це ще одна характеристика схеми і дорівнює вона числу фіксованих позицій у рядку, тобто загальному числу "0" і "1". Для схем "01*0*1" і "**0**1*" порядки рівні 4 і 2 відповідно.

Тепер про те, яке відношення схема має до генетичних алгоритмів. Справа в тім, що хромосома, по суті, є двоїчним рядком. У той же час особі, якій належить хромосома, що містить набір генів-параметрів задачі, поставлена у відповідність величина, що характеризує її пристосованість. Так як схема є узагальненням декількох бінарних рядків (хромосом), то можна говорити, що особи, які володіють хромосомами, що відповідають одній схемі більш пристосовані, а особи з хромосомами, які відповідають іншій схемі - менш пристосовані.

Можна сказати, що зміст роботи ГА полягає в пошуку двоїчного рядка визначеного виду з усієї множині бінарних рядків. Простір пошуку складає $2^L$ рядків, а його мірність дорівнює $L$ ($L$-мірний простір), де $L$ - довжина хромосоми. Схема відповідає деякій гіперплощині в цьому просторі. Дане твердження можна проілюструвати в такий спосіб. Нехай розрядність хромосоми дорівнює 3, тоді усього можна закодувати $2^3$=8 рядків. Представимо куб у 3-мірному просторі. Позначимо вершини цього куба 3-розрядними бінарними рядками так, щоб мітки сусідніх вершин відрізнялися рівно на один розряд, причому вершина з міткою "000" знаходилася б на початку координат. Варіант позначення зображено на рис. 3.20.

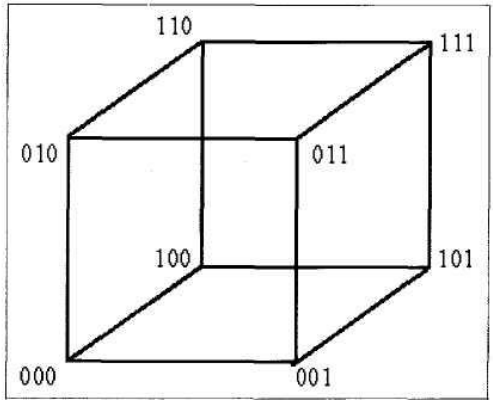



Рис. 3.20. 3-мірний куб

Якщо взяти схему виду "**0", то вона опише ліву грань куба, а схема "*10" - верхнє ребро цієї грані. Очевидно, що схема "***" відповідає всьому просторові. Якщо взяти двоїчні рядки довжиною 4 розряди, то розбивка простору схемами можна зобразити на прикладі 4-мірного куба з пойменованими вершинами (рис. 3.21).

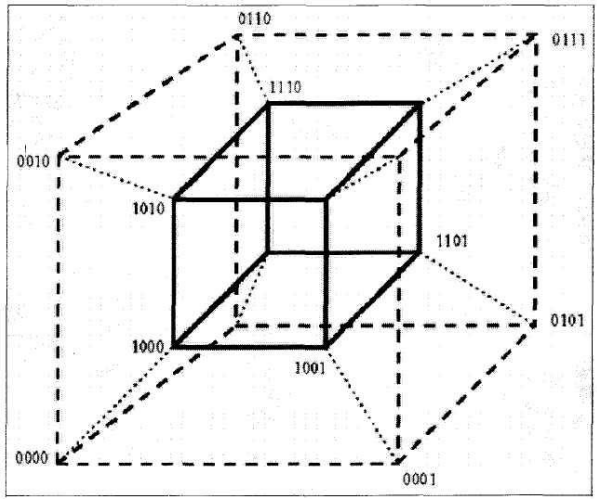



Рис. 3.21. 4-мірний куб

Тут схемі "*1**" відповідає гіперплощина, що включає задні грані зовнішнього і внутрішнього куба, а схемі "**10" - гіперплощина з верхніми ребрами лівих граней обох кубів. Розбивка простору пошуку можна представити і по іншому. Представимо координатну площину, у якій по одній осі ми будемо відкладати значення двоїчних рядків, а по іншійій - значення цільової функції (рис. 3.22).

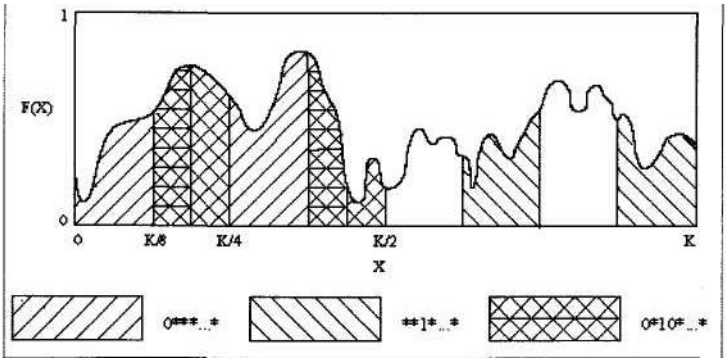


Рис. 3.22. Розбивка простору

Ділянки простору, які заштриховані різним стилем, відповідають різним схемам. Число *K* в правої частини горизонтальної осі відповідає максимальному значенню бінарного рядка - "111...111". З рисунка видно, що схема "0***...*" покриває всю ліву половину відрізка, схема "**1*...*" - 4 ділянки шириною в одну восьму частину, а схема "0*10*...*" - ліві половини ділянок, що знаходяться на перетинні перших двох схем. У такий спосіб і відбувається розбивка простору в цьому випадку. Як ми вже говорили, поняття *схема* було введено для визначення множини хромосом, що володіють деякими загальними властивостями, тобто подібних одна одній. Якщо аллели приймають значення 0 або 1 (розглядаються хромосоми з двоїчним алфавітом), то схема являє собою множину хромосом, що містять нулі й одиниці на деяких заздалегідь визначених позиціях. При розгляді схем зручно використовувати розширений алфавіт {0, 1, *}, у який крім 0 і 1 уведено додатковий символ *, що позначає будь-яке припустиме значення, тобто 0 або 1; символ * у конкретній позиції означає «усе рівно» *(don't care).* Наприклад,

10*1 ={1001, 1011}

*01*10 = {001010, 001110, 101010,101110}

Вважається, що *хромосома належить до даної схеми,* якщо для кожної *j*-ї позиції (локусу), $j = 1, 2, ..., L$, де $L$ - довжина хромосоми; символ, що займає *j*-у позицію хромосоми, відповідає символові, що займає *j*-у позицію схеми, причому символові * відповідають як 0, так і 1. Те ж саме означають твердження *хромосома відповідає схемі* і *хромосома представляє схему.* Відзначимо, що якщо в схемі присутньо *m* символів *, то ця схема містить $2^m$ хромосом. Крім того, кожна хромосома (ланцюжок) довжиною $L$ належить до $2^L$ схемам. У таблицях 3.2 і 3.3 представлені схеми, до яких належать ланцюжка довжиною 2 і 3 відповідно.

**Таблиця 3.2**. Схеми, до яких належать ланцюжки довжиною 2

| ЛАНЦЮГИ | СХЕМИ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 00 | ** | *0 | 0* | 00 |
| 01 | ** | *1 | 0* | 01 |
| 10 | ** | *0 | 1* | 10 |
| 11 | ** | *1 | 1* | 11 |

**Таблиця 3.3**. Схеми, до яких належать ланцюжки довжиною 3

| ЛАНЦЮГИ | СХЕМИ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 000 | *** | **0 | *0* | 0** | *00 | 0*0 | 00* | 000 |
| 001 | *** | **1 | *0* | 0** | *01 | 0*1 | 00* | 001 |
| 010 | *** | **0 | *1* | 0** | *10 | 0*0 | 01* | 010 |
| 011 | *** | **1 | *1* | 0** | *11 | 0*1 | 01* | 011 |
| 100 | *** | **0 | *0* | 1** | *00 | 1*0 | 10* | 100 |
| 101 | *** | **1 | *0* | 1** | *01 | 1*1 | 10* | 101 |
| 110 | *** | **0 | *1* | 1** | *10 | 1*0 | 11* | 110 |
| 111 | *** | **1 | *1* | 1** | *11 | 1*1 | 11* | 111 |

Ланцюжки довжиною 2 відповідають чотирьом різним схемам, а ланцюжки довжиною 3 - вісьмом схемам.

Генетичний алгоритм базується на принципі трансформації найбільш пристосованих особей (хромосом). Нехай $P(0)$ означає вихідну популяцію особей, а $P(k)$ - поточну популяцію (на $k$-й ітерації алгоритму). З кожної популяції $P(k)$, $k$ = 0, 1, ... методом селекції вибираються хромосоми з найбільшою пристосованістю, які включаються в так званий батьківський пул *(mating pool)* $M(k)$. Далі, у результаті об'єднання особей з популяції $M(k)$ у батьківські пари і виконання операції схрещування з ймовірністю $p_c$, а також операції мутації з ймовірністю $p_m$ формується нова популяція $P(k+1)$, у яку входять нащадки особей з популяції $M(k)$. Отже, для будь-якої схеми, що представляє гарне рішення, було б бажаним, щоб кількість хромосом, що відповідають цій схемі, зростало зі збільшенням кількості ітерацій $k$.

На відповідне перетворення схем у генетичному алгоритмі впливають 3 фактори: *селекція хромосом, схрещування і*

*мутація.* Проаналізуємо вплив кожного з них з погляду збільшення очікуваної кількості представників окремо узятої схеми.

Позначимо розглянуту схему символом *S*, а кількість хромосом популяції *P(k),* що відповідають цій схемі - *c(S, k).* Отже, *c(S, k)* можна вважати кількістю елементів (тобто потужністю) множини $P(k) \cap S$.

Почнемо з дослідження *впливу селекції.* При виконанні цієї операції хромосоми з популяції *P(k)* копіюються в батьківський пул *M(k)* з ймовірністю, обумовленою виразом (3.3). Нехай *F(S, k)* позначає середнє значення функції пристосованості хромосом з популяції *P(k),* що відповідають схемі *S*. Якщо

$$P(k)S = \{ch_1.....ch_{c(S,k)}\},$$

то

$$c(S,k)=\frac{\sum_{i=1}^{c(S,k)} F(ch_i)}{c(S,k)} \qquad (3.6)$$

Величина *F(S, k)* також називається пристосованістю схеми *S* на *k*-й ітерації.

Нехай $\Im(k)$ позначає суму значень функцій пристосованості хромосом з популяції *P(k)* потужністю *N*, тобто

$$\Im(k) = \sum_{i=1}^{N} F(ch_i^{(k)}) \qquad (3.7)$$

Позначимо через $\overline{F}(k)$ середнє значення функції пристосованості хромосом цієї популяції, тобто

$$\overline{F}(k) = \frac{1}{N}\,\Im(k). \qquad (3.8)$$

Нехай $ch_i^{(k)}$ позначає елемент батьківського пула *M(k).* Для кожного $ch_r^{(k)} \in M(k)$ і для кожного $i = 1,..., c(S, к)$ ймовірність того, що $ch_r^{(k)}=ch_i$ визначається відношенням $F(ch_i)IF(k).$ Тому очікувана кількість хромосом у популяції *M(k),* що рівні $ch_i$, складе

$$N\frac{F(ch_i)}{\Im(k)} = \frac{F(ch_i)}{\overline{F}(k)}$$

Таким чином, очікувана кількість хромосом з множини

$P(k) \cap S$, відібраних для включення в батьківський пул $M(k)$, буде дорівнювати

$$\sum_{i=1}^{c(S,k)} \frac{F(ch_i)}{\overline{F}(k)} = c(S,k) \frac{F(S,k)}{\overline{F}(k)},$$

що випливає з виразу (3.6).

Оскільки кожна хромосома з батьківського пулу $M(k)$ одночасно належить популяції $P(k)$, то хромосоми, що складають множину $M(k) \cap S$ - це ті ж самі особи, що були відібрані з множини $P(k) \cap S$ для включення в популяцію $M(k)$. Якщо кількість хромосом батьківського пулу $M(k)$, що відповідають схемі S (тобто кількість елементів множини $M(k) \cap S$), позначити $b(S, k)$, то з приведених міркувань можна зробити наступний висновок:

**Висновок 3.1**

Очікуване значення $b(S, k)$, тобто очікуване значення кількості хромосом батьківського пулу $M(к)$, що відповідають схемі $S$, визначається виразом

$$E[b(S,k)] = c(S,k) \frac{F(S,k)}{\overline{F}(k)} . \qquad (3.9)$$

З цього випливає, що якщо схема $S$ містить хромосоми зі значенням функції пристосованості, що перевищує середнє значення (тобто пристосованість схеми $S$ на $k$-й ітерації виявляється більшою, ніж середнє значення функції пристосованості хромосом з популяції $P(k)$, і тому $F(S, k)IF(k)>1$), то очікувана кількість хромосом з батьківського пулу $M(k)$, що відповідають схемі $S$, буде більше кількості хромосом з популяції $P(k)$, що відповідають схемі $S$. Тому можна стверджувати, що селекція викликає поширення схем із пристосованістю «краще середньої» і зникнення схем з «гіршою» пристосованістю.

Перш ніж приступити до аналізу впливу генетичних операторів схрещування і мутації на хромосоми з батьківського пулу, визначимо необхідні для подальших міркувань поняття *порядку* й *охоплення схеми.* Нехай $L$ позначає довжину хромосом, що відповідають схемі $S$.

**Означення 3.1**

*Порядок (order)* схеми $S$, інакше називаний *рахунковістю* схеми і що позначається $o(S)$ - це кількість сталих позицій у

схемі, тобто нулів і одиниць у випадку алфавіту {0, 1, *}. Наприклад,

$$o(10*1) = 3\ o(*01*10) = 4\ o(**0*Г) = 2\ o(*101**) = 3$$

Порядок схеми $o(S)$ дорівнює довжині $L$ за винятком кількості символів *, що легко перевірити на представлених прикладах (для $L = 4$ з одним символом * і для $L = 6$ із двома, чотирма і трьома символами *). Легко помітити, що порядок схеми, що складається винятково із символів *, дорівнює нулеві, тобто $o(****) = 0$, а порядок схеми без єдиного символу * дорівнює $L$; наприклад, $o(10011010) = 8$. Порядок схеми $o(S)$ - це завжди ціле число з інтервалу $[0, L]$.

**Означення 3.2**

*Охоплення (defining length)* схеми $S$, яке називається також *довжиною* схеми (не плутати з довжиною $L$) і що позначається $d(S)$ - це відстань між першим і останнім сталим символом (тобто різниця між правими і лівої крайніми позиціями, що містять сталі символи). Наприклад,

$$d(10*1) = 4-1 = 3$$
$$d(**0*1*) = 5-3 = 2$$
$$d(*01*10) = 6-2 = 4$$
$$d(*101**) = 4-2 = 2$$

Охоплення схеми $d(S)$ - це ціле число з інтервалу $[0, L - 1]$. Відзначимо, що охоплення схеми зі сталими символами на першій і останній позиції дорівнює $L$-1 (як у першому з приведених прикладів). Охоплення схеми з єдиною сталою позицією дорівнює нулеві, зокрема, $d(**1*)=0$. Охоплення характеризує змістовність інформації, яка міститься в схемі.

Перейдемо до міркувань про вплив операції схрещування на обробку схем у генетичному алгоритмі. Насамперед відзначимо, що одні схеми виявляються більш підданими знищенню в процесі схрещування, ніж інші. Наприклад, розглянемо схеми $S_1 = 1****0*$ і $S_2 = **01***$, а також хромосому ch=[1001101], що відповідає обом схемам. Видно, що схема $S_2$ має більше шансів «пережити» операцію схрещування, ніж схема $S_1$, що більше піддана «розщепленню» у точках схрещування 1, 2, 3, 4 і 5. Схему $S_2$ можна розділити тільки при виборі точки схрещування, рівної 3. Звертаємо увагу на охоплення обох схем, що - це очевидно - виявляється істотним у процесі схрещування.

У ході аналізу впливу операції схрещування на батьківський пул $M(k)$ розглянемо деяку хромосому з множини $M(k) \cap S$, тобто

хромосому з батьківського пулу, що відповідає схемі *S*. Ймовірність того, що ця хромосома буде відібрана для схрещування, дорівнює $p_c$. Якщо жоден з нащадків цієї хромосоми не буде належати до схеми *S*, то це означає, що точка схрещування повинна знаходитися між першим і останнім сталим символом даної схеми. Ймовірність цього дорівнює $d(S)/(L-1)$. З цього можна зробити наступні висновки:

**Висновок 3.2 (вплив схрещування)**

Для деякої хромосоми з $M(k) \cap S$ ймовірність того, що вона буде відібрана для схрещування і жоден з її нащадків не буде належати до схеми *S*, обмежена зверху величиною

$$p_c \frac{d(S)}{L-1}$$

Ця величина називається *ймовірністю знищення схеми S.*

**Висновок 3.3**

Для деякої хромосоми з $M(k) \cap S$ ймовірність того, що вона не буде відібрана для схрещування або, що хоча б один з її нащадків після схрещування буде належати до схеми *S*, обмежена знизу величиною

$$1-p_c \frac{d(S)}{L-1}$$

Ця величина називається *ймовірністю виживання схеми S.*

Легко показати, що якщо дана хромосома належить до схеми *S* і відбирається для схрещування, а друга батьківська хромосома також належить до схеми *S*, то обидва їх нащадка теж будуть належати до схеми *S*. Висновки 3.2 і 3.3 підтверджують значимість показника охоплення схеми *d(S)* для оцінки ймовірності знищення або виживання схеми.

Розглянемо тепер вплив мутації на батьківський пул *M(k).* Оператор мутації з ймовірністю $p_m$ випадковим чином змінює значення в конкретній позиції з 0 на 1 і зворотно. Очевидно, що схема переживе мутацію тільки в тому випадку, коли всі її сталі позиції залишаться після виконання цієї операції незмінними.

Хромосома з батьківського пулу, що належить до схеми *S* (тобто хромосома з множини $M(k) \cap S$) залишиться в цій схемі тоді і тільки тоді, коли жоден символ цієї хромосоми, що відповідає сталим символам схеми *S*, не зміниться в процесі

мутації. Ймовірність такої події дорівнює $(1-p_m)^{o(S)}$. Цей результат можна представити у формі наступного висновку:

**Висновок 3.4 (вплив мутації)**

Ймовірність того, що деяка хромосома з $M(k) \cap S$ буде належати до схеми $S$ після операції мутації, визначається виразом

$$(1-p_m)^{o(S)}.$$

Ця величина називається *ймовірністю виживання схеми S після мутації.*

**Висновок 3.5**

Якщо ймовірність мутації $p_m$ мала ($p_m \ll 1$), то можна вважати, що ймовірність виживання схеми $S$ після мутації, яка визначена у висновку 3.4, приблизно дорівнює

$$1 - p_m o(S).$$

Ефект спільного впливу селекції, схрещування і мутації (висновки 3.1 - 3.4) з урахуванням факту, що якщо хромосома з множини $M(k) \cap S$ дає нащадка, що відповідає схемі $S$, то він буде належати до $P(k+1) \cap S$, веде до побудови наступної схеми репродукції :

$$E[c(S,k+1)] \geq c(S,k) \frac{F(S,k)}{\overline{F}(k)} \left(1-p_c \frac{d(S)}{L-1}\right)(1-p_m)^{o(S)}. \qquad (3\ 10)$$

Залежність (3.10) показує, як змінюється від популяції до популяції кількість хромосом, що відповідають даній схемі. Ця зміна викликається трьома факторами, представленими в правій частині виразу (3.10), зокрема: $F(S,k)IF(k)$ відбиває роль середнього значення функції пристосованості, $1-p_c d(S)/(L-1)$ показує вплив схрещування і $(1-p_m)^{o(S)}$ - вплив мутації. Чим більше значення кожного з цих факторів, тим більшим виявляється очікувана кількість відповідностей схемі $S$ у наступній популяції. Висновок 3.5 дозволяє представити залежність (3.10) у виді

$$E[c(S,k+1)] \geq c(S,k) \frac{F(S,k)}{\overline{F}(k)} \left(1-p_c \frac{d(S)}{L-1} - p_m o(S)\right). \qquad (3.11)$$

Для великих популяцій залежність (3.11) можна апроксимувати виразом

$$c(S,k+1) \geq c(S,k)\ \frac{F(S,k)}{\overline{F}(k)}\left(1-p_c\frac{d(S)}{L-1}-p_m o(s)\right). \qquad (3.12)$$

З формул (3.11) і (3.12) випливає, що очікувана кількість хромосом, що відповідають схемі *S* у наступному поколінні, можна вважати функцією від фактичної кількості хромосом, що належать цій схемі, відносній пристосованості схеми, а також порядку й охоплення схеми. Помітно, що схеми з пристосованістю вище середньої і з малим порядком і охопленням характеризуються зростанням кількості своїх представників у наступних популяціях. Подібний ріст має показниковий характер, що випливає з виразу (3.9). Для великих популяцій цю формулу можна замінити рекурентною залежністю виду

$$c(S,k+1) = c(S,k)\ \frac{F(S,k)}{\overline{F}(k)} \qquad (3.13)$$

Якщо допустити, що схема *S* має пристосованість на ε% вище середньої, тобто

$$F(S,k) = \overline{F}(k) + \varepsilon\,\overline{F}(k), \qquad (3.14)$$

то при підстановці виразу (3.12) у нерівність (3.11) у припущенні, що ε не змінюється в часі, при старті від $k = 0$ одержуємо

$$c(S,k) = c(S,0)(1+\varepsilon)^k,$$

$$\varepsilon = (F(S,k) - \overline{F}(k))/\overline{F}(k), \qquad (3.15)$$

тобто ε > 0 для схеми з пристосованістю вище середньої і ε<0 - у противному випадку.

Рівність (3.15) описує геометричну прогресію. З цього випливає, що в процесі репродукції схеми, які виявилися кращими (гіршими) середніх, вибираються на чергових ітераціях генетичного алгоритму в показниково зростаючих (убутних) кількостях. Зверніть увагу, що залежності (3.9) - (3.13) засновані на припущенні, що функція пристосованості *F* приймає тільки додатні значення. При використанні генетичних алгоритмів для рішення оптимізаціоних задач, у яких цільова функція може приймати і від'ємні значення, необхідні деякі додаткові співвідношення між оптимізуємою функцією і функцією пристосованості. Кінцевий результат, одержуваний з виразів (3.10) - (3.12), можна сформулювати у формі теореми. Це

основна теорема генетичних алгоритмів, інакше називана *теоремою про схеми* .

**Теорема 3.1**

*Схеми малого порядку, з малим охопленням і з пристосованістю вище середньої формують показниково зростаючу кількість своїх представників у наступних поколіннях генетичного алгоритму.*

Відповідно до приведеної теореми важливим питанням стає кодування, що повинне забезпечувати побудову схем малого порядку, з малим охопленням і з пристосованістю вище середньої. Непрямим результатом теореми 3.1 (про схеми) можна вважати наступну гіпотезу, називану *гіпотезою про цеглинки* (або про *будівельні блоки)*.

**Гіпотеза 3.1**

Генетичний алгоритм прагне досягти близького до оптимального результату за рахунок комбінування гарних схем (із пристосованістю вище середньої) малого порядку і малого охоплення. Такі схеми називаються *цеглинками* (або *будівельними блоками).*

Гіпотеза про будівельні блоки висунута на підставі теореми про схеми з урахуванням того, що генетичні алгоритми досліджують простір пошуку за допомогою схем малого порядку і малого охоплення, які згодом беруть участь в обміні інформацією при схрещуванні. Незважаючи на те, що для доведення цієї гіпотези виконувалися значні дослідження, однак у більшості нетривіальних доданків приходиться спиратися на емпіричні результати. Протягом останніх років опубліковано численні роботи, присвячені застосуванням генетичних алгоритмів, що підтверджують цю гіпотезу. Якщо вона вважається правильною, то проблема кодування здобуває критичне значення для генетичного алгоритму; кодування повинне реалізувати концепцію малих будівельних блоків. Якість, яка забезпечує генетичним алгоритмам явну перевагу перед іншими традиційними методами, безсумнівно полягає в обробці великої кількості різних схем.

Звернемося знову до прикладів 3.4 і 3.5 і на їхній основі проаналізуємо обробку схем генетичним алгоритмом.

**Приклад 3.7**

В умовах приклада 3.4 розглянемо схему $S_0$ = **********11

і покажемо, як змінюється кількість представників цієї схеми і пристосованість у процесі виконання генетичного алгоритму. Довжина $L$=12, а охоплення і порядок схеми $S_0$ складають відповідно $d(S_0)$=1 і $o(S_0)$=2. У вихідній популяції з приклада 3.4 схемі $S_0$ відповідають дві наступні хромосоми:

$ch_3$ = [011101110011]
$ch_7$ = [101011011011]

З формули (3.10) випливає, що після селекції і схрещування кількість хромосом, що відповідають схемі $S_0$, повинна бути більше або дорівнює 2,5. Нагадаємо, що ймовірності схрещування і мутації вважаються рівними відповідно $p_c$ = 1 і $p_m$ = 0. Пристосованість схеми $S_0$ у вихідній популяції, що позначається $F(S_0,$ 0), дорівнює 8 і перевищує середню пристосованість усіх хромосом цієї популяції F=5,75, що легко розрахувати по формулах (3.6) - (3.8).

У прикладі 3.4 після селекції і схрещування в новій популяції отримані чотири хромосоми, що відповідають схемі $S_0$:

$Ch_1$ = [001111011011]
$Ch_3$ = [111011011011]
$Ch_7$ = [011101011011]
$Ch_8$ = [101011110011]

Пристосованість схеми $S_0$ у новій популяції, тобто $F(S_0,$ 1), складе 8,25, тоді як середня пристосованість хромосом цієї популяції F(1)=7, що також випливає з формул (3.6) - (3.8). Нова популяція характеризується великим середнім значенням функції пристосованості особей у порівнянні з попередньою (вихідною) популяцією, що уже відзначалося в прикладі 3.4. Крім того, у новій популяції пристосованість схеми $S_0$ виявляється кращою, а кількість представників цієї схеми - великим у порівнянні з попередньою популяцією.

**Приклад 3.8**

В умовах приклада 3.5 розглянемо схему $S_1$=1****
і простежимо її обробку при виконанні генетичного алгоритму.

У цьому випадку $L$=5, а охоплення і порядок схеми $S_1$, складають $d(S_1)$ = 0 і $o(S_1)$ = 1 відповідно. У вихідній популяції з приклада 3.5 цій схемі відповідають три хромосоми

$ch_1$ =[10011]
$ch_4$ =[10101]
$ch_6$ =[11101]

Пристосованість схеми $S_1$, у вихідній популяції $F(S_1, 0)$=1096 і перевищує середню пристосованість особей цієї популяції $F(0)$ = 589, що випливає з виразів (3.6) - (3.8). На основі формули (3.9) легко розрахувати очікувану кількість хромосом батьківського пулу, які відповідають схемі $S_1$. Вона складе C * 1096/589 = 5,58. У прикладі 3.5 за результатами селекції в батьківський пул включені 6 таких хромосом: $ch_6$, $ch_4$, $ch_6$, $ch_1$, $ch_4$, $ch_6$. Очікувана кількість хромосом, які відповідають схемі $S_1$, після схрещування з ймовірністю $p_c$=1 (ймовірність мутації $p_m$=0), як легко розрахувати по формулі (3.10), повинна перевищувати 5,58. У нову популяцію включено 6 представників схеми $S_1$. Це всі хромосоми даної популяції.

**Приклад 3.9**

В умовах приклада 3.5 розглянемо схему $S_2$ = 11*** і простежимо її обробку при виконанні генетичного алгоритму. Довжина $L$ = 5, а охоплення і порядок схеми $S_2$ складають $d(S_2)$= 1 і $o(S_2)$=2 відповідно. У вихідній популяції з приклада 3.5 цій схемі відповідає одна хромосома $ch_6$=[11101]. Тому пристосованість схеми $S_2$ у вихідній популяції дорівнює функції пристосованості хромосоми $ch_6$ і складає 1683. Вона перевищує середню пристосованість особей вихідної популяції, рівну 589. По формулі (3.9) розраховуємо очікувану кількість хромосом батьківського пулу, які відповідають схемі $S_2$. Вона складе 1683/589 = 2,86. У прикладі 3.5 за результатами селекції в батьківський пул включені 3 однакові хромосоми [11101], які відповідають схемі $S_2$. Очікувана кількість хромосом у новій популяції, які відповідають схемі $S_2$, після схрещування з ймовірністю $p_c$=1 (ймовірність мутації $p_m$=0), повинна перевищувати 5,58. У прикладі 3.5 у нову популяцію включені 3 хромосоми, які відповідають схемі $S_2$. Це

$$Ch_4=Ch_5 = Ch_6 = [11101].$$

**Приклад 3.10**

В умовах приклада 3.5 розглянемо схему $S_3$ = ***11 і простежимо її обробку при виконанні генетичного алгоритму. Довжина $L$= 5, а охоплення і порядок схеми $S_3$ складають $d(S_3)$ = 1 і $o(S_3)$ = 2 відповідно. У вихідній популяції з приклада 3.5 цій схемі відповідають три хромосоми

$ch_1$ = [10011]
$ch_2$ = [00011]
$ch_3$ = [00111]

На відміну від прикладів 3.8 і 3.9 пристосованість схеми $S_3$ у вихідній популяції виявляється менше середньої пристосованості особей цієї популяції $\overline{F}$ (0)=589 і складає F($S_3$, 0)=280. Очікувана кількість хромосом батьківського пулу, що відповідають схемі $S_3$ і розрахована по формулі (3.9), дорівнює 3*280/589=1,426. У прикладі 3.5 у батьківський пул була включена одна хромосома [10011], що відповідає схемі $S_3$. На основі формули (3.10) одержуємо значення 1,068, що визначає кількість представників схеми $S_3$ у новій популяції. У прикладі 3.5 після схрещування з ймовірністю $p_c$=1 (ймовірність мутації $p_m$=0) у нову популяцію була включена одна хромосома, що відповідає схемі $S_3$, тобто $Ch_2$ = [10111].

**Приклад 3.11**

В умовах приклада 3.5 розглянемо схему

$$S_4 = *10**$$

і простежимо її обробку при виконанні генетичного алгоритму. Довжина $L$ = 5, а охоплення і порядок схеми $S_4$ складають $d(S_4)$ = 1 і $o(S_A)$ = 2 відповідно. У вихідній популяції з приклада 3.5 цій схемі відповідає тільки одна хромосома

$$ch_5 = [01000]$$

Тому пристосованість схеми $S_4$ у вихідній популяції дорівнює значенню функції пристосованості хромосоми $ch_5$ і складає 129. Аналогічно прикладові 3.10, вона менше середньої пристосованості особей початкової популяції, що дорівнює 589. Очікувана кількість представників схеми $S_4$ у батьківському пулі складає 129/589 = 0,22, що випливає з формули (3.9). У прикладі 3.5 батьківський пул (після селекції) не містить ні однієї хромосоми, що відповідає схемі $S_4$. При розрахунку очікуваної кількості представників схеми $S_4$ у новій популяції по формулі (3.10) для ймовірності схрещування $p_c$ = 1 і ймовірності мутації $p_m$=0 одержуємо значення 0,165. У прикладі 3.5 після схрещування в нову популяцію не була включена жодна хромосома, що відповідає схемі $S_4$.

З прикладів 3.7 - 3.11, присвячених обробці схем, можна зробити наступні висновки. Ці приклади ілюструють основну теорему генетичних алгоритмів - теорему про схеми. Вони торкаються обробки схем низького (малого) порядку з малим охопленням. Приклади 3.7 - 3.9 демонструють збільшення кількості представників даної схеми в наступному поколінні для випадку, коли пристосованість цієї схеми перевищує середню

пристосованість всіх особей популяції. Приклади 3.10 і 3.11 показують ситуацію, коли пристосованість схеми виявляється менша середньої пристосованості особей популяції. Кількість представників таких схем у наступних поколіннях не збільшується, а навпаки - спостерігається зменшення кількості відповідних їм хромосом. При аналізі подібних прикладів для схем більшого порядку і більшого охоплення також не реєструється збільшення кількості їхніх представників у наступному поколінні, що погоджується з теоремою про схеми.

Графічна інтерпретація схем, що обговорювалися в прикладах 3.8 і 3.11, представлена на рис. 3.23; аналогічним чином можна проілюструвати схеми з прикладів 3.9 і 3.10.

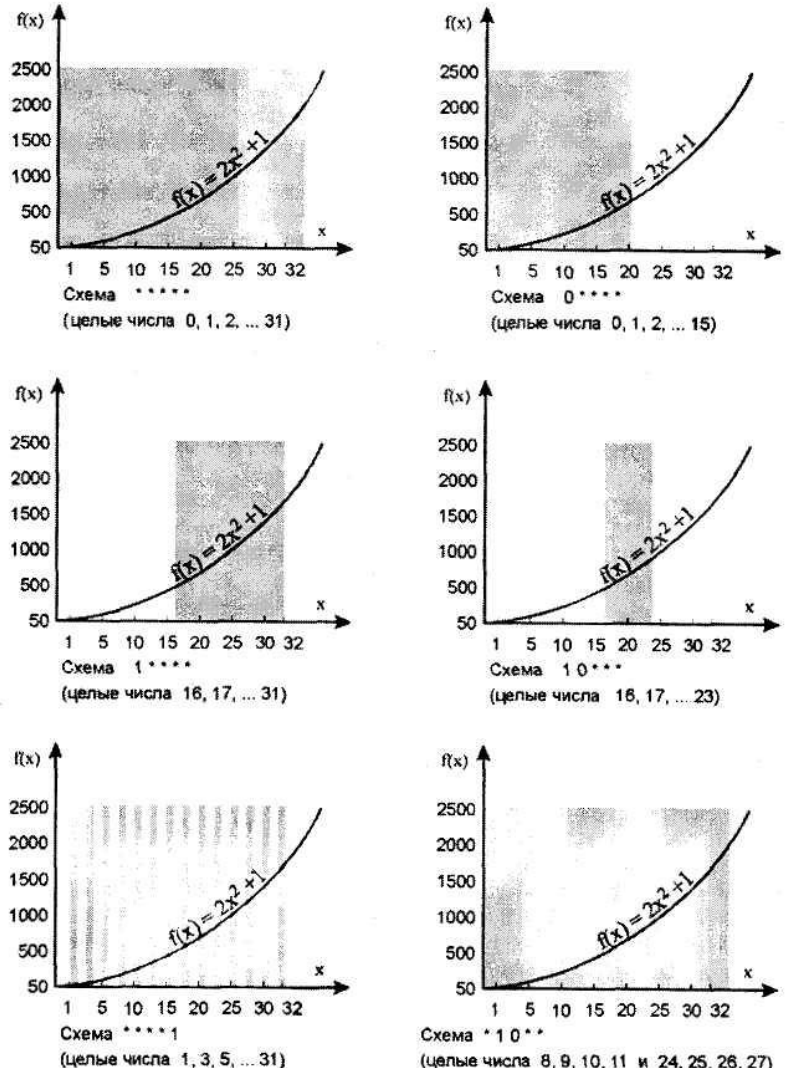



Рис. 3.23. Графічне представлення схем для цілочислових значень *x* від 0 до 31, закодованих у формі 5-бітових двоїчних послідовностей для оптимізації функції $f(x) = 2x^2 + 1$ (приклади 3.5, а також 3.8 і 3.11).

На рис. 3.23 видно, що до схеми 1*** (приклад 3.8) у вихідній популяції з приклада 3.5 належать хромосоми $ch_1$, $ch_4$ і $ch_6$ з фенотипами 19, 21, 29 відповідно, а після селекції і схрещування до цієї схеми вже належать усі включені в нову популяцію хромосоми, тобто $Ch_1$, $Ch_2$, $Ch_3$, $Ch_4$, $Ch_5$, і $Ch_6$ з фенотипами 17, 23, 21, 29, 29, 29 відповідно. У той же час до схеми *10** (приклад 3.11) у вихідній популяції з приклада 3.5 належить тільки одна хромосома $ch_5$, фенотип якої дорівнює 8; у наступній популяції вже немає жодної хромосоми, що належить цій схемі. Зверніть увагу (рис. 3.23), що оптимальне рішення, що максимізує функцію, задану виразом (3.1), належить до схеми 1**** і не відповідає схемі *10**.

Виконання генетичних алгоритмів засновано на обробці схем. Схеми малого порядку, з малим охопленням і пристосованістю вище середньої вибираються, розмножуються і комбінуються, у результаті чого формуються всі кращі кодові послідовності. Тому оптимальне рішення будується (відповідно до гіпотези цеглинок) шляхом об'єднання найкращих з отриманих на даний момент часткових рішень. Просте схрещування не занадто часто знищує схеми з малим охопленням, однак ліквідує схеми з досить великим охопленням. Однак незважаючи на згубність операцій схрещування і мутації для схем високого порядку й охоплення, кількість оброблюваних схем настільки велике, що навіть при відносно низькій кількості хромосом у популяції досягаються досить непогані результати виконання генетичного алгоритму.

Кількість ефективна оброблюваних схем, розраховане Холландом, складає $O(N^3)$. Це означає, що для популяції потужністю $N$ кількість оброблюваних у кожнім поколінні схем має порядок $N^3$.

## 3.12. Будівельні блоки (Building blocks)

Будівельний блок (ББ) (*building block (BB)*), як ми вже відзначали, - це одне з ключових понять у теорії генетичних алгоритмів, поряд зі схемою і теоремою схем. До будівельних блоків (як правило) прийнято відносити схеми з малою визначальною довжиною, малим порядком і високою

пристосованістю. Пристосованість ББ найчастіше визначається як середня пристосованість особей, хромосоми яких містять даний будівельний блок. У гіпотезі про будівельні блоки (*building blocks hypothesis*) вважається, що в процесі роботи генетичного алгоритму, у міру наближення до глобального оптимуму, порядок і пристосованість будівельного блоку збільшуються. Тобто на початку еволюції відносно високою пристосованістю володіють будівельні блоки малої визначальної довжини, потім, у результаті добору і рекомбінації, з'являються більш пристосовані і "довгі" будівельні блоки. Таким чином, говорять про обробку будівельних блоків (*building blocks processing*) у результаті роботи генетичного алгоритму.

Виділяють наступні проблеми в обробці будівельних блоків:
1. Ідентифікація.
2. Змішування.

Під *ідентифікацією* розуміють проблему знаходження (локалізації) будівельного блоку. Іншими словами, яку саме ділянку хромосоми можна вважати будівельним блоком. Очевидно, що мінімальний по розмірах ББ представляє один розряд, а максимальний - усю хромосому. Але це в найпростішому варіанті, а у випадку "не граничних умов" кількість "претендентів" на звання будівельного блоку дуже велика. Зокрема, для хромосоми довжиною *n* може існувати *C(n, 2)* різних позицій для будівельних блоків 2-го порядку, *C(n, 3)* різних позицій для будівельних блоків третього порядку і т.д., де *C(n, m)* - кількість сполучень з *n* по *m* (обчислюється як *n!/(m!(n-m)!)*, де "*!*" - факторіал, тобто *n! = 1*2*...*(n-1)*n*, де "*" - позначає операцію множення).

Проблема *змішування* полягає в тім, що, навіть якщо будівельні блоки знайдені, важко визначити, які з них варто змішувати (грубо говорячи, поміщати в одну хромосому), а які – ні, так як висока пристосованість різних будівельних блоків не гарантує високої пристосованості хромосом, отриманих у результаті їхньої комбінації.
У силу позначених проблем приділяється досить багато уваги

розробці операторів і підходів, що дозволили б забезпечити ефективну (*BB-wise*) обробку будівельних блоків. Також існує думка, що ефективна ідентифікація і змішування є критичними умовами для забезпечення успішної роботи генетичного алгоритму.

Крім ідентифікації і змішування існує проблема *невизначеності пристосованості будівельного блоку* (*building block fitness noise*). Оскільки той самий будівельний блок може входити як у гарні (з погляду розв'язуваної задачі) хромосоми, так і в не дуже, то пристосованість цього будівельного блоку практично завжди визначається з деяким "шумом". Це ускладнює задачу вибору між двома будівельними блоками, тому що навіть якщо пристосованість одного з них вища, ніж пристосованість іншого, завжди є ймовірність зробити неправильний вибір. Говорячи по-іншому, у результаті селекції може бути обрана особа з менш пристосованим будівельним блоком. Графічно це можна представити в такий спосіб (рис.3.24)

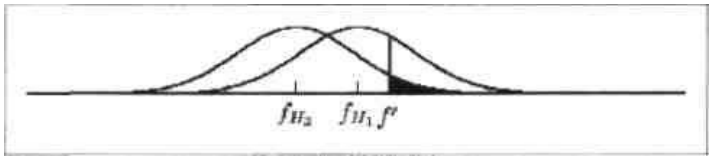



Рис. 3.24.

Нехай *H1* і *H2* два будівельних блоки, пристосованості яких розподілені по нормальному закону, причому $f_{H_1}$ і $f_{H_2}$ відповідно середні пристосованості блоків 1 і 2. Нехай також пристосованість $H_1$ більша, ніж пристосованість $H_2$. Якщо є невелике число хромосом, що містять розглянуті блоки, то їх пристосованості визначені з досить великою похибкою, іншими словами, дисперсія розподілів велика. Таким чином, площа взаємного "накладання" розподілів також значна, і, отже, значна ймовірність вибрати в результаті селекції менш пристосований будівельний блок. У випадку, представленому на рис. 3.24, ймовірність вибрати хромосому з пристосованістю більше *f'* і яка

утримує схему $H_2$, дорівнює площі зафарбованої частини розподілу пристосованості 2-ї схеми. Якщо збільшити кількість хромосом які утримують розглянуті будівельні блоки (наприклад, збільшивши розмір популяції), то тоді розподіли пристосувань обох блоків "стискуються", так як значення пристосувань визначаються більш точно (рис.3.25). Тим самим зменшується ймовірність вибору менш пристосованого будівельного блоку.

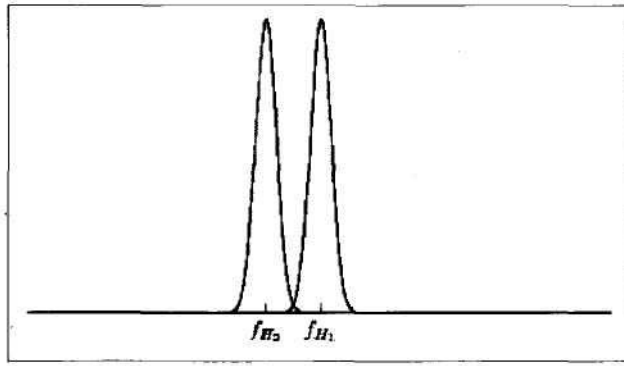



Рис. 3.25.

Незважаючи на те, що споконвічно будівельні блоки визначалися як схеми з визначеними властивостями, тобто стосовно до ГА з бінарним кодуванням, аналоги схем, а отже і будівельних блоків, можна знайти й в інших видах еволюційних обчислень (не тільки в генетичних алгоритмах). Вочевидь, концепція будівельних блоків давно вже живе "своїм життям", окремо від теореми схем, так як за ББ можна вважати будь-яку ділянку хромосоми визначеного виду поза залежністю від того, чи можливо описати хромосоми, що містять цей блок, однією схемою чи ні.

## 3.13. Модифікації класичного генетичного алгоритму

У класичному генетичному алгоритмі (представленому в разд. 3.4 - 3.6 і в прикладах 3.4 і 3.5) використовується двоїчне представлення хромосом, селекція методом колеса рулетки і точкове схрещування (з одною точкою схрещування). Для підвищення ефективності його роботи створена множина модифікацій основного алгоритму. Вони зв'язані з застосуванням інших методів селекції, з модифікацією генетичних операторів (у першу чергу оператора схрещування), з перетворенням функції пристосованості (шляхом її масштабування), а також з іншими способами кодування параметрів задачі у формі хромосом. Існують також версії генетичних алгоритмів, що дозволяють знаходити не тільки глобальний, але і локальні оптимуми. Це алгоритми, що використовують так називані ніші, введені в генетичні алгоритми за аналогією з природними екологічними нішами. Інші версії генетичних алгоритмів служать для багатокритеріальної оптимізації, тобто для одночасного пошуку оптимального рішення для декількох функцій. Зустрічаються також спеціальні версії генетичного алгоритму, створені для рішення проблем малої розмірності, не потребуючих ні великих популяцій, ні довгих хромосом. Їх називають *генетичними мікроалгоритмами.*

### 3.13.1. Методи селекції

Заснований на принципі колеса рулетки метод селекції, який представлено у разд. 3.4 і продемонстрований у прикладах 3.4 і 3.5, вважається для генетичних алгоритмів основним методом добору особей для батьківської популяції з метою наступного їхнього перетворення генетичними операторами, такими як схрещування і мутація. Незважаючи на випадковий характер процедури селекції, батьківські особи вибираються пропорційно значенням їхніх функцій пристосованості, тобто відповідно до ймовірності селекції, обумовленої по формулі (3.3). Кожна особа одержує в батьківському пулі таку кількість своїх копій, яка установлюється виразом

$$e(ch_i) = p_s(ch_i)\cdot N, \qquad (3.16)$$

де $N$ - кількість хромосом $ch_i$, i=1,2.....$N$ у популяції, а $p_s(ch_i)$ - ймовірність селекції хромосоми $ch_i$, що розраховується по формулі (3.3). Строго говорячи, кількість копій даної особи в батьківському пулі дорівнює цілій частині від $e(ch_i)$. При використанні формул (3.3) і (3.16) необхідно звертати увагу на

те, що $e(ch_1)=F(ch_i)/\overline{F}$, де $\overline{F}$ - середнє значення функції пристосованості в популяції. Очевидно, що метод рулетки можна застосовувати тоді, коли значення функції пристосованості додатні. Цей метод може використовуватися тільки в задачах максимізації функції (але не мінімізації).

Очевидно, що проблему мінімізації можна легко звести до задачі максимізації функції і зворотно. У деяких реалізаціях генетичного алгоритму метод рулетки застосовується для пошуку мінімуму функції (а не максимуму). Це результат відповідного перетворення, виконуваного програмним шляхом для зручності користувачів, оскільки в більшості прикладних задач вирішується проблема мінімізації (наприклад, витрат, відстані, похибки і т.п.). Як приклад такої реалізації можна назвати програму **FlexTool**. Однак можливість застосування методу рулетки усього лише для одного класу задач, тобто тільки для максимізації (або тільки для мінімізації) можна вважати його безсумнівним недоліком. Інша слабка сторона цього методу полягає в тім, що особи з дуже малим значенням функції пристосованості занадто швидко виключаються з популяції, що може призвести до передчасної збіжності генетичного алгоритму. Для запобігання такого ефекту застосовується масштабування функції пристосованості (п. 3.13.5).

З урахуванням відзначених недоліків метода рулетки створені і використовуються альтернативні алгоритми селекції. Один з них називається *турнірним методом (tournament selection).* Поряд з методом рулетки і ранговим методом він застосовується як один з основних алгоритмів селекції в програмі **FlexTool.** Представимо ці методи докладніше.

При *турнірній селекції* всі особи популяції розбиваються на підгрупи з наступним вибором у кожній з них особи з найкращою пристосованістю. Розрізняються два способи такого вибору: *детермінованний вибір (deterministic tournament selection)* і *випадковий вибір (stochastic tournament selection).* Детермінованний вибір здійснюється з ймовірністю, рівною 1, а випадковий вибір - з ймовірністю, меншою 1. Підгрупи можуть мати довільний розмір, але найчастіше популяція розділяється на підгрупи по 2 - 3 особи в кожній. Турнірний метод придатний для рішення задач як максимізації, так і мінімізації функції. Крім того, він може бути легко розповсюджений на задачі, зв'язані з багатокритеріальною

оптимізацією, тобто на випадок одночасної оптимізації декількох функцій. У турнірному методі допускається зміна розміру підгруп, на які підрозділяється популяція *(tournament size).* Дослідження підтверджують, що турнірний метод діє ефективніше, ніж метод рулетки.

На рис. 3.26 представлена схема, що ілюструє метод турнірної селекції для підгруп, що складаються з двох особей. Таку схему легко узагальнити на підгрупи більшого розміру. Це один з можливих доданків розглянутого алгоритму селекції (він використовується в програмі **FlexTool** ).

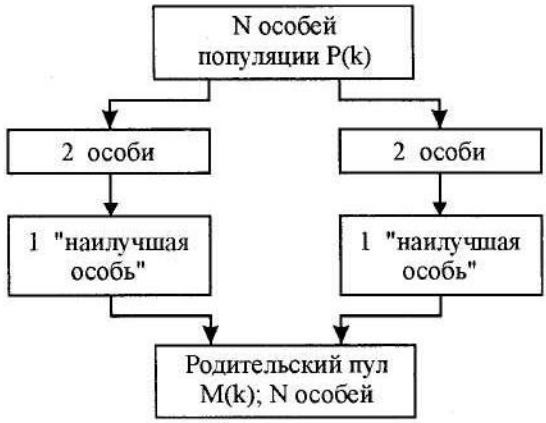



Рис. 3.26. Схема турнірної селекції для підгруп, що складаються з двох особей.

При *ранговій селекції (ranking selection)* особи популяції ранжируются за значеннями їхньої функції пристосованості. Це можна уявити собі як відсортований список особей, упорядкованих по напрямку від найбільш пристосованих до найменш пристосованих (або навпаки), у якому кожній особі приписується число, що визначає її місце в списку і називане *рангом (rank).* Кількість копій *M(k)* кожної особи, введених у батьківську популяцію, розраховується по апріорно заданій функції в залежності від рангу особи. Приклад такої функції показано на рис. 3.27.

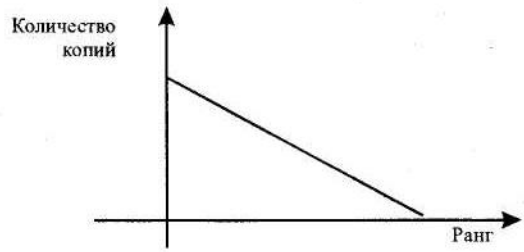



Рис. 3.27. Приклад функції, що визначає залежність кількості копій особи в батьківському пулі від його рангу при ранговій селекції.

Достоїнство рангового методу полягає в можливості його застосування як для максимізації, так і для мінімізації функції. Він також не вимагає масштабування через проблему передчасної збіжності, актуальної для методу рулетки.

Існують різні варіанти алгоритмів селекції. Представлені раніше методи (рулетки, турнірний і ранговий) застосовуються найчастіше. Інші методи являють собою або їх модифікації, або комбінації - наприклад, метод рулетки з турнірним методом, коли пари батьківських хромосом вибираються випадковим чином, після чого з кожної пари вибирається хромосома з найбільшим значенням функції пристосованості. Більшість методів селекції засновано на формулах (3.3) і (3.16), по яких розраховується ймовірність селекції і кількість копій, що вводяться в батьківський пул. У так називаному детермінованному методі кожна особа одержує число копій, рівне цілій частині від $e(ch_i)$, після чого популяція упорядковується відповідно до дробової частини $e(ch_i)$, а інші хромосоми, які необхідні для поповнення нової популяції, послідовно вибираються з верхньої частини сформованого в такий спосіб списку. В іншому методі (називаному випадковим) дробові частини $e(ch_i)$ розглядаються як ймовірності успіху по Бернуллі і, наприклад, хромосома $ch_i$, для якої $e(ch_i)=1,5$, одержує одну копію гарантированно і ще одну - з ймовірністю 0,5. У ще одному методі для усунення розбіжності між розрахунковим значенням $e(ch_i)$ і кількістю копій хромосом $ch_i$, обираних по методу рулетки, виробляється модифікація $e(ch_i)$ шляхом

збільшення або зменшення його значення для кожної хромосоми, обраної для схрещування і/або мутації.

### 3.13.2. Особливі процедури репродукції

Як особливі процедури репродукції можна розглядати так називану *елітарну стратегію* і *генетичний алгоритм із частковою заміною популяції.*

*Елітарна стратегія (elitist strategy)* полягає в захисті найкращих хромосом на наступних ітераціях. У класичному генетичному алгоритмі саме пристосовані особи не завжди переходять у наступне покоління. Це означає, що нова популяція *P(k*+1) не завжди містить хромосому з найбільшим значенням функції пристосованості з популяції *P(k).* Елітарна стратегія застосовується для запобігання втрати такої особи. Ця особа гарантированно включається в нову популяцію.

*Генетичний алгоритм із частковою заміною популяції,* інакше називаний *генетичним алгоритмом із зафіксованим станом (steady-state),* характеризується тим, що частина популяції переходить у наступне покоління без яких-небудь змін. Це означає, що хромосоми, які входять в цю частину, не піддаються операціям схрещування і мутації. Часто в конкретних реалізаціях алгоритму даного типу на кожній ітерації замінюються тільки одна або дві особи замість схрещування і мутації в масштабі всієї популяції. Саме такий підхід прийнятий, наприклад, у програмі **Evolver**. В інших програмах, зокрема, у **FlexTool**, користувач може сам установити - яка частина популяції (у відповідності зі значеннями функції пристосованості) повинна передаватися без змін у наступне покоління. Ця підмножина хромосом не піддається регулярній селекції і без змін включається в нову популяцію.

### 3.13.3. Генетичні оператори

У класичному генетичному алгоритмі операція схрещування являє собою так називане точкове схрещування, яке було розглянуто в разд. 3.4 і в прикладах 3.4 і 3.5. Також застосовуються й інші види схрещування: двоточкове, багатоточкове і рівномірне.

*Двоточкове схрещування (two-point crossover),* як випливає з його назви, відрізняється від точкового схрещування тим, що нащадки успадковують фрагменти батьківських хромосом, обумовлені двома випадково обраними точками схрещування.

Для пари хромосом із приклада 3.4 схрещування в точках 4 і 6 показано на рис. 3.28. Звертаємо увагу, що таке схрещування не приводить до знищення схеми 1**********1, яку представляє батько 2.

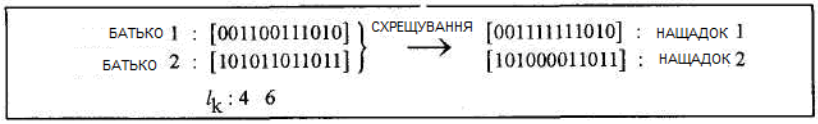



Рис. 3.28. Приклад двоточкового схрещування.

*Багатоточкове схрещування (multiple-point crossover)* являє собою узагальнення попередніх операцій і характеризується відповідно великою кількістю точок схрещування. Наприклад, для трьох точок схрещування, рівних 4, 6 і 9, і такої ж кількості батьків, як на рис. 3.28, результати схрещування показані на рис. 3.29.


Рис. 3.29. Приклад трьохточкового схрещування.

Аналогічно виконується схрещування для п'яти або більшої непарної кількості точок. Очевидно, що одноточкове схрещування може вважатися частковим випадком багатоточкового схрещування.

Приклад двоточкового схрещування, який представлено на рис. 3.28, можна проілюструвати способом, показаним на рис. 3.30.

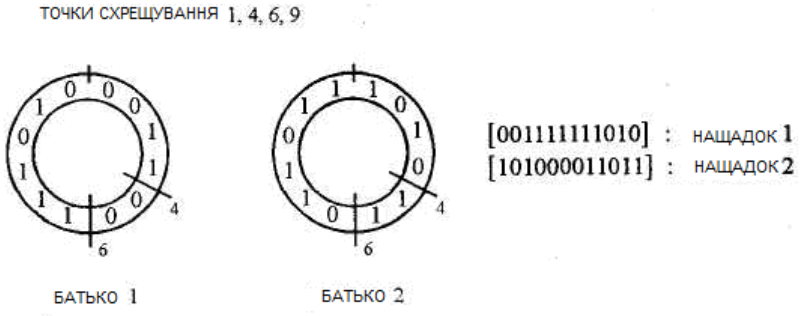



Рис. 3.30. Двоточкове схрещування з точками схрещування 4 і 6.

Багатоточкове схрещування для чотирьох точок, рівних 1,4,6, 9 і 4, 6, 9, 11 для тієї ж пари батьків з попередніх прикладів ілюструється на рис. 3.31.

Багатоточкове схрещування з великою парною кількістю точок схрещування протікає аналогічно показаному на рис. 3.31.

Схрещування з непарною кількістю точок можна представити в такий же спосіб, якщо додати ще одну точку схрещування в позицію, яка рівна 0. Приведений вище приклад для трьох точок можна представити також, як на рис. 3.31, із точками схрещування 0, 4, 6, 9. При парній кількості точок хромосома розглядається як замкнене кільце (див. рис. 3.30 і 3.31), а точки схрещування вибираються з рівною ймовірністю по всьому її колі.

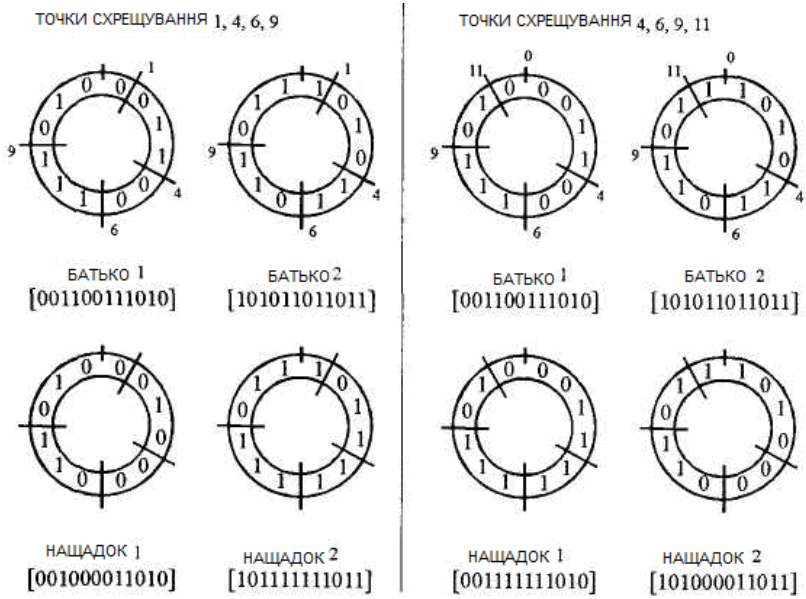



Рис. 3.31. Багатоточкове схрещування з чотирма точками схрещування, рівними 1, 4, 6, 9 і 4, 6, 9, 11.

*Рівномірне схрещування (uniform crossover),* інакше називане *монолітним* або *одностадійним,* виконується у відповідності з випадково обраним еталоном, що вказує, які гени повинні успадковуватися від першого батька (інші гени беруться від другого батька). Припустимо, що для пари батьків із прикладів на рис. 3.28 - 3.31 обрано еталон 010110111011, у якому 1 означає прийняття гена на відповідній позиції *(locus)* від батька 1, а 0 - від батька 2. У такий спосіб формується перший нащадок. Для другого нащадка еталон необхідно зчитувати аналогічно, причому 1 означає прийняття гена на відповідній позиції від батька 2, а 0 - від батька 1. У цьому випадку рівномірне схрещування протікає так, як показано на рис. 3.32.

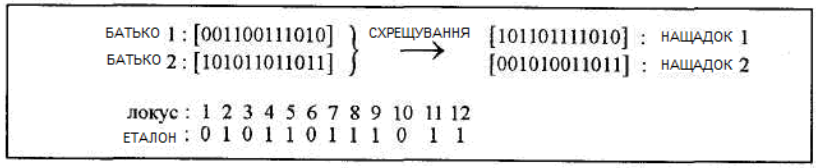



Рис. 3.32. Приклад рівномірного схрещування.

*Оператор інверсії.* Холланд запропонував три технології для одержання нащадків, що відрізняються від батьківських хромосом. Це уже відомі нам операції схрещування і мутації, а також операція інверсії. Інверсія виконується на одиночній хромосомі; при її здійсненні змінюється послідовність аллелей між двома випадково обираними позиціями *(locus)* у хромосомі. Незважаючи на те, що цей оператор був визначений за аналогією з біологічним процесом хромосомної інверсії, він не занадто часто застосовується в генетичних алгоритмах. Як приклад виконання інверсії розглянемо хромосому [001100111010] і припустимо, що обрано позиції 4 і 10. Тоді в результаті інверсії одержимо [001101110010].

## 3.13.4. Методи кодування

У класичному генетичному алгоритмі застосовується двоїчне кодування хромосом. Воно засновано на відомому способі запису десяткових чисел у двоїчній системі, де кожен біт двоїчного коду відповідає черговому степеню цифри 2. Наприклад, двоїчна послідовність [10011] являє собою код числа 19, оскільки $1*2^4+0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 19$. Такий спосіб кодування застосовувався в прикладах 3.1 і 3.5. Для кодування дійсних чисел $x_i \in [a_i, b_i] \in R$ реалізується відображення (3.5) так, як це робилося в прикладах 3.2 і 3.6.

У генетичних алгоритмах можна, наприклад, використовувати код Грея, який характеризується тим, що двоїчні послідовності, що відповідають двом послідовним цілим числам, відрізняються тільки одним бітом. Такий спосіб кодування хромосом може виявитися виправданим при використанні операції мутації.

*Логарифмічне кодування (logarithmic coding)* застосовується в генетичних алгоритмах для зменшення довжини хромосом.

Воно використовується, головним чином, у задачах багатомірної оптимізації з великими просторами пошуку рішень.

При логарифмічному кодуванні перший біт (α) кодової послідовності - це біт знака показникової функції, другий біт (β) - біт знака степеня цієї функції, а інші біти (bin) представляють значення самого степеня:

$$[\alpha\beta\ bin] = (-1)^{\beta}\, e^{(-1)^{\alpha}[bin]_{10}}$$

де $[bin]_{10}$ означає десяткове значення числа, закодованого у виді двоїчної послідовності bin. Наприклад,

[10110]

являє собою кодову послідовність числа

$$x_1 = (-1)^0 e^{(-1)^1[110]_{10}} = e^{-6} = 0{,}002478752\ ,$$

а

[01010]

являє собою кодову послідовність числа

$$x_2 = (-1)^1 e^{(-1)^0[010]_{10}} = -e^2 = -7{,}389056099\ .$$

Помітимо, що в такий спосіб за допомогою п'яти бітів можна закодувати числа з інтервалу $[-e^7, e^7]$. Це значно більший інтервал, чим [0, 31] із приклада 3.5. Логарифмічне кодування було реалізовано в програмі **FlexTool** як додаткову опцію для задач підвищеної складності.

Ще одна модифікація класичного генетичного алгоритму заснована на кодуванні дійсними, а не двоїчними числами. Це означає, що гени хромосом приймають дійсні значення (аллелі є дійсними числами). Такий спосіб кодування застосовується, зокрема, у програмі **Evolver**.

### 3.13.5. Масштабування функції пристосованості

Масштабування функції пристосованості виконується, найчастіше, по двох причинах. По-перше (про це вже говорилося під час обговорення методів селекції), для запобігання передчасної збіжності генетичного алгоритму. По-друге (у кінцевій фазі виконання алгоритму), у випадку, коли в популяції зберігається значна неоднорідність, однак середнє значення пристосованості ненабагато відрізняється від максимального значення. Масштабування функції пристосованості дозволяє попередити виникнення ситуації, у якій середні і найкращі особи формують практично однакову кількість нащадків у наступних поколіннях, що вважається небажаним явищем. Передчасна

збіжність алгоритму полягає в тім, що в популяції починають домінувати найкращі, але ще не оптимальні хромосоми. Така можливість характерна для алгоритмів із селекцією по методу колеса рулетки. Через кілька поколінь при селекції, яка пропорційна значенню функції пристосованості, популяція буде складатися винятково з копій найкращої хромосоми вихідної популяції. Представляється малоймовірним, що саме ця хромосома буде відповідати оптимальному рішенню, оскільки вихідна популяція - це, як правило, невелика випадкова вибірка з усього простору пошуку. Масштабування функції пристосованості охороняє популяцію від домінування неоптимальної хромосоми і тим самим запобігає передчасній збіжності генетичного алгоритму.

Масштабування полягає у відповідному перетворенні функції пристосованості. Розрізняють 3 основних методи масштабування: лінійне, сігма-відсікання і степеневе.

*Лінійне масштабування (linear scaling)* полягає в перетворенні функції пристосованості F до форми F' через лінійну залежність виду

$$F' = a \cdot F + b,$$

де *a* і *b* - константи, які варто підбирати таким чином, щоб середнє значення функції пристосованості після масштабування дорівнювало б її середньому значенню до масштабування, а максимальне значення функції пристосованості після масштабування було б кратним її середньому значенню. Коефіцієнт кратності найчастіше вибирається в межах від 1,2 до 2. Необхідно стежити за тим, щоб функція F' не приймала від'ємні значення.

*Сігма-відсікання (sigma truncation)* - метод масштабування, заснований на перетворенні функції пристосованості *F* до форми F' відповідно до виразу

$$F' = F + (\vec{F} - c \cdot \sigma),$$

де $\vec{F}$ позначає середнє значення функції пристосованості по всій популяції, *c* - мале натуральне число (як правило, від 1 до 5), а $\sigma$ - стандартне відхилення по популяції. Якщо розрахункові значення F' від'ємні, то вони приймаються рівними нулеві.

*Степеневе масштабування (power law scaling)* являє собою метод масштабування, при якому функція пристосованості *F* перетвориться до форми F' відповідно до виразу

$$F' = F^k,$$

де $k$ - число, близьке 1. Значення $k$ зазвичай підбирається емпірично з урахуванням специфіки розв'язуваної задачі. Наприклад, можна використовувати $k$ = 1,005.

### 3.13.6. Ніші в генетичному алгоритмі

У різних оптимізаційних задачах часто приходиться мати справу з функціями, що мають кілька оптимальних рішень. Основний генетичний алгоритм у таких випадках знаходить тільки глобальний оптимум, але якщо є кілька оптимумів з тим самим значенням, то він відшукує тільки один з них. У деяких задачах буває важливим знайти не тільки глобальний оптимум, але і локальні оптимуми (не обов'язково усі). Концепція реалізації в генетичних алгоритмах підходу, заснованого на відомих з біології поняттях *ніш і видів*, дозволяє знаходити велику частину оптимумів. Практично застосовуваний у генетичному алгоритмі метод утворення ніш і видів заснований на так називаній *функції співучасті (sharing function).* Ця функція визначає рівень близькості і степінь співучасті для кожної хромосоми в популяції. Функція співучасті позначається $s(d_{ij})$, де $d_{ij}$ - міра відстані між хромосомами $ch_i$ і $ch_j$. У програмі **FlexTool** ця відстань визначається по формулі

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \frac{x_{k,i} - x_{k,j}}{(x_{k\,max} - x_{k,min})^2}}$$

де $p$ означає розмірність задачі, $x_{k,min}$ і $x_{k,max}$ визначають відповідно мінімальне і максимальне значення $k$-го параметра, $x_{ki}$ і $x_{kj}$ - позначають відповідно $k$-й параметр $i$-ї і $j$-ї особей. Очевидно, що відстань між хромосомами розраховується на основі відповідних їм фенотипів.

Функція співучасті $s(d_{ij})$ повинна мати наступні властивості:

$$0 \le s(d_{ij}) \le 1 \text{ для кожного } d_{ij},$$

$$s(0) = 1,$$

$$\lim_{d_{ij} \to \infty} s(d_{ij}) = 0$$

Одна з функцій, для якої ці умови виконуються, має вигляд

$$s(d_{ij})=\begin{cases}1-(d_{ij}/\sigma_s)^{\omega}, & якщо\ d_{ij}<\sigma_s,\\ 0 & в\ противному\ випадку\end{cases}$$

де $\sigma_s$ і $\omega$ - константи.

У програмі **FlexTool** $\sigma_s=0{,}5 * q^{-1/p}$, де $q$ позначає зразкову кількість піків функції, яка оптимізується. Ця функція задається користувачем. Значення $\omega$ приймається рівним 1, що означає однаковий степінь співучасті сусідніх особей. У цьому випадку нове значення функції пристосованості хромосоми $ch_i$, розраховується по формулі

$$F_s(ch_i)=\frac{F(ch_i)}{\sum_{i=1}^{N} s(d_{ij})} \qquad (3.17)$$

де $N$ позначає кількість хромосом у популяції.

Якщо хромосома $ch_i$ знаходиться у своїй ніші на самоті, то $F_s(ch_i)=F(ch_i)$. У противному випадку значення функції пристосованості зменшується пропорційно кількості і степеневі близькості хромосом, що є сусідами. З виразу (3.17) випливає, що збільшення кількості схожих одна на одну (тобто приналежних до однієї і тій же ніші) хромосом обмежено, оскільки таке збільшення приводить до зменшення значення функції пристосованості. У програмі **FlexTool** при реалізації генетичного алгоритму з нішами метод, що представляється, використовується на завершальному етапі обробки кожного покоління.

Є також і різні модифікації процедури утворення ніш для генетичного алгоритму. Наприклад, можна визначити міру відстані між хромосомами не на рівні фенотипу (тобто параметрів задачі), а на рівні генотипу. У цьому випадку аргументом функції співучасті буде відстань Хеммінга між кодовими послідовностями. Відомі й інші підходи до модифікації функції пристосованості для генетичного алгоритму з нішами.

## 3.13.7. Генетичні алгоритми для багатокритеріальної оптимізації

Більшість задач, розв'язуваних за допомогою генетичних алгоритмів, мають один критерій оптимізації. У свою чергу, багатокритеріальна оптимізація заснована на відшуканні рішення, яке одночасно оптимізує більш ніж одну функцію. У

цьому випадку шукається деякий компроміс, у ролі якого виступає рішення, оптимальне в змісті Парето. При багатокритеріальній оптимізації вибирається не єдина хромосома, що представляє собою закодовану форму оптимального рішення в звичайному змісті, а множину хромосом, оптимальних у змісті Парето. Користувач має можливість вибрати оптимальне рішення з цієї множини. Розглянемо означення рішення, оптимального в змісті Парето (символами *x, y* будемо позначати фенотипи).

**Означення 3.3**

Рішення *x* називається *домінуючим,* якщо існує рішення *y*, не гірше чим *x*, тобто для будь-якої оптимізуємої функції $f_{i,j=1,2,...,m}$,

$f_i(x) \le f_i(y)$ при максимізації функції $f_i$,

$f_i(x) \ge f_i(y)$ при мінімізації функції $f_i$.

**Означення 3.4**

Якщо рішення не доминуємо ніяким іншим рішенням, то воно називається *недоминуємим* або *оптимальним у змісті Парето.*

Існує кілька класичних методів, що відносяться до багатокритеріальної оптимізації. Один з них - це метод *зваженої функції (method of objective weighting),* відповідно до якого оптимізуємі функції $f_i$, з вагами $w_i$ утворюють єдину функцію

$$f(x) = \sum_{i=1}^{m} w_i f_i(x),$$

де $w_i \in [0,1]$ і $\sum_{i=1}^{m} w_i = 1$

Різні ваги дають різні рішення в змісті Парето.

Інший підхід відомий як *метод функції відстані (method of distance function).* Ідея цього методу полягає в порівнянні значень $f_i(x)$ із заданим значенням $y_i$, тобто

$$f(x) = \left( \sum_{i=1}^{m} | f_i(x) - y_i |^r \right)^{\frac{1}{r}}$$

При цьому, як правило, приймається $r = 2$. Це метрика Эвкліда.

Ще один підхід до багатокритеріальної оптимізації зв'язаний з поділом популяції на підгрупи однакового розміру *(sub-populations),* кожна з яких «відповідає» за одну оптимізуєму функцію. Селекція виробляється автономно для кожної функції, однак операція схрещування виконується без урахування границь підгруп.

Алгоритм багатокритеріальної оптимізації реалізовано у програмі **FlexTool**. Селекція виконується турнірним методом, при цьому «краща» особа у кожній підгрупі вибирається на основі функції пристосованості, яка унікальна для даної підгрупи. Схема такої селекції у випадку оптимізації двох функцій представлена на рис. 3.33; на цьому рисунку $F_1$ і $F_2$ позначають дві різні функції пристосованості. Ця схема аналогічна схемі, зображеної на рис. 3.26, з тією різницею, що на більш ранній схемі всі підгрупи оцінювалися по одній і тій же функції пристосованості. «Найкраща» особа з кожної підгрупи змішується з іншими особами, і всі генетичні операції виконуються так само, як у генетичному алгоритмі для оптимізації однієї функції. Схему на рис. 3.33 можна легко узагальнити на більшу кількість оптимізуємих функцій. Програма **FlexTool** забезпечує одночасну оптимізацію чотирьох функцій.

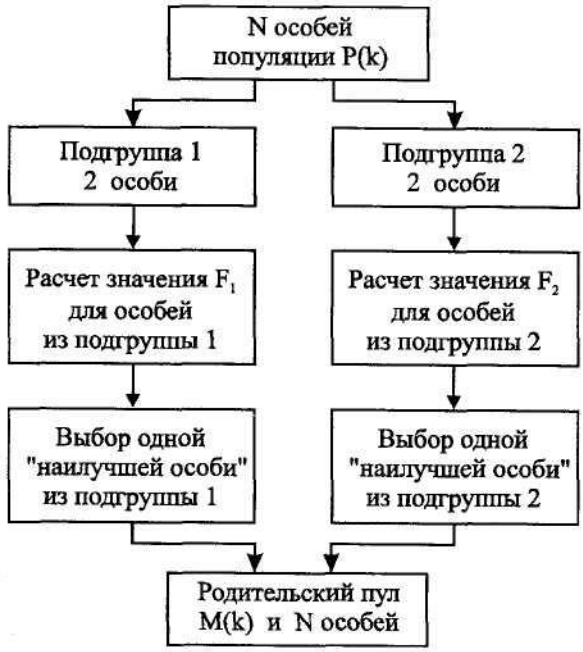



Рис. 3.33. Схема турнірної селекції у випадку багатокритеріальної оптимізації по двох функціях.

### 3.13.8. Генетичні мікроалгоритми

Генетичний мікроалгоритм - це модифікація класичного генетичного алгоритму, яка призначена для рішення задач, що не вимагають великих популяцій і довгих хромосом. Такі алгоритми застосовуються при обмеженому часі обчислень у випадку, коли рішення (не обов'язково глобальне) необхідно знайти швидко. Мова йде про те, щоб не робити трудомістких обчислень, зв'язаних з великою кількістю ітерацій. Генетичні мікроалгоритми зазвичай знаходять трохи гірші рішення, однак заощаджують обчислювальні ресурси комп'ютера. Як приклад можна привести генетичний мікроалгоритм програми **FlexTool.** Він підрозділяється на шість кроків.

1. Сформувати популяцію з числом особей, рівним п'яти. Можна або випадковим чином вибрати всі п'ять хромосом, або зберегти одну «гарну» хромосому, отриману на попередніх ітераціях, і випадковим чином «добрати» чотири інші хромосоми.

2. Розрахувати значення функції пристосованості хромосом у популяції і вибрати кращу хромосому. Позначити її номером 5 і перенести в наступне покоління (елітарна стратегія).

3. Вибрати для репродукції інші чотири хромосоми на основі детермінованного методу турнірної селекції (найкраща хромосома також бере участь у змаганні за право включення її копії в батьківський пул). У ході турнірної селекції хромосоми групуються випадковим чином, при цьому пари, що знаходяться по сусідству, змагаються за чотири місця, що залишилися. Варто звертати увагу на те, щоб батьківська пара не складалася з двох копій однієї і тієї ж хромосоми.

4. Виконати схрещування з ймовірністю 1; ймовірність мутації прийняти рівну 0.

5. Перевірити збіжність алгоритму (з використанням відповідної міри збіжності генотипів або фенотипів). У випадку виявлення збіжності повернутися до кроку 1.

6. Перейти до кроку 2.

Помітимо, що в генетичному мікроалгоритмі розмір популяції передбачається невеликим і фіксованим. Застосовується елітарна стратегія, що запобігає втраті «гарних» хромосом. Оскільки розмір популяції невеликий, то виконується детермінована селекція. Схрещування проводиться з ймовірністю 1. Мутація не потрібна, тому що достатня розмаїтість забезпечується формуванням нової популяції при кожнім «рестарті» алгоритму, тобто у випадку переходу до кроку 1 при виявленні збіжності. Процедура «старту» і «рестарту» алгоритму призначена для запобігання передчасної збіжності; генетичний мікроалгоритм завжди шукає найкращі рішення. Головна мета його застосування полягає в якнайшвидшому знаходженні оптимального (або майже оптимального) рішення.

# 3.14. Моделі генетичних алгоритмів і стратегії відбору і формування нового покоління

Нижче приведено невеликий опис деяких моделей генетичних алгоритмів. При реалізації власного алгоритму дотримуватись цих моделей не обов'язково, тому що дуже багато чого залежить від розв'язуваної задачі, однак, описувані принципи (наприклад, паралелізм) можуть виявитися корисними.

### *1. Canonical GA (J. Holland)*

Дана модель алгоритму є класичною. Вона була запропонована Джоном Холландом у його роботі "Адаптація в природних і штучних середовищах" (1975). Часто можна зустріти опис *простого ГА* (Simple GA, D. Goldberg), він відрізняється від канонічного тим, що використовує або рулеточний, або турнірний відбір. Модель канонічного ГА має наступні характеристики:

- Фіксований розмір популяції.
- Фіксована розрядність генів.
- Пропорційний відбір.
- Особи для схрещування вибираються випадковим чином.
- Одноточковий кросовер і одноточкова мутація.
- Наступне покоління формується з нащадків поточного покоління без "елітизму". Нащадки займають місця своїх батьків.

### *2. Genitor (D. Whitley)*

У даній моделі використовується специфічна стратегія добору. Спочатку, як і покладається, популяція ініціюється і її особи оцінюються. Потім вибираються випадковим чином дві особи, схрещуються, причому виходить тільки один нащадок, що оцінюється і займає місце найменш пристосованої особи. Після

цього знову випадковим чином вибираються 2 особи, і їхній нащадок займає місце особи з найнижчою пристосованістю. У такий спосіб на кожнім кроці в популяції обновляється тільки одна особа. Підводячи підсумки можна виділити наступні характерні риси:

- Фіксований розмір популяції.
- Фіксована розрядність генів.
- Особи для схрещування вибираються випадковим чином.
- Обмежень на тип кросовера і мутації немає.
- У результаті схрещування особей виходить один нащадок, що займає місце найменш пристосованої особи.

### *3. Hybrid algorithm (L. "Dave" Davis)*

Використання гібридного алгоритму дозволяє об'єднати переваги ГА з перевагами класичних методів. Справа в тім, що ГА є робастними алгоритмами, тобто вони дозволяють знаходити гарне рішення, але знаходження оптимального рішення найчастіше виявляється набагато більш важкою задачею в силу стохастичності принципів роботи алгоритму. Тому виникла ідея використовувати ГА на початковому етапі для ефективного звуження простору пошуку навколо глобального экстремума, а потім узявши кращу особу, застосувати один з "класичних" методів оптимізації. Характеристики алгоритму:

- Фіксований розмір популяції.
- Фіксована розрядність генів.
- Будь-які комбінації стратегій відбору і формування наступного покоління
- Обмежень на тип кросовера і мутації немає.
- ГА застосовується на початковому етапі, а потім у роботу включається класичний метод оптимізації.

### *4. Island Model GA*

Уявимо собі наступну ситуацію. У деякому океані є група близьколежачих островів, на яких живуть популяції особей одного виду. Ці популяції розвиваються незалежно і тільки зрідка відбувається обмін представниками між популяціями. Острівна модель ГА використовує описаний принцип для пошуку рішення. Варіант, безумовно, цікавий і є одним з різновидів паралельних ГА. Дана модель генетичного алгоритму має наступні властивості:

- Наявність декількох популяцій, як правило однакового фіксованого розміру.
- Фіксована розрядність генів.
- Будь-які комбінації стратегій добору і формування наступного покоління в кожній популяції. Можна зробити так, що в різних популяціях будуть використовуватися різні комбінації стратегій, хоча навіть один варіант дає різноманітні рішення на різних "островах".
- Обмежень на тип кросовера і мутації немає.
- Випадковий обмін особами між "островами". Якщо міграція буде занадто активною, то особливості острівної моделі будуть згладжені і вона буде не дуже сильно відрізнятися від моделей ГА без паралелізму.

### *5. CHC (Eshelman)*

CHC розшифровується як Cross-population selection, Heterogenous recombination and Cataclysmic mutation. Даний алгоритм досить швидко сходиться через те, що в ньому немає мутацій, використовуються популяції невеликого розміру, і добір особей у наступне покоління ведеться і між батьківськими особами, і між їх нащадками. У силу цього після знаходження деякого рішення алгоритм перезапускається, причому краща особа копіюється в нову популяцію, а особи, що залишилися, є сильною мутацією (мутує приблизно третина бітів у хромосомі) існуючих і пошук повторюється. Ще однією специфічною рисою є стратегія схрещування: всі особи розбиваються на пари, причому схрещуються тільки ті пари, у яких хромосоми особей істотно різна (хеммінгова відстань більша деякої граничної плюс можливі обмеження на мінімальну відстань між крайніми бітами, що

розрізняються,). При схрещуванні використовується так називаний HUX-оператор (Half Uniform Crossover) - це різновид однорідного кросовера, але в ньому до кожного нащадка попадає рівно половина бітів хромосоми від кожного батька. Таким чином, модель має наступні властивості:

- Фіксований розмір популяції.
- Фіксована розрядність генів.
- Перезапуск алгоритму після знаходження рішення.
- Невелика популяція.
- Особи для схрещування розбиваються на пари і схрещуються за умови істотних відмінностей.
- Відбір у наступне покоління проводиться між батьківськими особами і нащадками.
- Використовується половинний однорідний кросовер (HUX).
- Макромутація при перезапуску.

## Стратегії відбору (selection strategies)

ГА представляє із себе ітераційний процес, у якому особи спочатку відбираються для схрещування, потім схрещуються, потім з їхніх нащадків формується нове покоління й усе починається спочатку. Стратегії відбору є складовою частиною ГА і визначають "гідних" для схрещування особей. Нижче розглядаються декілька найбільш розповсюджених стратегій.

### *Пропорційний відбір (Proportional selection)*

При даному виді відбору спочатку підраховується пристосованість кожної особи $f_i$. Після цього знаходять середню пристосованість у популяції $f_{cp}$ як середнє арифметичне значення пристосованості всіх особей. Потім для кожної особи обчислюється відношення $f_i/f_{cp}$. Якщо це відношення більше 1, то особа вважається добре пристосованою і допускається до схрещування, у противному випадку особа, швидше за все, залишиться "за бортом". Наприклад, якщо дріб дорівнює 2.36, то дана особа має подвійний шанс на схрещування і буде мати

ймовірність рівну 0.36 третього схрещування. Якщо ж пристосованість дорівнює 0.54, то особа візьме участь у єдиному схрещуванні з ймовірністю 0.54. Реалізувати це можна, наприклад, у такий спосіб. Нехай є масив двоїчних рядків (популяція) і додатковий масив для особей допущених до схрещування. Визначимо для кожної особи популяції значення описаного вище відношення. Далі, будемо записувати рядки в проміжний масив відповідно до наступного правила: візьмемо цілу частину зрозуміло-якого-відншення і рівно стільки разів запишемо даний рядок у допоміжний масив, після цього за допомогою випадкової величини (ВВ) визначимо, чи будемо ми записувати даний рядок ще раз: якщо ВВ більша дробової частини відношення, то "так", якщо ні, то "ні". Надалі, особи для схрещування вибираються тільки з проміжного масиву випадковим чином, так що, якщо рядок присутній в ньому в декількох екземплярах, то в неї більше шансів залишити слід в історії. Для вже розглянутих прикладів з числами 2.36 і 0.51 ситуація буде виглядати в такий спосіб: перший рядок запишеться в проміжний масив два рази і з ймовірністю 0.36 запишеться в третій раз. Другий же рядок має ймовірність рівну 0.51 того, що вона взагалі буде присутня у допоміжному масиві. Дана стратегія відбору знаменита тим, що саме для неї створена класична теорема схем (от звідки там дріб).

### *Турнірний відбір (Tournament selection)*

Турнірний відбір може бути описаний у такий спосіб: з популяції, що містить *N* рядків, вибирається випадковим чином *t* рядків і кращий рядок записується в проміжний масив (між обраними рядками проводиться турнір). Ця операція повторюється *N* раз. Рядки в отриманому проміжному масиві потім використовуються для схрещування (також випадковим чином). Розмір групи рядків, що відбираються для турніру часто дорівнює 2. У цьому випадку говорять про *двоїчний/парний турнір (binary tournament)*. Узагалі ж *t* називається *чисельністю турніру (tournament size).* Чим більше турнір, тим більш жорсткий варіант селекції, тобто тим менше шансів в особей потрапити в відбір.

Перевагою даної стратегії є те, що вона не вимагає додаткових обчислень і упорядкування рядків у популяції по зростанню пристосованості. Також, вочевидь, такий варіант селекції ближчий до реальності, тому що успішність тієї або іншої особи багато в чому визначається її оточенням, наскільки воно краще або гірше її

### *Добір усіканням (Truncation selection)*

Дана стратегія використовує відсортовану по зростанню популяцію. Число особей для схрещування вибирається відповідно до *порога* T[0; 1]. Поріг визначає яка частка особей, починаючи з найпершої (=самої пристосованої) буде брати участь у відборі. У принципі, поріг можна задати і числом більше 1, тоді він буде просто дорівнювати числу особей з поточної популяції, допущених до відбору. Серед особей, що потрапили "під поріг" випадковим чином *N* раз вибирається сама везуча і записується в проміжний масив, з якого потім вибираються особи безпосередньо для схрещування.
Через те, що в цій стратегії використовується відсортована популяція, час її роботи може бути великим для популяцій великого розміру і залежати також від алгоритму сортування. Існують і інші стратегії відбору, наприклад, з лінійним і експонентним ранжируванням.

## Стратегії формування нового покоління

Після схрещування особей необхідно вирішити проблему про те, які з нових особей ввійдуть у наступне покоління, а які - ні, і що робити з їх предками. Є два способи:

1. Нові особи (нащадки) займають місця своїх батьків. Після чого настає наступний етап, у якому нащадки оцінюються, відбираються, дають потомство і поступаються місцем своїм "дітям".
2. Створюється проміжна популяція, що містить у собі як батьків, так і їхніх нащадків. Члени цієї популяції оцінюються, а потім з них вибираються *N* найкращих, котрі і ввійдуть у наступне покоління.

## 3.15. Перевірка ефективності ГА з використанням тестових функцій

Після того як написаний свій власний ГА вам хочеться знати наскільки він гарний. Нижче приведені деякі тестові функції для ГА. Усі тестові функції можуть мати різне число параметрів (*n*). Тому має сенс запустити алгоритм для оптимізації деякої функції спочатку з невеликим *n* (наприклад, 10 або 20), а потім з *n*=50, 100, 200,... Це дасть можливість перевірити масштабуванність алгоритму. Ефективність роботи ГА прийнято оцінювати кількістю обчислень цільової функції. Чим менше, тим краще. Після деяких функцій приведені результати роботи QGA генетичного алгоритма. Результати цільової функції менші 0.001 теж зараховувалися як знайдений глобальний мінімум. От характеристики QGA:

- Фіксований розмір популяції;
- Фіксована розрядність генів;
- Кількість точок розриву кросовера дорівнює числу генів (на кожен ген приходиться рівно одна точка);
- Для схрещування відбираються 50% популяцій;
- Ймовірність мутації 95%;
- Дві "елітні" особи;
- Видалення однакових особей з популяції (за допомогою мутації);
- Точність обчислень: 0.001;
- Результат підрахований по 50 запускам алгоритму;

Так як генетичні алгоритми використовують стохастичність, то для того, щоб визначити, наскільки ефективний ваш ГА потрібно запустити його на одній і тій же тестовій функції кілька разів і тільки після цього аналізувати результат. Наприклад, QGA ГА для функції Растригіна від 50 змінних знаходить глобальний мінімум у ~40% випадків, використовуючи не більш 10000 обчислень функції. Для деяких функцій є графіки для випадку двох змінних. Графіки представляють "перетин" поверхні функції площиною, що проходить через глобальний мінімум, перпендикулярною однієї з осей координат.

Нижче приводиться набір тестових функцій.

***Sphere model.***

Вважається легкою функцією для будь-якого методу оптимізації.

$$f(x) = \sum_{i=1}^{n} x_i^2$$

*xO*(-5,12; 5,12)

Один мінімум рівний 0 у точці, де $x_i$=0.0.

**Результат QGA**:

*n=10*, кількість нахождень глобального мінімуму (він тут один) 86%, число обчислень цільової функції не більш 1250, максимальне значення 0,008325.

*n=30*, кількість нахождень глобального мінімуму 34%, число обчислень цільової функції не більш 1250, максимальне значення 0,108851.

*n=50*, кількість нахождень глобального мінімуму 46%, число обчислень цільової функції не більш 2500, максимальне значення 0,016291, для схрещування відбиралося 40% популяції.

***Rosenbrock's saddle***

$$f(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$

*xO*(-2,048; 2,048)

Мінімум рівний 0 у точці, де $x_i$=1.0. Функція має велике повільно убутне плато.

**Результат QGA**:

*n=2*, кількість нахождень глобального мінімуму 92%, число обчислень цільової функції не більш 1250, максимальне

значення функції 0,00169118.

*n=4*, кількість нахождень глобального мінімуму 86%, число обчислень цільової функції не більш 1250, максимальне значення функції 0,00504982.

*n=10*, кількість нахождень глобального мінімуму 0%, число обчислень цільової функції не більш 10000, максимальне значення функції 0,0186537, мінімальне значення функції 0,0019092, для схрещування відбиралося 50% популяцій.

*n=10*, кількість нахождень глобального мінімуму 8%, число обчислень цільової функції не більш 10000, максимальне значення функції 0,0127758, для схрещування відбиралося 40% популяцій.

***Step function***

$$f(x) = \sum_{i=1}^{n} |[x_i]|$$

x*O*(-5,12; 5,12)
$|[x_i]|$ - модуль цілої частини
Мінімум рівний 0 у точці, де $x_i$=0.0.

***Gaussian quartic***

$$f(x) = \sum_{i=1}^{n} (x_i^4 + gauss(0,1))$$

x*O*(-1.28; 1.28)
gauss(0,1) - функція, що повертає випадкову величину з нормальним розподілом з математичним очікуванням у 0 і дисперсією рівною 1.
Мінімум рівний 0 у точці, де $x_i$=0.0.

***Rastrigin's function***

Функція зі складним рельєфом. Вважається складною для оптимізації.

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$$

x*O*(-5,12; 5,12)
Мінімум рівний 0 у точці, де $x_i$=0.0.
Локальний мінімум у точці, де одна координата дорівнює 1.0, а інші рівні 0.0

**Результат QGA**:

*n=20*, кількість нахождень глобального мінімуму 16%, число обчислень цільової функції не більш 5000, максимальне значення функції 19,8992.
*n=20*, кількість нахождень глобального мінімуму 26%, число обчислень цільової функції не більш 10000, максимальне значення функції 17,997.
*n=50*, кількість нахождень глобального мінімуму 38%, число обчислень цільової функції не більш 10000, максимальне значення функції 49,869.
Можливо, такі дивні результати обумовлені рельєфом функції Растригіна від 20 і 50 змінних, а може бути, і роботою ГА. Наприклад, з використанням генетичного алгоритму **QGA** ніяк не вдається точно апроксимувати квадратичною функцією (всього три параметри) набір з п'яти точок, що строго належать графікові параболи.

***Schwefel's (Sine Root) Function***

$$f(x) = 418{,}9829\,n + \sum_{i=1}^{n} (-x_i \sin(\sqrt{|x_i|}))$$

x*O*(-500; 500)
Мінімум рівний 0 у точці, де $x_i$=420.9687.
Локальний мінімум у точці, де одна координата дорівнює -302.5232, а інші рівні 420.9687. Так як локальний мінімум знаходиться далеко від глобального, тобто є небезпека, що

алгоритм "зіб'ється зі шляху". Кут під знаком синуса виходить у радіанах.

***Griewangk's Function***

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\frac{x_i}{\sqrt{i}}$$

x*O*(-600; 600)
Мінімум рівний 0 у точці, де $x_i$=0.0.
При *n*=10 є ще 4 локальних мінімуми рівних 0,0074 приблизно в точці (+-$P_i$, +-$P_i$*sqrt(2), 0, ..., 0).

***Ackley's Function***

$$f(x) = 20 + e - 20e^{-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$

x*O*(-30; 30)
Мінімум рівний 0 у точці, де $x_i$=0.0.

## 3.16. Приклади оптимізації функції за допомогою програми FlexTool

Застосування генетичного алгоритму для оптимізації різних функцій будемо ілюструвати прикладами, реалізованими на IBM PC сумісному комп'ютері з використанням програми **FlexTool**.
Програма **FlexTool** дозволяє вибирати різні варіанти генетичного алгоритму - такі, як класичний *(regular),* з частковою заміною популяції *(steady-state)* або мікроалгоритм *(micro).* Крім того, надається можливість вибрати метод селекції (рулетка, турнірний або ранговий), а також кількість точок схрещування і спосіб кодування (двоїчне або логарифмічне). У даному випадку

під двоїчним кодуванням розуміється спосіб, що застосовувався в прикладах 3.1, 3.5 і 3.6, а логарифмічне кодування було представлено в п. 3.13.4. Програма **FlexTool** також дозволяє одночасно оптимізувати кілька функцій і містить алгоритм створення ніш для знаходження більш ніж одного оптимуму (крім єдиного глобального рішення). **FlexTool** взаємодіє з програмою **MATLAB** і ***запускається в її вікні***. Для визначення оптимізуємих функцій використовуються файли з расширенням *.m* (так називані *m-файли*), які служать для запису математичних функцій у програмі **MATLAB.** У робочому вікні **MATLAB'**а також можна зчитувати значення різних змінних, які використовуються програмою **FlexTool,** що дає, зокрема, можливість перегляду популяцій хромосом.

*Програма **FlexTool** вимагає від користувача ввести розмірність популяцій, ймовірності схрещування і мутації, інтервали зміни параметрів задачі (простору пошуку), а також умови останова алгоритму*. Надається можливість перервати виконання алгоритму в довільний момент часу з наступним поновленням його роботи з точки переривання (так називаний *warm-start)* або з початкової точки *(cold-start).*

***Розглянемо приклади оптимізації функції однієї, двох і декількох змінних***. Як ілюстрації беруться графіки більшості оптимізуємих функцій (отримані засобами програми **MATLAB)** і сформовані програмою **FlexTool** графіки функцій пристосованості для конкретних ітерацій генетичного алгоритму.
Приклад 3.12 демонструє спосіб рішення задачі, розглянутої в прикладі 3.5, але при використанні генетичного алгоритму, реалізованого в програмі **FlexTool.** Для такої простої задачі найкраще підходить генетичний мікроалгоритм, представлений у п. 3.13.8. На відміну від приклада 3.5, у прикладі 3.12 шукається мінімум функції, заданою формулою (3.1).

**Приклад 3.12**

За допомогою генетичного мікроалгоритму програми **FlexTool** знайти мінімум функції, заданою формулою (3.1), тобто

$$f(x) = 2x^2 + 1$$

для цілих значень *x* в інтервалі від 0 до 31.

Генетичний мікроалгоритм програми **FlexTool** виконується на популяції з розмірністю, рівною п'яти. Селекція виконується детермінованим турнірним методом із застосуванням елітарної стратегії, завдяки якій краща хромосома поточної популяції

завжди переходить у наступне покоління. Виконується одноточкове схрещування. Ймовірність схрещування приймається рівною 1, а ймовірність мутації - рівною 0.

Довжина хромосом дорівнює п'яти, що очевидно випливає з можливості кодування 32 цілих чисел (для зазначеного інтервалу зміни змінної *x)* п'ятьма бітами двоїчної послідовності. У програмі **FlexTool** застосовується двоїчне кодування, аналогічне представленому в прикладах 3.1 і 3.5, однак для зручності програмної реалізації використовується зворотний код, тобто на першій позиції знаходиться найменш значущий біт, а на останній - найбільш значущий. Такий спосіб запису прямо протилежний повсюдно розповсюдженої формі запису двоїчних чисел.

Істотним елементом виконання генетичного мікроалгоритму вважається процедура «рестарту», тобто запуску його з початкової точки при виявленні збіжності (п. 3.13.8). У програмі **FlexTool** рестарт виконується періодично після виконання встановленої кількості ітерацій. За замовчуванням воно дорівнює 7, приклади виконувалися саме при цьому значенні.

Вихідна популяція - на першій ітерації генетичного мікроалгоритму - складається з наступних хромосом:

[01100] [11000] [01111] [10101] [11001]

зі значеннями фенотипів

6 3 30 21 19,

які є дійсними цілими числами з інтервалу від 0 до 31.

Найменше значення функції пристосованості в цій популяції має хромосома, що стїть на другому місці, з фенотипом, рівним трьом. Воно дорівнює 19. Найбільше значення функції пристосованості в хромосоми, що стїть на третім місці, з фенотипом, рівним 30, складає 1801. Легко підрахувати, що середнє значення функції пристосованості буде дорівнювати 699,8.

Популяція, яка отримана в результаті селекції і схрещування, стає поточною для наступної (другої) ітерації. Вона характеризується середнім значенням функції пристосованості, рівним 173,8. Помітно, що це значення менше, ніж розраховане для першого покоління. Найбільше значення функції пристосованості, рівне 723, має хромосома з фенотипом 19. Це найгірша особа даної популяції. «Найкраща на сучасний момент» хромосома з фенотипом, рівним двом, має мінімальне значення функції пристосованості, що дорівнює дев'яти.

У третім поколінні середнє значення функції пристосованості зменшується до 13. Найбільше значення для хромосоми з фенотипом, рівним трьом, складає 19, а найменше значення функції пристосованості для цієї популяції, також, як і для попередньої, дорівнює дев'яти. «Найкращою на сучасний момент» продовжує залишатися хромосома з фенотипом, рівним двом.

У наступних чотирьох поколіннях (від четвертого до сьомого) середнє значення функції пристосованості по популяції збігається з найбільшим і найменшим значеннями, рівними дев'яти. Це означає, що популяція складається з однакових хромосом з фенотипами, рівними двом. Отже, спостерігається збіжність до рішення, що не є оптимальним.

До цього моменту алгоритм виконувався аналогічно класичному генетичному алгоритмові, причому селекція проводилася детермінованим турнірним методом зі збереженням найкращої хромосоми з популяції (елітарна стратегія).

Після семи ітерацій починається новий цикл виконання генетичного мікроалгоритму. Виконується його «рестарт», тобто повторний запуск алгоритму з початкової точки - випадкового вибору чотирьох нових хромосом для включення в популяцію. З особей попереднього покоління зберігається тільки одна - «найкраща на сучасний момент» хромосома з фенотипом, рівним двом.

Після селекції і схрещування в черговому (дев'ятому) поколінні одержуємо популяцію із середнім значенням функції пристосованості, рівним 481,4. Найбільше значення функції пристосованості, рівне 1579, має хромосома з фенотипом 28, а найменше значення, рівне дев'яти, - з фенотипом, рівним двом.

На наступних п'ятьох ітераціях другого циклу генетичного мікроалгоритму ми знову одержуємо популяцію, що складається з однакових хромосом з фенотипом, рівним двом, для яких середнє, найбільше і найменше значення функції пристосованості рівні дев'яти. Отже, знову спостерігається збіжність до рішення, що не є оптимальним.

З п'ятнадцятого покоління починається черговий (третій) цикл виконання генетичного мікроалгоритму. Він також відкривається випадковим вибором чотирьох нових хромосом і формуванням популяції зі збереженням «найкращої на сучасний момент» особи з фенотипом, рівним двом.

На дев'ятнадцятій ітерації генетичного мікроалгоритму, тобто в п'ятому поколінні третього циклу ( що складається із семи ітерацій) виникає збіжність до оптимального рішення, яким виявляється хромосома з фенотипом, рівним 0. Середнє, найбільше і найменше значення функції пристосованості по всій популяції з 19 по 21 покоління залишається рівним 1.

Далі виконуються чергові цикли по сімох ітераціях, що починаються «рестартом» алгоритму, і в кожнім з них спостерігається збіжність до оптимального рішення, тобто до хромосоми з фенотипом, рівним 0.

Звичайно, задачу з приклада 3.12 можна вирішити з застосуванням генетичного алгоритму з довільною розмірністю популяцій, при завданні користувачем значень ймовірностей схрещування і мутації, а також при виборі їм одного з методів селекції (рулетки, турнірного або рангового). Таким чином, замість мікроалгоритму можна застосувати реалізований у програмі **FlexTool** звичайний генетичний алгоритм *(regular).* Однак необхідно відзначити, що при його використанні на популяціях малої розмірності з ймовірностями схрещування і мутації, відповідно рівними 1 і 0, а також при виконанні селекції методом рулетки (так, як у прикладі 3.5) доволі часто встає проблема передчасної збіжності цього алгоритму. Результат значно поліпшується, якщо ймовірність мутації відрізняється від нуля (наприклад, якщо вона приймається рівною 0,1). Незважаючи на те, що мутація грає виразно другорядну роль стосовно схрещування, вона виявляється необхідною, оскільки забезпечує розмаїтість хромосом у популяції.

Необхідно ще раз підкреслити, що при виконанні генетичного мікроалгоритму розмаїтість у популяції досягається завдяки процедурі рестарту алгоритму у випадку виявлення його збіжності. У цьому випадку мутація виявляється зайвою.

Наступний приклад відноситься до задачі, схожої на задачу приклада 3.12. Мінімізується та ж функція, однак змінна *x* приймає дійсні значення з інтервалу [-5, 5]. З цієї причини простір пошуку виявляється більшим, ніж у прикладі 3.12. Для рішення даної задачі застосуємо звичайний генетичний алгоритм програми **FlexTool** і селекцію методом рулетки. Помітимо, що

цей метод, що придатний і для селекції у випадку максимізації функції, у програмі **FlexTool** може використовуватися тільки для задач мінімізації. Це обумовлено адаптацією програми до вимог користувачів, що частіше вирішують задачі мінімізації (наприклад, похибок, витрат і т.п.), чим задачі максимізації.

**Приклад 3.13**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум функції, заданою формулою (3.1) для $x \in$ [-5, 5] з точністю до 0,1.

Поставлена задача вирішується шляхом використання звичайного генетичного алгоритму (regular) із селекцією методом рулетки. Помітимо, що пошук проводиться в просторі, що складається з 100 можливих рішень. Довжина хромосом (відповідно до формули 3.4)) повинна бути рівною семи. Нехай розмірність популяції складає 11 особей. Будемо застосовувати одноточкове схрещування з ймовірністю 0,9, а ймовірність мутації установимо рівної 0,1.

Вихідна популяція складається з 11 хромосом довжиною сім бітів, що відповідають наступним фенотипам:

3,4 2,4 2,0 5,0 1,4 0,1 3,0 -2,3 2,3 5,0 -4,8

Найкраще рішення, тобто хромосома з фенотипом, рівним 0, для якої значення функції пристосованості складає 1, отримано на сьомій ітерації алгоритму. На першій ітерації найбільше значення функції пристосованості по всій популяції дорівнює 51, середнє - 22,0745, а найменше - 1,02. Однак у сьомому поколінні найбільше значення функції пристосованості дорівнює 45,18, середнє по популяції складає 5,9909, а найменше значення відповідає кращій хромосомі і дорівнює 1. Графік на рис. 3.34 показує найменше значення функції пристосованості в популяції на послідовних ітераціях генетичного алгоритму.

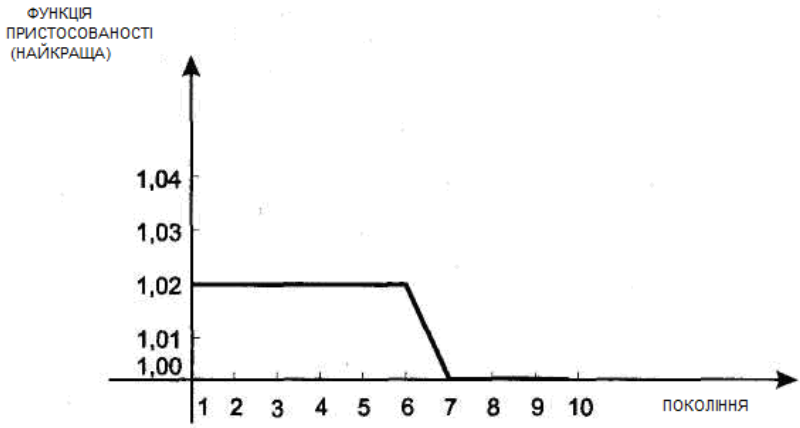



Рис. 3.34. Найменше значення функції пристосованості в популяції на послідовних ітераціях генетичного алгоритму для приклада 3.13.

Аналогічно можна застосовувати генетичний алгоритм для знаходження рішення з ще більшою точністю, наприклад, для двох або трьох знаків після коми. При цьому буде відповідно розширюватися простір пошуку і збільшуватися довжина хромосом.

Наступний приклад відноситься до задачі знаходження максимуму функції, що має локальні максимуми. Покажемо, що генетичний алгоритм знаходить глобальний оптимум. Для пошуку максимуму будемо застосовувати реалізований у програмі **FlexTool** турнірний метод, оскільки метод колеса рулетки в ній працює тільки для задач мінімізації.

У програмі **FlexTool** за замовчуванням прийняті наступні (найкращі для більшості розв'язуваних задач) значення параметрів генетичного алгоритму:

- розмірність популяції = 77,
- ймовірність схрещування = 0,77,
- ймовірність мутації = 0,0077.

У прикладах, що далі приводяться, використовуються зазначені ймовірності схрещування і мутації, однак розмірність

популяцій буде вибиратися в залежності від розмірності розв'язуваних задач.

**Приклад 3.14**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум функції, заданою формулою

$$f(x)=\frac{1}{(x-0{,}3)^2+0{,}01}+\frac{1}{(x-0{,}9)^2+0{,}04}-6$$

для $x\in$ [-1, 2] з точністю до 0,01.

Графік оптимізуємої функції представлено на рис. 3.35.

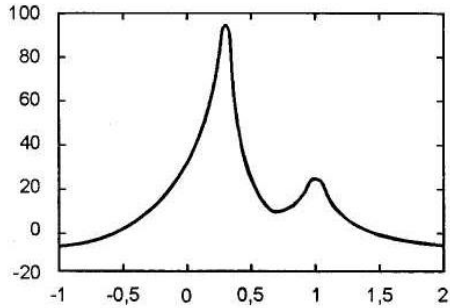



Рис. 3.35. Графік функції *f(x)* із приклада 3.14.

Звертвємо увагу на те, що функція має не тільки глобальний максимум, але і локальні оптимуми. Для знаходження глобального максимуму застосовується генетичний алгоритм із турнірною селекцією в підгрупах по дві особи. Розмірність популяції дорівнює 21; ймовірності схрещування і мутації складають, відповідно 0,77 і 0,0077; використовується одна точка схрещування. Хромосоми складаються з дев'яти генів зі значеннями 0 або 1. На рисунку 3.36 більш світлою лінією показана динаміка зміни «найкращого», у розглянутому випадку - максимального значення функції пристосованості в популяції при збільшенні кількості поколінь.

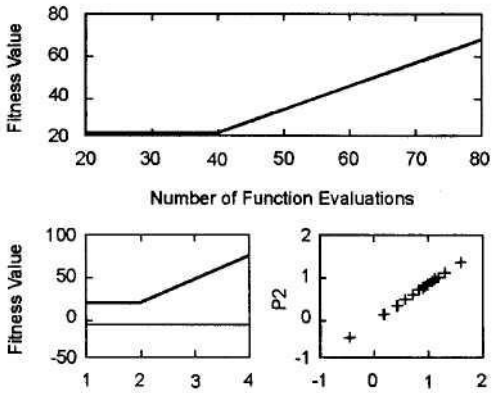



Рис. 3.36. Графіки, що показують значення функції пристосованості для перших чотирьох поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.14.

Більш темна лінія на другому графіку ілюструє зміну «найгіршого» значення функції пристосованості в популяції при послідовній зміні поколінь. На осі ординат обох графіків зазначені значення функції пристосованості *(fitness value).* На осі абсцис другого графіка зазначені номери поколінь, що відповідають числам на осі абсцис першого графіка, помноженим на розмірність популяції, тобто на 21.

З графіків видно, що «найкраща» хромосома в першому і другому поколіннях характеризується значенням функції пристосованості, близьким до 20. У той же час значення функції пристосованості «найгіршої» хромосоми приблизно дорівнює -5. У четвертому поколінні це значення наблизилося до -4, а значення функції пристосованості «найкращої» хромосоми зросло до 70. У нижній правій частині рис. 3.36 показані значення параметра задачі, тобто змінної *x* зі значеннями в інтервалі від -1 до 2; іншими словами, це фенотипи, що відповідають окремим хромосомам розглянутої популяції. Помітно, що популяція характеризується значною розмаїтістю. На рис. 3.37 популяція більш однорідна, а на рис. 3.38 зафіксовано тільки одне значення параметра, рівне 0,3; це означає, що всі хромосоми в популяції рівні між собою.

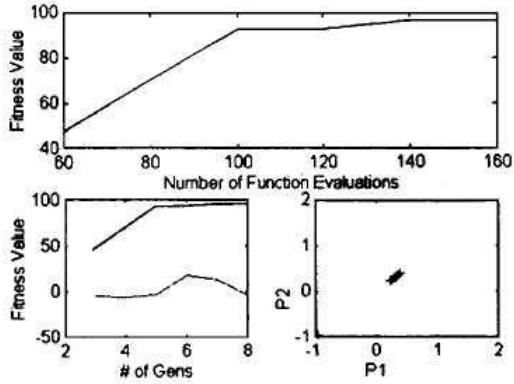



Рис. 3.37**.** Графіки, що показують значення функції пристосованості з третього по восьме покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.14.

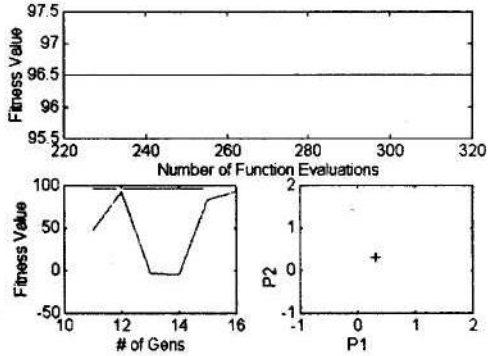



Рис. 3.38**.** Графіки, що показують значення функції пристосованості для останніх чотирьох поколінь у генетичному алгоритмі програми **FlexTool** у прикладі 3.14.

У розглянутому прикладі ми маємо справу тільки з одним параметром задачі P1, що збігається з P2, тому всі точки розташовуються уздовж прямої так, як це показано на рис. 3.36 і 3.37.

Графіки на рис. 3.37 демонструють збільшення «найкращого» значення функції пристосованості від близько 47 у третім поколінні до приблизно 70 у четвертому і приблизно 93 - у п'ятому поколінні, а в сьомому поколінні досягається значення,

рівне 96,5. Це максимальна величина, що залишається незмінною в наступних поколіннях, як видно на рис. 3.38. Таким чином, на сьомій ітерації алгоритму знайдено «найкраще» рішення - хромосома зі значенням фенотипу, рівним 0,3, для якого значення функції пристосованості складає 96,5. Нижні графіки на рис. 3.37 і 3.38 також показують, як змінюється «найгірше» значення функції пристосованості від 3 до 16 покоління. У шістнадцятому поколінні це значення дорівнює 92,81. Середнє значення функції пристосованості в популяціях чергових поколінь змінюється від 2,24 на першій ітерації алгоритму до 96,32 на шістнадцятій ітерації, що показує таблиця 3.4.

**Таблиця 3.4.** Середнє значення функції пристосованості в популяціях чергових поколінь генетичного алгоритму програми **FlexTool** для приклада 3.14

| НОМЕР ПОКОЛІННЯ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| СЕРЕДНЄ ЗНАЧЕННЯ ФУНКЦІЇ ПРИСТОСОВАНОСТІ | 2,24 | 7,76 | 14,15 | 20,08 | 32,49 | 50,59 | 65,47 | 75,89 |
| НОМЕР ПОКОЛІННЯ | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| СЕРЕДНЄ ЗНАЧЕННЯ ФУНКЦІЇ ПРИСТОСОВАНОСТІ | 85,66 | 89,61 | 93,39 | 96,15 | 86,98 | 91,66 | 95,86 | 96,32 |

**Приклад 3.15**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум функції, заданою формулою

$$f(x)=-x-\sin(10\pi x)+1$$

для $x \in [-1, 2]$ з точністю до 0,0001.

Графік оптимізуємої функції представлено на рис. 3.39. Ця функція має багато локальних оптимумів.

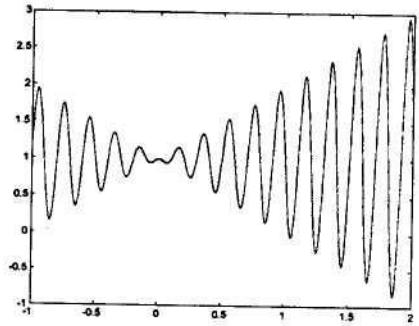


Рис. 3.39. Графік функції *f(x)* із приклада 3.15.

Для знаходження глобального мінімуму на інтервалі від -1 до 2 застосовується (так само, як і в прикладі 3.14) генетичний алгоритм із турнірною селекцією в підгрупах по дві особи. Виконується одноточкове схрещування з ймовірністю 0,77; ймовірність мутації дорівнює 0,0077. Розмірність популяції збільшена до 55. Довжина хромосом у цьому випадку складає 15 бітів. Графіки, що демонструють зміни значень функції пристосованості при зміні перших чотирьох поколінь, за аналогією з прикладом 3.14 зображені на рис. 3.40, а графіки наступних змін - на рис. 3.41.

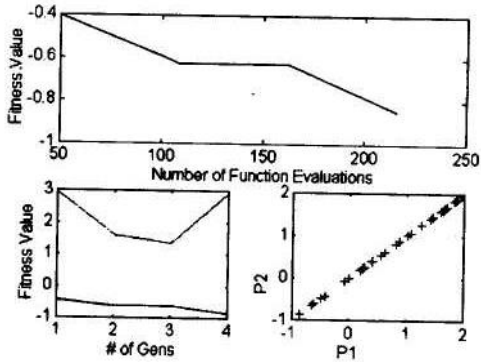



Рис. 3.40. Графіки, що показують значення функції пристосованості для перших чотирьох поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.15.

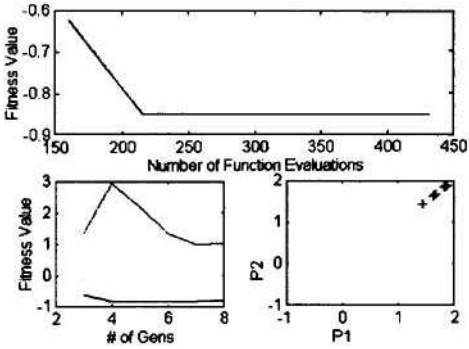



Рис. 3.41. Графіки, що показують значення функції пристосованості з третього по восьме покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.15.

«Найкраще» рішення дає хромосома зі значенням фенотипу 1,85, для якої функція пристосованості дорівнює -0,8503. Це рішення отримане в десятому поколінні. У таблицю 3.5 зібрані «найкращі» значення функції пристосованості на перших десятьох ітераціях алгоритму.

**Таблиця 3.5.** «Найкраще» значення функції пристосованості для перших десяти поколінь генетичного алгоритму програми **FlexTool** для приклада 3.15

| НОМЕР ПОКОЛІННЯ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| "НАЙКРАЩЕ" ЗНАЧЕННЯ ФУНКЦІЇ ПРИСТОСОВАНОСТІ | –0,4197 | –0,6247 | –0,6256 | –0,8498 | –0,8499 |
| НОМЕР ПОКОЛІННЯ | 6 | 7 | 8 | 9 | 10 |
| "НАЙКРАЩЕ" ЗНАЧЕННЯ ФУНКЦІЇ ПРИСТОСОВАНОСТІ | –0,8499 | –0,8502 | –0,8502 | –0,8502 | –0,8503 |

Значення фенотипу хромосоми з «найкращим» значенням функції пристосованості для другого покоління дорівнює 1,645, а для четвертого і наступного поколінь - 1,85.

**Приклад 3.16**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум функції двох змінних

$$f(x_1,x_2) = x_1^2 + x_2^2$$

для $x_1, x_2 \in$ [-10, 10] з точністю до 0,0001.
Графік оптимізуємої функції представлено на рис. 3.42.

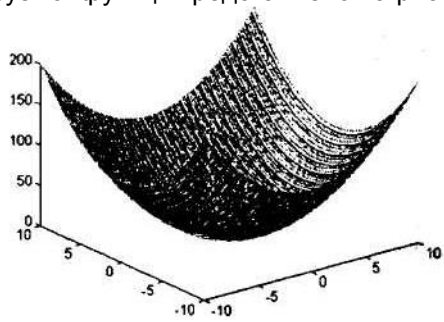



Рис. 3.42**.** Графік функції $f(x_1, x_2)$ із приклада 3.16.

Ця функція має один мінімум, рівний 0, у точці (0, 0).
Застосовується генетичний алгоритм із турнірною селекцією в підгрупах по дві особи, з одною точкою схрещування і прийнятими за замовчуванням значеннями ймовірностей схрещування 0,77 і мутації 0,0077, а також з розмірністю популяції, рівною 77.

Довжина хромосом складає 36 бітів, при цьому кожну змінну $x_1, x_2$ представляють 18 генів. Графіки, що демонструють зміни значень функції пристосованості при зміні перших чотирьох поколінь при виконанні генетичного алгоритму, зображені на рис. 3.43, а графіки наступних змін - на рис. 3.44 -3.47.

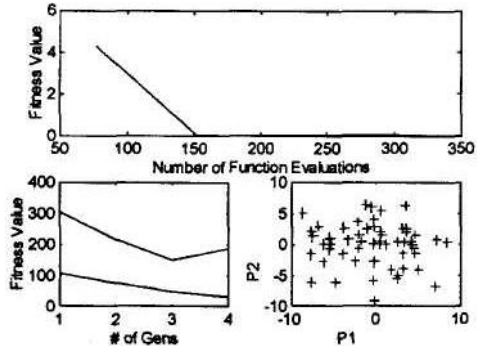


Рис. 3.43. Графіки, що показують значення функції пристосованості для перших чотирьох поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.16.

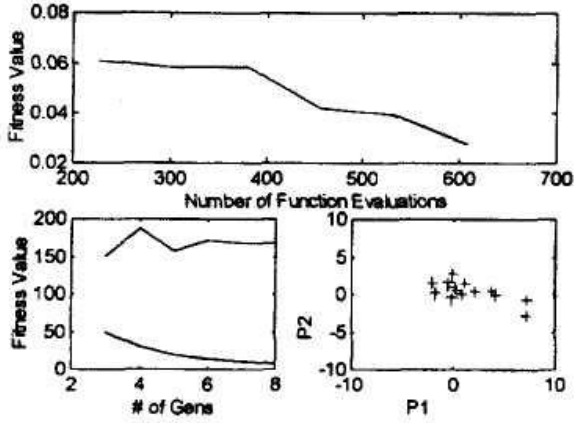



Рис. 3.44. Графіки, що показують значення функції пристосованості з другого по восьме покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.16.

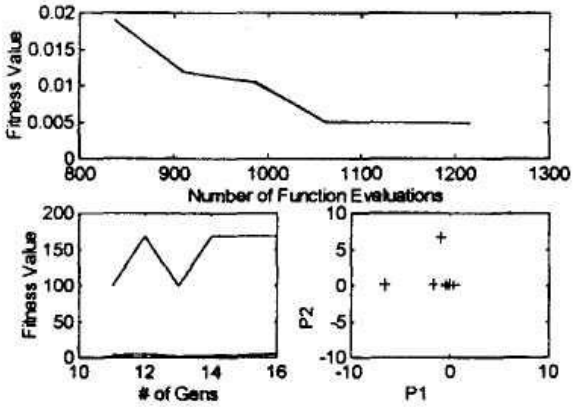


Рис. 3.45. Графіки, що показують значення функції пристосованості з десятого по шістнадцяте покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.16.

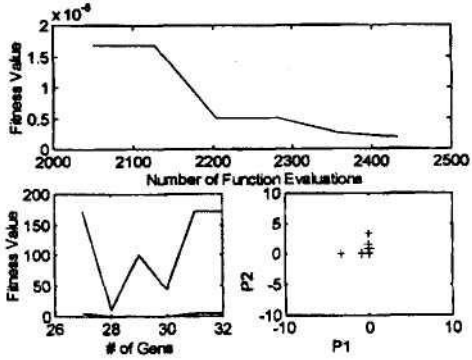



Рис. 3.46. Графіки, що показують значення функції пристосованості з двадцять шостого по тридцятьох друге покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.16.

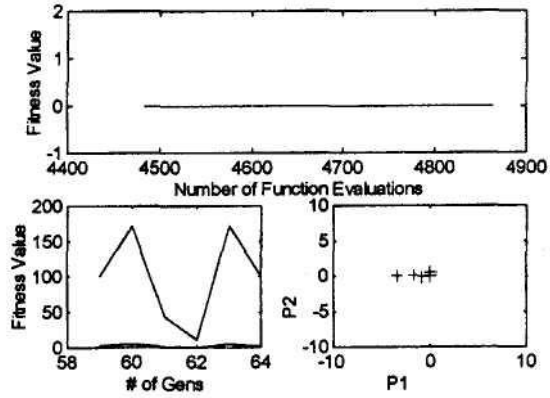



Рис. 3.47. Графіки, що показують значення функції пристосованості з п'ятдесятьох дев'ятого по шістдесяти четверте покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.16.

За графіком видно, як змінюється «найкраще» значення функції пристосованості - від значення 4,3543 у першому

поколінні (по 77 розрахункових точках) до нульового значення. В другому і третім поколіннях ця функція мала значення 0,0608, у четвертому і п'ятому - 0,0586, у шостому - 0,0421; у сьомому - 0,0397, у восьмому - 0,0278, у дев'ятому - 0,0192, у десятому і одинадцятому - 0,0191, у дванадцятому - 0,0119, у тринадцятому - 0,0105, з чотирнадцятого по шістнадцяте - 0,0050, у сімнадцятому і вісімнадцятому - 0,0011, у дев'ятнадцятому - 0,0001, а в двадцятому і наступному поколіннях - значення 0,0000. Весь комплекс змін показано на рис. 3.49, а перераховані «найкращі» значення функції пристосованості виділені на рис. 3.48. Таким чином, у результаті виконання генетичного алгоритму «найкраще» рішення знайдене в двадцятому поколінні.

```
mini =

Columns 1 through 7

  4.3543   0.0608   0.0608   0.0586   0.0586   0.0421   0.0397

Columns 8 through 14

  0.0278   0.0192   0.0191   0.0191   0.0119   0.0105   0.0050

Columns 15 through 21

  0.0050   0.0050   0.0011   0.0011   0.0001   0.0000   0.0000

Columns 22 through 28

  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Columns 29 through 35

  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Columns 36 through 42

  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Columns 43 through 49

  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Columns 50 through 56

  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Columns 57 through 63

  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Column 64

  0.0000
```

Рис. 3.48. Перелік найкращих значень функції пристосованості в популяціях з першого по шістдесяти четверте покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.16.

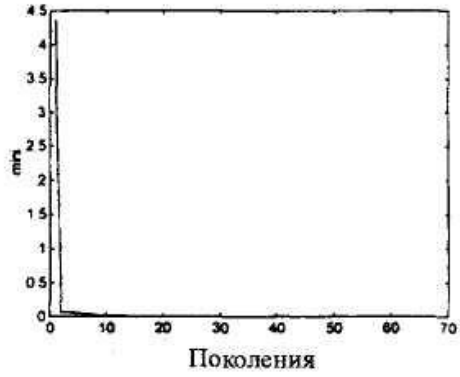



Рис. 3.49. Динаміка зміни «найкращих» значень функції пристосованості при послідовній зміні поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.16.

Нижні ліві графіки на рис. 3.43 - 3.47 показують зміни «найгіршого» (верхня крива) і середнього значення функції пристосованості особей популяції при зміні поколінь. Зміна «найкращого» значення функції пристосованості на цих графіках майже непомітна, оскільки відповідна крива практично збігається з горизонтальною віссю, тому що значення близькі до 0. Комплексна динаміка середніх значень функції пристосованості протягом усіх поколінь демонструється на рис. 3.50.

Рис. 3.50. Динаміка зміни середніх значень функції пристосованості при послідовній зміні поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.16.

Звертаємо увагу на те, що значення функції пристосованості «найгірших» хромосом в окремих популяціях досить великі і, мабуть, значно відрізняються від оптимального значення. Динаміка зміни «найгірших» значень функції пристосованості при зміні поколінь показана на рис. 3.51.

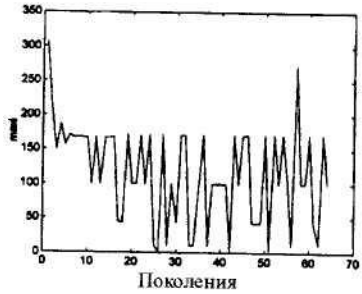



Рис. 3.51**.** Динаміка зміни «найгірших» значень функції пристосованості при послідовній зміні поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.16.

На правих нижніх графіках рисунків 3.43 - 3.47 показані значення параметрів задачі P1 і P2, що відповідають змінним $x_1$, $x_2$, тобто фенотипи (в інтервалі від -10 до 10). Найбільша розмаїтість особей спостерігається в початкових поколіннях. У наступному в складі популяцій з'являється усе більше однакових хромосом.

Нагадаємо, що «найкраще» рішення, рівне 0,0000, отримано в двадцятому поколінні (рис. 3.48). Однак варто звернути увагу на динаміку зміни «найкращого» значення функції пристосованості, показану на рис. 3.46. Це значення знаходилося в інтервалі від 0 до 0,0002, починаючи з 27 і по 32 покоління (тобто з 27*77 до 32*77 розрахункової точки функції пристосованості). Для «найкращої» хромосоми зі значеннями фенотипів (після 64 поколінь), рівними 0,0000 і 0,0001, значення функції $f(x_1, x_2)$ складає 0,00000001.

**Приклад 3.17**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум функції двох змінних

$$f(x_1,x_2) = x_1^2+(1/2)x_2^2+3$$

для $x_1, x_2 \in$ [-10, 10] з точністю до 0,001.

Дана задача аналогічна попередній (приклад 3.16). Графік оптимізуємої функції представлений на рис. 3.52. Ця функція має один мінімум, рівний 3, у точці (0, 0).

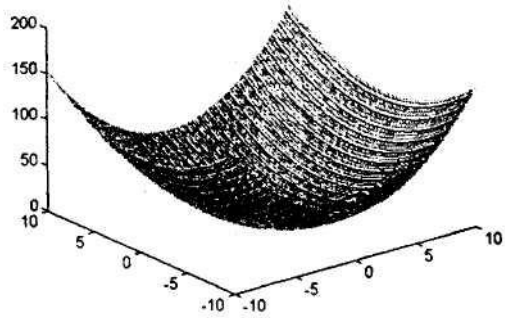



Рис. 3.52**.** Графік функції $f(x_1, x_2)$ із приклада 3.17.

Застосовується генетичний алгоритм із турнірною селекцією в підгрупах по дві особи з однієї або двома точками схрещування, а також із селекцією по методу рулетки з одною точкою схрещування. Також як і в попередньому прикладі, використовуються прийняті за замовчуванням значення ймовірностей схрещування 0,77 і мутації 0,0077, а також розмірність популяції, рівна 77. У цьому випадку з урахуванням меншої необхідної точності довжина хромосом складає 22 біта - по 11 генів на кожну змінну.

Через подібність розглянутого і попередніх прикладів графіки зміни значень функції пристосованості для конкретних поколінь не приводяться. Вони аналогічні графікам на рис. 3.43 - 3.47. Принципове розходження полягає в тім, що в даному випадку «найменше» значення функції пристосованості прагне до значення, рівному 3.

У даному випадку становлять інтерес графіки загальної динаміки зміни «найкращого» значення функції пристосованості (подібні до графіка на рис. 3.49) для використаних методів селекції. Графік для турнірної селекції з одною точкою

схрещування показано на рис. 3.53, а з двома точками схрещування - на рис. 3.54.

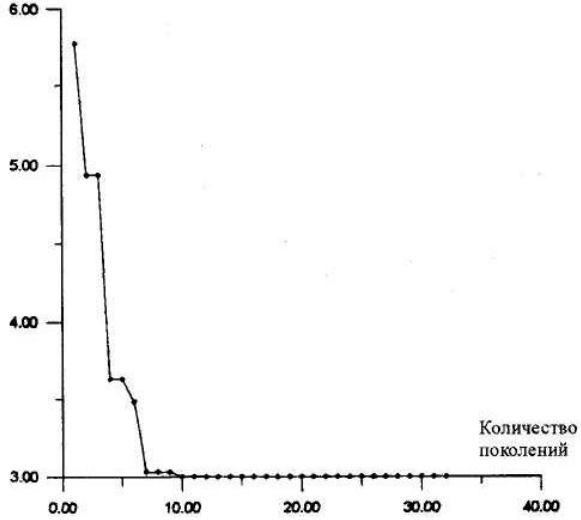



Рис. 3.53**.** Динаміка зміни «найкращих» значень функції пристосованості при послідовній зміні поколінь у генетичному алгоритмі з турнірною селекцією й одною точкою схрещування з приклада 3.17.

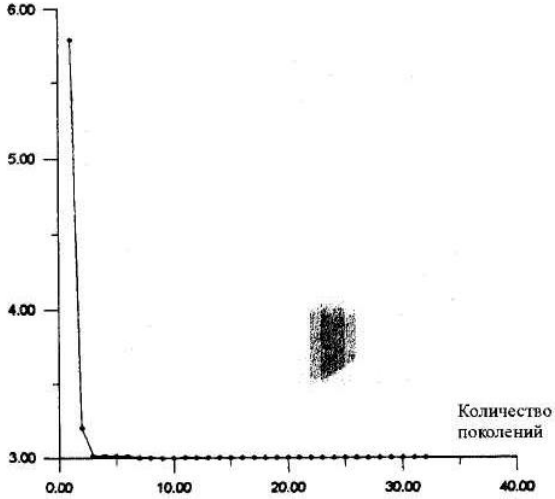


Рис. 3.54. Динаміка зміни «найкращих» значень функції пристосованості при послідовній зміні поколінь у генетичному алгоритмі з турнірною селекцією і двома точками схрещування з приклада 3.17.

Помітно, що в другому випадку «найкраще» рішення (тобто значення функції пристосованості, рівне 3) було знайдено швидше. Аналогічний графік для селекції методом рулетки представлений на рис. 3.55.

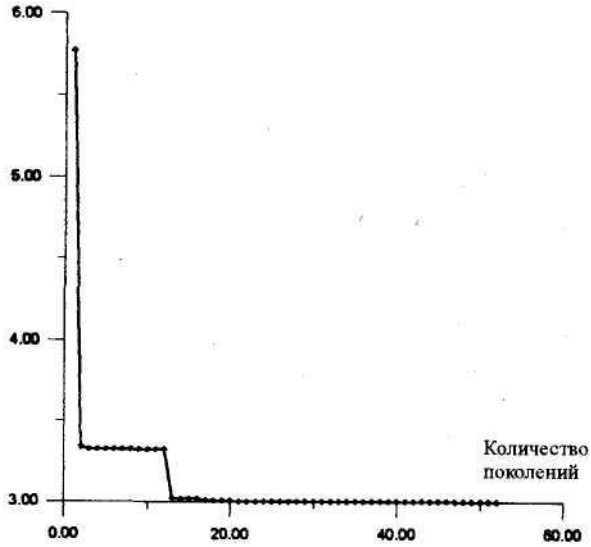



Рис. 3.55**.** Динаміка зміни «найкращих» значень функції пристосованості при послідовній зміні поколінь у генетичному алгоритмі із селекцією методом рулетки з приклада 3.17.

Можна зробити висновок, що турнірний метод дозволяє швидше знаходити мінімальне значення оптимізуємої функції, чим метод рулетки.

**Приклад 3.18**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум функції двох змінних

$$f(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

для $x_1, x_2 \in [-10, 10]$ з точністю до 0,001.

Графік оптимізуємої функції представлений на рис. 3.56.

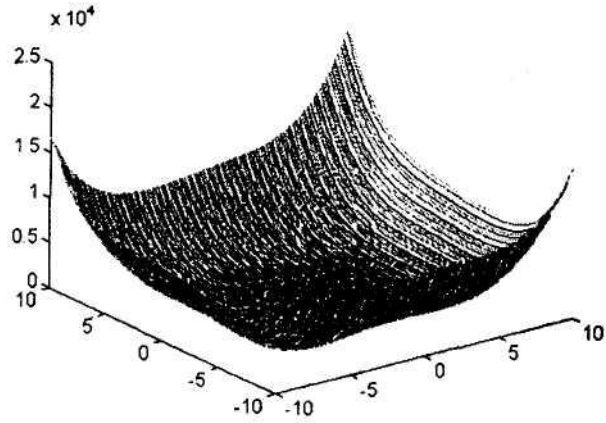



Рис. 3.56**.** Графік функції $f(x_1, x_2)$ із приклада 3.18.

Ця функція має мінімум, рівний 0, у наступних точках: (3, 2), (3,58, -1,85), (-2,80, 3,13), (-3,78, -3,28).

У випадку, коли функція має мінімальне (або максимальне) значення в декількох різних точках, і це значення є глобальним оптимумом, генетичний алгоритм знаходить, як правило, одну з цих точок. Повторний запуск алгоритму може принести той же самий результат або знайти координати іншої точки з таким же оптимальним значенням функції.

У прикладі спочатку застосовувався генетичний алгоритм із турнірною селекцією в підгрупах по дві особи з одною точкою схрещування і прийняті за замовчуванням значення ймовірностей схрещування 0,77 і мутації 0,0077, а також розмірність популяції, рівна 77. Найкращим рішенням, знайденим у цих умов, стала точка з координатами (-3,78, -3,28).

Динамічні зміни «найкращого» значення функції пристосованості показують графіки на рис. 3.57 - 3.59.

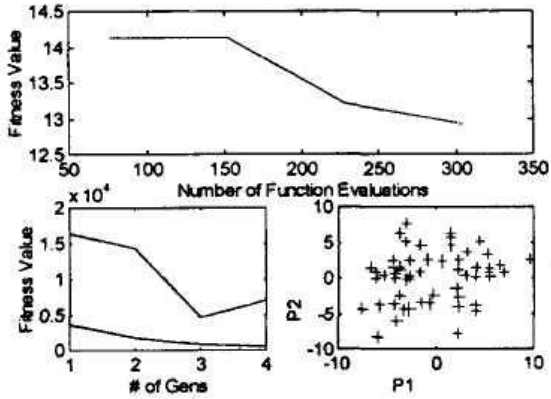



Рис. 3.57. Графіки, що показують значення функції пристосованості для перших чотирьох поколінь у генетичному алгоритмі з турнірною селекцією з приклада 3.18.

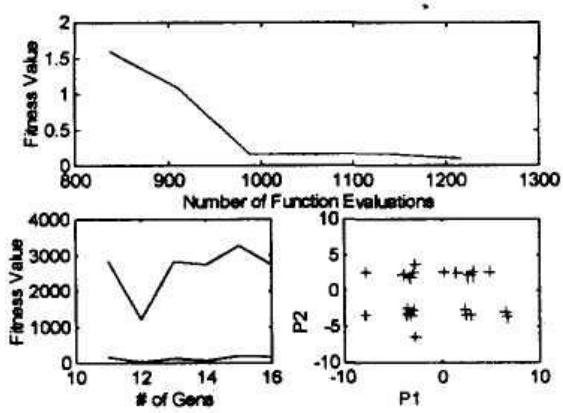



Рис. 3.58. Графіки, що показують значення функції пристосованості з десятого по шістнадцяте покоління в генетичному алгоритмі з турнірною селекцією з приклада 3.18.

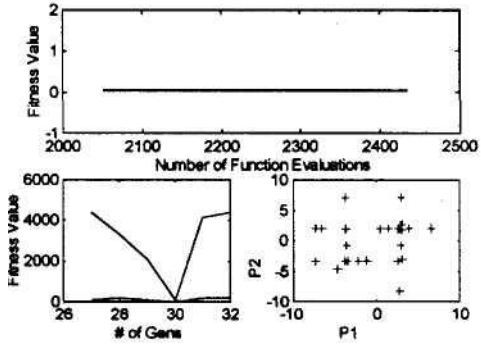



Рис. 3.59**.** Графіки, що показують значення функції пристосованості з двадцять шостого по тридцятьох друге покоління в генетичному алгоритмі з турнірною селекцією з приклада 3.18.

Для 32 поколінь це значення дорівнює 0,0474, а в 46 поколіннях досягається величина 0,0005. На цих же рисунках представлена і зміна «найгіршого» і середнього значень функції пристосованості по популяції при послідовній зміні поколінь, а також розподіл особей у популяції (параметри P1 і P2).

Надалі алгоритм був виконаний ще один раз, причому для експерименту застосовувалася селекція методом рулетки, а ймовірності схрещування і мутації були встановлені рівними 0,6 і 0,001 відповідно. Графіки, що ілюструють зміну значень функції пристосованості при послідовній зміні поколінь, представлені на рис. 3.60 і 3.61.

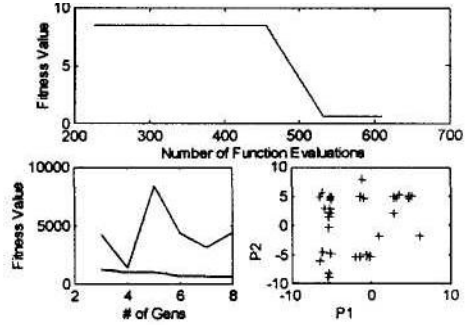


Рис. 3.60. Графіки, що показують значення функції пристосованості з третього по восьме покоління в генетичному алгоритмі із селекцією методом рулетки з приклада 3.18.

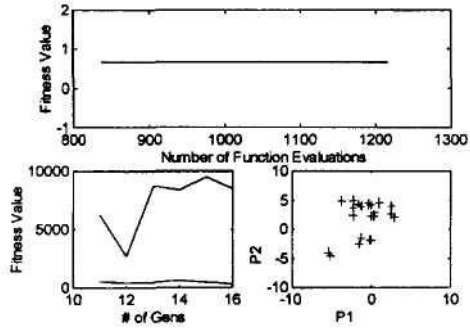



Рис. 3.61. Графіки, що показують значення функції пристосованості з десятого по шістнадцяте покоління в генетичному алгоритмі із селекцією методом рулетки з приклада 3.18.

«Найкраще» значення функції пристосованості в 70 поколіннях дорівнює 0,0038, а починаючи з 83 покоління воно складає 0,0009. «Найкращою» особою виявилася хромосома зі значеннями фенотипів -2,80 і 3,13. Таким чином, у цьому експерименті була знайдена інша точка з координатами (-2,80, 3,13), у якій мінімізуєма функція приймає нульове значення.

Спроби пошуку інших оптимальних точок, що відповідають «найкращому» рішенню, можна продовжувати із застосуванням того ж алгоритму або з використанням альтернативних методів селекції. Іноді такі спроби приходиться повторювати по кілька разів, якщо вони приводять до отриманого раніше рішенням.

**Приклад 3.19**

За допомогою генетичного алгоритму програми **FlexTool** знайти мінімум і максимум функції

$$f(x_1,x_2)=2(1-x_1)^2 e^{-x_1^2-(x_2+1)^2} - 7\left(\frac{x_1}{3} - x_1^3 - x_2^2\right) e^{-x_1^2-x_2^2} - \frac{1}{5} e^{-(x_1+1)^2-x_2^2}$$

для $x_1, x_2 \in [-3, 3]$ з точністю до 0,01.

Графік оптимізуємої функції представлено на рис. 3.62.

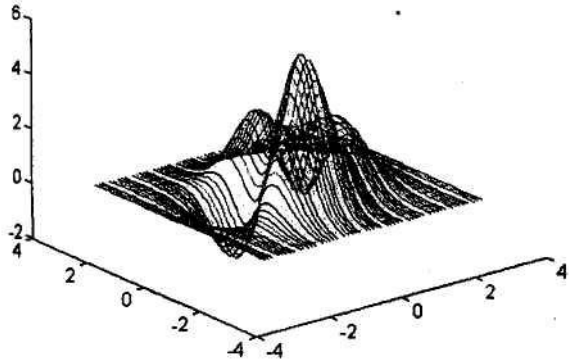



Рис. 3.62. Графік функції $f(x_1, x_2)$ із приклада 3.19.

Ця функція двох змінних має декілька так званих піків. Її контурний графік зображено на рис. 3.63.

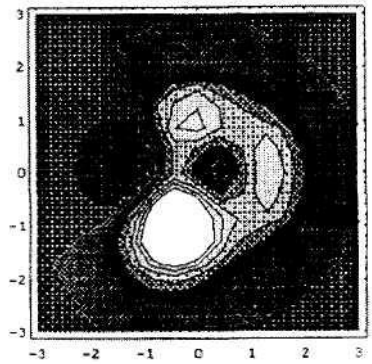



Рис. 3.63**.** Контурний графік функції $f(x_1, x_2)$ із приклада 3.19.

Застосовується генетичний алгоритм із турнірною селекцією в підгрупах по дві особи з одною точкою схрещування; за замовчуванням прийняті значення ймовірностей схрещування 0,77 і мутації 0,0077, а також розмірність популяції, рівна 77. У точці (-1,4, 0,17) знайдений мінімум, рівний -1,907, а в точці (-0,39, -0,99) - максимум, рівний 5,638. Точніше кажучи, при мінімізації найкращим рішенням виявилася хромосома зі значеннями фенотипів -1,4 і 0,17, для якої значення функції

пристосованості складає -1,907, а при максимізації - хромосома зі значеннями фенотипів -0,39 і -0,99 зі значенням функції пристосованості 5,638. Динаміка зміни значень функції пристосованості при послідовній зміні поколінь для випадку мінімізації показана на рис. 3.64 - 3.68, а для випадку максимізації - на рис. 3.69 - 3.71.

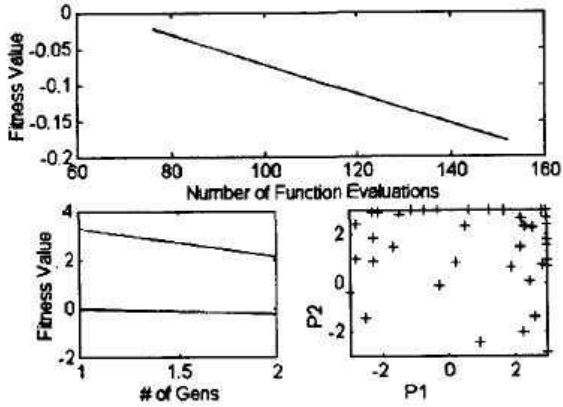



Рис. 3.64**.** Графіки, що показують значення функції пристосованості для перших двох поколінь у випадку пошуку мінімуму з приклада 3.19.

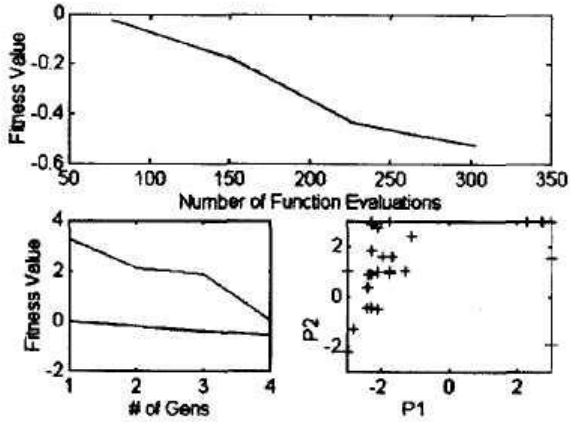


Рис. 3.65. Графіки, що показують значення функції пристосованості для перших чотирьох поколінь у випадку пошуку мінімуму з приклада 3.19.

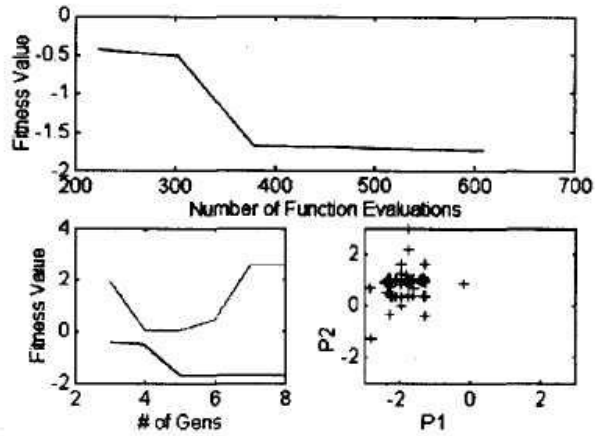



Рис. 3.66. Графіки, що показують значення функції пристосованості з другого по восьме покоління у випадку пошуку мінімуму з приклада 3.19.

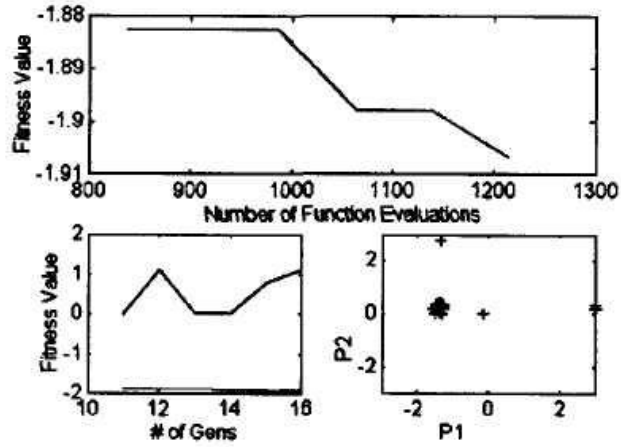


Рис. 3.67. Графіки, що показують значення функції пристосованості з десятого по шістнадцяте покоління у випадку пошуку мінімуму з приклада 3.19.

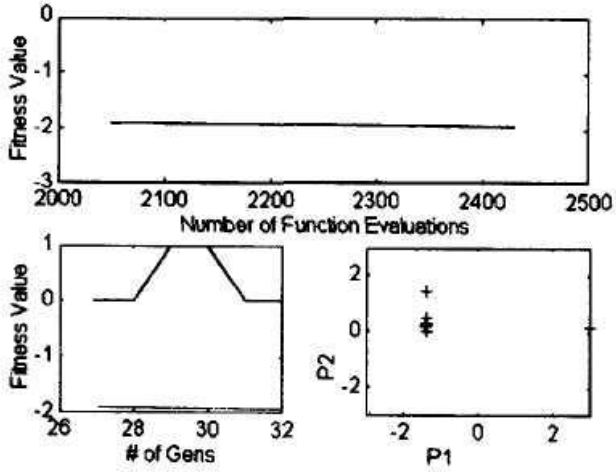



Рис. 3.68. Графіки, що показують значення функції пристосованості з двадцять шостого по тридцять друге покоління у випадку пошуку мінімуму з приклада 3.19.

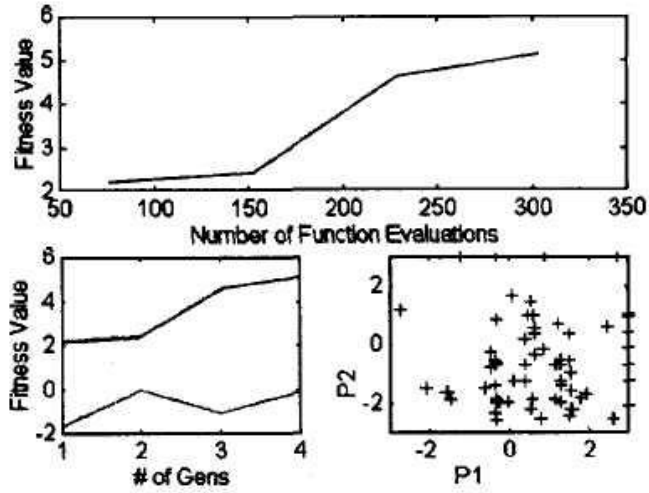



Рис. 3.69. Графіки, що показують значення функції пристосованості для перших чотирьох поколінь у випадку пошуку максимуму з приклада 3.19.

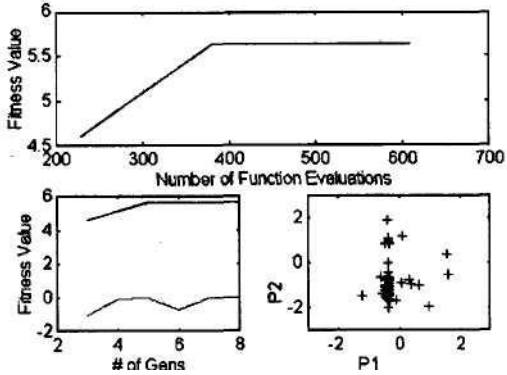



Рис. 3.70. Графіки, що показують значення функції пристосованості з третього по восьме покоління у випадку пошуку максимуму з приклада 3.19.

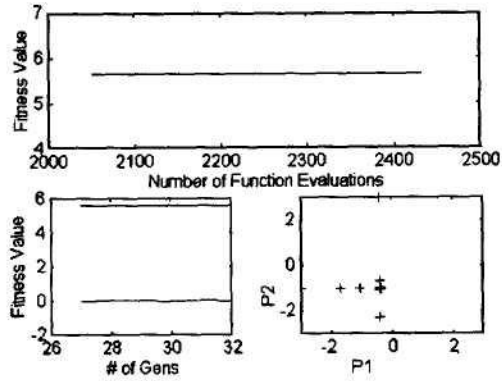



Рис. 3.71. Графіки, що показують значення функції пристосованості з двадцять сьомого по тридцятьох друге покоління у випадку пошуку максимуму з приклада 3.19.

Звертаємо увагу на розподіл особей у популяції в залежності від номера покоління. Спочатку (рис. 3.64) помітна досить велика розмаїтість і відносно рівномірний розподіл окремих точок на площині, заданої осями P1 і P2 в інтервалі від -3 до 3. На рис. 3.65 (четверте покоління) спостерігається виразний їхній зсув у напрямку значень P1, близьких до -2, а на рис. 3.66 - групування поблизу значень P2, приблизно рівних 0,5. На рис. 3.67 більшість точок знаходиться в околиці оптимального рішення з координатами P1 = -1,4 і P2 = 0,17. З рис. 3.68 випливає, що більшість хромосом у популяції збігається з «найкращою» особою, для якої функція пристосованості приймає мінімальне значення, рівне -1,907. Також необхідно додати, що цього мінімального значення функція пристосованості досягла в шістнадцятому поколінні виконання генетичного алгоритму. Верхні графіки на рис. 3.64 - 3.68 ілюструють зміну «найкращого» значення функції пристосованості при послідовній зміні поколінь (числа на осі абсцис відповідають номерам поколінь, помноженим на 77). Нижні ліві графіки на цих рисунках відображають динамічні зміни «найкращого» і «найгіршого» значень функції пристосованості при послідовній зміні поколінь.

Аналогічні графіки приведені на рис. 3.69 - 3.71; вони демонструють, як змінюються «найкраще» і «найгірше» значення функції пристосованості при послідовній зміні поколінь у випадку пошуку максимуму. На цих рисунках можна простежити зменшення розмаїтості хромосом у популяції. На рис. 3.71 практично всі особи однакові і рівні «найкращій» хромосомі зі значеннями фенотипів -0,39 і -0,99. Функція пристосованості цієї хромосоми дорівнює 5,638. Ця точка максимуму функції $f(x_1, x_2)$ знайдена (також, як і у випадку мінімізації) у шістнадцятому поколінні виконання генетичного алгоритму.

**Приклад 3.20**

За допомогою генетичного алгоритму програми **FlexTool** знайти оптимальний набір ваг $w_{11}$, $w_{12}$, $w_{21}$, $w_{22}$, $w_{31}$, $w_{32}$, $w_{10}$, $w_{20}$, $w_{30}$ у діапазоні від -10 до 10 для нейронної мережі, зображеної на рис. 3.11.

Отже, необхідно вирішити задачу, поставлену в прикладі 3.3, тобто знайти значення дев'яти змінних, що позначають ваги нейронної мережі, для яких функція похибки $Q$ досягає мінімального значення, рівного 0. Для мережі, що реалізує логічну систему XOR, функцію похибки можна визначити у виді

$$Q = \frac{1}{4}\sum_{i=1}^{4} \varepsilon_i^2$$

де

$\varepsilon_i = d_i$ - (1/ (1 + exp((-β)($w_{31}$(1/ (1 + $exp((-\beta)(w_{11}u_{1,i} + w_{12}u_{2,i} + w_{10})))$ + + $w_{32}$(1 / (1 +exp((-β)($w_{21}u_{1,i} + w_{22}u_{2i} + w_{20}$)))) + $w_{30}$))))

для $i$=1,2,3, 4. Значення $u_{1,i}$ , $u_{2,i}$, і $d_{1,i}$ - приведені в таблиці

| $u_1$ | $u_2$ | $d$ = XOR($u_1$,$u_2$) |
|---|---|---|
| +1 | +1 | –1 |
| +1 | –1 | +1 |
| –1 | +1 | +1 |
| –1 | –1 | –1 |

Передбачається, що β = 1.

Застосовується генетичний алгоритм із турнірною селекцією в підгрупах по дві особи з одною точкою схрещування. За замовчуванням прийняті встановлені в програмі **FlexTool**

значення ймовірностей схрещування 0,77 і мутації 0,0077, а також розмірність популяції, рівна 77. Довжина хромосом для розв'язуваної задачі дорівнює 99 бітам - по 11 генів на кожну змінну.

Динаміка зміни значень функції пристосованості при послідовній зміні поколінь ілюструється графіками на рисл. 3.72 - 3.80.

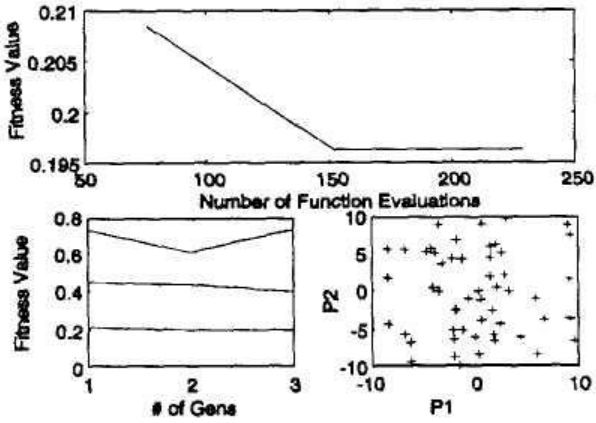



Рис**.** 3.72. Графіки, що показують значення функції пристосованості для перших трьох поколінь у генетичному алгоритмі програми **FlexTool** із приклада 3.20.

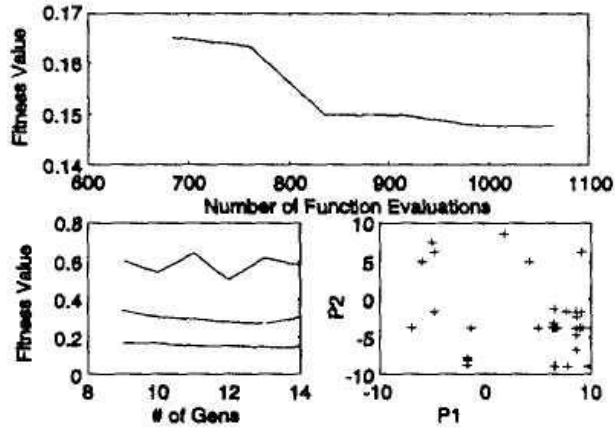



Рис. 3.73. Графіки, що показують значення функції пристосованості з дев'ятого по чотирнадцяте покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

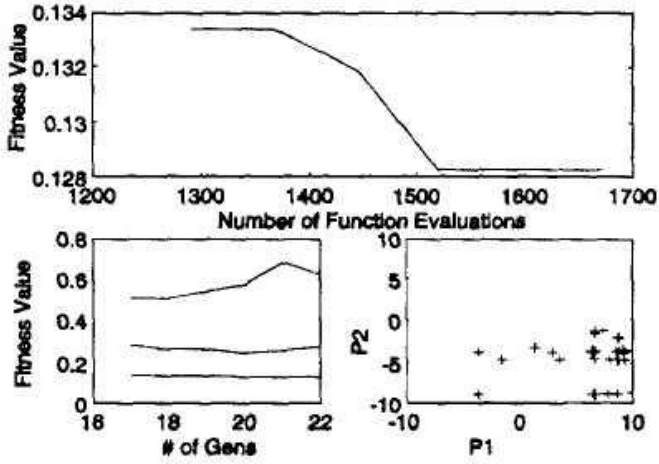



Рис. 3.74. Графіки, що показують значення функції пристосованості із сімнадцятого по двадцятьох друге покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

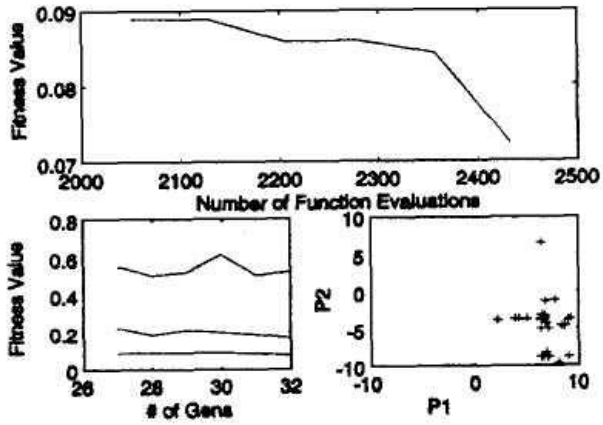



Рис. 3.75. Графіки, що показують значення функції пристосованості з двадцять сьомого по тридцять друге покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

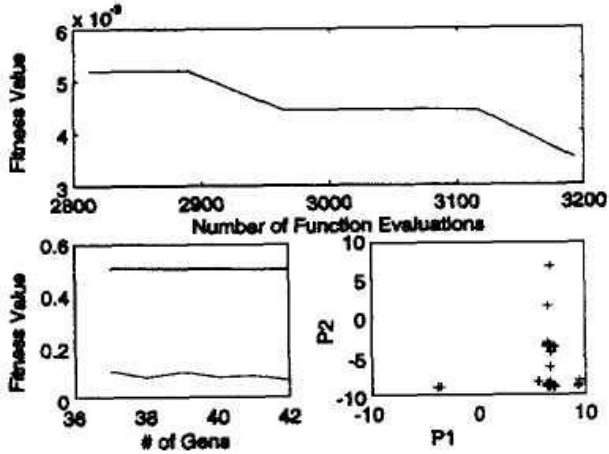



Рис. 3.76. Графіки, що показують значення функції пристосованості з тридцять сьомого по сорока друге покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

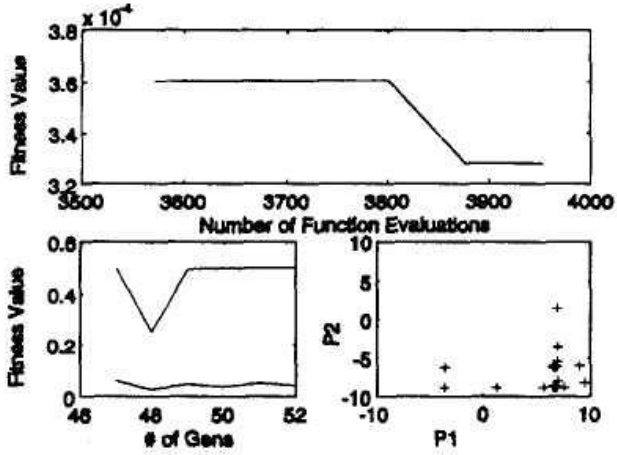



Рис. **3.77.** Графіки, що показують значення функції пристосованості із сорок сьомого по п'ятдесят друге покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

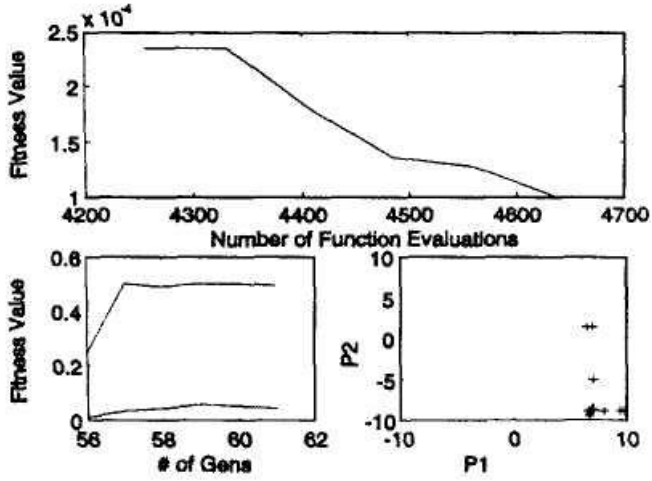



Рис. 3.78. Графіки, що показують значення функції пристосованості з п'ятдесят шостого по шістдесят перше

покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

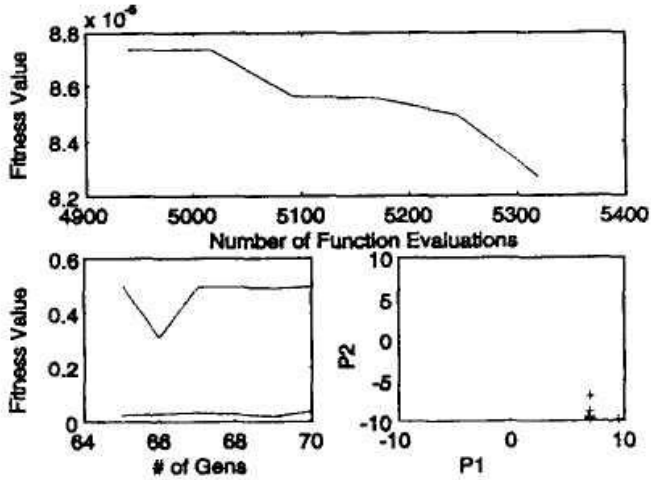



Рис. 3.79**.** Графіки, що показують значення функції пристосованості із шістдесят п'ятого по сімдесяте покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

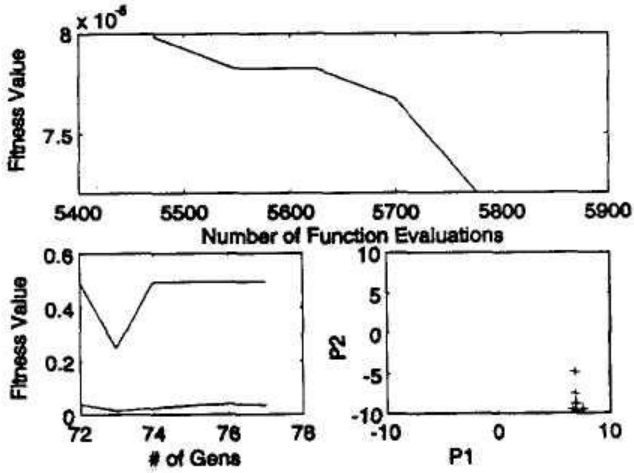


Рис. 3.80. Графіки, що показують значення функції пристосованості із сімдесят другого по сімдесят сьоме покоління в генетичному алгоритмі програми **FlexTool** із приклада 3.20.

На них також показано розподіл особей у залежності від номера покоління, причому відзначено тільки параметри P1 і P2, що відповідають змінним $w_{12}$ і $w_{21}$. «Найкраще» значення функції пристосованості змінюється від значення, приблизно рівного 0,21, до 0,00007. Нагадаємо, що функція пристосованості відповідає функції похибки *Q*, значення якої може змінюватися від 1 до 0.

Нижні ліві графіки ілюструють динамічні зміни «найкращого» (нижня крива), «найгіршого» (верхня крива) і середнього (середня крива) значень функції пристосованості в популяції при послідовній зміні поколінь. На рис. 3.72-3.80 «найкраще» значення функції пристосованості приймає настільки мале значення, що воно стає непомітним на графіках. У 75-м поколінні воно стає рівним $7*10^{-5}$, тобто таким, котре приймається як мінімальне значення. Таким чином, ця величина може вважатися значенням функції пристосованості «найкращої» хромосоми з наступними фенотипами - значеннями окремих змінних, що позначають ваги нейронної мережі:

$w_{11}$= 7,65
$w_{12}$ = - 8,43
$w_{21}$ = -10,00
$w_{22}$ = 9,98
$w_{31}$ = - 4,28
$w_{32}$ = - 3,96
$w_{10}$ = - 4,28
$w_{20}$ - - 3,96
$w_{30}$ = - 4,96

Обчислення були завершені після 77 ітерацій алгоритму. У випадку їхнього продовження можна було б очікувати одержання результату зі значеннями для $w_{11}$, $w_{12}$, $w_{21}$, $w_{22}$, $w_{31}$, $w_{32}$, які виявилися б приблизно рівні відповідно 10, -10, -10, 10, -10, -10, а $w_{10}$, $w_{20}$, $w_{30}$ - величині -5. Також очевидно, що зменшилося б і значення похибкиі *Q*; швидше за все, воно наблизилося б до величини, отриманої за допомогою програми

**Evolver** (приклад 3.24). Розраховані значення являють собою одну з можливих комбінацій ваг для нейронної мережі, що реалізує логічну систему XOR і зображеної на рис. 3.11.

## 3.17. Генетичні алгоритми і математичний апарат

Як нам відомо, генетичні алгоритми призначені, в основному, для рішення задач оптимізації. Прикладом подібної задачі може служити навчання нейромережі, тобто підбора таких значень ваг, при яких досягається мінімальна похибка. При цьому в основі генетичного алгоритму лежить метод випадкового пошуку. Основним недоліком випадкового пошуку є те, що нам невідомо скільки знадобиться часу для рішення задачі. Для того, щоб уникнути таких витрат часу при рішенні задачі, застосовуються методи, що проявилися в біології. При цьому використовуються методи відкриті при вивченні еволюції і походження видів. Як відомо, у процесі еволюції виживають найбільш пристосовані особи. Це приводить до того, що пристосованість популяції зростає, дозволяючи їй краще виживати в умовах, що змінюються.

Як ми вже знаємо, уперше подібний алгоритм був запропонований у 1975 році Джоном Холландом (John Holland) у Мічіганському університеті. Він одержав назву "репродуктивний план Холланда" і ліг в основу практично усіх варіантів генетичних алгоритмів. Однак, для виконання генетичного алгоритму, необхідно закодувати інформацію, що вводиться в нього. Розглянемо питання про те, яким чином об'єкти реального світу можуть бути закодовані для використання в генетичних алгоритмах. Для цілісності викладу матеріалу, опишемо деякі елементарні зведення про генетичні алгоритми, що вже були нами розглянуті раніше.

### Представлення об'єктів

З біології ми знаємо, що будь-який організм може бути представлений своїм *фенотипом*, що фактично визначає, чим є об'єкт у реальному світі, і *генотипом*, що містить всю інформацію про об'єкт на рівні хромосомного набору. При цьому кожен ген, тобто елемент інформації генотипу, має своє відображення у фенотипі. Таким чином, для рішення задач нам необхідно представити кожну ознаку об'єкта у формі, що підходить для використання в генетичному алгоритмі. Усе подальше функціонування механізмів генетичного алгоритму виконується на рівні генотипу, дозволяючи обійтися без інформації про внутрішню структуру об'єкта, що й обумовлює його широке застосування в самих різних задачах. У найбільше часто зустрічаючогося різновиді генетичного алгоритму для представлення генотипу об'єкта застосовуються бітові рядки. При цьому кожному атрибутові об'єкта у фенотипі відповідає один *ген* у генотипі об'єкта. Ген являє собою бітовий рядок, найчастіше фіксованої довжини, що являє собою значення цієї ознаки.

**Кодування ознак, представлених цілими числами**

Для кодування таких ознак можна використовувати самий простий варіант – бітове значення цієї ознаки. Тоді нам буде досить просто використовувати ген визначеної довжини, достатньої для представлення всіх можливих значень такої ознаки. Але, на жаль, таке кодування не позбавлене недоліків. Основний недолік полягає в тім, що сусідні числа відрізняються в значеннях декількох бітів, так наприклад числа 7 і 8 у бітовому представленні розрізняються в 4-х позиціях, що утруднює функціонування генетичного алгоритму і збільшує час, необхідний для його збіжності. Для того, щоб уникнути цієї проблеми краще використовувати кодування, при якому сусідні числа відрізняються меншою кількістю позицій, в ідеалі значенням одного біта. Таким кодом є код Грея, який доцільно використовувати в реалізації генетичного алгоритму. Значення кодів Грея розглянуті в таблиці 3.4.

Таким чином, при кодуванні цілочислової ознаки ми розбиваємо його на тетради і кожну тетраду перетворюємо по

коду Грея. У практичних реалізаціях генетичних алгоритмів зазвичай не виникає необхідності перетворювати значення ознаки в значення гена. На практиці має місце зворотна задача, коли за значенням гена необхідно визначити значення відповідної йому ознаки.

Таким чином, задача декодування значення генів, яким відповідають цілочислові ознаки, тривіальна.

Таблиця 3.4. Відповідність десяткових кодів і кодів Грея.

| **Двоїчне кодування** | | | **Кодування по коду Грея** | | |
|---|---|---|---|---|---|
| Десятковий код | Двоїчне значення | Шістнадцятеричне значення | Десятковий код | Двоїчне значення | Шістнадцятеричне значення |
| 0 | 0000 | 0h | 0 | 0000 | 0h |
| 1 | 0001 | 1h | 1 | 0001 | 1h |
| 2 | 0010 | 2h | 3 | 0011 | 3h |
| 3 | 0011 | 3h | 2 | 0010 | 2h |
| 4 | 0100 | 4h | 6 | 0110 | 6h |
| 5 | 0101 | 5h | 7 | 0111 | 7h |
| 6 | 0110 | 6h | 5 | 0101 | 5h |
| 7 | 0111 | 7h | 4 | 0100 | 4h |
| 8 | 1000 | 8h | 12 | 1100 | Ch |
| 9 | 1001 | 9h | 13 | 1101 | Dh |
| 10 | 1010 | Ah | 15 | 1111 | Fh |
| 11 | 1011 | Bh | 14 | 1110 | Eh |
| 12 | 1100 | Ch | 10 | 1010 | Ah |
| 13 | 1101 | Dh | 11 | 1011 | Bh |
| 14 | 1110 | Eh | 9 | 1001 | 9h |
| 15 | 1111 | Fh | 8 | 1000 | 8h |

**Кодування ознак, яким відповідають числа з точкою, що плаває**

Найпростіший спосіб кодування, що лежить на поверхні – використовувати бітове представлення. Хоча такий варіант має ті ж недоліки, що і для цілих чисел. Тому на практиці зазвичай застосовують наступну послідовність дій:

1. Розбивають весь інтервал припустимих значень ознаки на ділянки з необхідною точністю.
2. Приймають значення гена як цілочислове число, що визначає номер інтервалу (використовуючи код Грея).
3. Як значення параметра приймають число, що є серединою цього інтервалу.

Розглянемо вищеописану послідовність дій на прикладі: Припустимо, що значення ознаки лежать в інтервалі [0,1]. При кодуванні використовувалася розбивка ділянки на 256 інтервалів. Для кодування їхнього номера в такий спосіб нам буде потрібно 8 біт. Припустимо значення гена: 00100101b (заголовна буква G показує, що використовується кодування по коду Грея). Для початку, використовуючи код Грея, знайдемо відповідний йому номер інтервалу:

25h->36h->54d.

Тепер подивимося, який інтервал йому відповідає. Після нескладних підрахунків одержуємо інтервал [0,20703125, 0,2109375]. Значить значення нашого параметра буде (0,20703125+0,2109375)/2=0,208984375.

### Кодування нечислових даних

При кодуванні нечислових даних необхідно попередньо перетворити їх у числа.

### Визначення фенотипу об'єкта по його генотипу

Таким чином, для того, щоб визначити фенотип об'єкта (тобто значення ознак, що описують об'єкт) нам необхідно тільки знати значення генів, що відповідають цим ознакам, тобто

генотип об'єкта. При цьому сукупність генів, що описують генотип об'єкта, являє собою *хромосому*. У деяких реалізаціях її також називають особою. Таким чином, у реалізації генетичного алгоритму хромосома являє собою бітовий рядок фіксованої довжини. При цьому кожній ділянці рядка відповідає ген. Довжина генів усередині хромосоми може бути однаковою або різною. Найчастіше застосовують гени однакової довжини. Розглянемо приклад хромосоми й інтерпретації її значення. Допустимо, що в об'єкта є 5 ознак, кожний закодований геном доввжиною в 4 елементи. Тоді довжина хромосоми буде 5*4=20 біт

0010 1010 1001 0100 1101

тепер ми можемо визначити значення ознак

| **Ознака** | **Значення гена** | **Двоїчне значення ознаки** | **Десяткове значення ознаки** |
|---|---|---|---|
| Ознака 1 | 0010 | 0011 | 3 |
| Ознака 2 | 1010 | 1100 | 12 |
| Ознака 3 | 1001 | 1110 | 14 |
| Ознака 4 | 0100 | 0111 | 7 |
| Ознака 5 | 1101 | 1001 | 9 |

## Безперервні генетичні алгоритми - математичний апарат

Фіксована довжина хромосоми і кодування рядків двоїчним алфавітом переважали в теорії ГА з моменту початку її розвитку, коли були отримані теоретичні результати про доцільність використання саме двоїчного алфавіту. До того ж, реалізація такого ГА на ЕОМ була порівняно легкою. І все-таки, невелика група дослідників йшла по шляху застосування в ГА відмінних від двоїчних алфавітів для рішення часткових прикладних задач. Однієї з таких задач є знаходження рішень, представлених у формі дійсних чисел, що називається не інакше як "пошукова оптимізація в безперервних просторах". Виникла наступна ідея: *рішення в хромосомі представляти прямо у виді набору дійсних*

*чисел*. Природно, що потрібні були спеціальні реалізації біологічних операторів. Такий тип генетичного алгоритму одержав назву *безперервного ГА* (real-coded GA), або *генетичного алгоритму з дійсним кодуванням*. Спочатку безперервні гени стали використовуватися в специфічних доданках (наприклад, хемометрика, оптимальний підбор параметрів операторів стандартних ГА й ін.). Пізніше вони починають застосовуватися для рішення інших задач оптимізації в безперервних просторах (роботи дослідників Wright, Davis, Michalewicz, Eshelman, Herrera у 1991-1995 рр). Оскільки до 1991 теоретичних обґрунтувань роботи безперервних ГА не існувало, використання цього нового підвиду були спірними; учені, знайомі з фундаментальною теорією еволюційних обчислень, у якій була доведена перевага двоїчного алфавіту перед іншими, критично сприймали успіхи real-coded алгоритмів. Після того, як через деякий час теоретичне обґрунтування з'явилося, безперервні ГА цілком витиснули двоїчні хромосоми при пошуку в безперервних просторах.

Далі в тексті за аналогією з англомовною термінологією для ГА з двоїчним кодуванням буде використовуватися абревіатура BGA (Binary coded), для ГА з безперервними генами – RGA (Real coded).

### Переваги і недоліки двоїчного кодування

Перш ніж викладати особливості математичного апарата безперервних ГА, зупинимося на аналізі достоїнств і недоліків двоїчної схем кодування.

Як відомо, висока ефективність відшукання глобального мінімуму або максимуму генетичним алгоритмом з двоїчним кодуванням теоретично обгрунтована у фундаментальній теоремі генетичних алгоритмів ("теоремі про схему"), доведеної Холландом. Її докладне висвітлення і доведення було приведені нами раніше. Нагадаємо, що її суть полягає в тім, що двоїчний алфавіт дозволяє обробляти максимальну кількість інформації в порівнянні з іншими схемами кодування. Однак двоїчне представлення хромосом спричиняє значні труднощі при пошуку в безперервних просторах великої розмірності, і коли потрібна

висока точність знайденого рішення. У BGA для перетворення генотипу у фенотип використовується спеціальний прийом, заснований на тім, що *весь інтервал припустимих значень ознаки об'єкта* [$a_i$,$b_i$] *розбивається на ділянки з необхідною точністю*. Задана точність *p* визначається виразом

$$p = \frac{b_i - a_i}{2^N - 1},$$

де *N* – кількість розрядів для кодування бітового рядка.

Ця формула показує, що *p* сильно залежить від *N*, тобто точність представлення визначається кількістю розрядів, використовуваних для кодування однієї хромосоми. Тому при збільшенні *N* простір пошуку розширюється і стає величезним. Відомий приклад: нехай для 100 змінних, що змінюються в інтервалі [-500; 500], потрібно знайти экстремум з точністю до шостого знака після коми. У цьому випадку при використанні ГА з двоїчним кодуванням довжина рядка складе 3000 елементів, а простір пошуку – близько $10^{1000}$ хромосом. Ефективність BGA у цьому випадку буде невисокою. На перших ітераціях алгоритм витратить багато зусиль на оцінку молодших розрядів числа, закодованих у фрагменті двоїчної хромосоми. Але оптимальне значення на перших ітераціях буде залежати від старших розрядів числа. Отже, поки в процесі еволюції алгоритм не вийде на значення старшого розряду в околиці оптимуму, операції з молодшими розрядами виявляться марними. З іншого боку, коли це відбудеться, стануть непотрібні операції зі старшими розрядами – необхідно поліпшувати точність рішення пошуком у молодших розрядах. Таке "ідеальне" поводження не забезпечує сімейство алгоритмів BGA, тому що ці алгоритми оперують бітовим рядком цілком, і на перших же епохах молодші розряди чисел "застигають", приймаючи випадкове значення. У класичних ГА розроблено спеціальні прийоми по виходу з цієї ситуації. Наприклад, послідовний запуск ансамблю генетичних алгоритмів з поступовим звуженням простору пошуку. Є й інша проблема: при збільшенні довжини бітового рядка необхідно збільшувати і чисельність популяції.

## Математичний апарат безперервних ГА

Як уже відзначалося, при роботі з оптимізаційними задачами в безперервних просторах цілком природно представляти гени прямо дійсними числами. У цьому випадку хромосома є вектор дійсних чисел. Їхня точність буде визначатися винятково розрядною сіткою тієї ЕОМ, на якій реалізується real-coded алгоритм. Довжина хромосоми буде збігатися з довжиною вектора-рішення оптимізаційної задачі, інакше кажучи, *кожен ген буде відповідати за одну змінну*. Генотип об'єкта стає ідентичним його фенотипові. Вищесказане визначає список основних переваг real-coded алгоритмів:

1. Використання безперервних генів уможливлює пошук у великих просторах (навіть у невідомих), що важко робити у випадку двоїчних генів, коли збільшення простору пошуку скорочує точність рішення при незмінній довжині хромосоми.
2. Однією з важливих рис безперервних ГА є їхня здатність до локального настроювання рішень.
3. Використання RGA для представлення рішень зручно, оскільки близько до постановки більшості прикладних задач. Крім того, відсутність операцій кодування/декодування, що необхідні в BGA, підвищує швидкість роботи алгоритму.

Як відомо, поява нових особей у популяції канонічного ГА забезпечують кілька біологічних операторів: *добір, схрещування і мутація*. В якості операторів добору особей у батьківську пару тут підходять будь-які відомі з BGA: *рулетка, турнірний, випадковий.* Однак оператори схрещування і мутації не підходять: у класичних реалізаціях вони працюють з бітовими рядками. Потрібні власні реалізації, що враховують специфіку real-coded алгоритмів. Оператор схрещування безперервного ГА, або кросовер, породжує одного або декількох нащадків від двох хромосом. Власне кажучи, потрібно з двох векторів дійсних чисел одержати нові вектори за якими-небудь законами. Більшість real-coded алгоритмів генерують нові вектори в околиці батьківських пар. Для початку розглянемо прості і популярні кросовери.

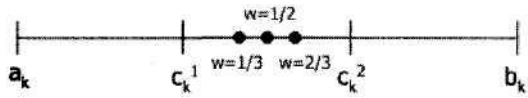
Нехай $C_1=(c_1^1,c_2^1,\ldots,c_n^1)$ і $C_2=(c_1^2,c_2^2,\ldots,c_n^2)$ – дві хромосоми, які обрані оператором селекції для схрещування. Після формули для деяких кросоверів приводиться малюнок – геометрична інтерпретація його роботи. Передбачається, що $c_k^1<=c_k^2$ і $f(C_1)>=f(C_2)$.

***Плоский кросовер*** (flat crossover): створюється нащадок $H=(h_1,\ldots,h_k,\ldots,h_n),h_k$, $k$=1,…,$n$ – випадкове число з інтервалу $[c_k^1,c_k^2]$.

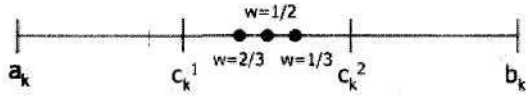
***Найпростіший кросовер*** (simple crossover): випадковим чином вибирається число $k$ з інтервалу {1,2,…,$n$-1} і генеруються два нащадки

$$H_1=(c_1^1,c_2^1,\ldots,c_k^1,c_{k+1}^2,\ldots,c_n^2) \text{ і } H_2=(c_1^2,c_2^2,\ldots,c_k^2,c_{k+1}^1,\ldots,c_n^2)\ldots$$

***Арифметичний кросовер*** (arithmetical crossover): створюються два нащадки $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^2,\ldots,h_n^2)$, де $h_k^1=w*c_k^1+(1-w)*c_k^2$, $h_k^2=w*c_k^2+(1-w)*c_k^1$, $k=1,\ldots,n$, $w$ - або константа (рівномірний арифметичний кроссовер) з інтервалу [0;1], або змінюється зі збільшенням епох (нерівномірний арифметичний кроссовер).

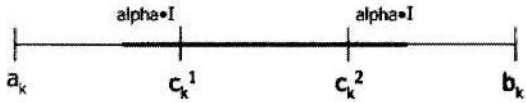



***Геометричний кросовер*** (geometrical crossover): створюються два нащадки $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^2,\ldots,h_n^2)$, де $h_k^1= (c_k^1)^{w*}(c_k^2)^{1-w}$, $(c_k^2)^{w*}(c_k^1)^{1-w}$, $w$ – випадкове число з інтервалу [0;1].




***Змішаний кросовер*** (blend, BLX-alpha crossover): генерується один нащадок $H=(h_1,\ldots,h_k,\ldots,h_n)$, де $h_k$ – випадкове число з

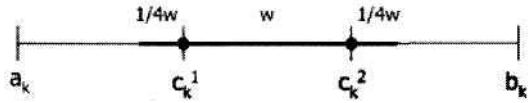
інтервалу $[c_{min}–I*alpha, c_{max}+I*alpha]$, $c_{min}=min(c_k^1,c_k^2)$, $c_{max}=max(c_k^1,c_k^2)$, $I=c_{max}–c_{min}$. *BLX*-0. 0 кроссовер перетворюється в плоский.




***Лінійний кросовер*** (linear crossover): створюються три нащадки $H_q=(h_1^q,…,h_k^q,…,h_n^q), q=1,2,3$, де $h_k^1=0.5*c_k^1+0.5*c_k^2$, $h_k^2=1.5*c_k^1-0.5*c_k^2$, $h_k^3=–0.5*c_k^1+1.5*c_k^2$. На етапі селекції в цьому кросовері відбираються два найбільш сильні нащадки.


***Дискретний кросовер*** (discrete crossover): кожен ген $h_k$ вибирається випадково по рівномірному закону з кінцевої множини $\{c_k^1,c_k^2\}$.


***Розширений лінійчатий кросовер*** (extended line crossover): ген $h_k=c_k^1+w*(c_k^2–c_k^1)$, $w$ – випадкове число з інтервалу [-0.25;1.25].

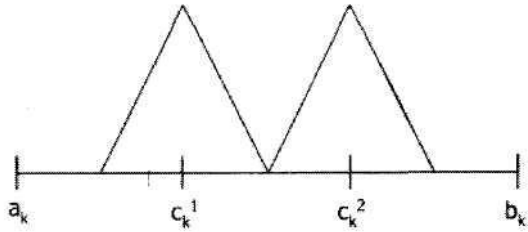


***Евристичний кросовер*** (Wright's heuristic crossover). Нехай $C_1$ – один із двох батьків із кращою пристосованістю. Тоді $h_k=w*(c_k^1-c_k^2)+c_k^1$, $w$ – випадкове число з інтервалу [0;1].


***Нечіткий кросовер*** (fuzzy recombination, FR-d crossover): створюються два нащадки $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^1,\ldots,h_n^2)$... Ймовірність того, що в $i$-тім гені з'явиться число $v_i$ , задається розподілом $p(v_i)\{F(c_k^1), F(c_k^2)\}$, де $F(c_k^1)$, $F(c_k^2)$ – розподіли ймовірностей трикутної форми (трикутні нечіткі функції приналежності) з наступними властивостями ($c_k^1=c_k^2$ і $I=|c_k^1-c_k^2|$):

| **Розподіл ймовірностей** | **Мінімум** | **Центр** | **Максимум** |
|---|---|---|---|
| $F(c_k^1)$ | $c_k^1-d*I$ | $c_k^1$ | $c_k^1+d*I$ |
| $F(c_k^2)$ | $c_k^2-d*I$ | $c_k^2$ | $c_k^2+d*I$ |

Параметр $d$ визначає степінь перекриття трикутних функцій приналежності, за замовчуванням $d$=0.5.




В якості оператора мутації найбільше поширення одержали: випадкова і нерівномірна мутація (random and non-uniform mutation). При випадковій мутації ген, що роздлягає змінні, приймає випадкове значення з інтервалу своєї змінної. У

нерівномірній мутації значення гена після оператора мутації розраховується по формулі:

$$c_k^* = \begin{cases} c_k + \Delta(t, b_k - c_k), & w = 0 \\ c_k - \Delta(t, b_k - c_k), & w = 1 \end{cases},$$

$$\Delta(t, y) = y\left[1 - r^{\left(1 - \frac{t}{e\_max}\right)^b}\right].$$

Складно сказати, що більш ефективно в кожнім конкретному випадку, але численні дослідження доводять, що безперервні ГА не менш ефективно, а часто набагато ефективніше справляються з задачами оптимізації в багатомірних просторах, при цьому більш прості в реалізації через відсутність процедур кодування і декодування хромосом.

Розглянуті кросовери історично були запропоновані першими, однак у багатьох задачах їхня ефективність виявляється невисокою. Виключення складає BLX-кроссовер з параметром alpha=0.5 – він перевершує по ефективності більшість простих кросоверов. Пізніше були розроблені поліпшені оператори схрещування, аналітична формула яких і ефективність обгрунтовані теоретично. Розглянемо докладніше один з таких кросоверов – SBX.

### SBX кроссовер

SBX (англ.: Simulated Binary Crossover) – кросовер, що імітує двоїчний. Був розроблений у 1995 році дослідницькою групою під керівництвом K. Deb'a. Як випливає з його назви, цей кросовер моделює принципи роботи двоїчного оператора схрещування.

SBX кросовер був отриманий у такий спосіб. У двоїчного кросовера була виявлена важлива властивість – середнє

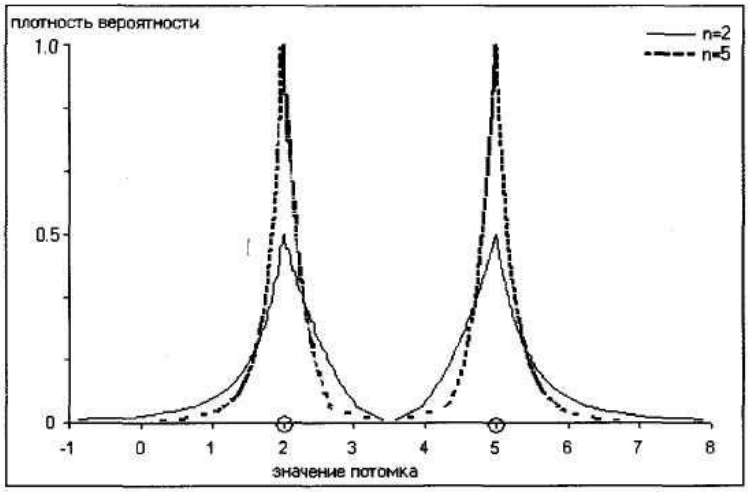
значення функції пристосованості залишалося незмінним у батьків і їхніх нащадків, отриманих шляхом схрещування. Потім автором було введене поняття сили пошуку кросовера (search power). Це кількісна величина, що характеризує розподіл ймовірностей появи будь-якого нащадка від двох довільних батьків. Спочатку була розрахована сила пошуку для одноточкового двоїчного кросовера, а потім був розроблений дійсний SBX кросовер з такою же силою пошуку. У ньому сила пошуку характеризується розподілом ймовірностей випадкової величини β:

$$P(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \beta \le 1 \\ 0.5(n+1)\beta^{-(n+2)}, & \beta > 1. \end{cases}$$

Для генерації нащадків використовується наступний алгоритм, що використовує вираз для P(β). Створюються два нащадки $H_k=(h_1^k, \ldots, h_j^k, \ldots, h_n^k)$, $k$=1,2,
де $h_j^1=0.5\cdot [(1- \beta_k)c_j^1 +(1- \beta_k)c_j^2]$ - число, отримане по формулі:

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{n+1}}, & u(0,1) \le 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & u(0,1) > 0.5. \end{cases}$$

У формулі $u(0,1)$ – випадкове число, розподілене по рівномірному закону, $n$ [2,5] – параметр кросовера. На рисунку приведена геометрична інтерпретація роботи SBX кросовера при схрещуванні двох хромосом, що відповідають дійсним числам 2 і 5.




З рисунка видно, як параметр *n* впливає на кінцевий результат: збільшення *n* спричиняє збільшенню ймовірності появи нащадка в околиці батька і навпаки.

Експерименти автора SBX кросовера показали, що він у багатьох випадках ефективніше BLX, хоча, мабуть, не існує жодного кросовера, ефективного у всіх випадках. Дослідження показують, що використання декількох різних операторів кросовера дозволяє зменшити ймовірність передчасної збіжності, тобто поліпшити ефективність алгоритму оптимізації в цілому. Для цього можуть використовуватися спеціальні стратегії, що змінюють ймовірність застосування окремого еволюційного оператора в залежності від його «успішності», або використання гібридних кросоверів, яких у даний час нараховується більше десятків. У будь-якому випадку, якщо перед Вами стоїть задача оптимізації в безперервних просторах, і Ви плануєте застосувати еволюційні алгоритми, то варто зробити вибір на користь безперервного генетичного алгоритму.

## 3.18. Еволюційні алгоритми

*Генетичні алгоритми (genetic algorithms)* разом з *еволюційною стратегією* й *еволюційним програмуванням* представляють *три головних напрямки розвитку так званого еволюційного моделювання (simulated evolution).*

Незважаючи на те, що кожен з цих методів виник незалежно від інших, вони характеризуються рядом важливих загальних властивостей. Для кожного з них формується вихідна популяція особей, що у наступному піддається селекції і впливові різних генетичних операторів (найчастіше схрещуванню і/або мутації), що дозволяє знаходити більш гарні рішення.

***Еволюційні стратегії*** *- це алгоритми, створені як методи рішення оптимізаційних задач і засновані на принципах природної еволюції.*

Еволюційне програмування являє собою підхід, запропонований американськими вченими спочатку в рамках *теорії кінцевих автоматів* і узагальнений пізніше для доданків до проблем оптимізації. Обидва напрямка виникли в шістдесятих роках XX століття.

Сконцентруємо увагу на найважливіших подібностях і розходженнях між еволюційними стратегіями і генетичними алгоритмами. Очевидно, що головна подібність полягає в тім, що обидва методи використовують популяції потенційних рішень і реалізують принцип селекції і перетворення найбільш пристосованих особей. Однак обговорювані підходи сильно відрізняються один від одного.

Перше розходження полягає в способі представлення особей. ***Еволюційні стратегії оперують векторами дійсних чисел,*** **тоді як** ***генетичні алгоритми - двоїчними векторами***.

Друге розходження між еволюційними стратегіями і генетичними алгоритмами криється в організації процесу селекції. При реалізації еволюційної стратегії формується проміжна популяція, що складається з усіх батьків і деякої кількості нащадків, створених у результаті застосування генетичних операторів. За допомогою селекції розмір цієї проміжної популяції зменшується до величини батьківської популяції за рахунок виключення найменш пристосованих особей. Сформована в такий спосіб популяція утворює чергове покоління. Навпроти, у генетичних алгоритмах передбачається, що в результаті селекції з популяції батьків відбирається кількість особей, рівна розмірності вихідної популяції, при цьому

деякі (найбільш пристосовані) особи можуть відбиратися багаторазово. У той же час, менш пристосовані особи також мають можливість з'явитися в новій популяції. Однак шанси їхнього відбору пропорційні величині пристосованості особей. Незалежно від застосовуваного в генетичному алгоритмі методу селекції (наприклад, рулетки або рангового) більш пристосовані особи можуть відбиратися багаторазово. *При реалізації еволюційних стратегій особи відбираються без повторень*. В еволюційних стратегіях застосовується детермінована процедура селекції, тоді як у генетичних алгоритмах вона має випадковий характер.

Третє розходження між еволюційними стратегіями і генетичними алгоритмами стосується послідовності виконання процедур селекції і рекомбінації (тобто зміни генів у результаті застосування генетичних операторів). При реалізації еволюційних стратегій спочатку виконується рекомбінація, а потім селекція. У випадку виконання генетичних алгоритмів ця послідовність інвертується. При здійсненні застосування еволюційних стратегій нащадок утворюється в результаті схрещування двох батьків і мутації. Формована в такий спосіб проміжна популяція, що складається з усіх батьків і отриманих від них нащадків, надалі піддається селекції, що зменшує розмір цієї популяції до розміру вихідної популяції. При виконанні генетичних алгоритмів спочатку виконується селекція, що приводить до утворення перехідної популяції, після чого генетичні оператори схрещування і мутації застосовуються до особів (обираних з ймовірністю схрещування) і до генів (обираних з ймовірністю мутації).

Наступне розходження між еволюційними стратегіями і генетичними алгоритмами полягає в тім, що параметри генетичних алгоритмів (такі, як ймовірності схрещування і мутації) залишаються сталими протягом усього процесу еволюції, тоді як при реалізації еволюційних стратегій ці параметри піддаються безперервним змінам (так називана самоадаптація параметрів).

Ще одне розходження між еволюційними стратегіями і генетичними алгоритмами стосується трактування обмежень, що накладаються на розв'язувані оптимізаційні задачі. Якщо при реалізації еволюційних стратегій на деякій ітерації нащадок не задовольняє всім обмеженням, то він відкидається і включається

в нову популяцію. Якщо таких нащадків виявляється багато, то еволюційна стратегія запускає процес адаптації параметрів, наприклад, шляхом збільшення ймовірності схрещування. У генетичних алгоритмах такі параметри не змінюються. У них може застосовуватися штрафна функція для тих особей, що не задовольняють накладеним обмеженням, однак ця технологія має багато недоліків.

В міру розвитку еволюційних стратегій і генетичних алгоритмів протягом останнього років істотні розходження між ними поступово зменшуються. Наприклад, при реалізації генетичних алгоритмів для рішення оптимізаційних задач усе частіше застосовується представлення хромосом дійсними числами і різні модифікації «генетичних» операторів, що має на меті підвищити ефективність цих алгоритмів. Подібні методи, що значно відрізняються від класичного генетичного алгоритму, за традицією зберігають колишню назву, хоча більш коректно було б називати їх «еволюційними алгоритмами». Проблему термінології ми будемо обговорювати трохи пізніше.

Еволюційне програмування, також як і еволюційні стратегії, робить основний упор на адаптацію і розмаїтість способів передачі властивостей від батька до нащадків у наступних поколіннях. Познайомимося зі стандартним алгоритмом еволюційного програмування.

Вихідна популяція рішень вибирається випадковим чином. У задачах оптимізації значень дійсних чисел (прикладом яких може служити навчання нейронних мереж) особа (хромосома) представляється ланцюгом значень дійсних чисел. Ця популяція оцінюється щодо заданої функції (функції пристосованості). Нащадки утворюються від вхідних у цю популяцію батьків у результаті випадкової мутації. Селекція заснована на ймовірнісному виборі (турнірний метод), при якому кожне рішення змагається з хромосомами, випадковим чином обираними з популяції. Рішення-переможці (які є найкращими) стають батьками для наступного покоління. Описана процедура повторюється так довго, поки не буде знайдені шукане рішення або не буде вичерпаний ліміт машинного часу.

Еволюційне програмування застосовується для оптимізації функціонування нейроих мереж. Так, як і інші еволюційні методи, воно не вимагає градиєнтної інформації і тому може використовуватися для рішення задач, у яких ця інформація

недоступна, або для її одержання потрібні значні обсяги обчислень. Одними з перших додатків еволюційного програмування вважаються задачі теорії штучного інтелекту, а самі ранні роботи стосувалися теорії кінцевих автоматів.

Спостерігається велика подібність між еволюційними стратегіями й еволюційним програмуванням у їхніх додатках до задач оптимізації безперервних функцій з дійсними значеннями. Деякі дослідники стверджують, що ці процедури, по суті, однакові, хоча вони і розвивалися незалежно одна від одної. Дійсно, обидва методи схожі на генетичні алгоритми. Принципове розходження між ними полягає в тім, що еволюційне програмування не зв'язане з конкретною формою представлення особей, оскільки оператор мутації не вимагає застосування якого-небудь спеціального способу кодування.

Перший контакт між науковими колективами, що розвивали еволюційні стратегії й еволюційне програмування, відбувся на початку 1992 р., безпосередньо перед першою міжнародною конференцією, присвяченою еволюційному програмуванню. Ці методи розвивалися незалежно протягом 30 років. Незважаючи на виділені розходження, вони мають багато принципово подібних властивостей.

Усі три представлених методи, тобто *генетичні алгоритми*, *еволюційні стратегії й еволюційне програмування* поєднуються під загальною назвою еволюційні алгоритми *(evolutionary algorithms).* Також застосовується термін еволюційні методи *(evolutionary methods).*

Еволюційними алгоритмами називаються й інші методи, що реалізують еволюційний підхід, зокрема, генетичне програмування *(genetic programming)*, що представляє собою модифікацію генетичного алгоритму з урахуванням можливостей комп'ютерних програм. При використанні цього методу популяція складається з закодованих відповідним чином програм, що піддаються впливові генетичних операторів схрещування і мутації для знаходження оптимального рішення, яким вважається програма, що щонайкраще вирішує поставлену задачу. Програми оцінюються щодо визначеної спеціальним чином функції пристосованості. Цікавою представляється модифікація еволюційних алгоритмів для рішення оптимізаційних задач методом так називаної м'якої селекції, що запропонована Р. Галаром.

Для позначення різноманітних алгоритмів, заснованих на еволюційному підході, також застосовується поняття еволюційних програм *(evolution programs)*. Цей термін поєднує як *генетичні алгоритми, еволюційні стратегії й еволюційне програмування, так і генетичне програмування*, а також інші аналогічні методи.

Еволюційні програми можна вважати узагальненням генетичних алгоритмів. Класичний генетичний алгоритм виконується при фіксованій довжині двоїчних послідовностей і в ньому застосовуються оператори схрещування і мутації. Еволюційні програми обробляють більш складні структури (не тільки двоїчні коди) і можуть виконувати інші «генетичні» операції. Наприклад, еволюційні стратегії можуть трактуватися як еволюційні програми, у яких хромосоми представляються дійсними (не двоїчними) числами, а мутація використовується як єдина генетична операція.

Структуру еволюційної програми досить точно відображає блок-схема, приведена на рис. 3.12. Вона збігається зі структурою генетичного алгоритму, оскільки ідеї еволюційної програми цілком запозичені з теорії генетичних алгоритмів. Розходження мають глибинний характер, вони стосуються способів представлення хромосом і реалізації генетичних операторів. Еволюційні програми допускають більшу розмаїтість структур даних, оскільки можливе не тільки двоїчне кодування хромосом, а також надають розширений набір генетичних операторів.

Згідно З. Михалевичу, еволюційна програма - це ймовірнісний алгоритм, застосовуваний на $k$-й ітерації до популяції особей

$$P(k) = \{ x^k_1 ..... x^k_n \}.$$

Кожна особа представляє потенційне рішення задачі, що у довільній еволюційній програмі може відображатися якоюсь (у тому числі і досить складної) структурою даних $D$. Будь-яке рішення $x_i^k$ оцінюється за значенням його «пристосованості». Далі в процесі селекції на ($k$+1)-й ітерації з найбільш пристосованих особей формується чергова популяція. Деякі особи цієї нової популяції трансформуються за допомогою «генетичних операторів», що дозволяє одержувати нові рішення. Існують перетворення $\mu_i$ (типу мутації), що змінюють конкретні хромосоми ($\mu_i$: D→D), а також трансформації більш високого порядку $\gamma_i$ (типу схрещування), що створюють нові особи шляхом

комбінування фрагментів декількох (двох або більш) хромосом, тобто $\gamma_i$: D × ... × D → D. Від еволюційної програми очікується, що після зміни деякої кількості поколінь найкраща особа буде представляти рішення, близьке до оптимального. Структура еволюційної програми може бути представлена у виді псевдокоду так, як це зображено на рис. 3.81 (рекомендується порівняти неї з рис. 3.12).

```
procedure эволюционная_программа
begin
 k = 0
 инициализация популяции P(k)
 оценивание приспособленности особей из P(k)
 while (not условие завершения) do
 begin
 k = k + 1
 селекция особей из P(k – 1) в P(k)
 применение генетических операторов
 оценивание приспособленности особей из P(k)
 end
end
```

Рис. 3.81. Представлення еволюційної програми у виді псевдокоду.

Розглянемо узагальнений приклад еволюційної програми. Припустимо, що шукається граф, що задовольняє певним обмеженням (наприклад, виконується пошук топології комунікаційної мережі, оптимальної за конкретними критеріями, наприклад, по вартості передачі і т.п.). Кожна особа в еволюційній програмі представляє одне з потенційних рішень, тобто в даному випадку деякий граф. Вихідна популяція графів P(0), сформована випадковим чином або створювана при реалізації якого-небудь евристичного процесу, вважається відправною точкою (*k*=0) еволюційної програми. Функція пристосованості, що зазвичай задається, зв'язана із системою обмежень розв'язуваної задачі. Ця функція визначає «пристосованість» кожного графа шляхом виявлення «кращих» і «гірших» особей. Можна запропонувати кілька різних операторів

мутації, призначених для трансформації окремих графів, і трохи операторів схрещування, що будуть створювати новий граф у результаті рекомбінації структур двох або більш графів. Дуже часто такі оператори обумовлюються характером розв'язуваної задачі. Наприклад, якщо шукається граф типу «дерево», то можна запропонувати оператор мутації, що видаляє галузь з одного графа і додає нову галузь, що поєднує два окремих підграфа. Інші можливості полягають у проектуванні мутації незалежно від семантики задачі, але з включенням у функцію пристосованості додаткових обмежень - «штрафів» для тих графів, що не є деревами.

Принципову різницю між класичним генетичним алгоритмом і еволюційною програмою, тобто еволюційним алгоритмом у широкому змісті, ілюструють рис. 3.82 і 3.83.


Рис. 4.82. Рішення задачі за допомогою класичного генетичного алгоритму.

Класичний генетичний алгоритм, що оперує двоїчними послідовностями, вимагає представити розв'язувану задачу в строго визначеному виді (відповідність між потенційними рішеннями і двоїчними кодами, декодування і т.п.). Зробити це не завжди просто.


Рис. 3.83. Рішення задачі за допомогою еволюційного алгоритму (еволюційної програми)

Еволюційні програми можуть залишити постановку задачі в незмінному виді за рахунок модифікації хромосом, що представляють потенційні рішення (з використанням «природних» структур даних), і застосування відповідних «генетичних» операторів. Іншими словами, для рішення нетривіальної задачі можна або перетворити її до виду, необхідному для використання генетичного алгоритму (рис. 3.82), або модифікувати генетичний алгоритм так, щоб він задовольняв задачі (рис. 3.83).

При реалізації першого підходу застосовується класичний генетичний алгоритм, а при реалізації другого підходу - еволюційна програма Таким чином, модифіковані генетичні алгоритми можна в загальному випадку називати еволюційними програмами. Однак найчастіше зустрічається термін еволюційні алгоритми. Еволюційні програми також можуть розглядатися як еволюційні алгоритми, підготовлені програмістом для виконання на комп'ютері. Основна задача програміста полягає при цьому у виборі відповідних структур даних і «генетичних» операторів.

Саме таке трактування поняття еволюційна програма представляється найбільш обґрунтованою.

Усі поняття, застосовувані в цьому розділі і стосовні головним чином до методів, заснованим на еволюційному підході, можна зіставити головному напрямкові досліджень - *комп'ютерному моделюванню еволюційних процесів*. Ця область інформатики називається *Evolutionary Computation,* що можна перевести як *еволюційні обчислення*.

До еволюційних алгоритмів також застосовується поняття технологія еволюційних обчислень. Можна додати, що назва генетичні алгоритми використовується як у вузькому змісті, тобто для позначення класичних генетичних алгоритмів і їхніх несуттєвих модифікацій, так і в широкому змісті - припускаючи будь-які еволюційні алгоритми, що значно відрізняються від «класики».

Останнім часом з'явилося багато нових методів, заснованих на еволюційному моделюванні, але базові технології, що використовуються - головним чином класичний генетичний алгоритм, еволюційні стратегії й еволюційне програмування.

## 3.19. Доданки еволюційних алгоритмів

Більшість доданків еволюційних алгоритмів, і особливо генетичних алгоритмів, стосується оптимізаційних задач. Найпростішими з них можна назвати задачі, представлені в прикладах 3.1 - 3.3, у прикладах 3.5 і 3.6, а також прорахованні на комп'ютері приклади з п. 3.16. У кожнім з них оптимізується цільова функція, задана конкретною формулою, і використовується характерне для основного генетичного алгоритму двоїчне кодування хромосом.

Як уже згадувалося в п. 3.16.4, наступна модифікація класичного генетичного алгоритму полягала в представленні хромосом дійсними числами. Про такий спосіб кодування говорилося й у п. 3.18. Однією з найбільш відомих комп'ютерних програм, призначених для рішення задач за допомогою генетичного алгоритму з кодуванням дійсними числами *(real coding),* вважається програма **Evolver**. У цій програмі застосовується алгоритм із частковою заміною популяції *(steady-state),* при якій у кожен момент часу заміняється тільки одна

особа. Селекція заснована на ранговому методі *(rank-based).* Якщо говорити про так званих генетичних операторів, то в програмі **Evolver** застосовуються два різних оператори схрещування і два різних оператори мутації - окремо для оптимізаційних і для комбінаторних задач.

Програма **Evolver** взаємодіє з табличним процесором **Excel,** у якому розв'язувана задача описується у відповідних осередках таблиці шляхом завдання її параметрів (змінних) і формули функції пристосованості.

## 3.19.1. Приклади оптимізації функції за допомогою програми Evolver

Перш ніж перейти до прикладів, дамо характеристику генетичним операторам, використовуваним програмою **Evolver** у задачах оптимізації функції. .

**Схрещування.** Це рівномірне схрещування *(uniform crossover),* визначене аналогічно застосованому в п. 3.16.3, але для хромосом, що складаються з генів з дійсними аллелями.

Схрещування виконується відповідно до так називаного *показника схрещування (crossover rate),* що визначає, який відсоток генів нащадок успадкує від кожного батька. У програмі **Evolver** показник схрещування вводиться користувачем і являє собою число з інтервалу від 0,01 до 1,0. Наприклад, значення показника схрещування 0,8 означає, що нащадок одержить близько 80 % генів зі значеннями (аллелями) такими ж, як у першого батька, а кількість, що залишилася, (порядку 20 %) - успадкує від другого батька. Якщо показник схрещування дорівнює 1, то ніякого схрещування практично не відбувається, а утворюються тільки так називані клони, тобто хромосоми, ідентичні батькам. За замовчуванням у програмі **Evolver** застосовується значення показника схрещування, рівне 0,5, що означає спадкування зразково однакової кількості генів від кожного батька.

**Мутація.** Кожен ген у хромосомі представляє один параметр задачі. Отже, аллелі відповідають фенотипам, тобто значенням конкретних змінних. Для кожного гена випадковим чином вибирається число з інтервалу від 0 до 1, що порівнюється з так називаним *показником мутації (mutation rate).* Якщо розігране число менше введеного значення показника або дорівнює йому, то виконується мутація даного гена. Ця операція полягає в заміні

значення гена іншим, випадково обраним числом з області припустимих значень параметра, що відповідає мутуйочому генові.

Значення показника, що вводиться користувачем, мутації являє собою число з інтервалу від 0,0 до 1,0. Чим більше це значення, тим більша кількість генів піддається мутації. Якщо показник мутації дорівнює 1, то мутації піддаються 100 % генів, обираних випадковим чином. З урахуванням того, що в програмі **Evolver** мутація завжди виконується після схрещування, завдання показника мутації рівним 1 означає, що в цьому випадку ефект схрещування не має ніякого значення. Очевидно, що якщо показник мутації дорівнює 0, то мутація взагалі не виконується. Як правило, значення показника мутації задається в межах від 0,06 до 0,2. Відзначимо, що обумовлений у такий спосіб показник мутації являє собою аналог ймовірності мутації $p_m$, описаної в п. 3.4. У той же час використовуваний у програмі **Evolver** показник схрещування має сенс, який відрізняється від введеної в п. 3.4 ймовірності схрещування $p_c$. Значення як показника мутації, так і показника схрещування можна змінювати в процесі виконання програми **Evolver.**

У класичному генетичному алгоритмі на кожній ітерації обновляється вся популяція, що відповідає формуванню чергового покоління. Стосовно до програми **Evolver** можна говорити про послідовні «такти» *(trials)* її виконання, причому на кожнім такті змінюється тільки одна особа. Якщо популяція нараховує *N* хромосом (особей, що в описі програми називаються організмами), то *N* таких «тактів» складають одну ітерацію класичного генетичного алгоритму. Тому для того, щоб при заданій кількості «тактів» виконання програми **Evolver** знайти відповідне йому кількість поколінь (у термінології класичного генетичного алгоритму), необхідно значення *N* розділити на кількість особей у популяції. У прикладах, що далі представляються, «такти» будуть позначатися символом *t.* Приклад 3.21 демонструє застосування програми **Evolver** для оптимізації функції трьох змінних.

**Приклад 3.21**

За допомогою програми **Evolver** знайти мінімум функції

$$f(x_1,x_2, x_3) = x_1^2 + x_2^2 + x_3^2$$

для цілочислових $x_1, x_2, x_3$ в інтервалі [-5, 5].

Обчислення виконується для популяції розмірністю $N$ = 6. Використовувалося значення показника схрещування рівне 0,9. Це означає, що в результаті схрещування нащадок успадковує приблизно (з округленням до цілого) 90 % генів від першого батька й інші гени (близько 10 %) - від другого батька. Помітимо, що для показника схрещування рівного 1 буде спостерігатися такий же ефект, що і при відсутності схрещування, оскільки нащадок буде успадковувати всі гени тільки від першого батька. З цієї причини при обраному значенні даного показника, рівному 0,9, схрещування або не буде проводитися взагалі (точніше кажучи, нащадок виявиться абсолютно ідентичним своєму першому батькові), або схрещування буде виконано, і нащадок успадкує велику частину генів від першого батька. Для мутації обрана величина показника мутації рівна 0,1, що вказує, що значення 10% генів (з округленням до цілого) повинні піддатися випадковим змінам. За допомогою програми **Evolver** сформована наступна вихідна популяція:

[-1 3 -3]
[-1 -5 2]
[5 3 -4]
[2 -4 -0]
[5 4 2]
[4 -3 5]

Після шести «тактів», тобто для $t$ = 6, що відповідає одній ітерації класичного генетичного алгоритму, отримана популяція особей, упорядкованих по убуванню функції пристосованості так, як це показано на рис. 3.84.

| № п/п | $f(x_1, x_2, x_3)$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| 1 | 50 | 5 | 3 | –4 |
| 2 | 50 | 5 | 3 | –4 |
| 3 | 45 | 5 | 4 | 2 |
| 4 | 30 | –1 | –5 | 2 |
| 5 | 20 | 2 | –4 | 0 |
| 6 | 19 | –1 | 3 | –3 |

Рис. 3.84. Популяція особей для $t$ = 6 (приклад 3.21).

Відзначимо, що в популяцію не входить особа [4 -3 5] зі значенням функції пристосованості, рівним 50. Вона була виключена як «найгірша» особа (з найбільшим значенням функції пристосованості), і на її місце введена копія особи [5 3 -4]. Ця особа також виявляється «найгіршою», тому вона буде виключена на наступному «такті» *(t* = 7). Популяції для *t* = 7 і *t* = 8 представлені на рис. 3.85 і 3.86.

| № п/п | $f(x_1, x_2, x_3)$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| 1 | 50 | 5 | 4 | 2 |
| 2 | 50 | 5 | 3 | –4 |
| 3 | 45 | 5 | 4 | 2 |
| 4 | 30 | –1 | –5 | 2 |
| 5 | 20 | 2 | –4 | 0 |
| 6 | 19 | –1 | 3 | –3 |

Рис. 3.85. Популяція особей для *t* = 7 (приклад 3.21).

| № п/п | $f(x_1, x_2, x_3)$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| 1 | 50 | 0 | –4 | 2 |
| 2 | 50 | 5 | 4 | 2 |
| 3 | 45 | 5 | 4 | 2 |
| 4 | 30 | –1 | –5 | 2 |
| 5 | 20 | 2 | –4 | 0 |
| 6 | 19 | –1 | 3 | –3 |

Рис. 3.86**.** Популяція особей для *t* = 8 (приклад 3.21).

На останнім місці (№п/п=6) розміщується «найкраща до даного моменту» особа, що має найменше значення функції пристосованості. На першому місці (№ п/п = 1) показана знову введена в популяцію особа. На рис. 3.85 це копія особи [5 4 2] з попередньої популяції *(t* = 6), а на рис. 3.86 - нова особа [0 -4 2], отримана в результаті схрещування з наступною мутацією від

пари особей з популяції для *t*=7. Виділене прямокутником значення функції пристосованості в першому рядку таблиці відноситься до особи, що виключається з популяції.
На рис. 3.87 і 3.88 представлені популяції для *t* =17 і *t* =18.

| № п/п | $f(x_1, x_2, x_3)$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| 1 | 25 | 0 | –4 | 0 |
| 2 | 20 | 2 | –4 | 0 |
| 3 | 20 | 0 | –4 | 2 |
| 4 | 20 | 2 | –4 | 0 |
| 5 | 20 | 2 | –4 | 0 |
| 6 | 19 | –1 | 3 | –3 |

Рис. 3.87. Популяція особей для *t* = 17 (приклад 3.21).

| № п/п | $f(x_1, x_2, x_3)$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| 1 | 20 | 2 | –3 | 0 |
| 2 | 20 | 0 | –4 | 2 |
| 3 | 20 | 2 | –4 | 0 |
| 4 | 20 | 2 | –4 | 0 |
| 5 | 19 | –1 | 3 | –3 |
| 6 | 16 | 0 | –4 | 0 |

Рис. 3.88**.** Популяція особей для *t* = 18 (приклад 3.21).

Помітимо, що *t* = 18 відповідає трьом ітераціям класичного генетичного алгоритму. Нова особа у популяції на рис. 4.87 отримана схрещуванням пари особей з попередньої популяції (для *t*=16), а нова особа у популяції на рис. 3.88 сформована в результаті мутації середнього гена особи [2 -4 0] з попередньої популяції (для f=17).

На рис. 4.89 показана популяція для *t* = 41. Зверніть увагу, що *t*=42 відповідає семи ітераціям (поколінням) класичного генетичного алгоритму. «Найкраще до даного моменту» рішення - це [-1 2 0] зі значенням функції пристосованості, рівним 5.

Одержання «найкращого» рішення, рівного [0 0 0], з нульовим значенням функції пристосованості можливо у випадку, якщо при подальшому виконанні алгоритму відбудеться мутація другого гена (заміна -3, 3 або 2 на 0).

| № п/п | $f(x_1, x_2, x_3)$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| 1 | 10 | 0 | –3 | 0 |
| 2 | 10 | –1 | 3 | 0 |
| 3 | 10 | –1 | 3 | 0 |
| 4 | 10 | –1 | 3 | 0 |
| 5 | 9 | 0 | –3 | 0 |
| 6 | 5 | –1 | 2 | 0 |

Рис. 3.89**.** Популяція особей для *t* - 41 (приклад 3.21).

Наступний приклад являє собою узагальнення попереднього приклада за рахунок збільшення кількості змінних і розширення простору пошуку.

**Приклад 3.22**

За допомогою програми **Evolver** знайти мінімум функції

$$f(x_1, x_2, ..., x_6) = x_1^2 + x_2^2 + \ldots + x_6^2$$

для цілочислових $x_1, x_2, ..., x_6$ в інтервалі [-10, 10].

Обчислення виконується для популяції розмірністю N=50. Використовувалися прийняті за замовчуванням у програмі **Evolver** значення показника схрещування = 0,5 і показника мутації =0,06. На рис. 3.90 представлена модель рішення розглянутого приклада в табличному процесорі **Excel, що** містить початкові значення змінних $x_1, x_2, ..., x_6$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | МІНІМІЗАЦІЯ ФУНКЦІЇ F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6 | | | | | | |
| | для X1, X2, ..., X6 З ІНТЕРВАЛУ [-10,10] | | | | | | |
| | | | | | | | |
| | X1 = | 10 | | | | | |
| | X2 = | -8 | | | | | |
| | X3 = | 9 | | | | | |
| | X4 = | 7 | | | | | |
| | X5 = | -6 | | | | | |
| | X6 = | -9 | | | | | |
| | | | | | | | |
| | F(X1, X2, ..., X6) = | 411 | | | | | |
| | | | | | | | |

Рис. 3.90. Початкові значення змінних і значення функції для приклада 3.22, представлені в табличному процесорі **Excel**.

У вихідну популяцію включені 50 особей, перша з яких містить гени зі значеннями, показаними на рис. 3.90, а інші хромосоми згенеровані випадковим чином (гени мають дійсні значення з інтервалу від -10 до 10). Довжина кожної хромосоми дорівнює 6 і збігається з кількістю змінних $x_1$, $x_2$,..., $x_6$. Помітно, що приведені на рис. 3.90 початкові значення змінних $x_1$, $x_2$, ..., $x_6$ досить далекі від оптимального рішення, яким для розглянутої задачі (також як і в прикладі 3.21) буде хромосома з нульовими значеннями генів, тобто [000000]. Тому не викликає подиву те, що хромосома з аллелями, показаними на рис. 3.90, дуже швидко виключається з популяції. На рис. 3.91 зображені графіки зміни «найкращого» (нижня крива) і середнього (верхня крива) значення функції пристосованості з часом (точніше, зі збільшенням кількості «тактів»).

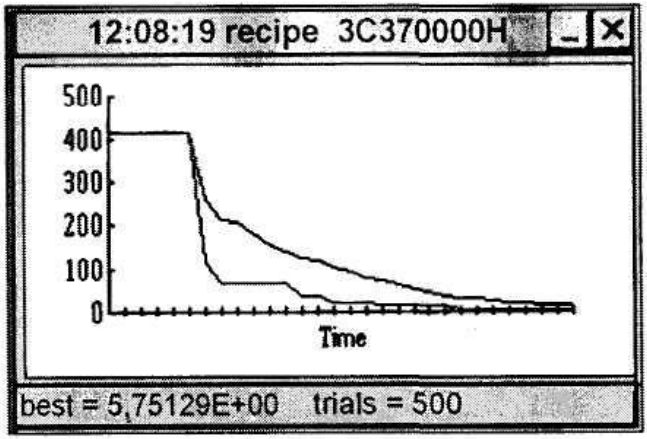



Рис. 3.91**.** Зміна «найкращого» (угорі) і середнього (унизу) значення функції пристосованості для приклада 3.22.

На цьому рисунку один момент часу відповідає 20 «тактам». Для *t*=500 (через 500 «тактів») найкраще значення функції пристосованості дорівнює 5,75129. На рис. 3.92 у виді стовпчастої діаграми показані значення функції пристосованості всіх особей популяції для *t* = 500.

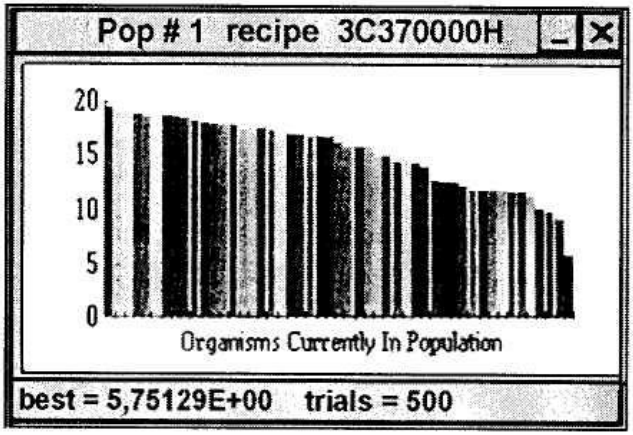



Рис. 3.92. Стовпчаста діаграма значень функції пристосованості особей у популяції при *t* = 500 для приклада 3.22.

Рис. 3.93 представляє значення змінних $x_1$, $x_2$, ..., $x_6$ і відповідне їм значення мінімізуємої функції, отримане після 7975 «тактів» виконання програми **Evolver.**

| | | |
|---|---|---|
| | Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6 | |
| | для X1, X2, ..., X6 из интервала [-10,10] | |
| | | |
| | X1 = | -0,021 |
| | X2 = | 0,024 |
| | X3 = | -0,096 |
| | X4 = | 0,005 |
| | X5 = | 0,009 |
| | X6 = | -0,055 |
| | | |
| | F(X1, X2, ..., X6) = | 0,013364 |
| | | |

Рис. 3.93. Значення змінних і функції для приклада 3.22 при $t$ = 7975.

На рис. 3.94 і 3.95 приведені аналогічні результати, але після 10623 і 11953 «тактів».

| | | |
|---|---|---|
| | Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6 | |
| | для X1, X2, ..., X6 из интервала [-10,10] | |
| | | |
| | X1 = | -0,021 |
| | X2 = | 0,024 |
| | X3 = | -0,011 |
| | X4 = | 0,005 |
| | X5 = | 0,009 |
| | X6 = | -0,055 |
| | | |
| | F(X1, X2, ..., X6) = | 0,004269 |
| | | |

Рис. 3.94**.** Значення змінних і функції для приклада 3.22 при $t$ = 10623.

| | | |
|---|---|---|
| Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6 | | |
| для X1, X2, ..., X6 из интервала [-10,10] | | |
| | | |
| X1 = | -0,02 | |
| X2 = | 0,001 | |
| X3 = | -0,011 | |
| X4 = | 0,002 | |
| X5 = | 0,009 | |
| X6 = | 0,001 | |
| | | |
| F(X1, X2, ..., X6) = | 0,000608 | |

Рис. 3.95**.** Значення змінних і функції для приклада 3.22 при $t$ = 11953.

Їхнє зіставлення показує, як одержувані рішення наближаються до оптимального. Рис. 3.96 демонструє «найкраще рішення», вироблене після приблизно 15000 «тактів» виконання програми **Evolver.** Наступна робота програми вже не змінює отриманий результат. Підсумкова популяція містить 50 особей зі значеннями $x_1, x_2, ..., x_6$, показаними на рис. 3.96.

| | | |
|---|---|---|
| Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6 | | |
| для X1, X2, ..., X6 из интервала [-10,10] | | |
| | | |
| X1 = | -0,02 | |
| X2 = | 0,001 | |
| X3 = | -0,011 | |
| X4 = | 0 | |
| X5 = | 0,009 | |
| X6 = | 0,001 | |
| | | |
| F(X1, X2, ..., X6) = | 0,000604 | |

Рис. 3.96**.** Значення змінних і функції для приклада 3.22 при $t$ = 15950.

Повторне поновлення цього алгоритму при тих же початкових даних (рис. 3.90), але інших випадкових значеннях інших членів вихідної популяції дає зовсім інший набір

«найкращих» значень змінних $x_1, x_2, ..., x_6$, що теж близькі до оптимального.

Наступні приклади зв'язані з мінімізацією більш складної функції 9 змінних, а саме - функції похибки нейронної мережі, що реалізує логічну систему XOR.

**Приклад 3.23**

За допомогою програми **Evolver** знайти оптимальний набір ваг нейронної мережі, зображеної на рис. 3.11 (приклад 3.3), якщо значення ваг лежать в інтервалі від -5 до 5.

Рішення цієї задачі полягає в підборі такого комплексу значень ваг $w_{11}$, $w_{12}$, $w_{21}$, $w_{22}$, $w_{31}$, $w_{32}$, а також $w_{10}$, $w_{20}$, $w_{30}$, що мінімізує значення похибки $Q$, заданої в прикладі 3.3. Це середня сума квадратів різностей між очікуваним і розрахунковим значенням на виході системи XOR для кожної з чотирьох пар входів, зазначених у табл. 2.1.

| $u_1$ | $u_2$ | $d = \mathrm{XOR}(u_1,u_2)$ |
|---|---|---|
| +1 | +1 | –1 |
| +1 | –1 | +1 |
| –1 | +1 | +1 |
| –1 | –1 | –1 |

Задача зводиться до мінімізації функції

$$Q = \frac{1}{4}\sum_{i=1}^{4} \varepsilon_i^{\,2}$$

щодо визначених вище дев'яти ваг, де $\varepsilon_i = d_i - y_i$, для $i$ = 1, 2, 3, 4, а $y_i$, розраховується по формулі

$y_i = 1/(1 + \exp((-\beta)(w_{31}(1/(1 + \exp((-\beta)(w_{11}u_{1,i} + w_{12}u_{2,i} + w_{10}))))) +$
$+ w_{32}(1/(1 + \exp((-\beta)(w_{21}u_{1,i} + w_{22}u_{2i} + w_{20}))))) + w_{30})))$.

Приймемо, що $\beta = 1$.

Задача моделювалася в табличному процесорі **Excel**, а оптимізація виконувалася за допомогою генетичного алгоритму програми **Evolver**. Для популяції з розмірністю, рівною 50, при значеннях показника схрещування рівного 0,5 і показника мутації рівного 0,06 після приблизно 30000 «тактів» отримано «найкраще рішення» у виді набору ваг, показаного на рис. 3.97. Мінімальне значення абсолютної похибки $Q$ для цих ваг складає 0,015171.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | веса | |
| | | | | | | w11 = | -4,9955 | |
| | | | | | | w12 = | -4,9985 | |
| входы | | заданное выходное значение | расчетное выходное значение | | | w21 = | 4,994 | |
| u1 | u2 | d | y | | | w22 = | -4,988 | |
| 0 | 0 | 0 | 0,123677 | | | w31 = | 4,9975 | |
| 0 | 1 | 1 | 0,878417 | | | w32 = | 4,9865 | |
| 1 | 0 | 1 | 0,876504 | | | w10 = | -2,8905 | |
| 1 | 1 | 0 | 0,123919 | | | w20 = | -2,9025 | |
| | | | | | | w30 = | -2,4805 | |
| | | | | | | | | |
| | | Минимальная абсолютная погрешность | | Q = | 0,015171 | | | |
| | | | | | | | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) | | | | | | |
| | | для каждой пары входных значений | | | | | | |

Рис. 3.97. «Найкращий» набір ваг з інтервалу [-5, 5] для нейронної мережі, що реалізує систему XOR (приклад 3.23).

Необхідно відзначити, що отримане «найкраще рішення» близьке наборові ваг $w_{11}$ = -5, $w_{12}$ = 5, $w_{21}$ = 5, $w_{22}$ = -5, $w_{31}$ = 5, $w_{32}$ = 5, $w_{10}$ = -3, $w_{20}$ = -3, $w_{30}$ = -2,5, для якого $Q$ = 0,015167. Нейронна мережа (див. рис. 3.11) з такими вагами демонструє симетрію, характерну для системи XOR. Очевидно, що знайдено не єдиний оптимальний набір ваг для мережі цього типу. При повторному запуску програми з тією же самою розмірністю популяції і таких же показників схрещування і мутації, швидше за все, буде знайдено «найкраще рішення», що містить інший «оптимальний» набір ваг. На рис. 3.98 приведено приклад альтернативного «найкращого» набору ваг, отриманий при повторному виконанні генетичного алгоритму програми **Evolver** і проведенні обчислення протягом 6000 «тактів».

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | веса | |
| | | | | | | w11 = | 3,333 | |
| | | | | | | w12 = | 3,3495 | |
| входы | | заданное выходное значение | расчетное выходное значение | | | w21 = | -4,979 | |
| u1 | u2 | d | y | | | w22 = | -4,977 | |
| 0 | 0 | 0 | 0,149766 | | | w31 = | -4,998 | |
| 0 | 1 | 1 | 0,83362 | | | w32 = | -4,989 | |
| 1 | 0 | 1 | 0,83522 | | | w10 = | -4,99 | |
| 1 | 1 | 0 | 0,171337 | | | w20 = | 1,9165 | |
| | | | | | | w30 = | 2,6465 | |
| | | | | | | | | |
| | | Минимальная абсолютная погрешность | | Q = | 0,026655 | | | |
| | | | | | | | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) | | | | | | |
| | | для каждой пары входных значений | | | | | | |

Рис. 3.98**.** Інший «найкращий» набір ваг для приклада 3.23.

При відомих припустимих комбінаціях ваг для системи XOR, легко пророчити оптимальний набір ваг, до якого прагне рішення при продовженні виконання цього алгоритму.

Наступні приклади відносяться до тієї ж задачі, однак при розширеному діапазоні зміни ваг.

**Приклад 3.24**

За допомогою програми **Evolver** знайти оптимальний набір ваг нейронної мережі, зображенної на рис. 3.11 (приклад 3.3), якщо значення ваг лежать в інтервалі від -10 до 10.

При початкових значеннях, що збігаються з приведеними в прикладі 3.23, було проведено три сеанси обчислень. Через 30000 «тактів» у першому, другому і третьому сеансах отримані «найкращі» рішення, показані на рис. 3.99, 3.100 і 3.102 відповідно.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | веса | |
| | | | | | w11 = | 9,997 | |
| | | | | | w12 = | -9,99 | |
| входы | | заданное выходное значение | расчетное выходное значение | | w21 = | 9,982 | |
| u1 | u2 | d | y | | w22 = | -9,993 | |
| 0 | 0 | 0 | 7,67E-03 | | w31 = | -9,983 | |
| 0 | 1 | 1 | 0,992914 | | w32 = | 9,972 | |
| 1 | 0 | 1 | 0,99262 | | w10 = | 5,139 | |
| 1 | 1 | 0 | 7,66E-03 | | w20 = | -5,462 | |
| | | | | | w30 = | 5,02 | |
| | | Минимальная абсолютная погрешность | | Q = | 5,56E-05 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | |

Рис**.** 3.99. «Найкращий» набір ваг з інтервалу [-10, 10] для нейронної мережі, що реалізує систему XOR (приклад 3.24).

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | веса | |
| | | | | | w11 = | -9,939 | |
| | | | | | w12 = | 9,972 | |
| входы | | заданное выходное значение | расчетное выходное значение | | w21 = | 9,975 | |
| u1 | u2 | d | y | | w22 = | -9,999 | |
| 0 | 0 | 0 | 0,007916 | | w31 = | 9,904 | |
| 0 | 1 | 1 | 0,992505 | | w32 = | 9,935 | |
| 1 | 0 | 1 | 0,992687 | | w10 = | -5,314 | |
| 1 | 1 | 0 | 0,007921 | | w20 = | -5,379 | |
| | | | | | w30 = | -4,925 | |
| | | Минимальная абсолютная погрешность | | Q = | 5,88E-05 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | |

Рис. 3.100**.** Інший «найкращий» набір ваг для приклада 3.24.

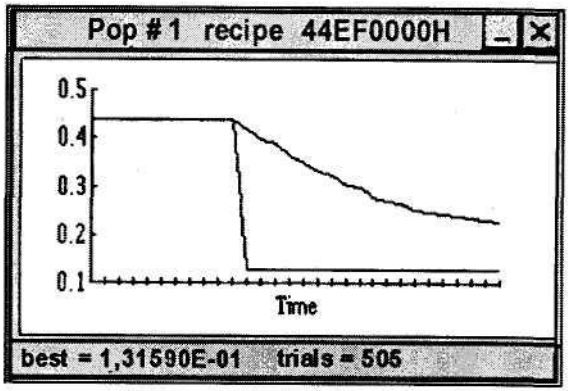



Рис. 3.101. Зміна «найкращого» і середнього значення функції пристосованості для приклада 3.24.

На рис. 3.101 зображені графіки зміни «найкращого» (нижня крива) і середнього (верхня крива) значення функції пристосованості для цього прикладу після 505 «тактів».

Функція пристосованості для системи XOR задається формулою розрахунку похибки $Q$ (див. приклад 3.23) і змінюється в межах від 0 до 1. На рис. 3.101 помітно стрімке зменшення найкращого значення функції пристосованості до «найкращого на даний момент» значення, рівного 0,13159. Одиниця на тимчасовій осі цього графіка відповідає 20 «тактам». Набір ваг, до якого прагне «найкраще рішення», характеризується тим, що $w_{11} = w_{12}$ і $w_{21} = w_{22}$, що типово для логічної системи XOR. Результати, що представлені на рис. 3.99 і 3.100, можна порівняти з показаними на рис. 3.97 для приклада 3.23, а інформацію на рис. 3.102 - зіставити з рис. 3.98.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | веса |
| | | | | | | | | w11 = | -6,778 |
| | | | | | | | | w12 = | -6,788 |
| входы | | заданное выходное значение | | расчетное выходное значение | | | | w21 = | -9,86 |
| u1 | u2 | d | | y | | | | w22 = | -9,937 |
| 0 | 0 | 0 | | 9,65E-03 | | | | w31 = | 9,995 |
| 0 | 1 | 1 | | 0,991612 | | | | w32 = | -9,989 |
| 1 | 0 | 1 | | 0,991623 | | | | w10 = | 9,941 |
| 1 | 1 | 0 | | 1,08E-02 | | | | w20 = | 4,226 |
| | | | | | | | | w30 = | -4,78 |
| | | | | | | | | | |
| | | Минимальная абсолютная погрешность | | | | Q = | 8,74E-05 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | | | |

Рис. 3.102. Ще один «найкращий» набір ваг для приклада 3.24.

Помітимо, що мінімальна похибка у всіх трьох випадках (рис. 3.99, 3.100 і 3.102) виявляється значно меншою, ніж для інтервалу зміни від -5 до 5, розглянутого в прикладі 3.23 (рис. 3.97 і 3.98).

**Приклад 3.25**

За допомогою програми **Evolver** знайти оптимальний набір ваг нейронної мережі, яка зображена на рис. 3.11 (приклад 3.3), якщо значення ваг лежать в інтервалі від -15 до 15.

Застосовувався той же генетичний алгоритм програми **Evolver**, що й у попередніх прикладах, але для розширеного інтервалу зміни ваг. Після 30000 «тактів» отримано «найкраще рішення», яке показано на рис. 3.103.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | веса |
| | | | | | | | | w11 = | 14,7225 |
| | | | | | | | | w12 = | -14,985 |
| входы | | заданное выходное значение | | расчетное выходное значение | | | | w21 = | 14,934 |
| u1 | u2 | d | | y | | | | w22 = | -14,9865 |
| 0 | 0 | 0 | | 5,26E-04 | | | | w31 = | 14,9775 |
| 0 | 1 | 1 | | 0,999369 | | | | w32 = | -14,946 |
| 1 | 0 | 1 | | 0,999389 | | | | w10 = | -7,107 |
| 1 | 1 | 0 | | 5,25E-04 | | | | w20 = | 7,4685 |
| | | | | | | | | w30 = | 7,3755 |
| | | | | | | | | | |
| | | Минимальная абсолютная погрешность | | | | Q = | 3,31E-07 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | | | |

Рис. 3.103. «Найкращий» набір ваг з інтервалу [-15, 15] для нейронної мережі, що реалізує систему XOR (приклад 3.25).

Повторне виконання того ж алгоритму при тих же початкових умовах через 30000 «тактів» дало «найкращий» набір ваг, представлений на рис. 3.104.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | веса |
| | | | | | | | w11 = | -14,9985 |
| | | | | | | | w12 = | 14,94 |
| входы | | заданное выходное значение | | расчетное выходное значение | | | w21 = | -14,7675 |
| u1 | u2 | d | | y | | | w22 = | 14,9835 |
| 0 | 0 | 0 | | 5,23E-04 | | | w31 = | 14,913 |
| 0 | 1 | 1 | | 0,999346 | | | w32 = | -14,9205 |
| 1 | 0 | 1 | | 0,999351 | | | w10 = | -7,662 |
| 1 | 1 | 0 | | 5,22E-04 | | | w20 = | 7,464 |
| | | | | | | | w30 = | 7,35 |
| | | Минимальная абсолютная погрешность | | | Q = | 3,49E-07 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | | |

Рис. 3.104. Інший «найкращий» набір ваг для приклада 3.25.

Головне розходження між рішеннями на рис. 3.103 і 3.104 полягає в тім, що в першому випадку ваги $w_{11}$ і $w_{21}$ додатні, а ваги $w_{12}$ і $w_{22}$ від'ємні, тоді як у другому випадку ситуація виявилася протилежною. У розглянутому прикладі отримано дві з множини можливих комбінацій ваг для нейронної мережі, що реалізує систему XOR. Ще одне альтернативне рішення приведено на рис. 3.105.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | веса |
| | | | | | | | w11 = | 14,223 |
| | | | | | | | w12 = | -14,5155 |
| входы | | заданное выходное значение | | расчетное выходное значение | | | w21 = | -14,2815 |
| u1 | u2 | d | | y | | | w22 = | 13,585 |
| 0 | 0 | 0 | | 5,92E-04 | | | w31 = | 14,9595 |
| 0 | 1 | 1 | | 0,999441 | | | w32 = | 14,9625 |
| 1 | 0 | 1 | | 0,999437 | | | w10 = | -7,554 |
| 1 | 1 | 0 | | 5,85E-04 | | | w20 = | -6,6645 |
| | | | | | | | w30 = | -7,458 |
| | | Минимальная абсолютная погрешность | | | Q = | 3,30E-07 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | | |

Рис. 3.105. Ще один «найкращий» набір ваг для приклада 3.25 (отриманий після приблизно 12000 «тактів»).

Значення вхідних у нього ваг отримані після приблизно 12000 «тактів». Нескладно угадати, як будуть змінюватися ці значення при збільшенні кількості «тактів». Отже, це ще одна комбінація, подібна до рішень на рисунках 3.103 і 3.104, причому в даному випадку ваги $w_{11}$, $w_{22}$, $w_{31}$ і $w_{32}$ додатні, а ваги $w_{12}$, $w_{21}$, $w_{10}$, $w_{20}$ і $w_{30}$ від'ємні.

**Приклад 3.26**

За допомогою програми **Evolver** знайти оптимальний набір ваг нейронної мережі, зображенної на рис. 3.11 (приклад 3.3), якщо значення ваг лежать в інтервалі від -20 до 20.

Це та ж задача, що й у попередніх прикладах, однак інтервал зміни ваг значно розширено. Задача вирішувалася при тій же розмірності популяції і таких же значеннях показників схрещування і мутації, як і в попередніх прикладах. Стовпчаста діаграма на рис. 3.106 демонструє значення функції пристосованості особей у популяції після приблизно 1500 «тактів».

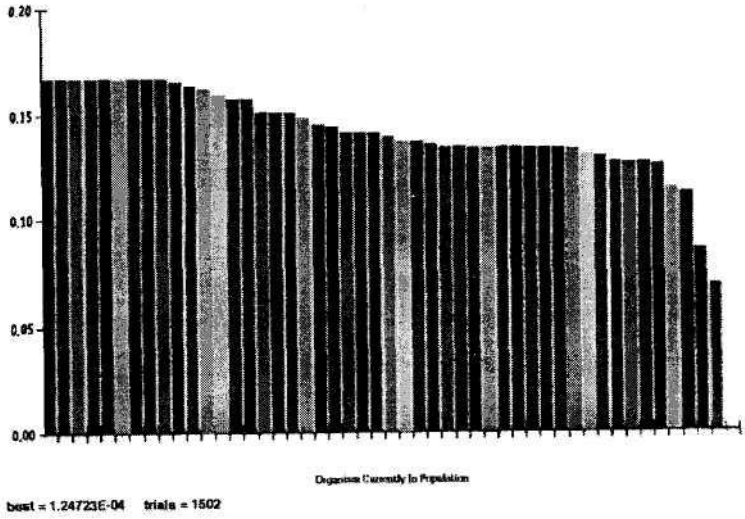



Рис. 3.106. Стовпчаста діаграма значень функції пристосованості особей у популяції при $f$ = 1502 для приклада 3.26.

Звичайно, це значення в інтервалі від 0 до 1. На рис. 3.107 зображено графіки зміни «найкращого» (нижня крива) і середнього (верхня крива) значення функції пристосованості для цього приклада після приблизно 1500 «тактів».

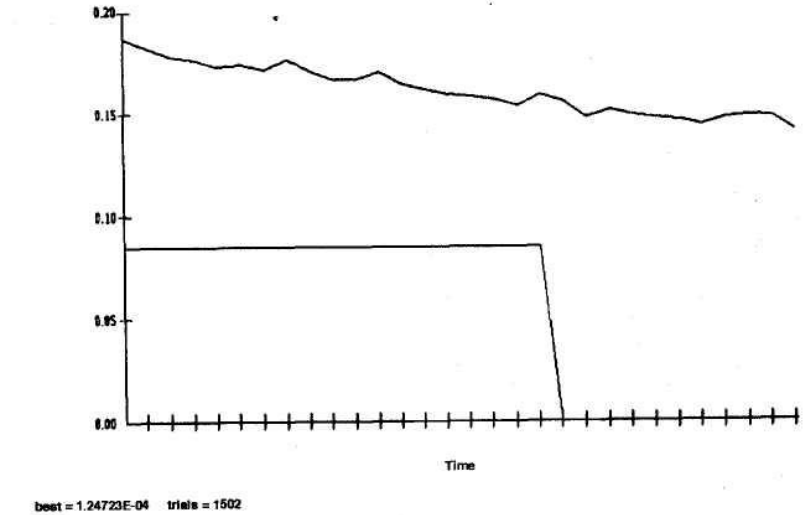



Рис. 3.107. Зміна «найкращого» (унизу) і середнього (угорі) значення функції пристосованості для приклада 3.26.

Одиниця на тимчасовій осі цього графіка відповідає 20 «тактам». «Найкраще на даний момент» (після 1502 «тактів») значення функції пристосованості складає $1{,}247 * 10^{-4}$. Рис. 3.108 представляє «найкращий» набір ваг, отриманий після 33000 «тактів», а рис. 3.109 - після 63000 «тактів». Помітно, що значення сполучних ваг прагнуть до 20 або до -20, а значення $w_{10}$, $w_{20}$ і $w_{30}$ - відповідно до 10, -10, 10.

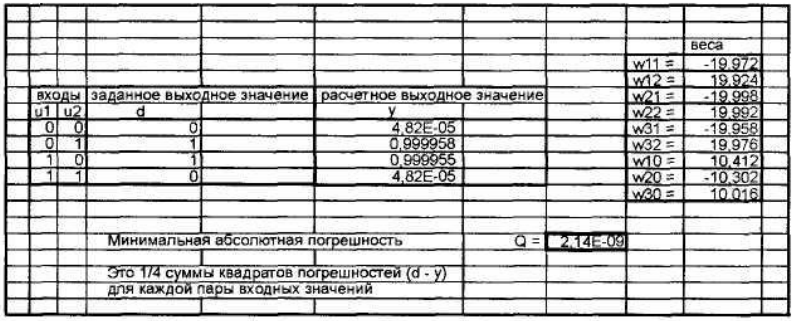
| входы | | заданное выходное значение | расчетное выходное значение |
|---|---|---|---|
| u1 | u2 | d | y |
| 0 | 0 | 0 | 4,82E-05 |
| 0 | 1 | 1 | 0,999952 |
| 1 | 0 | 1 | 0,999955 |
| 1 | 1 | 0 | 4,82E-05 |

| | веса |
|---|---|
| w11 = | 19,972 |
| w12 = | 19,826 |
| w21 = | -19,994 |
| w22 = | 19,814 |
| w31 = | -19,958 |
| w32 = | 19,89 |
| w10 = | 9,822 |
| w20 = | -10,064 |
| w30 = | 10,016 |

Минимальная абсолютная погрешность Q = 2,23E-09

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 3.108. «Найкращий» набір ваг з інтервалу [-20, 20] при $t$ = 33000 для приклада 3.26.



| входы | | заданное выходное значение | расчетное выходное значение |
|---|---|---|---|
| u1 | u2 | d | y |
| 0 | 0 | 0 | 4,82E-05 |
| 0 | 1 | 1 | 0,999958 |
| 1 | 0 | 1 | 0,999955 |
| 1 | 1 | 0 | 4,82E-05 |

| | веса |
|---|---|
| w11 = | -19,972 |
| w12 = | 19,924 |
| w21 = | -19,998 |
| w22 = | 19,992 |
| w31 = | -19,958 |
| w32 = | 19,976 |
| w10 = | 10,412 |
| w20 = | -10,302 |
| w30 = | 10,016 |

Минимальная абсолютная погрешность Q = 2,14E-09

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 3.109. «Найкращий» набір ваг з інтервалу [-20, 20] при $t$ = 63000 для приклада 3.26.

Якщо продовжити виконання алгоритму, то при подальшому збільшенні кількості «тактів» деякі ваги приймуть значення, рівні 20. Очевидно, що набори ваг на рис. 3.108 і 3.109 являють собою лише два елементи з множини припустимих комбінацій ваг нейронної мережі, що реалізує логічну систему XOR. На рис. 3.110 показано зовсім інший набір «найкращих» ваг, отриманих при черговому поновленні генетичного алгоритму програми **Evolver.**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | веса | |
| | | | | | | w11 = | -19,992 | |
| | | | | | | w12 = | -19,986 | |
| | входы | | заданное выходное значение | расчетное выходное значение | | w21 = | 13,376 | |
| | u1 | u2 | d | y | | w22 = | 13,356 | |
| | 0 | 0 | 0 | 4,65E-05 | | w31 = | -19,988 | |
| | 0 | 1 | 1 | 0,999954 | | w32 = | -19,992 | |
| | 1 | 0 | 1 | 0,999954 | | w10 = | 9,648 | |
| | 1 | 1 | 0 | 4,73E-05 | | w20 = | -20 | |
| | | | | | | w30 = | 10,01 | |
| | | | Минимальная абсолютная погрешность | | Q = | 2,17E-09 | | |
| | | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | |

Рис. 3.110**.** Інший «найкращий» набір ваг для приклада 3.26.

Результати зафіксовано після приблизно 15000 тактів при тих же, що і раніше, розмірностей популяції, значеннях показників схрещування і мутації й інтервалу зміни ваг. Відзначимо, що «найкраще рішення» на рис. 3.110 теж являє собою одну з припустимих комбінацій ваг системи XOR, оскільки воно прагне до оптимального рішення, у якому $w_{11} = w_{12}$ і $w_{21} = w_{22}$.

З прикладів 3.23 - 3.26 випливає висновок, що чим більше інтервал зміни ваг, тим менше мінімальне значення похибки. Тому для одержання нейронної мережі, яка зможе найбільш точно реалізувати систему XOR (тобто для якої різниця між еталоними і розрахунковим вихідними значеннями буде мінімальною), необхідно підбирати ваги з як можна більш широкого інтервалу припустимих значень. Очевидно, що підбір цих ваг повинен бути оптимальним і мінімізувати похибку так, як це робилося в прикладах 3.23 - 3.26. Якби було потрібно вибирати значення ваг з інтервалу, більш вузького, чим [-5, 5], то забезпечити мінімальну похибку у межах, наприклад, 0,1, виявилося б ще складніше. На рис. 3.111 показано «найкращий» набір ваг, знайдений також як у прикладах 3.23 - 3.26, але для інтервалу [-3, 3].

| входы | | заданное выходное значение | расчетное выходное значение |
|---|---|---|---|
| u1 | u2 | d | y |
| 0 | 0 | 0 | 0,323599 |
| 0 | 1 | 1 | 0,679728 |
| 1 | 0 | 1 | 0,681891 |
| 1 | 1 | 0 | 0,323578 |

| | веса |
|---|---|
| w11 = | 2,9946 |
| w12 = | -2,9964 |
| w21 = | -2,9934 |
| w22 = | 2,9949 |
| w31 = | 2,9997 |
| w32 = | 2,9991 |
| w10 = | -2,0394 |
| w20 = | -2,0559 |
| w30 = | 1,4229 |

Минимальная абсолютная погрешность Q = 0,103297

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 3.111. «Найкращий» набір ваг з інтервалу [-3, 3] для нейронної мережі, що реалізує систему XOR (приклад 3.26).

Після 40000 «тактів» мінімальне значення похибки перевищувало 0,1. Помітимо, що для інтервалу зміни ваг від -20 до 20 можна досягти похибку порядку $10^{-4}$ уже через 1500 «тактів» (рис. 3.106 і 3.107), а для інтервалу [-10, 10] мінімальна похибка на рівні 0,1 була досягнута вже через 500 «тактів» (рис. 3.101). У прикладах 3.23 - 3.26 початкові значення ваг, що враховуються програмою **Evolver** як гени однієї з особей вихідної популяції, у загальному випадку приймаються рівними 1. Для нейронної мережі, зображенної на рис. 3.11, у цьому випадку величина похибки буде складати $Q$ = 0,438, що сильно відрізняється від мінімального значення. Звичайно, можна прийняти й інші початкові значення, далекі від оптимальних.

**Приклад 3.27**

За допомогою програми **Evolver** знайти оптимальний набір ваг нейронної мережі, зображенної на рис. 3.11 (приклад 3.3), якщо значення ваг лежать в інтервалі від -7 до 8.

Початкові значення ваг, що складають генотип однієї з особей вихідної популяції, представлені на рис. 3.112.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | веса |
| | | | | | | | w11 = | 3,1592 |
| | | | | | | | w12 = | 4,5006 |
| входы | | заданное выходное значение | расчетное выходное значение | | | | w21 = | 1,715 |
| u1 | u2 | d | y | | | | w22 = | -2,5636 |
| 0 | 0 | 0 | 0,105235 | | | | w31 = | -2,5994 |
| 0 | 1 | 1 | 0,290778 | | | | w32 = | -2,0924 |
| 1 | 0 | 1 | 0,016872 | | | | w10 = | -0,321 |
| 1 | 1 | 0 | 0,204858 | | | | w20 = | -2,2872 |
| | | | | | | | w30 = | -0,8546 |
| | | | | | | | | |
| | | Минимальная абсолютная погрешность | | | Q = | 0,380644 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | | |

Рис. 3.112**.** Початкові значення ваг для приклада 3.27.

Видно, що значення в для конкретних пар входів зовсім не відповідають функції XOR. Значення похибки $Q$=0,38 вважається абсолютно незадовільним.

Для рішення задачі застосовувався той же генетичний алгоритм при тій же розмірності популяції і тих же показниках схрещування і мутації, що й у попередніх прикладах. Усього лише через 96 «тактів» отримано найкращі значення функції пристосованості конкретних особей у популяції, які показані на рис. 3.113 (стовпчаста діаграма).

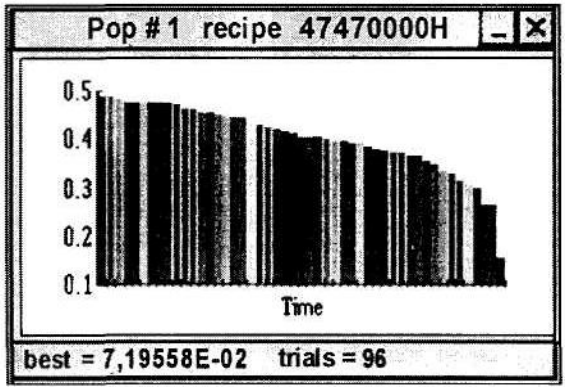



Рис. 3.113**.** Стовпчаста діаграма значень функції пристосованості особей у популяції при $t$ = 96 для приклада 3.27.

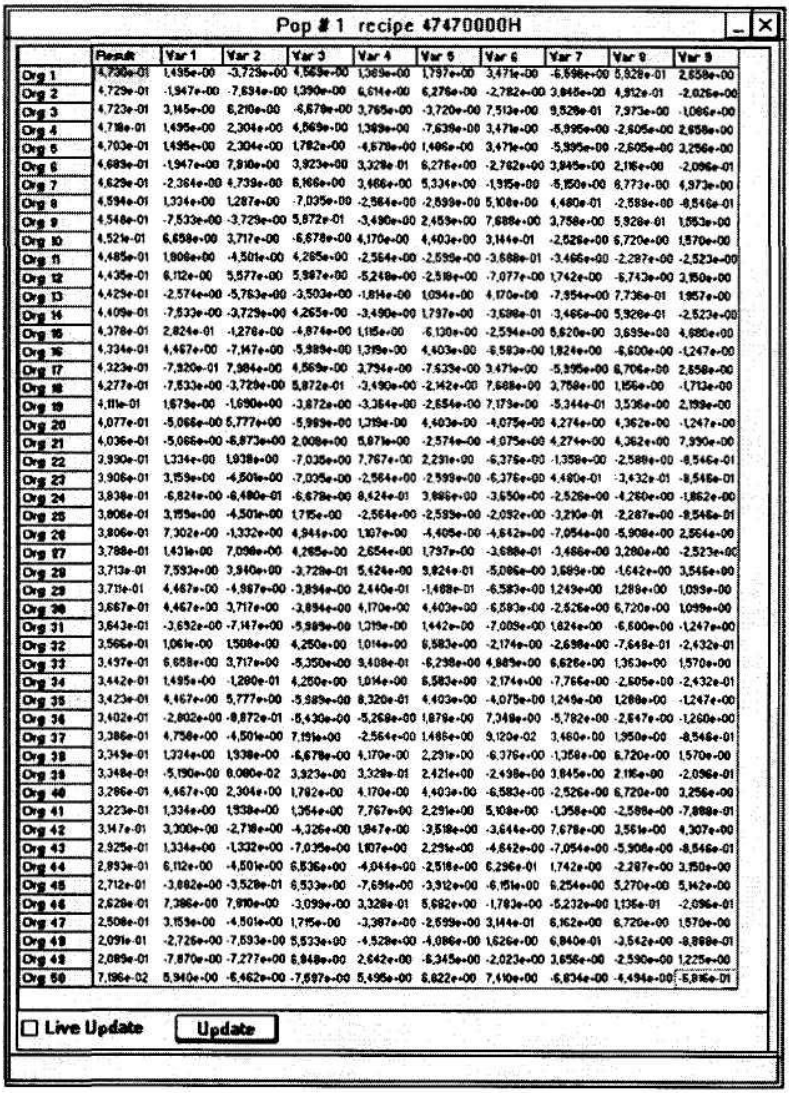



Рис. 3.114**.** Популяція особей при *t* = 96 для приклада 3.27.

У свою чергу, на рис. 3.114 представлена таблиця, що містить генотипи всіх 50 особей цієї популяції ( щоскладаються з дев'яти дійсних чисел, що відповідають конкретним вагам) і значення їхньої функції пристосованості, розміщені в першому стовпчику.

Звертаємо увагу на те, що початкова хромосома з аллелями, рівними значенням ваг з рис. 3.112, розташована в середній частині цієї таблиці і позначена «Організм 25». Варто пам'ятати, що особи в популяції упорядковані від «найгіршого» до «найкращого». Остання особа («Організм 50») - це хромосома, «найкраща на даний момент», тобто після 96 «тактів». Значення ваг, що входять до складу цієї хромосоми, показано на рис. 3.115 разом з вихідними значеннями для конкретних пар входів і значенням мінімізуємої похибки. Отже, після 96 «тактів» отримано винятково гарний результат.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | веса | |
| | | | | | w11 = | 5,94 | |
| | | | | | w12 = | -6,4616 | |
| входы | | заданное выходное значение | расчетное выходное значение | | w21 = | -7,597 | |
| u1 | u2 | d | y | | w22 = | 5,4952 | |
| 0 | 0 | 0 | 0,356101 | | w31 = | 6,8216 | |
| 0 | 1 | 1 | 0,991321 | | w32 = | 7,4104 | |
| 1 | 0 | 1 | 0,785702 | | w10 = | -6,8336 | |
| 1 | 1 | 0 | 0,33914 | | w20 = | -4,4936 | |
| | | | | | w30 = | -0,6816 | |
| | | | | | | | |
| | | Минимальная абсолютная погрешность | | Q = | 0,071956 | | |
| | | Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений | | | | | |

Рис. 3.115. Набір ваг, отриманий при $t$ = 96 для приклада 3.27.

При повторному виконанні цієї ж програми із самого початку при тому же вихідному наборі ваг (рис. 3.112) через приблизно 10000 «тактів» отримано похибку $Q$ = 0,0962, через 20000 «тактів» - $Q$ = 0,07956, а через 21000 «тактів» - $Q$ = 0,0687. Після 24000 тактів ця похибка зменшилася до значення $Q$=0,0077 (рис. 3.116), а після 40000 «тактів» отримано «найкращий» набір ваг, показаний на рис. 3.117.

| входы | | заданное выходное значение | расчетное выходное значение |
|---|---|---|---|
| u1 | u2 | d | y |
| 0 | 0 | 0 | 0,021682 |
| 0 | 1 | 1 | 0,864866 |
| 1 | 0 | 1 | 0,952198 |
| 1 | 1 | 0 | 0,096875 |

| | веса |
|---|---|
| w11 = | 7,176 |
| w12 = | -7,953 |
| w21 = | 5,8144 |
| w22 = | -3,4576 |
| w31 = | -6,0736 |
| w32 = | 7,9752 |
| w10 = | 3,936 |
| w20 = | -3,7112 |
| w30 = | 1,9576 |

Минимальная абсолютная погрешность Q = 7,60E-03

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 3.116**.** «Найкращий» набір ваг, отриманий через 21000 «тактів» .

| входы | | заданное выходное значение | расчетное выходное значение |
|---|---|---|---|
| u1 | u2 | d | y |
| 0 | 0 | 0 | 0,020804 |
| 0 | 1 | 1 | 0,975645 |
| 1 | 0 | 1 | 0,97539 |
| 1 | 1 | 0 | 0,020713 |

| | веса |
|---|---|
| w11 = | 7,9696 |
| w12 = | -7,953 |
| w21 = | 7,9816 |
| w22 = | -7,9976 |
| w31 = | -7,964 |
| w32 = | 7,9752 |
| w10 = | 3,936 |
| w20 = | -4,1104 |
| w30 = | 3,8312 |

Минимальная абсолютная погрешность Q = 5,16E-04

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 3.117**.** «Найкращий» набір ваг, отриманий для випадку рис. 3.116, після 40000 «тактів».

Таким чином, процеси виконання того самого алгоритму при однаковій розмірності популяції і повторюваних значеннях показників схрещування і мутації виявляються зовсім різними. Це, звичайно, визначається випадковим набором особей вихідної популяції і випадковим проводженням схрещування і мутації.

Для випадку на рис. 3.115 отримано результат, подібний представленому на рис. 3.117, причому ваги $w_{21}$, $w_{22}$, $w_{31}$, а також $w_{10}$ і $w_{30}$ мають протилежні знаки.

При використанні генетичного алгоритму програми **Evolver** для рішення задач, аналогічних розглянутим у прикладах 3.23 - 3.27, генетичні оператори змінюють значення окремих ваг (мутація) і здійснюють обмін значень ваг, обраних випадковим

чином, між двома хромосомами (схрещування). У той же час, у програмі **FlexTool** (приклад 3.20) генетичні оператори рекомбінують гени усередині хромосом, що являють собою двоїчні послідовності, які відповідають закодованим значенням конкретних ваг. У програмі **Evolver** хромосоми складаються з генів, що мають дійсні значення, що збігаються зі значеннями ваг. Програма **Evolver** - це прекрасний інструмент для оптимізації функції декількох змінних, при цьому вона може застосовуватися для пошуку максимуму або мінімуму функції двох і навіть одної змінної. Помітимо, що у випадку тільки одної змінної схрещування практично не виконується, і виконується тільки операція мутації, що характерно для так називаного еволюційного програмування. Усі приклади, що вирішені за допомогою програми **FlexTool,** можна вирішувати також і засобами програми **Evolver.** Наприклад, на рис. 3.118 показана зміна «найкращого» і середнього значення функції пристосованості у випадку пошуку максимуму функції, представленної на рис. 3.39.

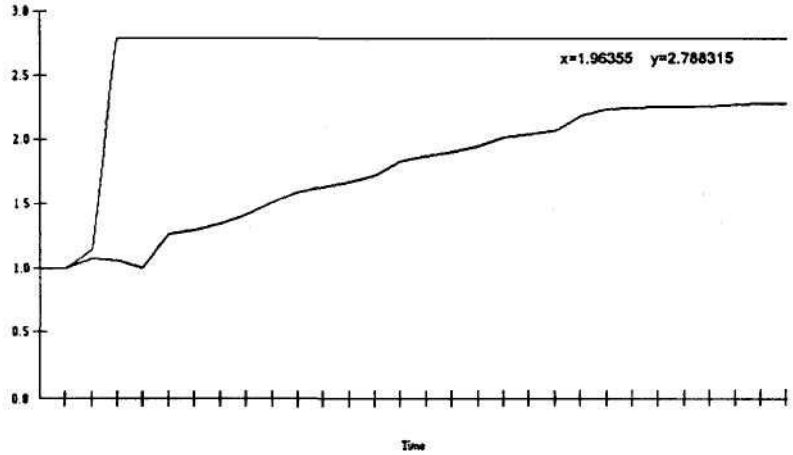



Рис. 3.118. Зміна «найкращого» і середнього значення функції пристосованості у випадку пошуку максимуму функції, зображенної на рис. 3.39, за допомогою програми **Evolver.**

Один розподіл на тимчасовій осі відповідає 20 «тактам». Обчислення проводилися з прийнятими за замовчуванням

значеннями показників схрещування (0,5) і мутації (0,06), а також розмірності популяції (50). Графіки і «найкращі» рішення реєструвалися після 700 «тактів» (*t*=700). Приклад 3.28 присвячений оптимізації функції двох змінних, графік якої зображено на рис. 3.56.

**Приклад 3.28**

За допомогою програми **Evolver** знайти мінімум функції з приклада 3.18.

Графік цієї функції зображено на рис. 3.56. У прикладі 3.18 приведено координати чотирьох точок, у яких ця функція має мінімальне значення, рівне 0. Генетичний алгоритм повинний знайти одну з цих точок. Застосовується програма **Evolver** із прийнятими за замовчуванням значеннями показників схрещування (0,5) і мутації (0,06). Розмірність популяції обрана рівною 30. На рис. 3.119 представлені початкові значення змінних $x_1$ і $x_2$, що введені у вихідну популяцію як гени однієї з хромосом.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | |
| 2 | | | ФУНКЦИЯ HIMMELBLAU | | | | | Вычисление минимума | | | |
| 3 | | | | | | | | | | | |
| 4 | | | F(X1,X2) = (X1^2 +X2 -11)^2 + (X1 +X2^2 -7)^2 | | | | | -10 <= X1,X2 <= 10 | | | |
| 5 | | | | | | | | | | | |
| 6 | | | | | | | | | | | |
| 7 | X1 = | 10 | | | | | | | | | |
| 8 | X2 = | 10 | | | | | | | | | |
| 9 | F(X1,X2)= | 20410 | | | | | | | | | |

Рис. 3.119**.** Початкові значення для приклада 3.28.

Значення функції пристосованості для цієї хромосоми дуже велике - воно складає 20410. Зрозуміло, що дана хромосома буде вже незабаром виключена з популяції, що підтверджується рис. 3.120.

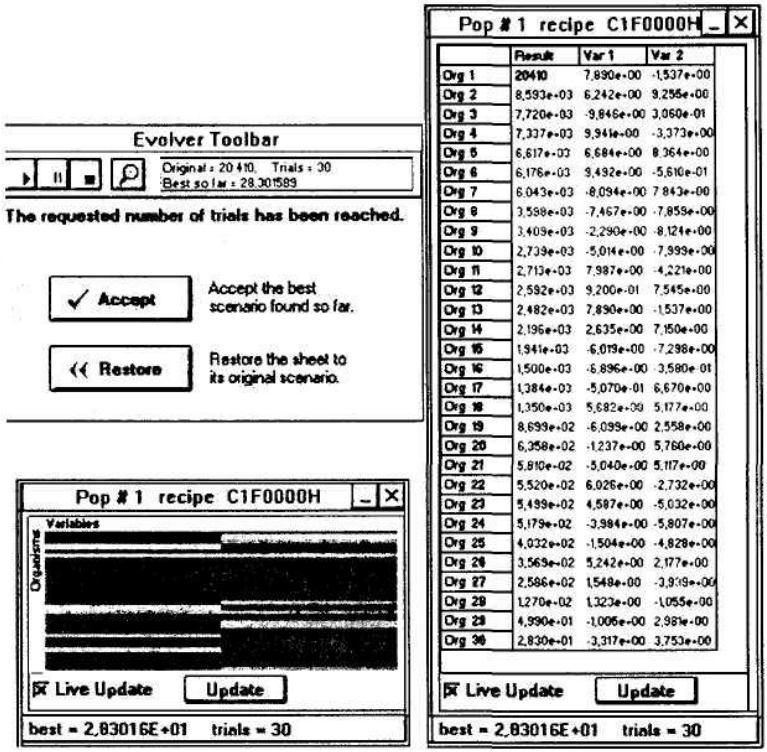



Рис. 3.120. Популяція особей при f=30 для приклада 3.28.

На цьому малюнку показані хромосоми популяції для *t*=30 (після 30 «тактів»), що відповідає першій ітерації (першому поколінню) класичного генетичного алгоритму. Змінні *var*1 і *var2* позначають відповідно $x_1$, і $x_2$, перший стовпець *(result)* містить значення функції пристосованості конкретних хромосом. Перше значення (20410) належить особі, яка виключена з популяції. На її місце вводиться нова хромосома з аллелями, рівними 7,89 і -1,537, для якої значення функції пристосованості ще тільки має бути розрахованим. У лівому нижньому куті рисунка демонструється різнорідність особей цієї популяції. У даному випадку вона також досить велика. На рис. 3.121 приведені

аналогічні графіки для популяції після 60 «тактів» *(t*=60), а також стовпчаста діаграма, що ілюструє конкретні особи.

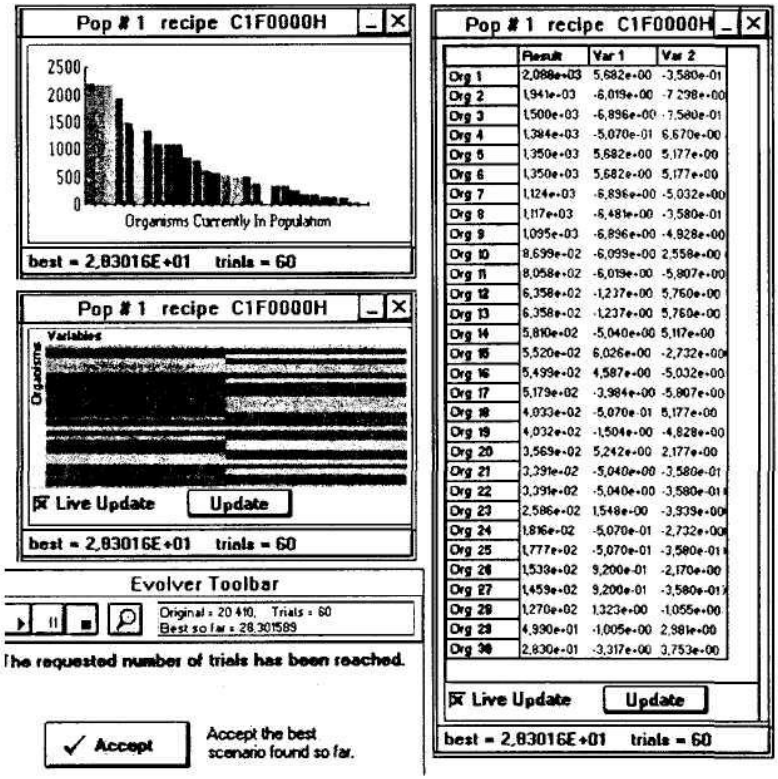



Рис. 3.121. Популяція особей при *t*=60 для приклада 3.28.

«Найкраще на даний момент» значення функції пристосованості усе ще занадто велике і складає 28,3.

На рис. 3.122 представлені ті ж графіки після 150 «тактів» *(t*=150), які доповнені (у лівому нижньому куті) графіком зміни «найкращого» значення функції пристосованості, що стрімко зменшується.

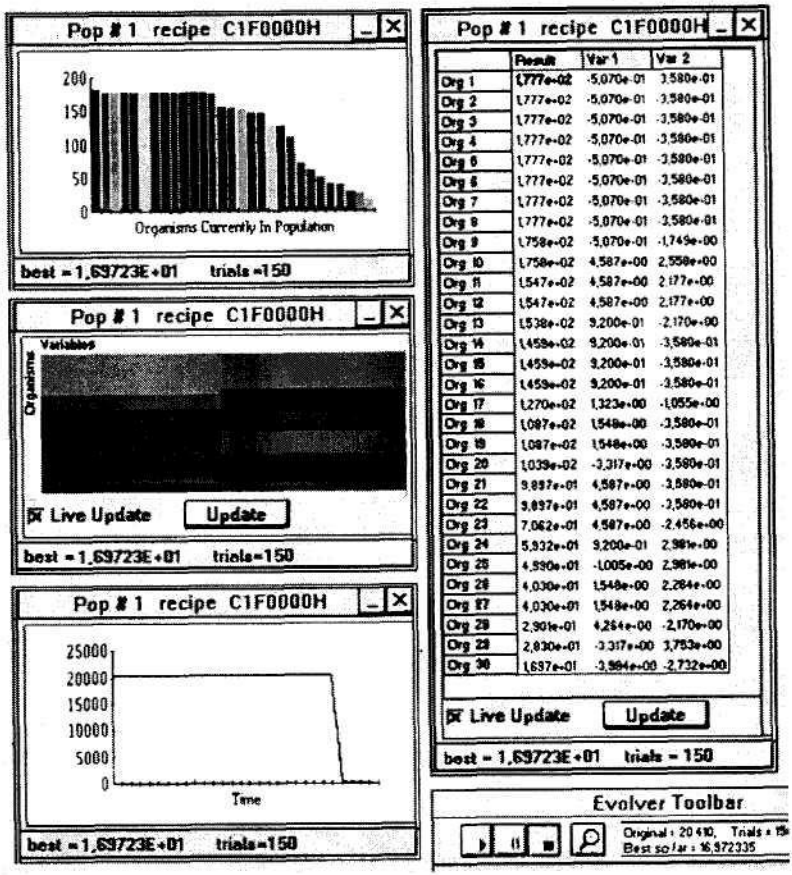



Рис. 3.122. Популяція особей і графік функції пристосованості при $t$=150 для приклада 3.28.

У середній частині ліворуч показана різнорідність популяції, що також значно знизилася.

Ще менша різнорідність популяції спостерігається на рис. 3.123, що містить також стовпчасту діаграму і значення генів конкретних хромосом популяції.

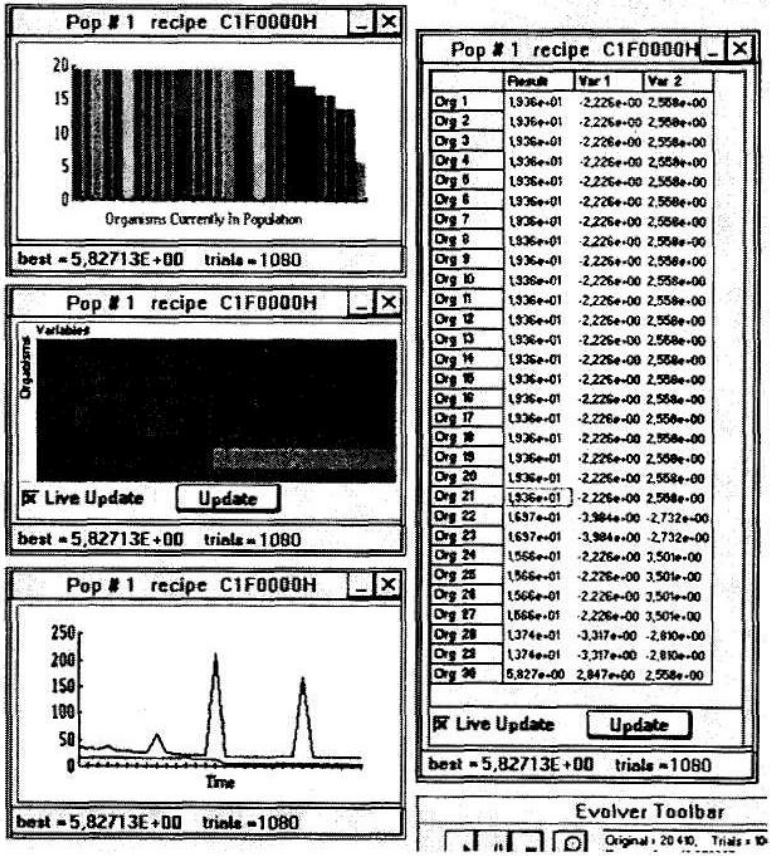



Рис. 3.123. Популяція особей і графік функції пристосованості при *t*=1080 для приклада 3.28.

«Найменше» значення функції пристосованості тут складає 5,83 для *t*=1080, тобто після 1080 «тактів». Це значення усе ще набагато більше мінімального. Графіки в лівій нижній частині рисунка показують зміну середнього (верхня крива) і «найкращого» (нижня крива) значення функції пристосованості.
На рис. 3.124 представлені ті ж графіки після 8000 «тактів» (*t*=8000). Помітно, що як «найкраще», так і середнє значення функції пристосованості в популяції приймають значення, близькі

до 0. На всіх графіках один розподіл на тимчасовій осі відповідає 20 «тактам».

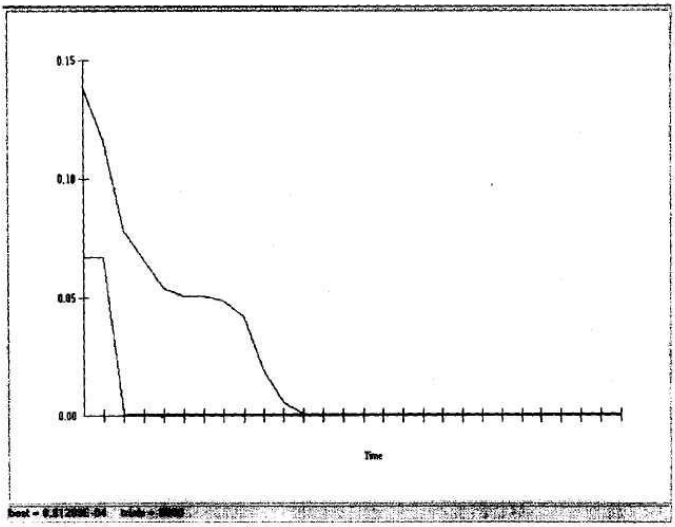



Рис. 3.124**.** Графік функції пристосованості при *t* = 8000 для приклада 3.28.

На рис. 3.125 зображена стовпчаста діаграма особей популяції для *t*=8000. Помітно, що всі хромосоми в популяції стали однаковими.

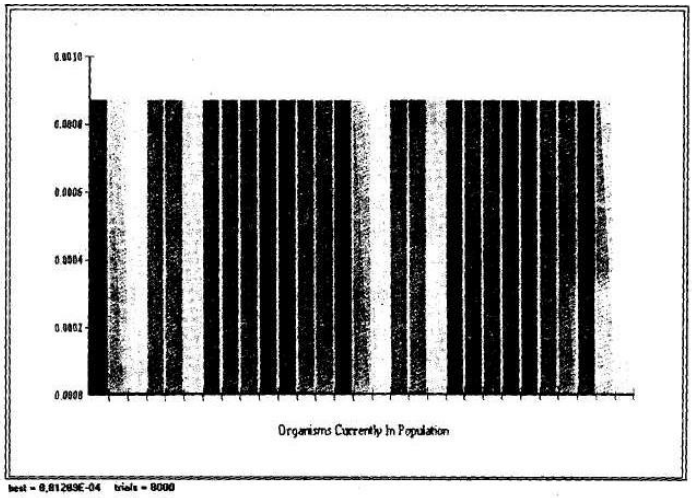



Рис. 3.125. Стовпчаста діаграма значень функції пристосованості особей у популяції при $t$ = 8000 для приклада 3.28.

Значення їхньої функції пристосованості, мабуть, такі ж, як і для «найкращого» рішення, тобто дорівнює 0,000881.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | |
| 2 | | | ФУНКЦИЯ HIMMELBLAU | | | | | Вычисление минимума | | | |
| 3 | | | | | | | | | | | |
| 4 | | | F(X1,X2) = (X1^2 +X2 -11)^2 + (X1 +X2^2 -7)^2 | | | | | -10 <= X1,X2 <= 10 | | | |
| 5 | | | | | | | | | | | |
| 6 | | | | | | | | | | | |
| 7 | X1 = | -3,78 | | | | | | | | | |
| 8 | X2 = | -3,279 | | | | | | | | | |
| 9 | F(X1,X2)= | 0,000881 | | | | | | | | | |

Рис. 3.126. «Найкраще» рішення для приклада 3.28 (отримане при $t$ = 8000).

«Найкраще» рішення показано на рис. 3.126. Це хромосома зі значеннями змінних $x_1$ = -3,78 і $x_2$ = -3,279.

## 3.19.2. Рішення комбінаторних задач за допомогою програми Evolver

При рішенні комбінаторних задач проблема полягає в пошуку найкращого рішення серед можливих перестановок параметрів задачі. Як приклад можна назвати сортування списку імен (приклад 3.29) або задачу комівояжера. У програмі **Evolver** для рішення комбінаторних задач застосовуються генетичні оператори, визначення яких трохи відрізняється від аналогічних операторів, орієнтованих на оптимізіційні задачі. Зокрема:

**Схрещування** розбивається на наступні кроки:

1) випадковим чином вибираються позиції в першого батька; їхня кількість залежить від показника схрещування;

2) знаходяться позиції з такими ж значеннями генів (аллелями) у другого батька;

3) значення позицій першого батька, що залишилися, копіюються на позиції другого батька, що залишилися, у послідовності, у якій вони записані в першого батька.

Описаний спосіб схрещування ілюструється на рис. 3.127.

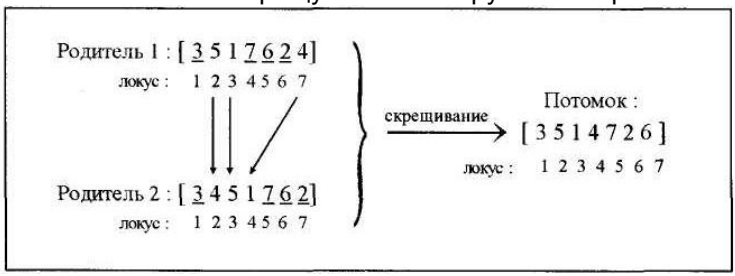



Рис. 3.127. Схрещування зі збереженням порядку в генетичному алгоритмі програми **Evolver**.

На цьому рисунку показано дві хромосоми батьків, що складаються із семи генів зі значеннями з інтервалу цілих чисел від 1 до 7. Кожен ген у хромосомі характеризується унікальним значенням. Кожна хромосома являє собою перестановку натуральних чисел від 1 до 7. Під кожним геном зазначений номер його позиції (locus). Припустимо, що показник схрещування дорівнює 0,5, і в першого батька випадковим чином обрано позиції 1, 4, 5, 6, на яких знаходяться значення 3, 7, 6, 2 відповідно. У другого батька ці значення знаходяться на позиціях 1, 5, 6, 7. У результаті копіювання значень позицій, що залишилися, першого батька (тобто чисел 5, 1, 4 з позицій 2, 3, 7 відповідно) на позиції другого батька, що залишилися, (тобто на позиції 2, 3, 4) у послідовності, у якій вони записані в першого батька, утвориться нащадок зі значеннями генів 3, 5, 1,4,7,2,6.

Представлений метод схрещування застосовується в програмі **Evolver** для рішення задач, що зводяться до пошуку хромосом з найкращим упорядкуванням генів. Описуваний спосіб подібний упорядкованому схрещуванню *(order crossover)*, показаному на рис. 3.128.

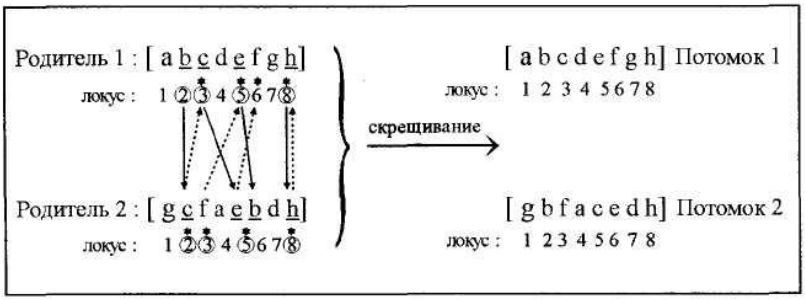



Рис. 3.128. Упорядковане схрещування *(order crossover).*

Номера позицій вибираються випадковим чином. Далі значення генів з обраних позицій одного з батьків переносяться на відповідні позиції другого батька. У результаті схрещування утворюються два нащадки.

**Мутація.** Оператор мутації реалізує так називану мутацію, засновану на упорядкуванні *(order-based mutation)*. Визначена в такий спосіб мутація полягає у випадковому виборі двох позицій у хромосомі й обміні значень генів на цих позиціях. Наприклад, після мутації хромосоми [3 5 1 4 7 2 6] на обраних позиціях 2 і 5 буде отримана хромосома [3714526]. Кількість обмінів зростає або знижується пропорційно збільшенню або зменшенню показника мутації.

Цей спосіб мутації відрізняється тим, що в результаті його виконання формується нова хромосома зі зміненою послідовністю генів. Така мутація застосовується для пошуку найкращої перестановки параметрів задачі.

**Приклад 3.29**

Цей приклад демонструє застосування генетичного алгоритму для сортування списку імен за абеткою.

Для спрощення будемо розглядати дуже короткий список із семи імен, що починаються буквами J, M, У, R, S, H, F. Таким чином, найкраще рішення шукається в просторі рішень, що складається з 7!=5040 можливих перестановок семи елементів. Найкраще рішення очевидне - це B, F, H, J, M, R, S. На цьому найпростішому прикладі познайомимося з тим, як генетичний алгоритм вирішує задачі цього типу. Припишемо кожному імені, включеному у вихідний (несортований) список, порядковий номер так, як це зроблено в першому стовпці на рис. 3.129.

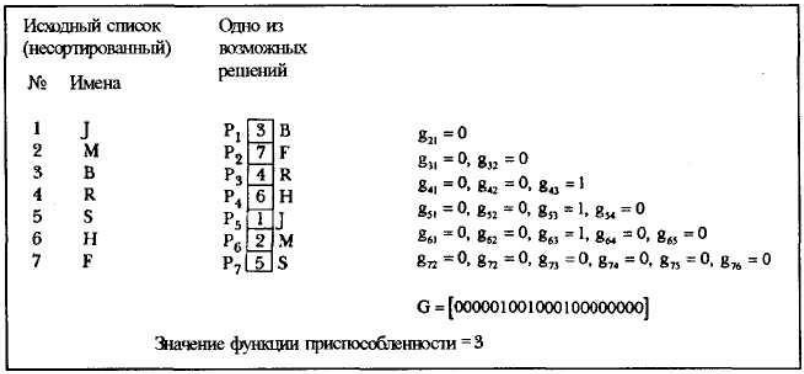



Рис. 3.129. Одне з припустимих рішень задачі з приклада 3.29.

Припустиме рішення, яке представлено на риснку - це B, F, R, H, J, M, S. Приведеним першим буквам імен передують відповідні їм порядкові номери з першого стовпця. Послідовність цих номерів (на риснку вони вписані в клітки) ідентифікує дане рішення. На рис. 3.130 у такий же спосіб представлено найкраще рішення, обумовлене послідовністю 3, 7, 6, 1,2,4,5.


Рис. 3.130. Найкраще рішення задачі з приклада 3.29.

Розглянута задача має сім змінних (параметрів задачі). Позначимо їх $P_1$, $P_2$, ..., $P_7$. Кожна з цих змінних може приймати цілі значення від 1 до 7. На рис. 3.129 показана одна з особей популяції, для якої значення конкретних змінних (параметрів

задачі) рівні 3, 7, 4, 6, 1, 2, 5, а найкраще рішення на рис. 3.130 характеризується значеннями цих змінних 3, 7, 6, 1, 2, 4, 5.

Приведені послідовності чисел розглядаються як аллелі хромосом, що представляють відповідні особи. Для кожної особи, що входить у популяцію, при виконанні генетичного алгоритму розраховується значення функції пристосованості і на цій основі вибираються найкращі і найгірші особи.

Перш ніж визначити функцію пристосованості для розглянутої задачі, уведемо ряд позначень. Нехай $n(P_i)$ відповідає імені, визначеному порядковим номером $P_i$, i=1,2.....7, і нехай $n(P_i)<n(P_j)$ означає, що ім'я з порядковим номером $P_i$ - повинне передувати імені з порядковим номером $P_j$, тобто вони упорядковуються за алфавітом. Нехай $G$ позначає послідовність, складену з елементів $g_{km}$, $k$ = 2.....7, $m$ = 1.....$k$-1, де

$$g_{km}=\begin{cases}1, & якщо\ n(P_k) < n(P_m)\\ 0 & в\ противному\ випадку\end{cases}$$

Очевидно, що якщо послідовність $G$ складається з одних нулів, то послідовність імен буде коректною. Тому найкраща особа повинна характеризуватися послідовністю $G$, всі елементи якої дорівнюють нулеві. Отже, функцію пристосованості можна визначити як суму елементів послідовності $G$, і нас, звичайно, буде цікавити мінімізація цієї функції (яка може приймати цілі значення в інтервалі від 0 до 21). Помітимо, що якби елемент $g_{km}$ приймав нульове значення при $n(P_k)<n(P_m)$ і значення 1 у противному випадку, то варто було б максимізувати кількість одиниць у послідовності $G$. Тепер перейдемо до інтерпретації функції пристосованості з приклада 3.4.

Задача, яка сформульована в прикладі 3.29, вирішується за допомогою програми **Evolver**, розмірність популяції прийнята рівною 10, показник схрещування дорівнює 0,5, показник мутації дорівнює 0. При $t$= 10, що відповідає першій популяції класичного генетичного алгоритму, отримано популяцію, яка представлена на рис. 3.131.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 15 | 3 | 5 | 1 | 4 | 7 | 2 | 6 |
| 2 | 13 | 5 | 3 | 1 | 4 | 2 | 7 | 6 |
| 3 | 13 | 2 | 6 | 1 | 5 | 4 | 3 | 7 |
| 4 | 11 | 3 | 4 | 5 | 1 | 7 | 2 | 6 |
| 5 | 11 | 7 | 2 | 6 | 5 | 4 | 1 | 3 |
| 6 | 11 | 1 | 7 | 7 | 4 | 5 | 6 | 3 |
| 7 | 11 | 7 | 4 | 1 | 5 | 6 | 3 | 2 |
| 8 | 9 | 6 | 4 | 1 | 3 | 2 | 7 | 5 |
| 9 | 7 | 3 | 5 | 1 | 7 | 6 | 2 | 4 |
| 10 | 5 | 3 | 6 | 1 | 4 | 2 | 7 | 5 |

Рис. 3.131. Популяція особей у генетичному алгоритмі програми **Evolver** для задачі з приклада 3.29.

Перший стовпець визначає послідовність хромосом у популяції (від «найгіршої» до «найкращої»). В другому приведені значення функції приналежності кожної хромосоми. У третьому стовпці записані самі хромосоми, що складаються із семи генів. Кожна з цих хромосом являє собою перестановку натуральних чисел від 1 до 7. Перше значення функції пристосованості, яке обкреслене прямокутником і рівне 15, відноситься до хромосоми, виключеної з попередньої популяції. Замість неї вводиться хромосома [3514726], яка отримана в результаті схрещування хромосом [3 5 1 7 6 2 4] і [3 4 5 1 7 2 6], що є присутнім на рис. 3.131.

Остання хромосома, що має функцію пристосованості, рівну п'яти, після 10 «тактів» є «найкращою на даний момент». При продовженні виконання алгоритму можна одержати найкращу хромосому [3 7 6 1 2 4 5] з функцією пристосованості, рівною 0. Ця оптимальна хромосома представлена на рис. 3.130. Помітимо, що в розглянутому прикладі функція пристосованості інтерпретується як похибка, яку, звичайно ж, варто мінімізувати. Для найкращого рішення ця похибка зобов'язана бути рівною 0.

Приклад 3.29 легко узагальнити на будь-які переліки імен або назв, що підлягають сортуванню за абеткою (або відповідно до іншого ключа). Задачі такого типу можна вирішувати за

допомогою еволюційного алгоритму програми **Evolver.** Ця програма також дозволяє вирішити відому з літератури задачу про комівояжера, що має набагато більш складну структуру, чим розглянута задача з приклада 3.29.

## 3.20. Еволюційні алгоритми в нейронних мережах

Об'єднання генетичних алгоритмів і нейронних мереж відомо в літературі під абревіатурою COGANN *(Combinations of Genetic Algorithms and Neural Networks).* Це об'єднання може бути допоміжним *(supportive)* або рівноправним *(collaborative)*. Допоміжне об'єднання двох методів означає, що вони *застосовуються послідовно один за одним*, причому один з них служить для підготовки даних, використовуваних при реалізації другого методу. При рівноправному об'єднанні обидва методи застосовуються одночасно. Класифікація цих типів об'єднань генетичних алгоритмів і нейронних мереж представлена в табл. 3.6.

У наступній частині цього розділу будуть обговорюватися конкретні комбінації, які відбиті в таблиці, що приводиться. Малюнки 3.132 - 3.134 ілюструють різні підходи до рішення задач, які розглянуто як допоміжні об'єднання генетичних алгоритмів і нейронних мереж.

Необхідно відзначити, що відповідно до зауважень, приведених в п. 3.18, термін «генетичні алгоритми» застосовується тут у більш широкому змісті, чим класичний генетичний алгоритм.

**Таблиця 3.6.** Об'єднання генетичних алгоритмів і нейронних мереж

| Вид объединения | Характеристика объединения | Примеры использования |
|---|---|---|
| | Генетические алгоритмы и нейронные сети независимо применяются для решения одной и той же задачи | Однонаправленные нейронные сети, сети Кохонена с самоорганизацией и генетические алгоритмы в задачах классификации |
| Вспомогательное | Нейронные сети для обеспечения генетических алгоритмов | Формирование исходной популяции для генетического алгоритма |
| | Генетические алгоритмы для обеспечения нейронных сетей | Анализ нейронных сетей |
| | | Подбор параметров либо преобразование пространства параметров |
| | | Подбор параметров либо правила обучения (эволюция правил обучения) |
| Равноправное | Генетические алгоритмы для обучения нейронных сетей | Эволюционное обучение сети (эволюция весов связей) |
| | Генетические алгоритмы для выбора топологии нейронной сети | Эволюционный подбор топологии сети (эволюция сетевой архитектуры) |
| | Системы, объединяющие адаптивные стратегии генетических алгоритмов и нейронных сетей | Нейронные сети для решения оптимизационных задач с применением генетического алгоритма для подбора весов сети |
| | | Реализация генетического алгоритма с помощью нейронной сети |
| | | Применение нейронной сети для реализации оператора скрещивания в генетическом алгоритме |

### 3.20.1. Незалежне застосування генетичних алгоритмів і нейронних мереж

Генетичні алгоритми і нейронні мережі можуть незалежно застосовуватися для рішення однієї і тієї ж задачі. Цей підхід ілюструється на рис. 3.132.


Рис. 3.132. Генетичний алгоритм і нейронна мережа незалежно застосовуються для рішення однієї і тієї ж задачі.

Наприклад, описані незалежні застосування нейронних мереж, генетичних алгоритмів і алгоритму KNN «найближчий сусід» *(K - means nearest neighbour)* для рішення задач класифікації. У літературі приводяться порівняння тришарової односпрямованої нейронної мережі з навчанням по методу зворотного поширення похибки (навчання з учителем), мережі Кохонена із самоорганізацією (навчання без учителя), системи класифікації, заснованої на генетичному алгоритмі, а також алгоритму KNN «найближчий сусід». Автори ряду робіт вважають незалежне застосування цих методів для рішення задачі автоматичної класифікації результатів ЭМГ (электроміографія - реєстрація електричної активності м'язів) допоміжним об'єднанням. Відомі й інші роботи, у яких порівнюються можливості застосування різних методів (зокрема, генетичних алгоритмів і нейронних мереж) для рішення тих самих задач. Прикладом задачі, яку можна вирішити за допомогою як

нейронної мережі, так і генетичного алгоритму, може служити задача про комівояжера.

### 3.20.2. Нейронні мережі для підтримки генетичних алгоритмів

Більшість дослідників вивчали можливості застосування генетичних алгоритмів для забезпечення роботи нейронних мереж. До нечисленних зворотних випадків відноситься гібридна система, призначена для рішення задачі трасування, що класифікується як приклад допоміжного об'єднання нейронних мереж і генетичних алгоритмів. У цій системі генетичний алгоритм використовується в якості оптимізаційної процедури, призначеної для знаходження найкоротшого шляху. Нейронна мережа застосовується при формуванні вихідної популяції для генетичного алгоритму. Цей підхід схематичиески ілюструється на рис. 3.133.

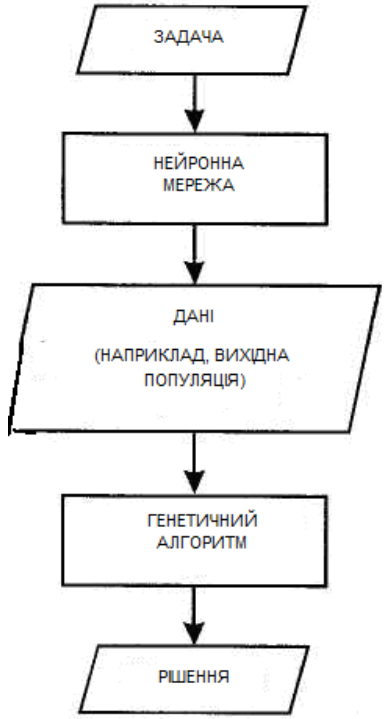



Рис. 3.133. Допоміжне об'єднання нейронної мережі з генетичним

алгоритмом.

### 3.20.3. Генетичні алгоритми для підтримки нейронних мереж

Підхід, заснований на використанні генетичного алгоритму для забезпечення роботи нейронної мережі, схематично представлено на рис. 3.134.


Рис. 3.134. Допоміжне об'єднання генетичного алгоритму з нейроНною мережею.

Відомо багато робіт, присвячених подібному об'єднанню розглядаємих методів. Можна виділити три області проблем:
- застосування генетичного алгоритму для підбору параметрів або перетворення простору параметрів, використовуваних нейронною мережею для класифікації;
- застосування генетичного алгоритму для підбору правила навчання або параметрів, керуючих навчанням нейронною мережою;
- застосування генетичного алгоритму для аналізу нейронної мережі.

Дві перші області доданка генетичних алгоритмів у нейронних мережах, узагалі говорячи, дозволяють поліпшувати функціонування мереж (тобто вирішують проблему синтезу), тоді як третя служить для аналізу їхнього функціонування. Почнемо обговорення з останньої позиції.

**Аналіз нейронних мереж.** Деякі дослідники застосовували генетичні алгоритми як допоміжний інструмент для з'ясування закономірностей функціонування нейронних мереж або аналізу ефективності їхньої роботи. Генетичний алгоритм використовувався для побудови «інструментальної системи», що полегшує розуміння функціонування мережі - попросту говорячи, для з'ясування, що і чому робить мережа. Таке розуміння необхідне для того, щоб нейросітьовий класифікатор не сприймався в якості «чорної шухляди», що формує відповідь якимось таємничим чином, і щоб рішення по класифікації об'єктів були пояснюванними. Подібний «інструментарій» *(explanation facilities)* використовується в більшості експертних систем. Побудова цих інструментів для їхнього застосування в нейронних мережах вважається більш масштабною проблемою, що відноситься до аналізу мереж. Генетичний алгоритм застосовувався для побудови так званих кодових векторів *(codebook vectors),* що представляють собою вхідні сигнали, при яких функція активації конкретного вихідного нейрона мережі приймає максимальне або близьке до нього значення. Вхідні вектори представлялися в хромосомах множиною дійсних чисел від 0,0 до 1,0. Аналізувалася нейронна мережа, яка призначена для рішення задачі класифікації. Аналогічний підхід застосовувався для мережі ART1 (часткового випадку ART із двоїчними вхідними сигналами). За допомогою генетичного алгоритму також проводився аналіз нейронної мережі,

використовуваної як модель асоціативного запам'ятовуючого пристрою. Приведені приклади характеризують допоміжне об'єднання генетичних алгоритмів і нейронних мереж, хоча і не можуть вважатися типовими стосовно схеми, представленої на рис. 3.134.

**Підбір параметрів або перетворення простору параметрів.** Генетичний алгоритм використовується при підготовці даних для нейронної мережі, яка грає роль класифікатора. Ця підготовка може виконуватися шляхом перетворення простору параметрів або виділенням деякого підпростору, що містить необхідні параметри.

Перший з цих методів, так називане перетворення простору параметрів, застосовується найчастіше в алгоритмах типу «найближчий сусід», хоча відомі також його доданки в нейромережних класифікаторах. Другий підхід полягає у виділенні підмножини параметрів, що враховуються. Виявляється, що обмеження множини параметрів часто поліпшує функціонування нейронної мережі як класифікатора і, до того ж, скорочує обсяги обчислень. Подібне обмеження множини що враховуються нейронною мережею параметрів застосовувалося, зокрема, для контролю сценаріїв подій на ядерних об'єктах, а також для розпізнавання китайських ієрогліфів. Відомі й інші приклади підготовки даних для нейронних мереж за допомогою генетичних алгоритмів.

**Підбір параметрів і правил навчання.** Генетичний алгоритм також застосовується для підбора параметрів навчання - найчастіше швидкості навчання (learning rate) і так називаного моменту для алгоритму зворотного поширення похибки. Таке адаптивне уточнення параметрів алгоритму зворотного поширення (вони кодуються в хромосомах) у результаті еволюції може розглядатися як перша спроба еволюційної модифікації правил навчання. Замість безпосереднього застосування генетичного алгоритму для підбору параметрів навчання розвивається еволюційний підхід, спрямований на побудову оптимального правила (алгоритму) навчання.

Еволюція правил навчання буде представлена в п. 3.20.7.3.

Помітимо, що еволюційна концепція вже може розглядатися як перехід від допоміжного до рівноправного об'єднання генетичного алгоритму і нейронних мереж.

## 3.20.4. Застосування генетичних алгоритмів для навчання нейронних мереж

Думка про те, що нейронні мережі можуть навчатися за допомогою генетичного алгоритму, висловлювалася різними дослідниками. Перші роботи на цю тему стосувалися застосування генетичного алгоритму як метод навчання невеликих односпрямованих нейронних мереж, але в наступному було реалізовано застосування цього алгоритму для мереж з більшою розмірністю.

Як правило, задача полягає в оптимізації ваг нейронної мережі, що має апріорі задану топологію. Ваги кодуються у виді двоїчних послідовностей (хромосом). Кожна особа популяції характеризується повною множиною ваг нейронної мережі. Оцінка пристосованості особей визначається функцією пристосованості, що задається у виді суми квадратів похибок, тобто різностей між очікуваними (еталонними) і фактично одержуваними значеннями на виході мережі для різних вхідних даних.

Приведемо два найважливіших аргументи на користь застосування генетичних алгоритмів для оптимізації ваг нейронної мережі. Насамперед, генетичні алгоритми забезпечують глобальний перегляд простору ваг і дозволяють уникати локальні мінімуми. Крім того, вони можуть використовуватися в задачах, для яких інформацію про градієнти одержати дуже складно або вона виявляється занадто дорогою.

Еволюційному навчанню нейронних мереж, тобто еволюції ваг зв'язків, присвячено п. 3.20.7.1.

## 3.20.5. Генетичні алгоритми для вибору топології нейронних мереж

Як найбільш очевидний спосіб об'єднання генетичного алгоритму з нейронною мережею інтуїтивно сприймається спроба закодувати в генотипі топологію об'єкта (у розглянутому випадку - нейронної мережі) із указівкою кількості нейронів і зв'язків між ними при наступному визначенні ваг мережі за допомогою будь-якого відомого методу.

Проектування оптимальної топології нейронної мережі може бути представлене у виді пошуку такої архітектури, яка забезпечує найкраще (щодо обраного критерію) рішення конкретної задачі. Такий підхід припускає перебір простору

архитектур, складеного з усіх можливих варіантів, і вибір точки цього простору, найкращої щодо заданого критерію оптимальності.

З урахуванням достоїнств еволюційного проектування архітектури в останні роки була виконана велика кількість досліджень, у яких основна увага приділялася еволюції з'єднань нейронної мережі, тобто кількості нейронів і топології зв'язків між ними. Лише в деяких роботах розглядалася еволюція функцій переходів, хоча ці функції вважаються важливим елементом архітектури і впливають на функціонування нейронної мережі.

Також, як і у випадку еволюційного навчання, перший крок еволюційного проектування архітектури полягає у формуванні вихідної множини розглянутих варіантів. Типовий цикл еволюції архітектур представлено у п. 3.20.7.2.

## 3.20.6. Адаптивні взаємодіючі системи

До рівноправного об'єднання генетичних алгоритмів і нейронних мереж варто віднести комбінацію адаптивних стратегій обох методів, що складає єдину адаптивну систему. Можна привести три приклади систем такого типу.

*Перший* з них - це нейронна мережа для оптимізаційної задачі з генетичним алгоритмом для визначення ваг мережі.

*Другий* приклад відноситься до реалізації генетичного алгоритму за допомогою нейронної мережі. У цьому випадку нейронні підсистеми застосовуються для виконання генетичних операцій репродукції і схрещування.

У *третьому* прикладі, трохи схожому на попередній, нейронна мережа також застосовується як оператор схрещування в генетичному алгоритмі, призначеному для рішення оптимізаційних задач.

Представлені приклади стосуються такого рівноправного об'єднання генетичних алгоритмів і нейронних мереж, що у результаті дозволяє одержати більш ефективний алгоритм, який поєднує кращі якості обох методів.

## 3.20.7. Типовий цикл еволюції

Як тільки визначений вид еволюції вводиться в штучну нейронну мережу, відразу виникає потреба у відповідній йому схемі хромосомного представлення даних, тобто повинний бути створений спосіб генетичного кодування особей популяції. Розробка способу кодування вважається першим етапом такого

еволюційного підходу, поряд з яким типовий процес еволюції включає наступні кроки:

- декодування;
- навчання;
- оцінювання пристосованості;
- репродукція;
- формування нового покоління.

Приведена на рис. 3.12 блок-схема зберігає свою актуальність, оскільки (як уже згадувалося в п. 3.18) вона відображає і класичний генетичний алгоритм, і так називані еволюційні програми, що засновані на генетичному підході й узагальнюють його принципи. Отже, цій універсальній блок-схемі відповідають різні еволюційні алгоритми, і в кожнім з них у першу чергу повинна бути згенерована вихідна популяція хромосом. За аналогією з класичним генетичним алгоритмом ініціалізація (тобто формування цієї вихідної популяції) полягає у випадковому виборі необхідної кількості хромосом, які включаються в неї, що припускає відповідне генетичне кодування кожної особи. У класичному генетичному алгоритмі хромосоми представляються тільки двоїчними послідовностями. При еволюційному підході вибір способу кодування являє собою важливу й актуальну задачу.

Далі відповідно до типового циклу еволюції випливає необхідність декодування кожної особи (хромосому) вихідної або поточної популяції для того, щоб одержати множину рішень (фенотипів) даної задачі. У випадку еволюції ваг, архітектур і/або правил навчання фенотипи представляють відповідно множини ваг, архітектур і правил навчання.

Згодом відповідно до генетичного алгоритму розраховуються значення функції пристосованості особей вихідної (або поточної) популяції. При нейромережному підході після декодування хромосом виходить множина нейронних мереж, для яких степінь пристосованості визначається за результатами навчання цих мереж.

При реалізації типового циклу еволюції необхідно сконструювати множину відповідних нейронних мереж (фенотипів):

- мережі з фіксованою архітектурою і множиною закодованих хромосомами ваг - у випадку *еволюції ваг*;

- мережі з закодованою хромосомами архітектурою - у випадку *еволюції архітектури*;
- мережі з випадково згенерованими архитектурами і початковими вагами - у випадку *еволюції правил навчання*.

Після навчання оцінюється пристосованість кожної особи, що входить у поточну популяцію. Помітимо, що також як і в прикладі максимізації функції (приклад 3.5), для оцінювання пристосованості хромосом необхідно їх спочатку декодувати і лише потім розрахувати значення функції пристосованості особей по їхніх фенотипах.

Наступний крок генетичного алгоритму - це *селекція хромосом*. Вибираються хромосоми, що підлягають репродукції, тобто формується батьківський пул, особи якого в результаті застосування генетичних операторів сформують популяцію нащадків. Селекція може бути заснована на методі рулетки або будь-якому іншому, наприклад, по алгоритму Уітлі (Whitley). Згідно з цими методами селекція виконується з ймовірністю, пропорційною пристосованості хромосом, або відповідно до їх рангу (при використанні рангового методу). Під репродукцією в даному випадку розуміється процес відбору (селекції) і копіювання (розмноження) хромосом для формування з них перехідної популяції (батьківського пула), особи якої будуть піддаватися впливові генетичних операторів схрещування, мутації і, можливо, інверсії.

Застосування генетичних операторів за обраним методом селекції хромосом відбувається аналогічно класичному генетичному алгоритмові, причому ці оператори можуть відрізнятися від схрещування і мутації базового алгоритму. Як відзначалося в п. 3.18, для конкретної задачі генетичні оператори можуть визначатися в індивідуальному порядку.

Також як і в класичному генетичному алгоритмі, у результаті застосування генетичних операторів з обраним методом селекції хромосом формується нова популяція особей (нащадків). Наступні кроки алгоритму повторюються для чергової популяції аж до виконання умови завершення генетичного алгоритму. На кожній ітерації формується нове покоління нащадків.

Найкраща особа з останнього покоління вважається шуканим рішенням даної задачі. У такий спосіб виходить найкраща множина ваг, найкраща архітектура або найкраще правило навчання.

### 3.20.7.1. Еволюція ваг зв'язків

Еволюційний підхід до навчання нейронних мереж складається з двох основних етапів. Як указувалося у введенні до п. 3.20.7, *перший* з них - це вибір відповідної схеми представлення ваг зв'язків. Він полягає в ухваленні рішення - чи можна кодувати ці ваги двоїчними послідовностями, чи потрібна якась інша форма. На *другому* етапі вже здійснюється сам процес еволюції, заснований на генетичному алгоритмі.

Після вибору схеми хромосомного представлення генетичний алгоритм застосовується до популяції особей (хромосом, що містять закодовану множину ваг нейронної мережі) з реалізацією типового циклу еволюції, що складається з чотирьох кроків:

1) Декодування кожної особи (хромосоми) поточного покоління для відновлення множини ваг і конструювання відповідній цій множині нейронної мережі з апріорно заданою архітектурою і правилом навчання.

2) Розрахунок загальної середньоквадратичної похибки між фактичними і заданими значеннями на усіх виходах мережі при подачі на її входи навчальних образів. Ця похибка визначає пристосованість особи (сконструйованої мережі); у залежності від виду мережі функція пристосованості може бути задана й іншим чином.

3) Репродукція особей з ймовірністю, що відповідає їхній пристосованості, або відповідно до їх рангу (у залежності від способу селекції - наприклад, по методу рулетки або ранговому методу).

4) Застосування генетичних операторів - таких як схрещування, мутація і/або інверсія для одержання нового покоління.

Блок-схема, ілюструюча еволюцію ваг, представлена на рис. 3.135.

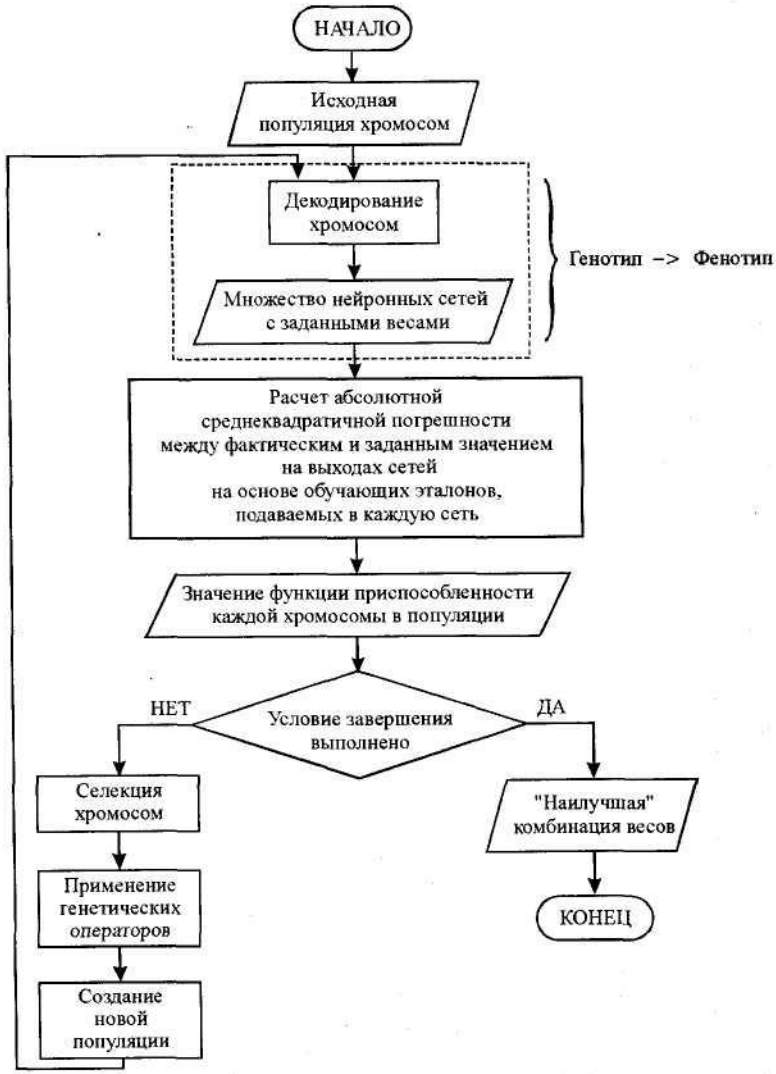



Рис. 3.135. Блок-схема генетичного алгоритму пошуку найкращого набору ваг нейронної мережі (випадок еволюції ваг).

Відповідно до цієї схеми були розраховані ваги для нейронної мережі, що реалізує систему XOR за допомогою програми **Evolver** (приклади 3.23 - 3.27) і за допомогою програми **FlexTool** (приклад 3.20). Відповідно до першого етапу типового циклу еволюції апріорно задаються і залишаються незмінними архітектура мережі, що визначає кількість шарів, число нейронів у кожнім шарі і топологію міжнейронних зв'язків, а також правило навчання мережі. Пристосованість кожної особи (генотипу) оцінюється значенням середньоквадратичної похибки, розрахованої по відповідно цій особи нейронній мережі (фенотипові).

У представленому процесі еволюційного навчання реалізується режим так називаного пакетного навчання *(batch training mode),* при якому значення ваг змінюються тільки після пред'явлення мережі всіх навчальних образів. Такий прийом відрізняється від застосовуваного в більшості послідовних алгоритмів навчання - наприклад, у методі зворотного поширення похибки ваги уточнюються після пред'явлення мережі кожної навчальної вибірки.

Розглянемо більш докладно перший етап еволюційного підходу до навчання, зв'язаний з фіксацією схеми представлення ваг. Як уже відзначалося, необхідно вибрати між бінарним представленням і кодуванням ваг дійсними числами. Крім традиційного двоїчного коду, може застосовуватися код Грея, логарифмічне кодування або інші більш складні форми запису даних. У ролі обмежувача виступає необхідна точність представлення значень ваг. Якщо для запису кожної ваги використовується занадто мало бітів, то навчання може продовжуватися занадто довго і не принести ніякого ефекту, оскільки точність апроксимації окремих комбінацій дійсних значень ваг дискретними значеннями часто виявляється недостатньою. З іншого боку, якщо використовується занадто багато бітів, то двоїчні послідовності, які представляють нейронні мережі великої розмірності, виявляються дуже довгими, що сильно подовжує процес еволюції і робить еволюційний підхід до навчання нераціональним із практичної точки зору. Питання оптимізації кількості бітів для представлення конкретних ваг усе ще залишається відкритим.

Для усунення недоліків схеми двоїчного представлення даних було запропоновано задавати значення ваг дійсними

числами, точніше - кожну вагу описувати окремим дійсним числом. Такий спосіб кодування реалізований, зокрема, у програмі **Evolver**. Він використовувався при рішенні прикладів 3.23 - 3.27.

Стандартні генетичні оператори, розроблені для схеми двоїчного представлення даних, можуть застосовуватися й у випадку завдання ваг двоїчними числами, однак для більшої ефективності еволюційного алгоритму і прискорення його виконання створені спеціальні генетичні оператори.

### 3.20.7.2. Еволюція архітектури мережі

У п. 3.20.7.1 при розгляді еволюційного навчання нейронних мереж передбачалося, що архітектура мережі задається апріорно і не змінюється в процесі еволюції ваг. Однак зберігає актуальність питання - як вибрати архітектуру мережі? Відомо, що архітектура впливає на весь процес обробки інформації нейронної мережею. На жаль, найчастіше вона підбирається експертами методом проб і помилок. У таких умовах спосіб оптимального (або майже оптимального) проектування архітектури нейронної мережі для конкретної задачі виявився б дуже корисним. Один з можливих підходів полягає в еволюційному формуванні архітектури з застосуванням генетичного алгоритму.

Також як і у випадку еволюційного навчання, на першому етапі еволюційного проектування архітектури приймається рішення щодо відповідної форми її опису. Однак у даній ситуації проблема не зв'язана з вибором між двоїчним і дійсним представленням (тобто дійсними числами), оскільки мова може йти тільки про дискретні значення. Необхідно вибрати більш загальну концептуальну структуру представлення даних, наприклад, у формі *матриць, графів* і т.п. Ключове питання складається в ухваленні рішення про кількість інформації про архітектуру мережі, що повинна кодуватися відповідною схемою. З одного боку, повна інформація про архітектуру може безпосередньо кодуватися у виді двоїчних послідовностей, тобто кожен зв'язок і кожен вузол (нейрон) прямо специфікується визначеною кількістю бітів. Такий спосіб представлення називається *схемою безпосереднього кодування.* З іншого боку, можуть представлятися тільки найважливіші параметри або властивості архітектури – такі, як кількість вузлів (нейронів), кількість зв'язків і вид перехідної функції нейрона. Цей спосіб

представлення називається *схемою непрямого кодування.* Існують і інші назви зазначених способів представлення даних, наприклад, замість «безпосереднього кодування» зустрічається термін *сильна схема специфікації,* а замість «непрямого кодування» - *слабка схема класифікації*.

**Схема безпосереднього кодування** означає, що кожен зв'язок нейронної мережі безпосередньо задається його двоїчним представленням. Матриця *C* розмірністю $n \times n$, $C = [c_{ij}]_{n \times n}$ може представляти зв'язки нейронної мережі, що складається з *n* вузлів (нейронів), причому значення $c_{ij}$ визначає наявність або відсутність зв'язку між *i-м* і *j-м* нейронами. Якщо $c_{ij} = 1$, то зв'язок існує, якщо $c_{ij} = 0$, то зв'язок відсутній. При такому підході двоїчна послідовність (хромосома), що представляє зв'язки нейронної мережі, має вигляд комбінації рядків (або стовпців) матриці *C.* Приклад розглянутого способу кодування для $n = 5$ приведено на рис. 3.136.

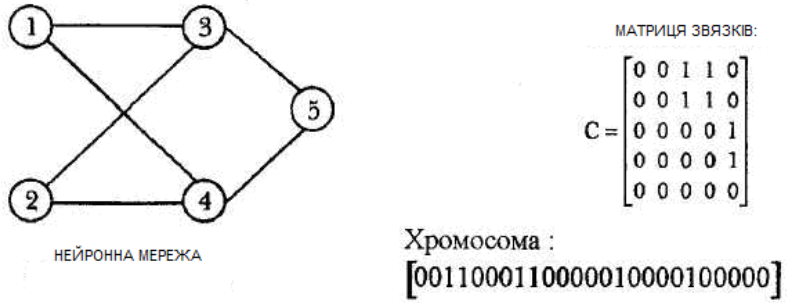



Рис. 3.136. Приклад безпосереднього кодування матриці зв'язків для нейронної мережі.

Якщо значення *n* позначає кількість нейронів у мережі, то зв'язки між цими нейронами будуть представлятися двоїчною послідовністю, що має довжину $n^2$. Очевидний недолік такого способу кодування полягає в стрімкому збільшенні довжини генотипу при розширенні нейронної мережі. Однак в обговорювану схему представлення можна легко внести обмеження, які скоротять довжину хромосом. Зокрема, можуть прийматися до уваги тільки односпрямовані зв'язки, що дозволить враховувати тільки ті елементи матриці *C,* що задають зв'язки даного вузла (нейрона) з наступним вузлом. У цьому

випадку хромосома для приклада на рис. 3.136 буде мати вигляд 0110110011.

Схема безпосереднього кодування може одночасно застосовуватися для визначення як зв'язків, так і їхніх ваг. Цей спосіб кодування застосовується, головним чином, для невеликих нейронних мереж.

**Схема непрямого кодування** - це спосіб скорочення довжини опису зв'язків, що полягає в кодуванні тільки найбільш важливих властивостей, але не кожного зв'язку нейронної мережі. З цієї причини помітним достоїнством схеми непрямого кодування стає компактне представлення зв'язків. Така схема виглядає більш обґрунтованою з біологічної точки зору. Відповідно до сучасної нейрології, неможливо прямо і незалежно описати закодованою в хромосомах генетичною інформацією всю нервову систему. Такий висновок випливає, наприклад, з факту, що генотип людини складається з набагато меншої кількості генів, чим число нейронів у його мозку.

Відомі різні методи непрямого кодування.

Другий етап еволюційного проектування архітектури нейронної мережі складається (відповідно до типового циклу еволюції) з наступних кроків:

1) Декодування кожної особи поточної популяції для опису архітектури нейронної мережі.

2) Навчання кожної нейронної мережі з архітектурою, отриманою на першому кроці, за допомогою заздалегідь заданого правила (деякі його параметри можуть адаптивно уточнюватися в процесі навчання). Навчання повинне починатися при різних випадково обираних початкових значеннях ваг і (при необхідності) параметрів правила навчання.

3) Оцінювання пристосованості кожної особи (закодованої архітектури) за досягнутими результатами навчання, тобто по найменшій цілій середньоквадратичній похибці навчання або на основі тестування, якщо найбільший інтерес викликає здатність до узагальнення, найменша тривалість навчання або спрощення архітектури (наприклад, мінімізація кількості нейронів і зв'язків між ними).

4) Репродукція особей з ймовірністю, що відповідає їхній пристосованості або рангові в залежності від використовуваного методу селекції.

5) Формування нового покоління в результаті застосування таких генетичних операторів, як схрещування, мутація і/або інверсія.

Блок-схема, що ілюструє еволюцію архитектур, представлена на рис. 3.137.

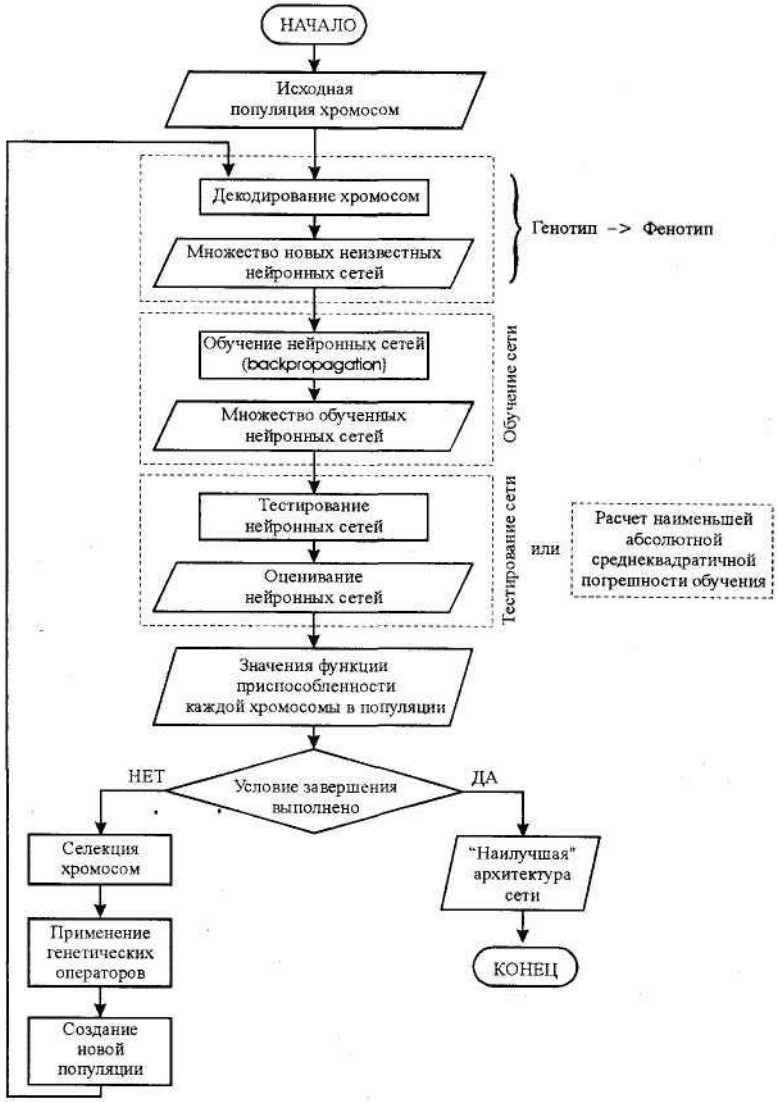



Рис. 3.137. Блок-схема генетичного алгоритму для пошуку найкращої архітектури нейронної мережі (випадок еволюції архитектур).

Якщо говорити про навчання мережі (крок 2), то найбільше часто зустрічається розвиток топології односпрямованих мереж із застосуванням алгоритму зворотного поширення похибки з метою локального навчання. Відомі роботи, у яких описується застосування генетичного алгоритму для одночасної адаптації і ваг і топології. В інших дослідженнях допускалися з'єднання в межах одного шару, зворотні зв'язки, а також навчання на основі конкуренції *(competetive learning)* і по Хеббу.

Достоїнством еволюційного підходу вважається той факт, що функцію пристосованості можна легко визначити спеціально для еволюції мережі зі строго визначеними властивостями. Наприклад, якщо для оцінювання пристосованості використовувати результати тестування замість результатів навчання, то буде отримана мережа з кращою здатністю до узагальнення.

### 3.20.7.3. Еволюція правил навчання

Відомо, що для різних архітектур і задач навчання потрібні різні алгоритми навчання. Пошук оптимального (або майже оптимального) правила навчання, як правило, відбувається з урахуванням експертних знань і часто - методом проб і помилок. Тому досить перспективним вважається розвиток *автоматичних методів оптимізації правил навчання нейронних мереж*. Розвиток людських здібностей до навчання від відносно слабких до досить сильних свідчить про потенційну можливість застосування еволюційного підходу в процесі навчання штучних нейронних мереж.

Схема хромосомного представлення у випадку еволюції правил навчання повинна відбивати динамічні характеристики. Статичні параметри (такі як архітектура або значення ваг мережі) кодувати значно простіше. Спроба створення універсальної схеми представлення, що дозволила б описувати довільні види динамічних характеристик нейронної мережі, свідомо приречена на невдачу, оскільки припускає невиправдано великий обсяг обчислень, необхідних для перегляду всього простору правил навчання. З цієї причини на тип динамічних характеристик зазвичай накладаються певні обмеження, що дозволяє вибрати *загальну структуру правила навчання*. Найчастіше встановлюється, що для всіх зв'язків нейронної мережі повинно застосовуватися те саме правило навчання, що може бути задано функцією виду

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1, i_2 \cdots i_k = 1}^{n} [\theta_{i_1, i_2 \cdots i_k} \prod_{j=1}^{k} x_{i_j}(t-1)] \text{ , (*)}$$

де $t$ - час, $\Delta w$ - збільшення ваги, $x_{i_j}$ - так називані локальні змінні, $\theta_{i_1, i_2, \ldots, i_k}$ - дійсні коефіцієнти.

Головна мета еволюції правил навчання полягає в підборі відповідних значень коефіцієнтів $\theta_{i_1, i_2, \ldots, i_k}$.

З урахуванням великої кількості компонентів рівняння (*), що може зробити еволюцію занадто повільною і практично неефективною, часто вводяться додаткові обмеження, засновані на евристичних посилках.

Представимо типовий цикл еволюції правил навчання.

1) Декодування кожної особи поточної популяції для опису правила навчання, що буде використовуватися як алгоритм навчання нейронних мереж.

2) Формування множини нейронних мереж з випадково згенерованими архітектурами і початковими значеннями ваг, а також оцінювання цих мереж з урахуванням їх навчання за правилом, отриманому на кроці 1, у категоріях точності навчання або тестування, тривалості навчання, складності архітектури і т.п.

3) Розрахунок значення пристосованості кожної особи (закодованого правила навчання) на основі отриманої на кроці 2 оцінки кожної нейронної мережі, що являє собою своєрідний вид зваженого усереднення.

4) Репродукція особей з ймовірністю, що відповідає їхній пристосованості або рангові в залежності від використовуваного методу селекції.

5) Формування нового покоління в результаті застосування таких генетичних операторів, як схрещування, мутація і/або інверсія.

Блок-схема, що ілюструє еволюцію архітектур, представлена на рис. 3.138.

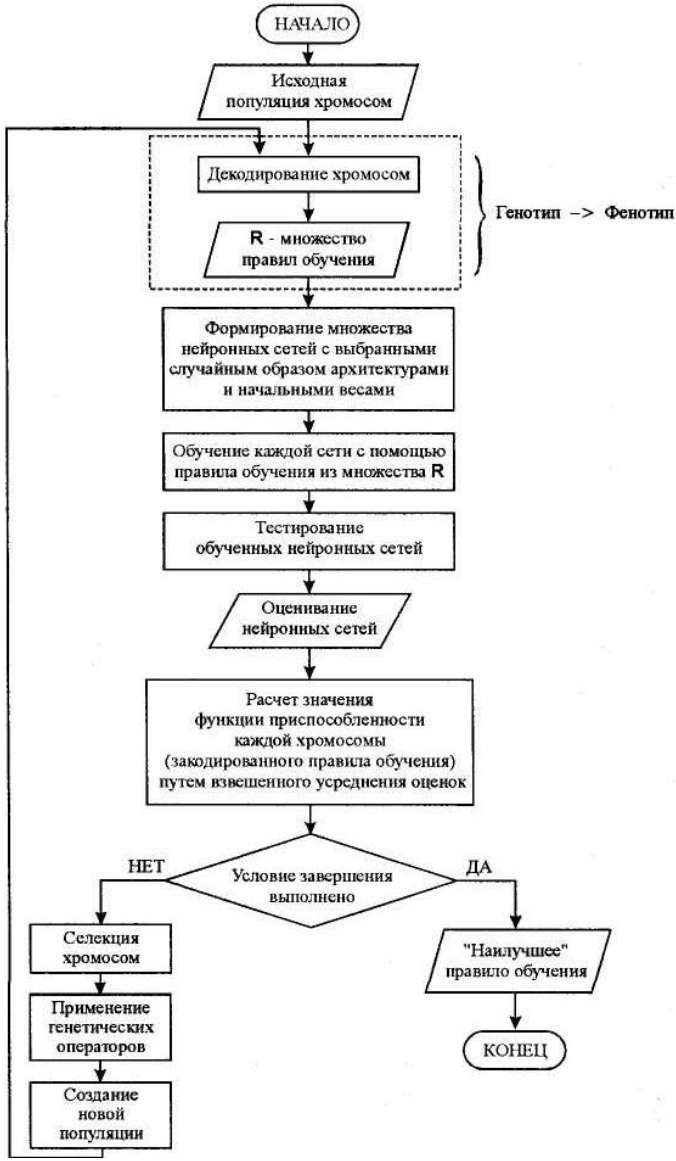

НАЧАЛО
Исходная популяция хромосом
Декодирование хромосом
R - множество правил обучения
Генотип –> Фенотип
Формирование множества нейронных сетей с выбранными случайным образом архитектурами и начальными весами
Обучение каждой сети с помощью правила обучения из множества R
Тестирование обученных нейронных сетей
Оценивание нейронных сетей
Расчет значения функции приспособленности каждой хромосомы (закодированного правила обучения) путем взвешенного усреднения оценок
Условие завершения выполнено
НЕТ
ДА
Селекция хромосом
Применение генетических операторов
Создание новой популяции
"Наилучшее" правило обучения
КОНЕЦ

Рис. 3.138. Блок-схема генетичного алгоритму для пошуку найкращого правила навчання (випадок еволюції правил навчання).

## 3.21. Приклади моделювання еволюційних алгоритмів у доданку до нейронних мереж

Представлені приклади 3.23 - 3.27, а також приклад 3.20 можна розглядати як ілюстрацію можливості застосування генетичного (зокрема, еволюційного) алгоритму для підбору ваг нейронної мережі. Цей алгоритм буде застосовуватися в перерахованих прикладах замість традиційного методу навчання, наприклад, замість алгоритму зворотного поширення помилки *(backpropagation).* Дуже часто застосовується так називаний гібридний підхід, що складається в об'єднанні обох методів. Як правило, спочатку за допомогою генетичного алгоритму знаходиться рішення, досить близьке до оптимального, і потім воно розглядається як відправна точка для традиційного пошуку оптимальної точки, наприклад, по методу зворотного поширення похибки.

### 3.21.1. Програми Evolver і BrainMaker

Метод зворотного поширення похибки застосовується для навчання нейронних мереж у програмі **BrainMaker**. У прикладі 3.30 з використанням цієї програми тестується нейрона мережа, що реалізує логічну систему XOR зі значеннями ваг, розрахованими в п. 3.19.1 за допомогою програми **Evolver.** При рішенні прикладів 3.31 і 3.32 демонструється гібридний підхід, тобто навчання цієї ж нейронної мережі програмою **BrainMaker,** але при початкових значеннях ваг, розрахованих генетичним алгоритмом програми **Evolver.** У свою чергу, приклад 3.33 ілюструє навчання нейронної мережі, що реалізує логічну систему XOR, при використанні тільки програми **BrainMaker.**

**Приклад 3.30**

Протестувати за допомогою програми **BrainMaker** нейронну мережу, що реалізує логічну систему XOR (рис. 3.11) з вагами, розрахованими в прикладі 3.27 генетичним алгоритмом програми **Evolver** і представленими на рис. 3.117.

На рис. 3.139 показані значення ваг, введені в програму **BrainMaker.** Вони згруповані в два блоки, розділені пустим

рядком. Перший блок містить ваги зв'язків між вхідним і схованим шаром так, що для кожного з двох нейронів схованого шару приведені ваги зв'язків із усіма входами, а останні елементи рядків - це $w_{10}$ і $w_{20}$. Другий блок містить ваги зв'язків між схованим і вихідним шаром, тобто три ваги вихідного нейрона, причому останнім зазначена вага $w_{30}$. У прикладі використовувався інтервал значень ваг [-7, 8].

Результати тестування мережі приведені на рис. 3.140-3.143. Можна зробити висновок про гарну навченість мережі. Значення на виходах для чотирьох пар вхідних значень практично збігаються з показаними на рис. 3.117. Абсолютна різниця між заданим значенням $d$ і вихідним значенням $y$ для кожної пари входів $u_1$ і $u_2$ (рис. 3.140-3.143) виявилася менше 0,025, тому мережа може вважатися добре навченою з толерантністю 0,025 і тим більше - з толерантністю 0,1. Зниження порога толерантності до 0,02 означало б, що ця мережа не вважається добре навченою і що необхідний рівень навчання не може бути досягнутий. Подальше навчання при толерантності, рівній або меншій 0,02, не змінює значень ваг, показаних на рис. 3.139.

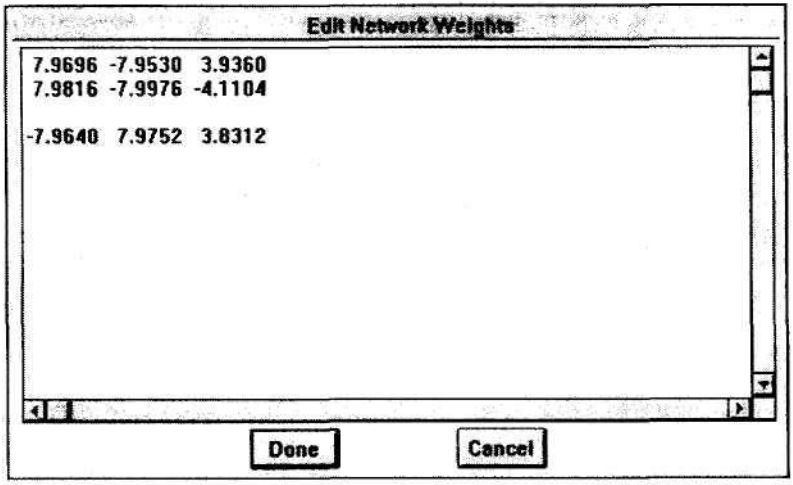



Рис. 3.139**.** Показані на рис. 3.117 ваги нейронної мережі, що реалізує систему XOR (рис. 3.11) і підготовлені до тестування програмою **BrainMaker.**

Однак, як буде видно з приклада 3.31, зменшення толерантності до 0,022 принесло б ефект у виді «донавчання» мережі.

**Приклад 3.31**

Навчити за допомогою програми **BrainMaker** нейронну мережу, що реалізує логічну систему XOR (рис. 3.11) з вагами, розрахованими в прикладі 3.27 генетичним алгоритмом програми **Evolver** і представленими на рис. 3.116.

Спочатку за допомогою програми **BrainMaker** була протестована нейронна мережа, яка показана на рис. 3.11. Як початкові значення для алгоритму навчання програми **BrainMaker** використовувалися ваги, які розраховано генетичним алгоритмом програми **Evolver.** Отже, це типовий приклад гібридного підходу, оскільки генетичний алгоритм використовується для знаходження початкових значень ваг для градієнтного алгоритму зворотного поширення похибки *(backpropagation).* Ці початкові значення у форматі програми **BrainMaker** представлені на рис. 3.144, а відповідні результати тестування мережі - на рис.3.145 - 3.148.

Аналіз результатів тестування свідчить про те, що вони виявляються гірші, ніж показані на рис. 4.140-4.143, оскільки в попередньому прикладі не було ні однієї помилки, а в поточному прикладі в п'ятьох випадках з 22 мережа дала невірну відповідь.

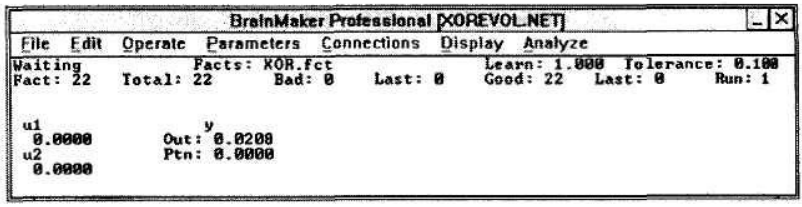



Рис. 3.140**.** Результат тестування програмою **BrainMaker** нейронної мережі з рис. 3.11 з початковими значеннями ваг, представленими на рис. 3.139, для $u_1 = 0$, $u_2 = 0$ і d = 0.

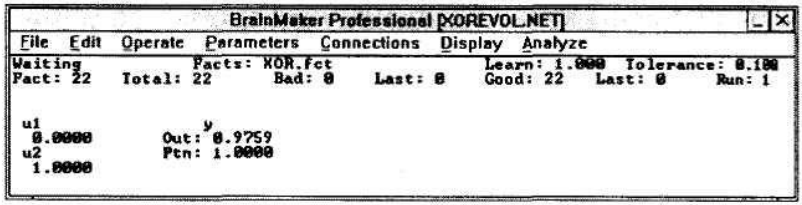



Рис. 3.141**.** Результат тестування програмою **BrainMaker** нейронної мережі з рис. 3.11 з початковими значеннями ваг, представленими на рис. 4.139, для $u_1 = 0$, $u_2 = 1$ і d = 1.

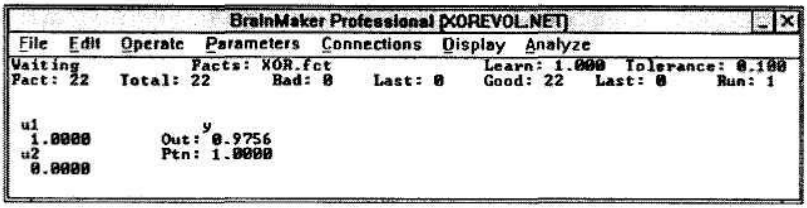



Рис. 3.142**.** Результат тестування програмою **BrainMaker** нейронної мережі з рис. 3.11 з початковими значеннями ваг, представленими на рис. 3.139, для $u_1 = 1$, $u_2 = 0$ і d = 1.

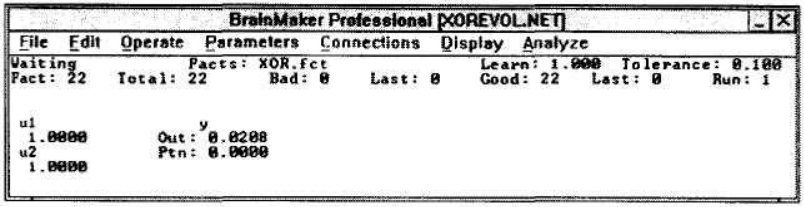



Рис. 3.143**.** Результат тестування програмою **BrainMaker** нейронної мережі з рис. 3.11 з початковими значеннями ваг, представленими на рис. 3.139, для $u_1 = 1$, $u_2 = 1$ і d = 0.

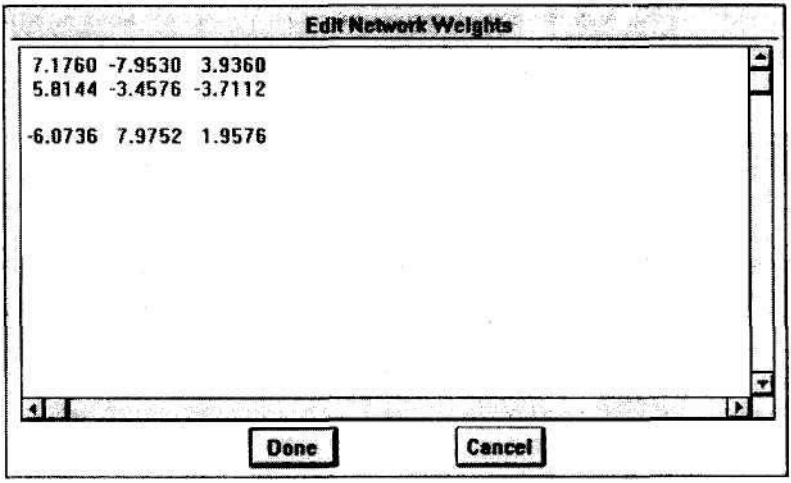



Рис. 3.144**.** Показані на рис. 3.116 ваги нейронної мережі, що реалізує систему XOR і підготовленої до навчання програмою **BrainMaker**

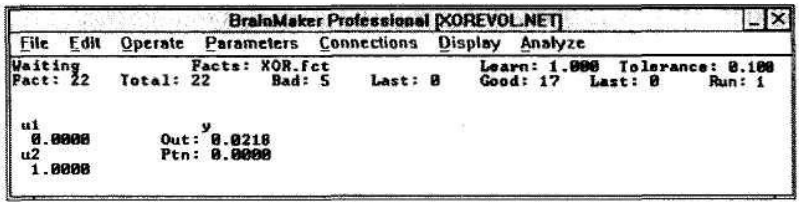



Рис. 3.145**.** Результат тестування нейронної мережі з рис. 3.11 з початковими значеннями ваг, представленими на рис. 3.144, для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

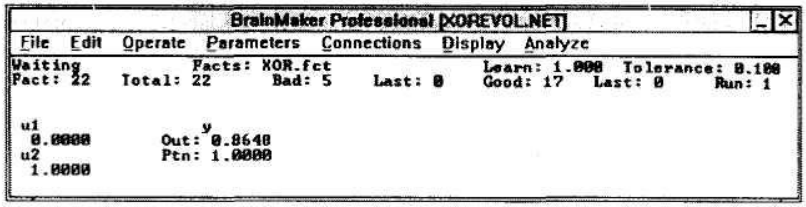



Рис. 3.146**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і

d = 1.

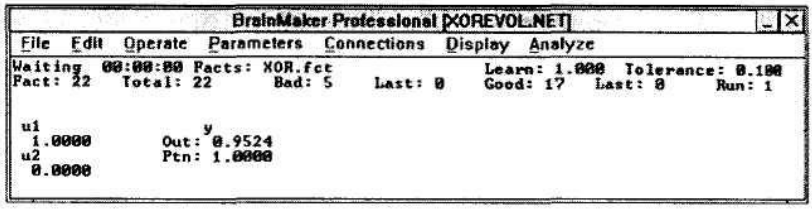



Рис. 3.147**.** Результат тестування тієї ж мережі для $u_1$= 1, $u_2$ = 0 і $d$ = 1.

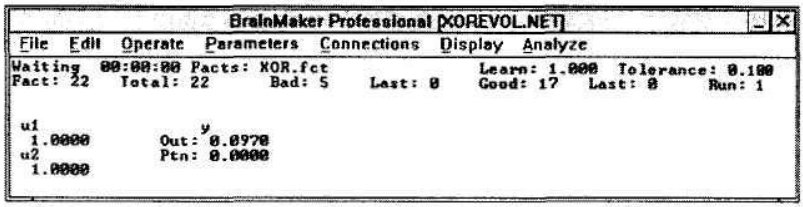



Рис. 3.148**.** Результат тестування тієї ж мережі для $u_1$= 1, $u_2$ = 1 і $d$ = 0.

Крім того, видно, що толерантність вихідного значення на рис. 3.146 перевищує рівень 0,1. Згодом мережа піддалася навчанню за допомогою програми **BrainMaker** з толерантністю, рівною 0,1. Результати навчання ілюструє рис. 3.149.

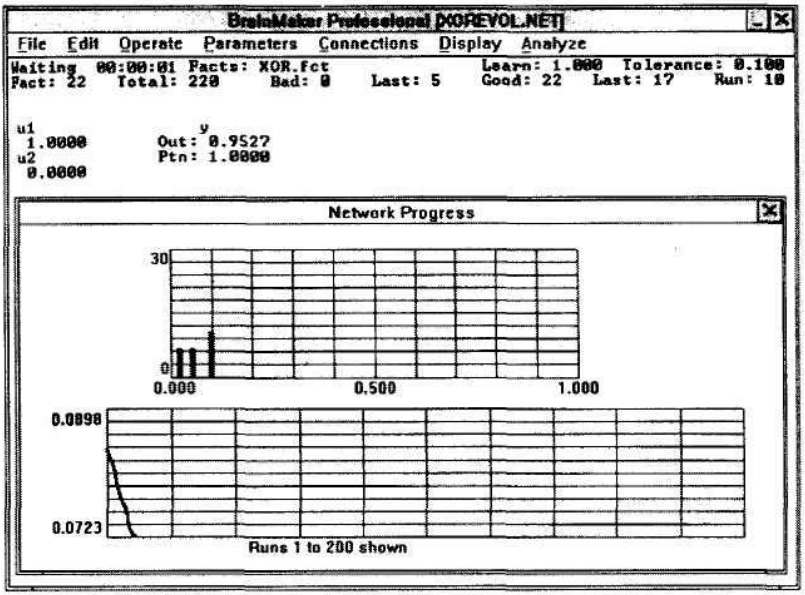



Рис. 3.149**.** Результат навчання програмою **BrainMaker** нейронної мережі з рис. 3.11 з початковими значеннями ваг, представленими на рис. 3.144.

На нижньому графіку помітне зменшення середньоквадратичної похибки RMS *(Root Mean Squared)* при виконанні 10 реалізацій *(runs)* алгоритму. Похибка RMS відрізняється від похибки $Q$, що мінімізувалася в прикладі 3.27, тим, що

$$Q=\frac{1}{N}\sum_{i=1}^{N}\varepsilon_{i^2}$$

$$\text{RMS}=\frac{1}{N}\sqrt{\sum_{i=1}^{N}\varepsilon_{i^2}}$$

де $\varepsilon_i = d_i - y_i$, а значення $N$ у прикладі 3.27 дорівнює 4.

З урахуванням розходжень, зв'язаних з узяттям квадратного кореня при розрахунку RMS, можна порівняти цю похибку зі значенням $Q$ із приклада 3.27. Верхній графік на рис. 3.149 являє собою гістограму розподілу похибки, розрахованою як абсолютна

різниця між заданим $d_i$ і фактичним $y_i$ вихідним значенням для кожної пари входів $u_{1i}$ і $u_{2i}$. На горизонтальній осі відкладено значення цієї похибки, а на вертикальній осі - кількість вихідних значень з такою похибкою. Представлено три рівні похибки з толерантністю 0,1. У верхній частині рис. 3.149 зазначене значення $y$ для $u_1 = 1$, $u_2 = 0$ і $d = 1$ (для спрощення тут опущено індекс $i$). Результати тестування цієї мережі для трьох інших комбінацій вхідних значень (для системи XOR) приведені на рис. 3.150-3.152.

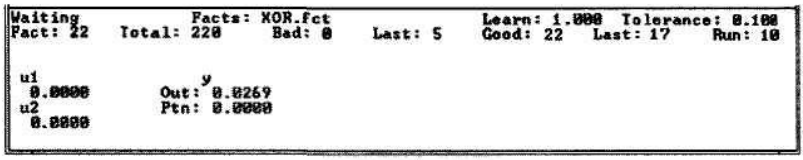



Рис. 4.150**.** Результат тестування програмою **BrainMaker** нейронної мережі з рис. 3.2 з початковими значеннями ваг, представленими на рис. 3.144, з толерантністю 0,1 для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

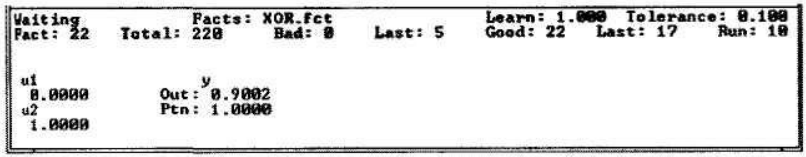



Рис. 3.151**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d = 1$**.**

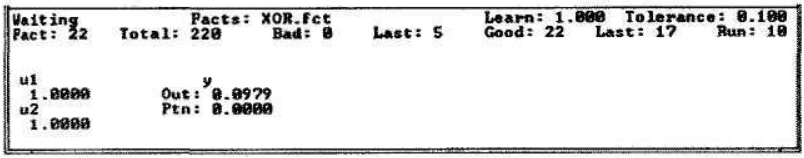



Рис. 3.152**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Абсолютна різниця між еталонним (Ptn) і вихідним (Out) значеннями для рис. 3.149 дорівнює 0,0473, для рис. 3.150 -

0,0269, для рис. 3.151 - 0,0998 і для рис. 3.152 - 0,0979. З гістограми на рис. 3.149 випливає, що для шести з 22 вхідних пар $u_1$, $u_2$ ця похибка дорівнює 0,0269, таку ж кількість вхідних пар має похибка 0,0473, а для вхідних пар, що залишилися, ця похибка склала близько 0,099. Ваги навченої в такий спосіб мережі показано на рис. 3.153.

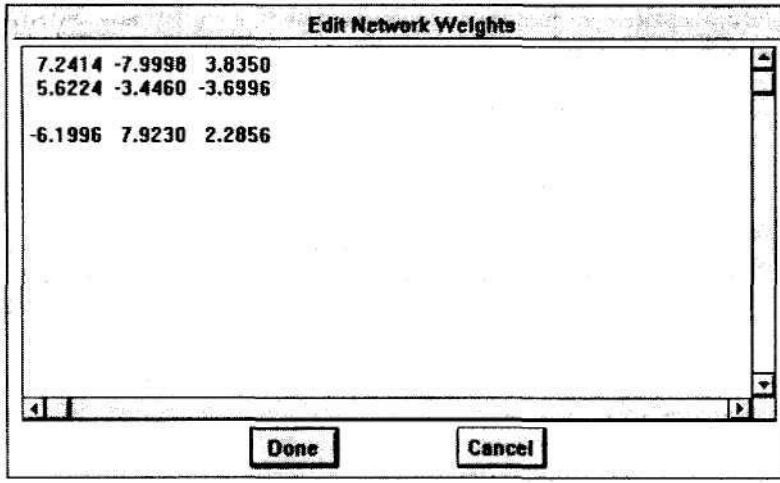



Рис. 3.153**.** Ваги нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,1

Мережа навчена з толерантністю 0,1. Це означає, що вихідні значення повинні не більше ніж на 10 % відрізнятися від значень 0 і 1 для того, щоб модель системи XOR була визнана коректною. Отже, при подачі еталона 0 вважаються правильними значення *y*, менші або рівні 0,1, а при подачі еталона 1 правильними будуть значення *y*, більші або рівні 0,9.

Звичайно, показані на рис. 3.153 значення ваг не можуть розглядатися в якості оптимальних для системи XOR. Тому толерантність була зменшена до 0,025, і процес навчання продовжився. Отримані результати приведено на рис. 3.154.

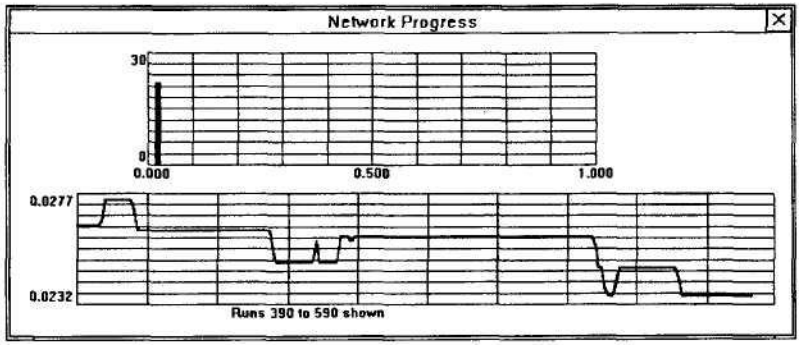



Рис. 3.154**.** Підсумкова фаза наступного навчання програмою **BrainMaker** мережі з вагами, показаними на рис. 3.153, при рівні толерантності 0,025.

Цікаво порівняти досягнуте значення похибки RMS з аналогічним показником на рис. 3.149. Результати тестування навченої мережі представлені на рис. 3.155 - 3.158.

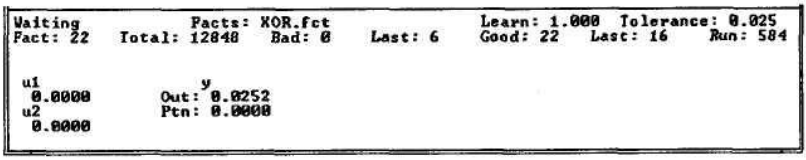



Рис. 4.155**.** Результат тестування нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,025 (рис. 3.154), для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

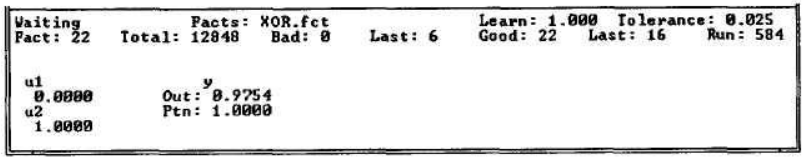



Рис. 3.156**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

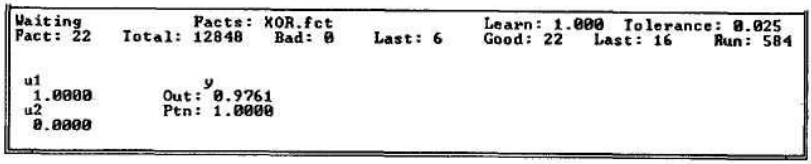



Рис. 3.157**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 0$ і $d = 1$.

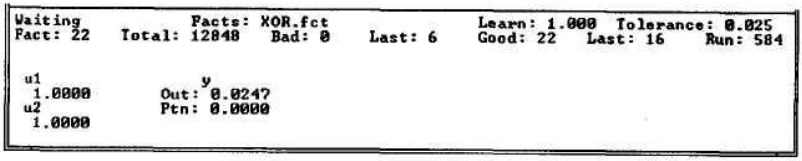



Рис. 3.158**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Значення абсолютної різниці між еталоном і фактичним вихідним значенням на наступних один за одним рисунках складає 0,0252, 0,0246, 0,0239, 0,0247 відповідно, що відбиває гістограма на рис. 3.154.

Значення ваг для мережі, навченої з толерантністю 0,025, приведено на рис. 3.159.

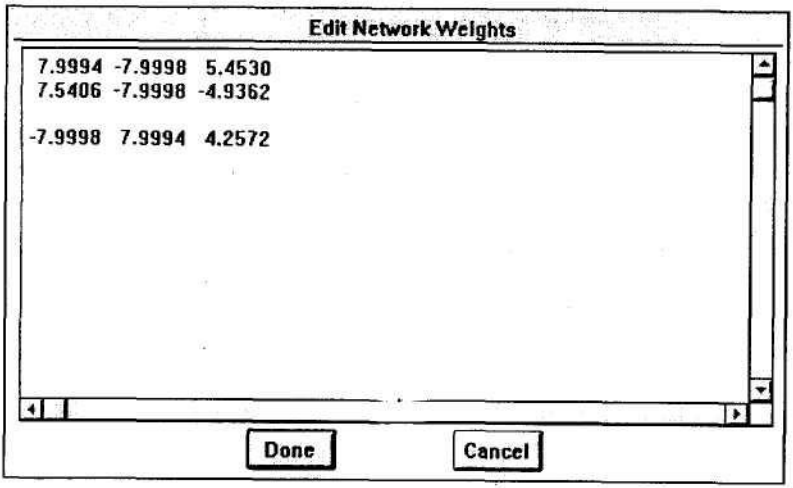


Рис**.** 3.159. Ваги нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,025 (рис. 3.154).

Цікаво порівняти його з рис. 3.153. Більш пізні результати виявляються набагато ближчі до оптимального. Далі була почата спроба ще краще навчити мережу зі зменшенням толерантності до 0,02. На жаль, ця спроба завершилася невдачею, оскільки здатності мережі до навчання виявилися вичерпаними. Однак при фіксації толерантності на рівні 0,023 був досягнутий кінцевий ефект, показаний на рис.3.160.

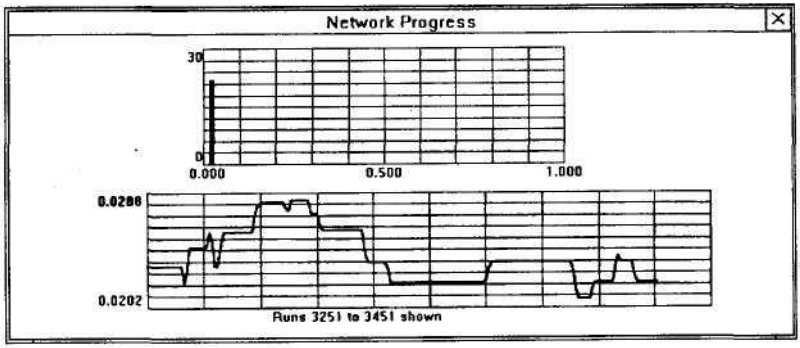



Рис. 3.160**.** Підсумкова фаза наступного навчання програмою **BrainMaker** мережі з вагами, показаними на рис. 3.159, при рівні толерантності 0,023.

Результати тестування мережі, навченої подібним чином, представлені на рис. 3.161 - 3.164, а значення ваг цієї мережі - на рис. 3.165.

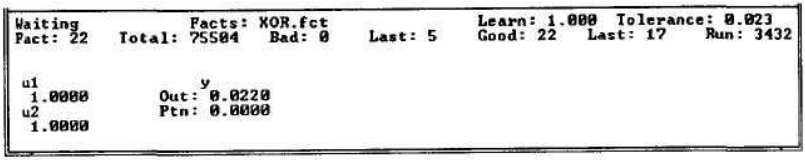



Рис. 3.161**.** Результат тестування нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,023 (рис. 3.160), для

$u_1 = 0$, $u_2 = 0$ і $d = 0$.

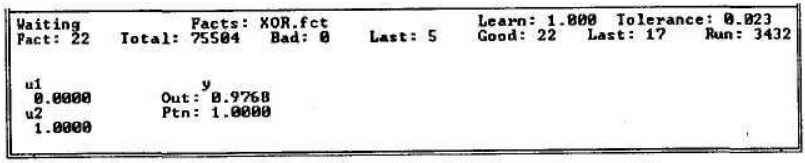



Рис. 3.162**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d=1$.

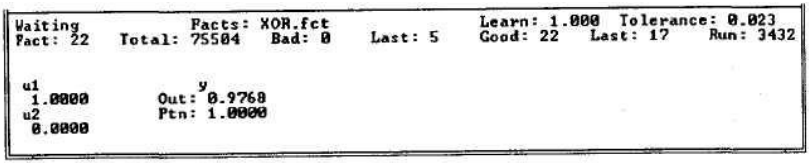



Рис. 3.163**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 0$ і $d=1$.

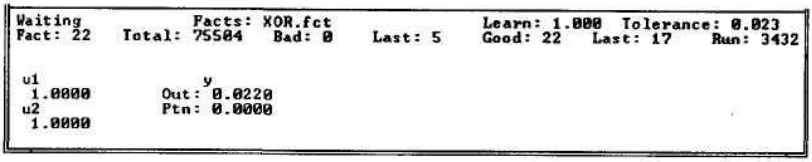



Рис. 3.154**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

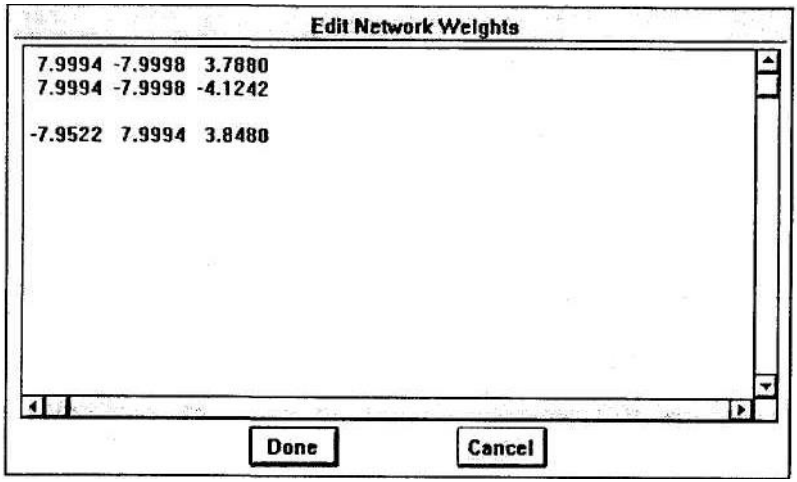



Рис. 3.165**.** Ваги нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,023 (рис. 3.160)**.**

Ще кращий результат навчання, що ілюструється рис. 3.166, удалося одержати при толерантності 0,022.

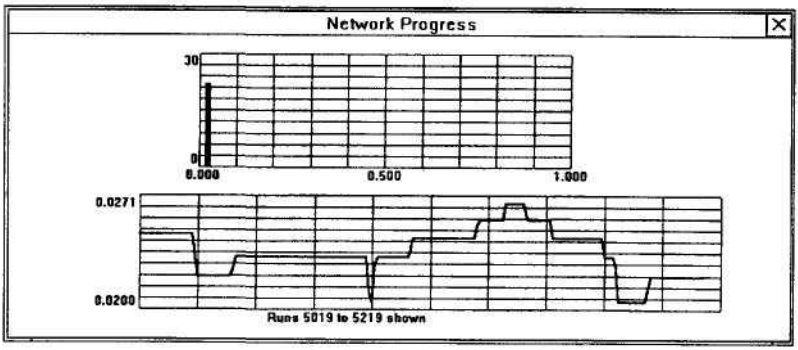



Рис. 3.166**.** Підсумкова фаза наступного навчання програмою **BrainMaker** мережі з вагами, показаними на рис. 3.165, при рівні толерантності 0,022.

Результати тестування мережі, навченої подібним чином, представлені на рис. 3.167-3.170, а значення ваг цієї мережі - на рис. 3.171.

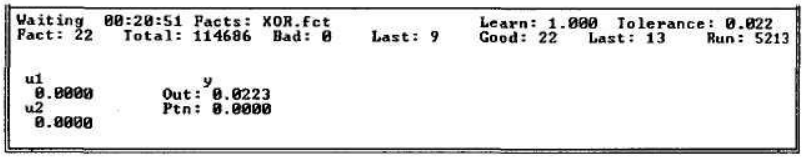



Рис. 3.167**.** Результат тестування нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,022 (рис. 3.166), для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

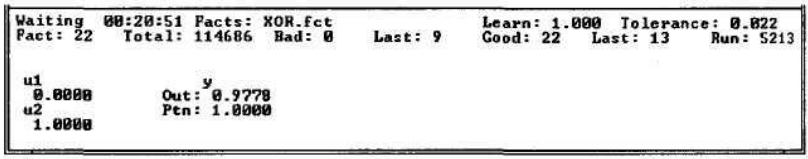



Рис. 3.168**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

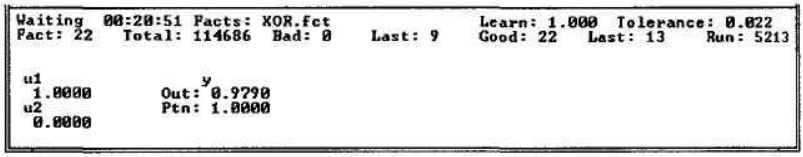



Рис. 3.169**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 0$ і $d = 1$.

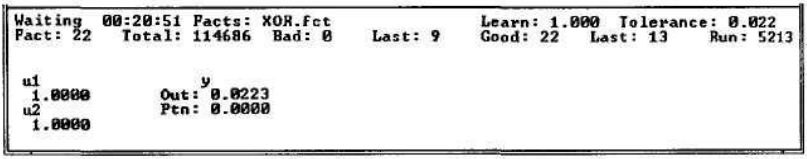



Рис. 3.170**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

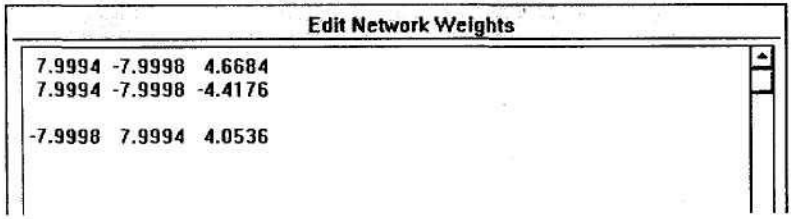



Рис. 3.171**.** Ваги нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,022 (рис. 3.166).

Подальше зниження рівня толерантності до 0,021, на жаль, уже не веде до більшої навченості мережі, навіть якби програма працювала ще протягом багатьох годин. Таким чином, найкращим рішенням вважається мережа зі значеннями ваг, показаними на рис. 3.171, які варто порівняти з набором ваг для приклада 3.27, приведеними на рис. 3.117. Неважко помітити, що об'єднання генетичного алгоритму програми **Evolver** із програмою **BrainMaker** дало кращі результати, чим на рис. 3.117. Якби обчислення відразу проводилися з толерантністю 0,022, а як початкові ваги приймалися значення з рис. 3.144, то результат був би кращим, ніж на рис. 3.117. Аналогічний результат можна було очікувати й у прикладі 3.30 при встановленні рівня толерантності, рівним 0,022.

**Приклад 3.32**

Навчити за допомогою програми **BrainMaker** нейронну мережу, що реалізує логічну систему XOR (рис. 3.11) з початковими значеннями ваг, розрахованими в прикладі 3.27 генетичним алгоритмом програми **Evolver** і представленими на рис. 3.115. Це ще один приклад гібридного підходу до навчання ваг нейронної мережі, що реалізує логічну систему XOR. Він дуже схожий на попередній приклад, однак відрізняється набором початкових значень ваг, отриманих за менший час виконання генетичного алгоритму і, отже, більш далеких від оптимальних. Толерантність похибки прийнята рівною 0,025, що означає 2,5 % припустимої похибки, тобто різниці між фактичним вихідним значенням і заданим значенням - еталоном. Тому для еталону, рівному 0, коректним буде визнаватися вихідне значення від 0 до 0,025, а для еталону, рівному 0 -

значення від 0,975 до 1. Початковий набір ваг для програми **BrainMaker** представлено на рис. 3.172, а на рис. 3.173 - 3.176 показані результати тестування мережі.


Рис. 3.172**.** Вихідна множина ваг для програми **BrainMaker,** отримана за допомогою генетичного алгоритму програми **Evolver.**

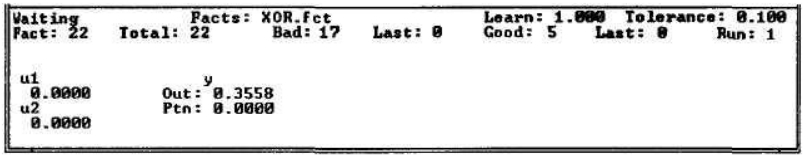



Рис. 3.173. Результат тестування нейронної мережі з вагами, показаними на рис. 3.172, для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

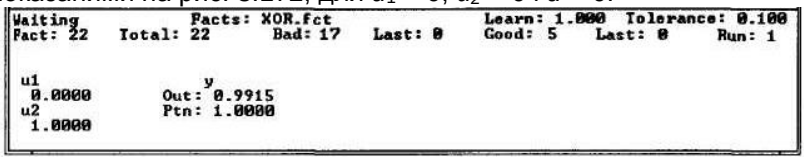



Рис. 3.174. Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

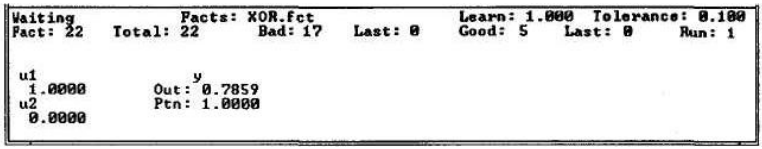



Рис. 3.175**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 0$ і $d = 1$.

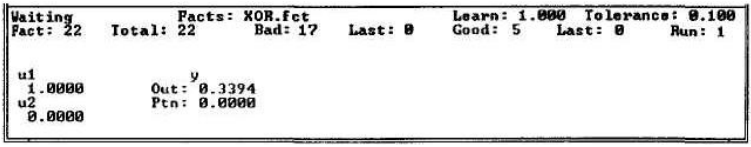



Рис. 3.176**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Видно, що вихідні значення в для конкретних пар входів досить близькі до приведених на рис. 3.115. Мережа не може вважатися навченою. На 22 тестуючих вибірках тільки 5 разів реакція на виході була коректною, а в 17 випадках - помилковою. Набір ваг з рис. 3.172 був отриманий після 96 «тактів» функціонування програми **Evolver** (що відповідає менш чим 2 ітераціям класичного генетичного алгоритму). Результати навчання нейронної мережі з цими вагами програмою **BrainMaker** з толерантністю похибки, рівною 0,025, представлені на рис. 3.177.

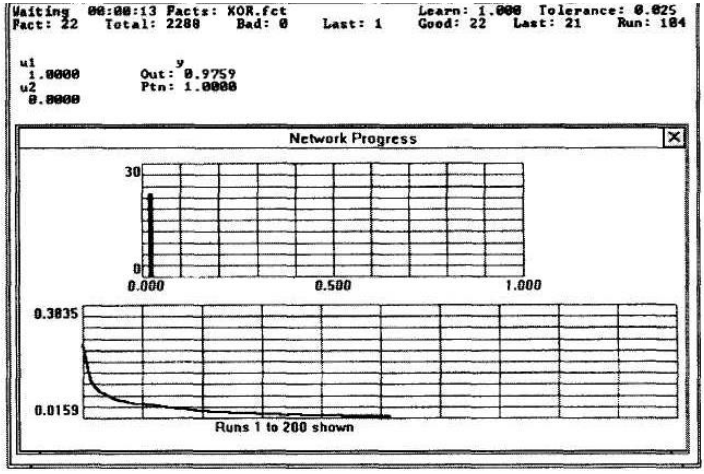



Рис. 3.177**.** Результат навчання програмой **BrainMaker** нейронної мережі з вагами, представленими на рис. 3.144, при рівні толерантності 0,025.

Значення похибки RMS із графіка цього рисунка легко порівняти зі значенням похибки *Q* на рис. 3.115, 3.116 і 3.117. Помітно, що вихідні значення в для кожної пари входів системи XOR укладаються в границі 2,5% толератності. Мережа навчилася досить швидко - за 104 прогона *(runs).* На рис. 3.181 представлені ваги навченої мережі, а на рис. 3.177 - 3.180 - результати її тестування.

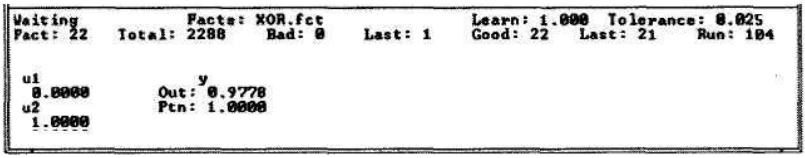



Рис. 3.178**.** Результат тестування нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,025 (рис. 3.177), для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

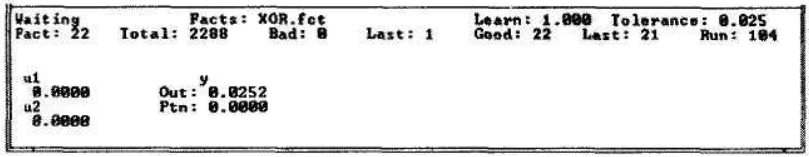



Рис. 3.179**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

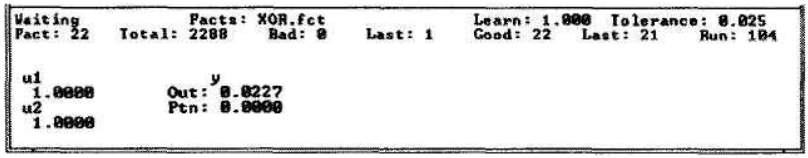



Рис. 3.180**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Становить інтерес зіставлення рис. 3.181 і 3.159, оскільки вони відбивають навчання з однієї і тією же толерантністю, рівною 0,025.

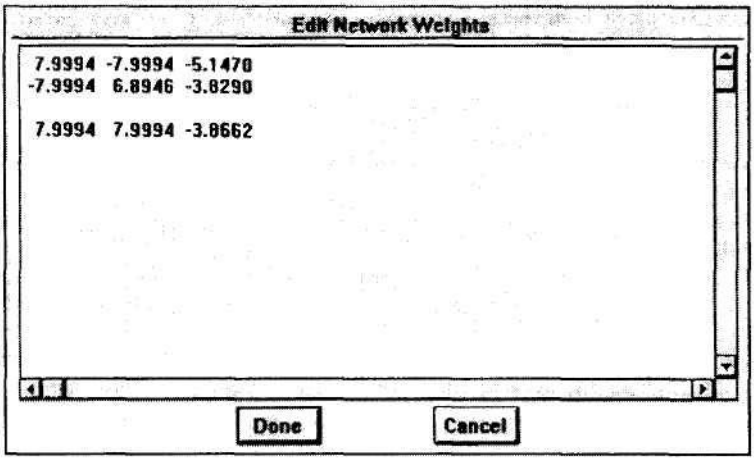



Рис. 3.181**.** Ваги нейронної мережі (при початкових значеннях, показаних на рис. 3.172), навченою програмою **BrainMaker** з толерантністю 0,025 .

Якщо продовжити навчання мережі при меншому значенні толерантності, то (за аналогією з прикладом 3.31) можна знайти значення ваг, ще більш близьких до оптимального і практично співпадаючі з показаними нарис. 3.171.

Розглянемо тепер ефект навчання нейронної мережі, що реалізує логічну систему XOR, за допомогою тільки програми **BrainMaker** без застосування генетичного алгоритму.

**Приклад 3.33**

Навчити за допомогою програми **BrainMaker** нейронну мережу, що реалізує логічну систему XOR (рис. 3.2) з початковими значеннями ваг, представленими на рис. 3.112.

Показані на рис. 3.112 значення ваг згенеровані випадковим чином. На рис. 3.182 ці ваги представлені у форматі програми **BrainMaker.**

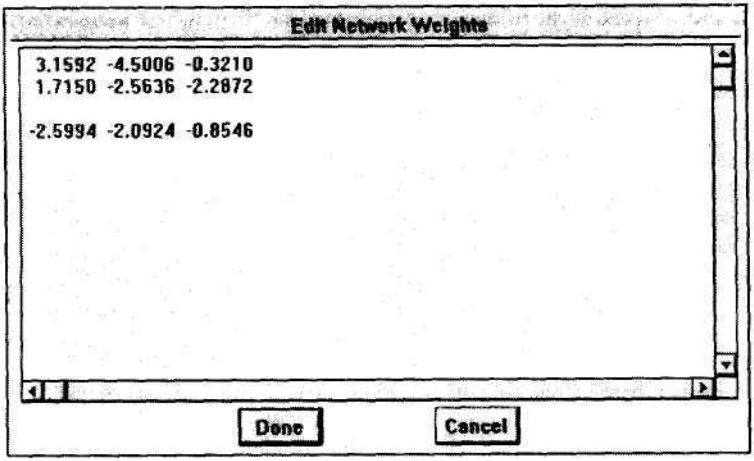



Рис. 3.182**.** Згенерований випадковим чином вихідний набір ваг для нейронної мережі, що реалізує систему XOR.

Нейронна мережа з цими вагами не може вважатися навченою. Результати її тестування приведені на рис. 3.183 - 3.186.

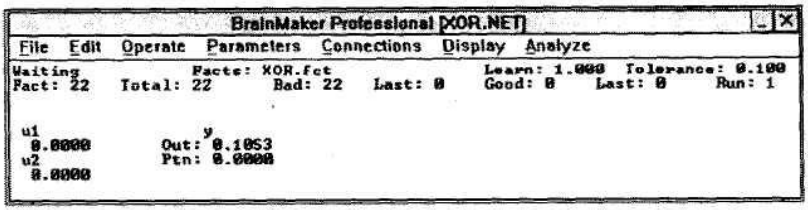



Рис. 3.183. Результат тестування нейронної мережі з вагами, показаними на рис. 3.166, для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

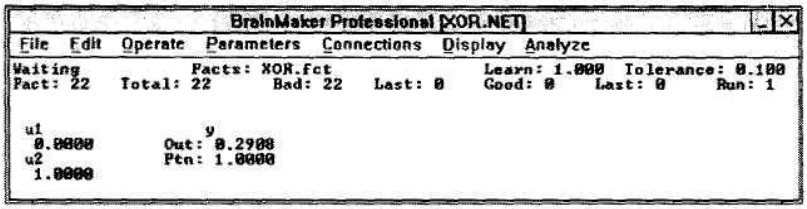


Рис. 3.184**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

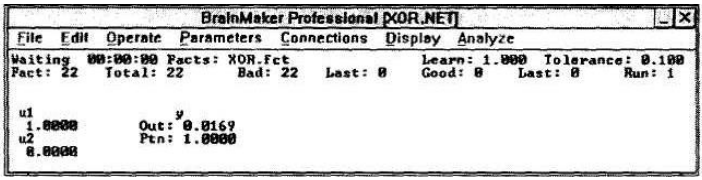



Рис. 3.185**.** Результат тестування тієї ж мережі для $u_1 = 1$,$u_2 = 0$ і $d$=1.

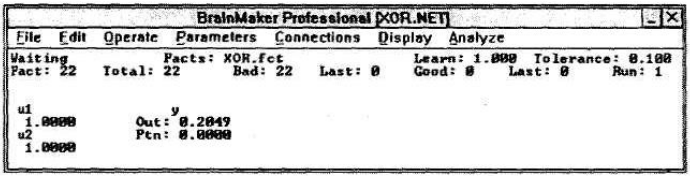



Рис. 3.186**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Помітно, що вихідні значення *y* зовсім не відповідають еталонові 1, а для еталону 0 вихідні значення також не попадають у границі 10% толерантності. Для всіх 22 тестуючих вибірок отримано помилкові вихідні сигнали. Спочатку навчання мережі проводилося з толерантністю похибки, рівною 0,1. Процес навчання ілюструють графіки на рис. 3.187.

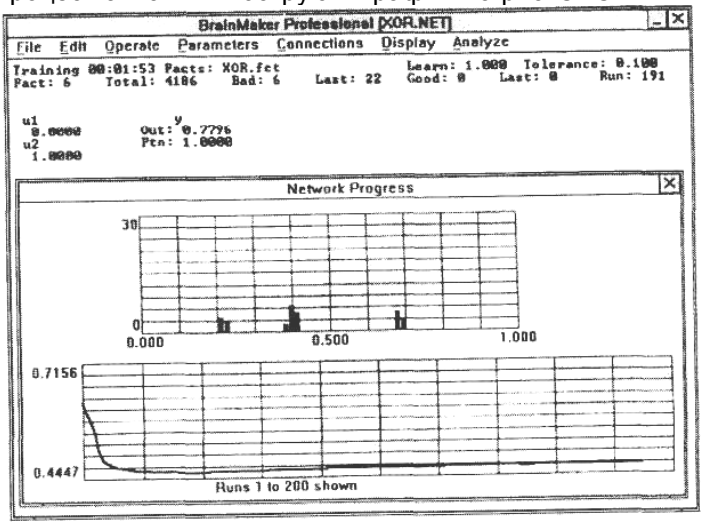


Рис. 3.187. Початкова фаза навчання програмою **BrainMaker** мережі з вагами, показаними на рис. 3.182, при рівні толерантності 0,1.

На рис. 3.188 показано значення ваг, які отримано після 191 прогону (runs) алгоритму.

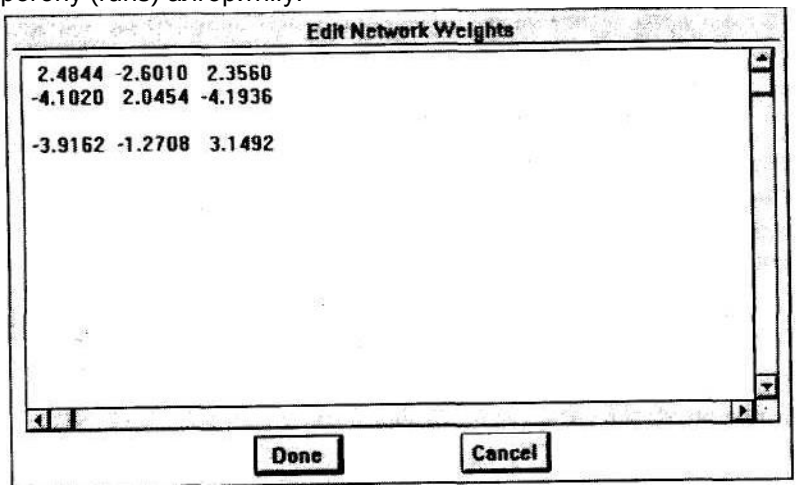



Рис. 3.188. Ваги, які отримано після 191 прогону алгоритму навчання програми **BrainMaker.**

Продовження графіка з рис. 3.187 демонструється на рис. 3.189, а його завершення - на рис. 3.190.

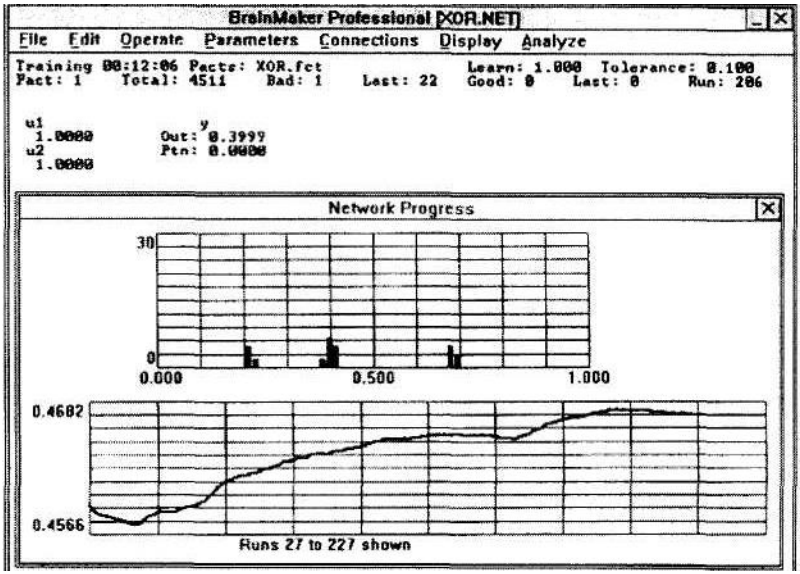



Рис. 3.189. Продовження навчання, показаного на рис. 3.187.

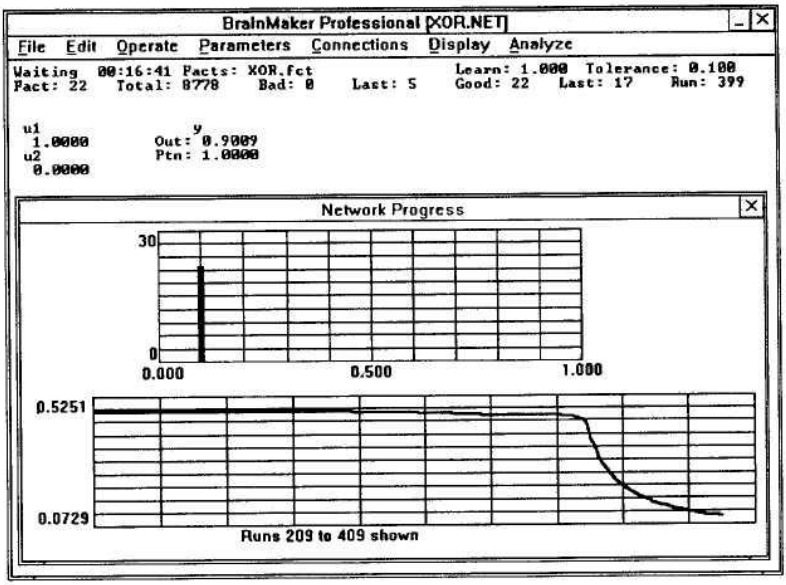


Рис. 3.190. Завершальна фаза навчання, показаного на рис. 3.187.

Отримані значення ваг нейронної мережі з толерантністю похибки, рівною 0,1, представлені на рис. 3.191.

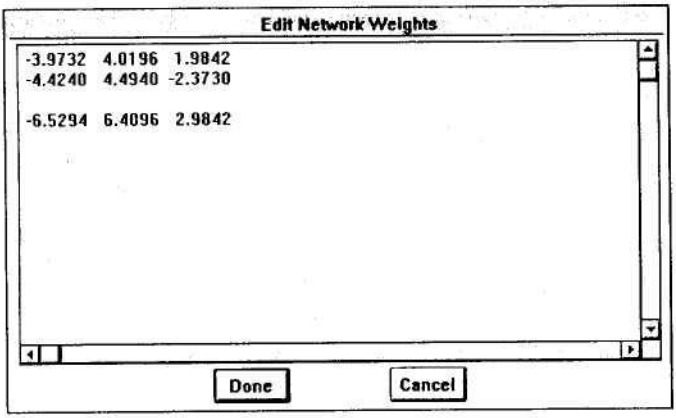



Рис. 3.191**.** Ваги нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,1**.**

Помітимо, що значення похибки RMS для цього випадку (див. рис. 3.190) досить близьке до значення похибки *Q*, розрахованому програмою **Evolver** у прикладах 3.23 - 3.27 для значень ваг з рис. 3.191 (середньоквадратична похибка *Q* = 0,009887). Нейронна мережа з вагами, показаними на рис. 3.182, була навчена з толерантністю похибки 0,1 за 399 прогонів алгоритму програми **BrainMaker.** Результати тестування сформованої мережі представлені на рис. 3.192-3.194.

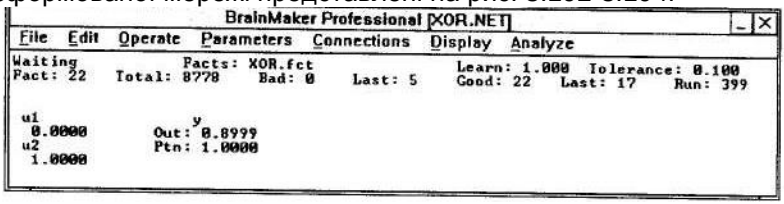



Рис. 3.192**.** Результат тестування нейронної мережі, навченою програмою **BrainMaker** з толерантністю 0,025, для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

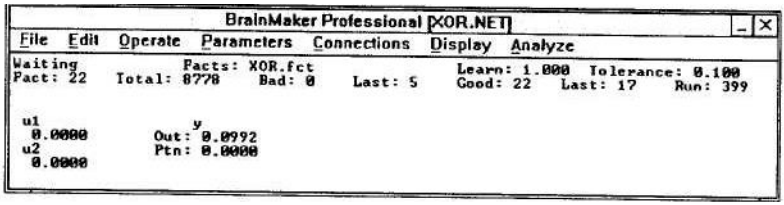



Рис. 3.193**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

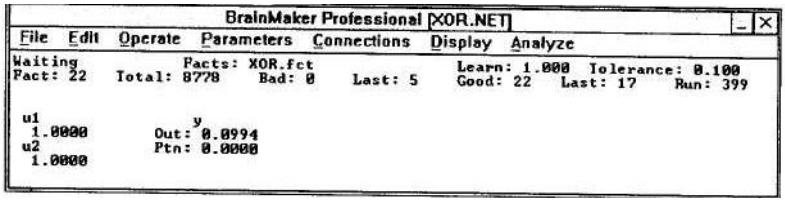



Рис. 3.194**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Для входів $u_1 = 1$ і $u_2 = 0$ рішення приведено на рис. 3.190. Мережа може вважатися навченою, оскільки в ході тестування не зареєстровано помилкових відгуків. Далі навчання мережі продовжилося з толерантністю похибки, рівною 0,025. Після 510 прогонів отримано графіки, які зображено на рис. 4.195, і значення ваг, які приведені на рис. 3.196.

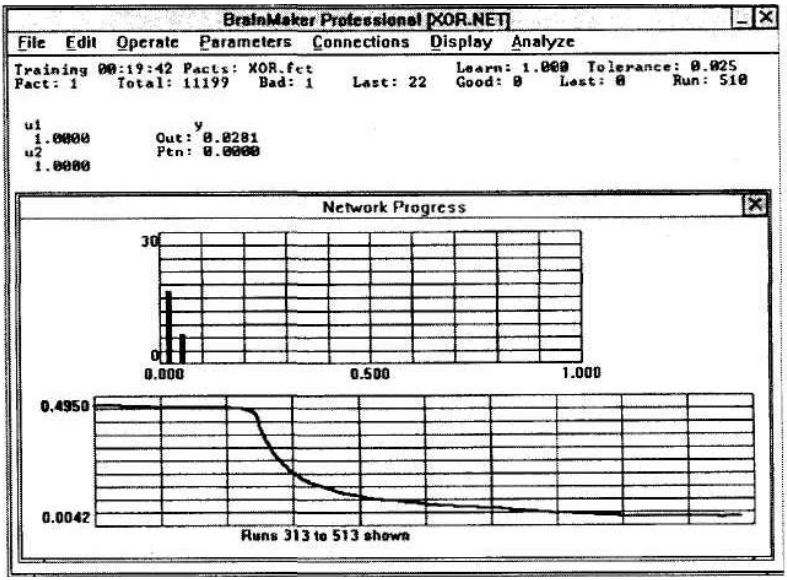



Рис. 3.195**.** Графіки, отримані при наступному навчанні програмою **BrainMaker** мережі з вагами, показаними на рис. 3.191, при рівні толерантності 0,025.

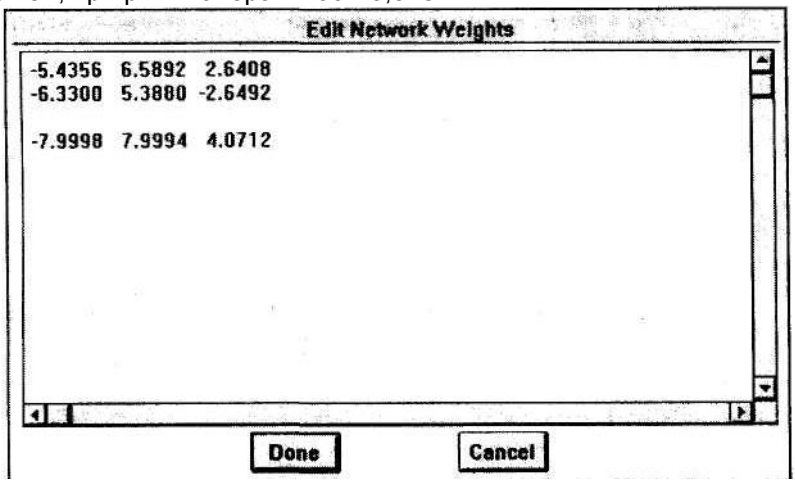



Рис. 3.196**.** Ваги, отримані в результаті наступного навчання мережі при рівні толерантності 0,025 (після 510 прогонів - мал. 3.195).

Значення похибки RMS для цієї комбінації ваг можна одержати з графіка на рис. 3.195. При толерантності похибки 0,025 мережа погано піддавалася навчанню, тому після 4000 прогонів рівень толерантності був збільшений до 0,03. Графіки і значення ваг на момент зміни рівня показано на рис.3.197 і 3.198 відповідно.

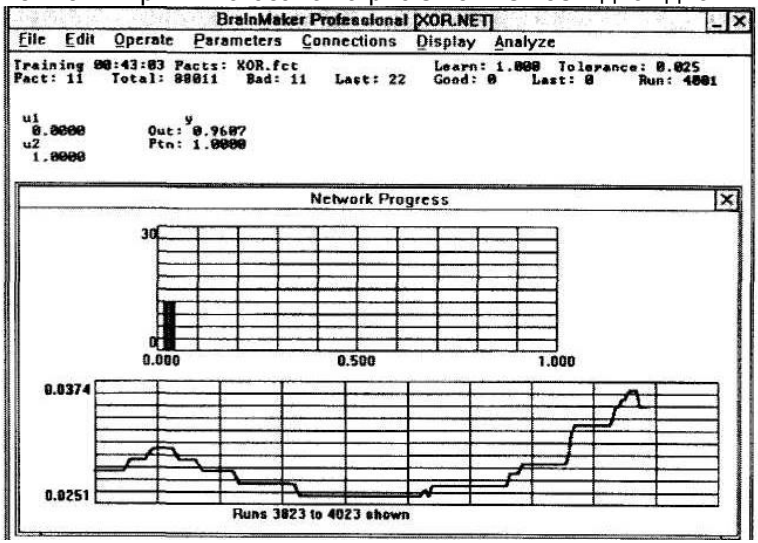



Рис. 3.197**.** Процес навчання мережі з толерантністю 0,025 після 4000 прогонів.

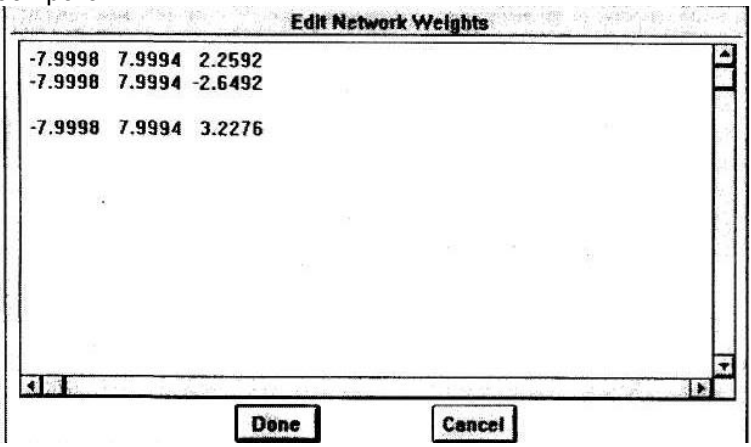


Рис. 3.198**.** Ваги, які отримані після 4000 прогонів навчання мережі з толерантністю 0,025.

Після збільшення значення толерантності виконання програми дуже швидко - після 402 прогонів - завершилося. Досягнутий ефект можна спостерігати на рис. 3.199, а отримані значення ваг - на рис. 3.200.

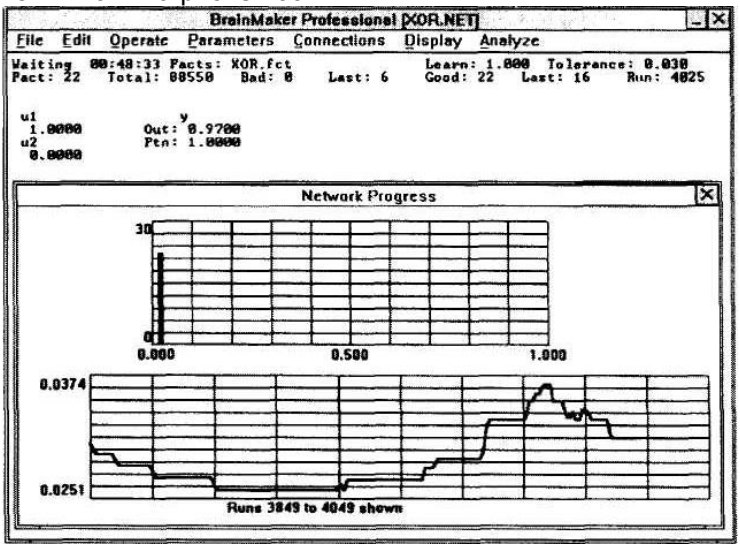



Рис. 3.199**.** Процес навчання мережі на рис. 3.197 при зміні толерантності до 0,03.

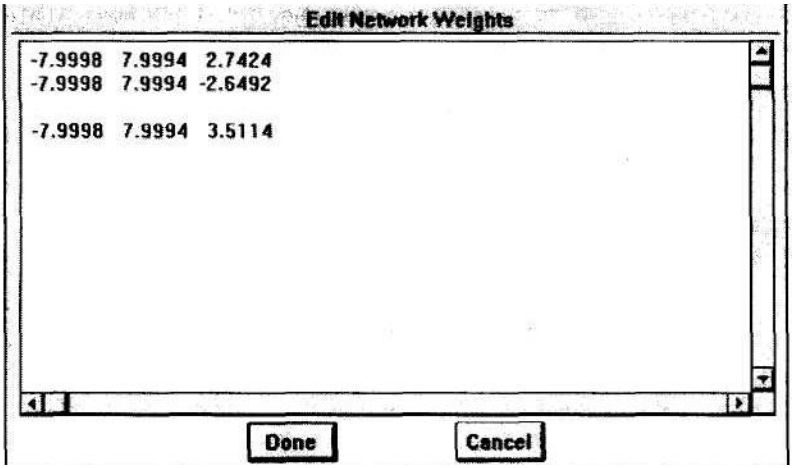


Рис. 3.200**.** Ваги, які отримані по завершенні навчання мережі з толерантністю 0,03 (рис. 3.199).

У результаті навчання нейронної мережі, що реалізує логічну систему XOR з початковими значеннями ваг, представленими на рис. 3.182 і 3.112, була знайдена ще одна комбінація ваг, подібна отриманим при мінімізації похибкиі *Q* за допомогою програми **Evolver.** У принципі, сформований у поточному прикладі і представлений на рис. 3.200 набір ваг мережі, навченої за допомогою програми **BrainMaker,** відрізняється від двох аналогічних комбінацій «найкращих» ваг із приклада 3.27 тільки знаками. Наприклад, якщо порівняти ваги з рис. 3.200 з вагами, показаними на рис. 3.117, то можна зафіксувати, що ваги $w_{11}$, $w_{12}$, $w_{21}$, $w_{22}$ змінили знаки на протилежні, а ваги $w_{31}$, $w_{32}$, $w_{10}$, $w_{20}$, $w_{30}$ мають в обох випадках ті самі знаки, тобто $w_{31}$ і $w_{20}$ залишилися від'ємними, а $w_{32}$, $w_{10}$, $w_{30}$ - додатними. Результати тестування сформованої мережі (з вагами на рис. 3.200) для входів $u_1 = 1$ і $u_2 = 0$ приведено на рис. 3.199. Для пари входів $u_1 = 0$ і $u_2 = 1$ значення *y* залишилося незмінним, тобто рівним 0,9700.

Для $u_1 = 0$ і $u_2 = 0$, а також для $u_1 = 1$ і $u_2 = 1$ вихідне значення *y* дорівнює 0,0301.

Розглянуті приклади ілюструють гібридний підхід, заснований на об'єднанні двох різних методів - *генетичного алгоритму і градієнтного методу навчання ваг нейронної мережі*, відомого

за назвою зворотного поширення похибки. У прикладі 3.27 для навчання нейронної мережі застосовувався генетичний алгоритм програми **Evolver.** Після 40000 «тактів» функціонування цього алгоритму (що відповідає 800 поколінням класичного генетичного алгоритму) отримано близьке до оптимального рішення. Однак, як показують приклади 3.30 і 3.31, цей результат можна ще поліпшити за рахунок «донавчання» за допомогою градієнтного алгоритму.

Приклади 3.31 і 3.32 представляють типовий спосіб такого гібридного підходу, коли генетичний алгоритм застосовується тільки для вибору початкової точки (у даному випадку - вихідної множини ваг) для градієнтного методу. У прикладі 3.31 генетичний алгоритм функціонував довше, ніж у прикладі 3.32, тому початкова точка для методу зворотного поширення похибки в прикладі 3.32 знаходиться на більшій відстані від точки оптимального рішення, чим у прикладі 3.31. Варто звернути увагу на факт, що тривалість «донавчання» не має великого значення. Градієнтний алгоритм виконується швидше генетичного, тому що в останньому передбачається перегляд усієї популяції можливих рішень.

У прикладі 3.33 продемонстровано спосіб навчання нейронної мережі методом зворотного поширення похибки (програма **BrainMaker).** Отриманий результат виявився ще більш близьким до оптимального, чим при використанні тільки генетичного алгоритму (приклад 3.30). Однак необхідно підкреслити, що градієнтний метод не завжди приводить до досягнення очікуваного результату, що залежить від початкової точки. При іншому, згенерованої випадковим чином вихідної множини ваг мережа може виявитися «ненавченою», що зустрічається досить часто. Крім того, принциповим недоліком градієнтних методів виявляється їх «затвердівання» у локальних оптимумах. Це можна запобігти застосуванням генетичного алгоритму - тільки для того, щоб знайти яку-небудь точку, настільки близьку до глобального оптимуму, що градієнтний алгоритм при старті з цієї точки не «застрягне» ні в якому локальному оптимумі і швидко знайде глобальне рішення.

У розглянутих прикладах використовувався генетичний (еволюційний) алгоритм програми **Evolver**. Звичайно, для досягнення тих ж цілей можна було б застосовувати і програму **FlexTool**, що допускає такий же інтервал зміни значень ваг від -7

до 8, як і програма **BrainMaker**. Для вибору вихідної множини ваг нейронної мережі також придатні й інші реалізації генетичного (еволюційного) алгоритму.

### 3.21.2. Програма GTO

Програма **GTO** (Genetic Training Option - Режим Генетичного Навчання) взаємодіє з програмою **BrainMaker.** У ній реалізовано еволюційний алгоритм із відповідно визначеними операторами мутації і схрещування. Мутація змінює значення ваг обраного нейрона, а схрещування полягає в обміні ваг обраних нейронів у пари батьків. Наприклад, для нейронної мережі, зображеної на рис. 3.11, можна одержати нащадка, що має ваги першого нейрона, успадковані від першого батька, і ваги другого і третього нейронів - від другого батька. Виражені у відсотках показники мутації (mutation rate) і схрещування (crossover rate) визначають кількість нейронів, що піддається мутації або схрещуванню. Користувач також може ввести значення додаткових параметрів схрещування і мутації. Вони відносяться до так називаного повного схрещування або до визначених функцій розподілу показників схрещування і/або мутації.

**Приклад 3.34**

Застосувати програму **GTO** для нейронної мережі, що реалізує логічну систему XOR (рис. 3.11) з початковими значеннями ваг, представленими на рис. 3.182, що після навчання програмою **BrainMaker** з толерантністю похибки, рівною 0,1 (приклад 3.33) має ваги, які представлено на рис. 3.191.

Усі параметри програми **GTO** мали значення, які установлено за замовчуванням. Зокрема, передбачалося навчання кожної мережі програмою **BrainMaker** протягом 100 прогонів (runs) і зміною 30 поколінь. Для оцінки пристосованості мережі використовувалися результати як навчання і тестування вже навченої мережі. Показники мутації і схрещування були рівні відповідно 10 і 50. Це означає, що 50% усіх нейронів мережі піддавалося схрещуванню, а 10% - мутації. Також приймалося, що всі нейрони, що схрещуються, піддаються повному схрещуванню, тобто нащадок успадковує усі ваги нейрона від другого батька. Крім того, усі підлягаючі мутації нейрони мутировали відповідно до розподілу Гаусса з показником 0,25. Це означає, що до ваги додавалася деяка величина, що вибиралася по розподілу Гаусса. Як критерій оцінювання

нащадків застосовувався max{1 - RMS}, де RMS (також, як і в прикладі 3.31) означав похибку, розраховану як квадратний корінь із суми квадратів різностей між заданим (еталонним) і вихідним значенням, ділений на кількість еталонів. У результаті виконання програми **GTO** при її взаємодії з програмою **BrainMaker** розраховано значення зазначеної вище похибки для конкретних мереж, які упорядковані в порядку їхнього зменшення, тобто від «найкращої» мережі до «найгіршої» (рис. 3.201).

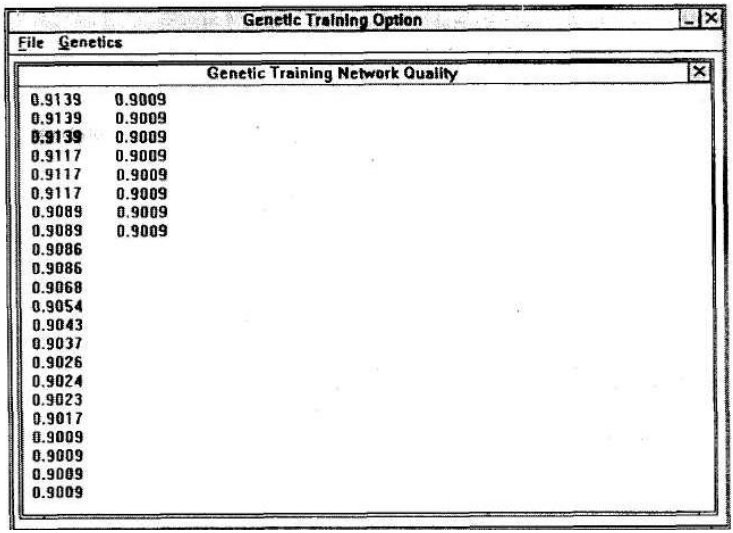



Рис. 3.201**.** Список нейронних мереж, сформованих програмою **GTO** для приклада 3.34 і упорядкованих у послідовності від «найкращої» до «найгіршої».

Найкращою з 30 сформованих програмою **GTO** нейронних мереж зі структурою, показаною на рис. 3.11, виявилася мережа, для якої значення max{1 - RMS} дорівнювало 0,9139. Значення ваг цієї мережі представлені на рис. 3.202, а результати її тестування програмою **BrainMaker** - на рис. 3.203 - 3.206.

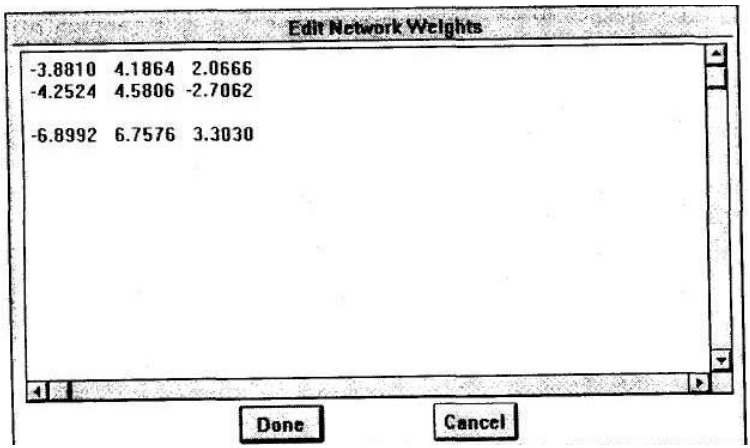



Рис. 3.202**.** Ваги нейронної мережі, що реалізує систему XOR, які отримані програмою **GTO** (приклад 3.34).

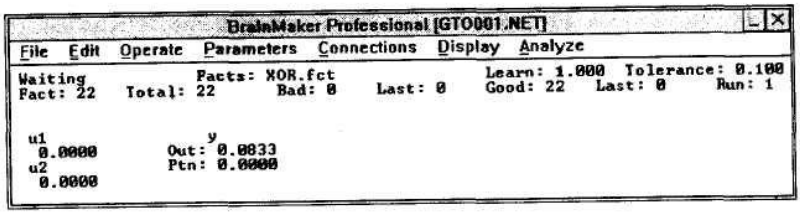



Рис. 3.203. Результат тестування сформованою програмою **GTO** нейронної мережі з вагами, показаними на рис. 3.202, для $u_1 = 0$, $u_2 = 0$ і $d = 0$.

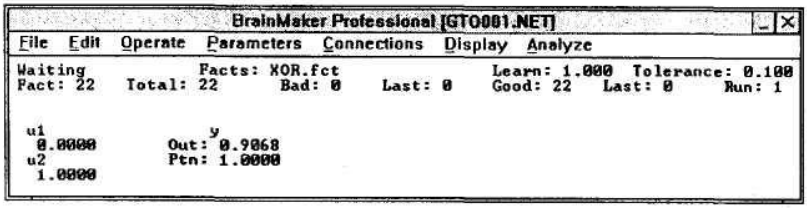



Рис. 3.204**.** Результат тестування тієї ж мережі для $u_1 = 0$, $u_2 = 1$ і $d = 1$.

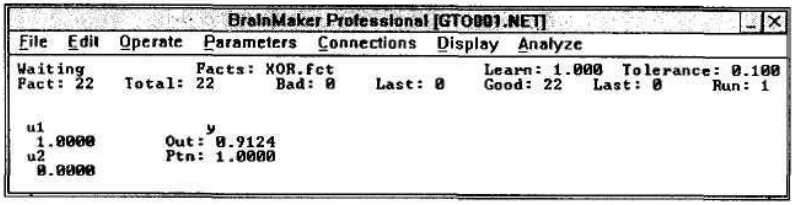



Рис. 3.205**.** Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 0$ і $d = 1$.

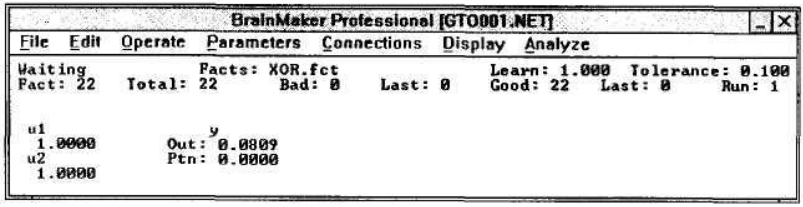



Рис. 3.206. Результат тестування тієї ж мережі для $u_1 = 1$, $u_2 = 1$ і $d = 0$.

Помітимо, що значення похибки $Q$, що розраховується програмою **Evolver**, для мережі з вагами, показаними на рис. 3.202, дорівнює 0,007451. Ця похибка менша значення $Q$, розрахованого для мережі з приклада 3.33 (ваги див. на рис. 3.191). Звичайно, нова мережа ненабагато краща мережі з приклада 3.33. Тепер варто знову застосувати програму **BrainMaker** для перевірки - чи можна поліпшити характеристики мережі, отриманої в результаті виконання програми **GTO.** Очевидно, що спроба донавчання цієї мережі з толерантністю похибки 0,1 нічого не змінить. Однак при завданні рівня толерантності 0,025 ми одержуємо процес, показаний на рис. 3.207 і значення ваг, які представлені на рис. 3.208.

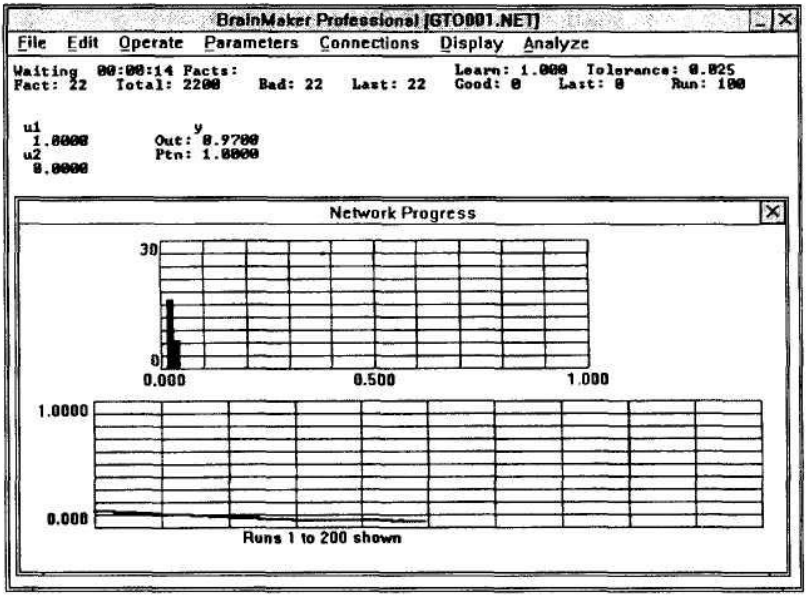



Рис. 3.207. Процес навчання нейронної мережі, який сформовано програмою **GTO**, здійснюваний програмою **BrainMaker** з толерантністю 0,025.

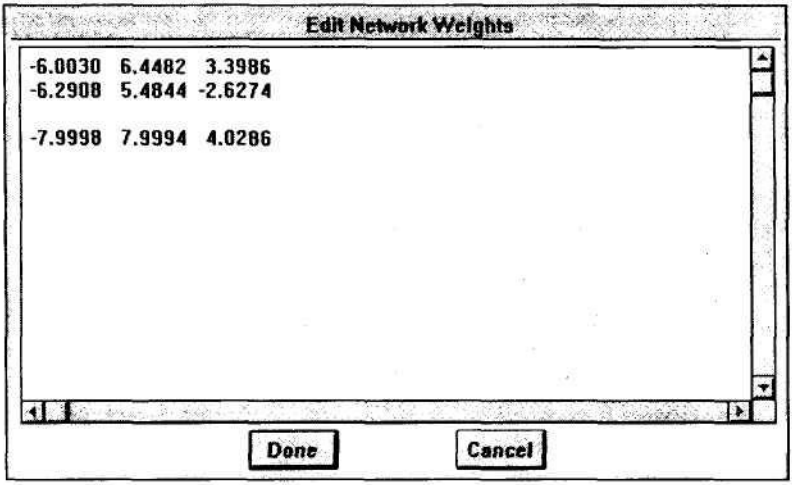


Рис. 3.208. Ваги, отримані в результаті гібридного навчання мережі програмами **BrainMaker, GTO** і повторно **BrainMaker**.

Вихідне значення в для $u_1 = 1$ і $u_2 = 0$ також приведено на рис. 3.207.

Приклад 3.34 ілюструє наступне об'єднання традиційного (градієнтного) алгоритму навчання, реалізованого в програмі **BrainMaker,** з генетичним алгоритмом програми **GTO:**

1) навчання нейронної мережі програмою **BrainMaker;**
2) застосування програми **GTO** до мережі, навченої в п.1;
3) продовження навчання найкращої мережі, отриманої в п.2, програмою **BrainMaker;**

Слід зазначити, що п.2 припускає використання як генетичного алгоритму програми **GTO,** так і відповідної процедури програми **BrainMaker,** яка викликається програмою **GTO.**

Можна запропонувати й інший підхід:

1) використання програми **BrainMaker** тільки для формування мережі зі значеннями ваг, обираними випадковим чином;
2) застосування програми **GTO** для пошуку найкращих ваг;
3) застосування програми **BrainMaker** для завершення навчання.

Другий підхід заснований на обробці програмою **GTO** мережі з випадковими значеннями ваг. При відключенні опції навчання кожної знову формованої мережі програмою **BrainMaker** програма **GTO** викликає її тільки для тестування мережі. У подальшому найкраща зі створених у такий спосіб мереж «донавчається» програмою **BrainMaker** (п. 3). Така методика подібна використовувамшися в прикладах 3.31 і 3.32 - спочатку виконується генетичний, а потім - традиційний алгоритм навчання. У програмі **GTO** додатково можливе навчання кожної знову сформованої мережі (нащадка) за визначену кількість прогонів з наступним її тестуванням. Це приклад підходу, що складається в «донавчанні» нейронних мереж традиційним (зокрема, градієнтним) методом щораз перед оцінюванням пристосованості. Програма **GTO** також надає можливість вибрати один з декількох критеріїв оцінювання нейронних мереж (функцію їхньої пристосованості). Рішення про застосування одного з двох розглянутих гібридних підходів до спільного застосування програм **GTO** і **BrainMaker** залежить від поставленої задачі і, як правило, приймається методом проб і

помилок. Практично неможливо апріорно оцінити - який підхід виявиться кращим для конкретної задачі.

Стосовно до нейронної мережі, призначеної для реалізації логічної системи XOR, найкращі результати можна було б очікувати від застосування другого підходу. Однак з урахуванням обмеженого обсягу цієї роботи ми не будемо розглядати додаткові приклади, що демонструють різні можливості його реалізації з використанням програми **GTO**.

Найбільш важливі зауваження щодо гібридного підходу, що складається в об'єднанні генетичного алгоритму з градієнтним методом навчання нейронних мереж (програма **BrainMaker**) представлені в п. 3.21.1. Програма **GTO** являє собою приклад рівноправного об'єднання обох методів, при якому відповідно до типового циклу еволюції (п. 3.20.7) пристосованість особей популяції розраховується генетичним алгоритмом за результатами навчання нейронних мереж. Існування такої програми, як **GTO**, підтверджує практичне застосування гібридного підходу, що поєднує достоїнства двох оптимізаційних методів: генетичного алгоритму, що легко знаходить точку, близьку до оптимального рішення, і градієнтного алгоритму, що стартує зі знайденої точки і швидко приводить до дійсного оптимуму.

## 3.22. Застосування ГА для автоматичної генерації тестів

При розробці і супроводі програмного забезпечення, значна частина зусиль витрачається на пошук і усунення помилок. Найпоширенішим методом пошуку помилок є тестування, тобто процес виконання програм з метою виявлення помилок. Тут слово «програма» розуміється в широкому змісті, як будь-який запис алгоритму. Зокрема, *програмами є окремі процедури, функції, класи і т.д*. Процес тестування включає виконання деякого набору тестів і аналіз отриманих результатів. ***Тест*** *- це послідовність звертань до тестируємої програми*. ***Результатом виконання тесту*** *є рішення (вердикт) про те, чи відробила програма коректно або некоректно*. Основною характеристикою тестового набору, що визначає якість тестування, є клас можливих помилок у програмі, який даний

тестовий набір здатний знайти. Для ***кількісної*** оцінки якості тестування використовуються різні *метрики тестового покриття*. Для ***якісного тестування*** необхідно побудувати *повний тестовий набір*, тобто набір, що задовольняє деякому *критерієві повноти*. Найчастіше критерій повноти для тестового набору визначають через граничне значення метрики тестового покриття. Побудова повного тестового набору для великих систем вручну може бути вкрай трудомісткою задачею. Автоматизація цього процесу дозволяє істотно знизити витрати на тестування. Існують різні підходи до рішення задачі *автоматичної генерації тестів*. Один з них заснований на застосуванні генетичних алгоритмів. Цей підхід у багатьох випадках дає гарні результати. На жаль, його ефективність істотно залежить від використовуваного критерію повноти. Ціль даного розділу - проаналізувати деякі широко розповсюджені критерії повноти тестового набору на їхню застосовність при використанні генетичних алгоритмів для генерації тестів.

Основні поняття

*Генетичні алгоритми*

Як ми знаємо, генетичні алгоритми - це метод рішення задач оптимізації. У методі використовуються ідеї, почерпнуті з еволюційної біології: спадкування ознак, мутація, природний відбір і кросовер. Визначається множина кандидатів, серед яких шукається рішення задачі. Кандидати представляються у виді списків, дерев або інших структур даних. Загальна схема роботи генетичного алгоритму докладно була описана в початкових розділах цього видання.

Генетичні алгоритми дозволяють вирішувати задачі, для яких не застосовні традиційні методи оптимізації. Однією з областей застосування генетичних алгоритмів є *автоматична генерація тестів для програмного забезпечення.*

*Критерії повноти тестового покриття*

Для тестування програмного забезпечення потрібно створити *репрезентативний набір тестів*, тобто набір, що охоплює всі можливі сценарії роботи системи. Для оцінки репрезентативності тестових наборів використовуються різні критерії повноти тестового покриття.

Нехай *P* - множина програмних систем, *T* - множина тестів, а Σ - множина тестових наборів, тобто множина усіх кінцевих підмножин множини *T*. Тоді задача генерації тестів може бути сформульована в такий спосіб: *для заданої тестуємої системи* $S \in P$ *побудувати тестовий набір* $\sigma \in \Sigma$*, що задовольняє заданому критерієві повноти тестового покриття* $F : P \times \Sigma \to \{\bullet, \perp\}$*, тобто такий набір* $\sigma$*, для якого* $F(S, \sigma) = \bullet$.

Багато критеріїв повноти тестового покриття, що мають практичне застосування, будуються за наступною схемою: для тестуємої системи *S* критерій *F* визначає множину елементів тестового покриття $Q_S^F$. *Елементом тестового покриття можна вважати деякий клас подій, що можуть відбутися в ході роботи тестуємої програмної системи*. По появі в процесі виконання програми елементів тестового покриття і різних їхніх комбінацій можна судити про повноту або якість перевірки, що виконує даний тестовий набір. Наприклад, елементами тестового покриття можуть бути рядки вихідного коду, що виконуються, (відповідні подіям їхнього виконання); ребра графа потоку керування; шляхи в графі потоку керування; логічні вирази, що зустрічаються у вихідному коді і т.п. Крім того, критерій *F* визначає логічну функцію $f : Q_S^F \times T \to \{\bullet, \perp\}$, що приймає значення $f(q, t) = \bullet$, якщо елемент тестового покриття *q* покривається тестом *t*. Тестовий набір σ для системи *S* задовольняє критерієві повноти тестового покриття *F*, якщо кожен елемент тестового покриття з множини $Q_S^F$ покривається хоча б одним тестом з тестового набору σ. Іншими словами:

$$F(S,\sigma) \equiv \forall q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet \quad (*)$$

Приведемо кілька прикладів критеріїв повноти, що згадуються часто, тестового покриття:

- кожен оператор у вихідному коді виконується хоча б один раз;
- кожна гілка графа потоку керування виконується хоча б один раз;
- кожен шлях графа потоку керування виконується хоча б один раз;
- кожний логічний вираз хоча б один раз обчислюється зі значенням «істина» і хоча б один раз - зі значенням «неправда»;
- тестовий набір убиває всіх мутантів із заданого набору.

Помітимо, що всі критерії, приведені як приклади, відповідають раніше викладеній схемі.

*Метрики тестового покриття*

З багатьма критеріями повноти тестового покриття можна зв'язати відповідну метрику тестового покриття. Метрика тестового покриття - це функція виду $M : P \times \Sigma \to R$. Значення цієї функції $M(S,\sigma)$ має сенс числової оцінки того, наскільки добре тестовий набір σ покриває тестуєму систему *S*. Сам критерій при цьому можна записати у виді $M(S,\sigma) \geq \alpha_S$, де $\alpha_S$ - це мінімальне граничне значення метрики *M* для тестуємої системи *S*. Зокрема, для критерія повноти тестового покриття *F*, представимого у виді (*), можна ввести наступну метрику:

$$M^F(S,\sigma) = |\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet\}| \quad (**)$$

Сам критерій при цьому прийме вид:

$$|\{q \in Q_S^F : \exists t \in \sigma: f(q,t) = \bullet\}| \geq |Q_S^F|$$

У деяких випадках, коли не вдається побудувати тестовий набір, що задовольняє такому критерієві повноти тестового покриття, можна використовувати ослаблений критерій:

$$|\{q \in Q_S^F : \exists t \in \sigma: f(q,t) = \bullet\}| \geq \lambda |Q_S^F| \quad (***)$$

Параметр $\lambda \in (0,1]$ указує, яка частка елементів тестового покриття повинна бути покрита тестовим набором. Приведемо кілька прикладів метрик тестового покриття, що згадуються часто:

- кількість покритих (виконаних хоча б один раз) операторів у вихідному коді;
- кількість покритих гілок графа потоку керування;
- кількість покритих шляхів графа потоку керування;
- кількість розпізнаних мутантів (версій тестируємої системи зі штучно привнесеними помилками).

Усі ці метрики можуть бути представлені у виді (**). Докладний опис цих і інших, використовуваних на практиці, метрик повноти тестового покриття можна знайти в літературі.

## Генетичний алгоритм генерації тестів

Розглянемо наступну задачу генерації тестів:

**Задача 1**. Для заданої тестової системи *S* побудувати тестовий набір $\sigma \in \Sigma$, що задовольняє критерієві (***).

Для побудови генетичного алгоритму рішення цієї задачі необхідно визначити:

- множину кандидатів;

- структуру представлення кандидатів;
- функцію, яка оцінює (оцінюється);
- оператор кросовера;
- оператор мутації;
- умова останову.

### *Найпростіший алгоритм*

Розглянемо найпростіший генетичний алгоритм для рішення задачі 1. Як множину кандидатів візьмемо множину Σ; як оцінну функцію візьмемо метрику тестового покриття $M^F(S,\sigma)$ для заданої тестируємої системи *S*. Умовою останова буде наявність у поточному поколінні рішення σ, що задовольняє критерієві (***). Структуру представлення кандидатів, а також оператори кросовера і мутації ми поки уточнювати не будемо. Помітимо, що такий алгоритм допускає ситуацію, у якій критерій (***) не виконується ні для одного тестового набору з поточного покоління, але, проте, виконується для деякого об'єднання тестових наборів з поточного і попереднього поколінь. Іншими словами, усі тести, які необхідні для побудови рішення, уже знайдені, але саме рішення ще не побудовано. У цій ситуації алгоритм не здатний ефективно побудувати шукане рішення, цілеспрямовано об'єднавши підходящі тести з різних тестових наборів. Причина проблеми в тім, що при побудові алгоритму не використовувалася наявна інформація про структуру критерію (***). Помітимо також, що кожне наступне покоління тестів формується шляхом застосування операторів кросовера і мутації до тестів з попереднього покоління. Якщо в попереднім поколінні не було ні одного тесту, що покриває деякий елемент тестового покриття *q*, то в наступному поколінні такий тест може з'явитися тільки як результат кросовера або мутації тестів, що не покривають *q*. Як би ми не визначали оператори кросовера і мутації, немає ніяких підстав думати, що одержати таким способом тест, що покриває *q*, простіше, ніж при цілком випадковій генерації.

З цих зауважень випливає, що ефективність даного генетичного алгоритму, узагалі говорячи, не вища, ніж у цілком випадкового алгоритму генерації тестів.

### *Цілеспрямований пошук*

З огляду на структуру критерію (***), з задачі 1 можна виділити наступну підзадачу:

**Задача 2**. Для заданої тестової системи *S* і заданого елемента тестового покриття *q*, побудувати тест $t \in T$, що задовольняє умові $f(q,t) = \bullet$ .

Для рішення вихідної задачі 1, досить рішити задачу 2 для $n \geq \lambda |Q_S^F|$ попарно різних елементів тестового покриття $q_1, q_2, \cdots, q_n \in Q_S^F$, тобто побудувати тести $t_1, t_2, \cdots, t_n$ такі, що

$$\begin{cases} f(q_1, t_1) = \bullet , \\ f(q_2, t_2) = \bullet , \\ \cdots \\ f(q_n, t_n) = \bullet . \end{cases}$$

Рішенням задачі 1 будем ножина $t_1, t_2, \cdots, t_n$.

Розглянемо генетичний алгоритм рішення задачі 2. Як множину кандидатів візьмемо множину тестів *T*. Умова останова: у поточній популяції присутній тест *q* такий, що $f(q,t) = \bullet$ . Оцінна функція $m_q : T \to R$ кожному тестові *t* ставить у відповідність числову міру $m_q(t)$ того, наскільки тест *t* близький до того, щоб покрити елемент тестового покриття *q*. При цьому оцінна функція $f_q$ досягає свого максимального значення на тих

і тільки на тих тестах, які задовольняють умові $f(q,t)=\bullet$ . Іншими словами:

$$f(q,t)=\bullet \Leftrightarrow m_q(t)=\max_{t\in T} m_q(t) \qquad (****)$$

Зокрема, як оцінну функцію можна використовувати наступну функцію, що задовольняє умові (****):

$$m_q(t)=\begin{cases} 0, \text{ при } f(q,t)=\perp, \\ 1, \text{ при } f(q,t)=\bullet \end{cases}.$$

У такій оцінній функції вважається, що всі тести, які не покривають елемент тестового покриття *q*, однаково далекі від того, щоб покрити елемент *q*. При використанні цієї оцінної функції ефективність генетичного алгоритму буде не вище, ніж при випадковому пошуку. Приклади більш ефективних оцінних функцій для деяких метрик повноти тестового покриття можна знайти в літературі.

## Оцінні функції

У цьому розділі докладно розглядаються три відомих критерії повноти тестового покриття, і для кожного з них пропонується оцінна функція.

### *Покриття операторів вихідного коду*

Тестовий набір задовольняє критерієві покриття операторів вихідного коду, якщо при виконанні цього тестового набору кожен оператор вихідного тексту програми виконується хоча б один раз. Елементами тестового покриття в даному випадку є

оператори вихідного тексту. Для заданого оператора *q* значення оцінної функції $m_q(t)$ тим більше, ніж ближче тест *t* до тесту, що покриває оператор *q*. Для побудови оцінної функції розглянемо граф потоку керування тестуємої системи *S*. Вершинами графа є оператори вихідного коду, тобто множина $Q_S^F$. У графі існує ребро, яке йде з вершини $q_1$ у вершину $q_2$ тоді і тільки тоді, коли оператор $q_2$ може бути виконаний безпосередньо після оператора $q_1$. Нехай $\Pi(q,t)$ - це множина всіх елементів *q'* з $Q_S^F$, для яких виконуються наступні умови:

- існує шлях у графі потоку керування ведучий з *q'* у *q* або *q' = q*;
- $f(q',t) = \bullet$ .

Позначимо через *dist(q', q)* довжину найкоротшого шляху в графі потоку керування, що веде з *q'* у *q* (*dist(q, q)*≡0). Тоді оцінну функцію можна визначити в такий спосіб:

$$-m_q(t) = \min_{q' \in \Pi(q,t)} dist(q',q) \qquad (*****)$$

Вираз, що стоїть праворуч, визначає, за яку мінімальну кількість переходів можна добратися до елемента покриття *q* від уже покритих елементів з множини $Q_S^F$. Функція $m_q(t)$ приймає значення 0 на тих і тільки тих тестів, що покривають елемент *q*. Помітимо, що

$$\begin{cases} m_q(t) = 0 & \Leftrightarrow \quad f(q,t) = \bullet \ ; \\ m_q(t) < 0 & \Leftrightarrow \quad f(q,t) = \perp . \end{cases}$$

***Покриття галузей потоку керування***

Тестовий набір задовольняє критерієві покриття галузей потоку керування, якщо при виконанні цього тестового набору

керування хоча б один раз проходить по кожному ребру графа потоку керування. Помітимо, що будь-який тестовий набір, що задовольняє цьому критерієві, задовольняє також і критерієві покриття операторів вихідного коду. Зворотне твердження, однак, невірне. Елементами тестового покриття є переходи в графі потоку керування. З кожним переходом у графі потоку керування можна зв'язати умову, при якій цей перехід може бути виконаний. Перехід від оператора *q* до оператора *r*, з яким зв'язана умова *p*, позначимо як $q \xrightarrow{p} r$. Для виконання переходу $q \xrightarrow{p} r$ необхідно і досить, щоб був виконаний оператор *q*, і щоб після цього умова *p* обернулася в істину. Відповідно, для тестів, що не покривають оператор *q*, у якості оцінної підходить функція $m_q$, визначена рівнянням (*****), тому що істинність умови *p* для оцінки таких тестів ролі не грає. Для тестів, що покривають оператор *q*, функція $m_q(t)$ обертається в 0. Для таких тестів оцінна функція повинна визначати, наскільки близький тест до тесту, для якого після виконання оператора *q* буде вірна умова *p*. Таким чином, у загальному виді оцінну функцію для критерію покриття галузей потоку керування можна визначити в такий спосіб:

$$m_{q \xrightarrow{p} r}(t) = \begin{cases} m_q(t), \text{при } f(q,t) = \perp, \\ \mu_{q,p}(t), \text{при } f(q,t) = \bullet \end{cases}.$$

Значення функції $\mu_{q,p} : T \to R^+$ тим більше, ніж ближче заданий тест до тесту, у якому умова *p* виконується після виконання оператора *q*. При цьому функція $\mu_{q,p}(t)$ досягає свого максимуму на тих і тільки тих тестах, у яких після виконання оператора *q* виконується умова *p*, тобто тих, котрі покривають перехід $q \xrightarrow{p} r$.

Функцію $\mu_{q,p}(t)$ можна визначати по-різному в залежності від характеру умови *p*. Якщо умова має форму простої (не)рівності $x \Diamond y$, де « $\Diamond$ » позначає одне з відношень «<», «>», «=», «≤» або «≥», то для визначення функції $\mu_{q,p}(t)$ можна використовувати значення $|x-y|$, наприклад, у такий спосіб:

$$\mu_{q,p}(t) = \begin{cases} 2, \text{при} x \Diamond y, \\ e^{-|x-y|}, \text{при} \neg(x \Diamond y). \end{cases}$$

Якщо умова являє собою кон'юнкцію $p = p_1 \wedge p_2 \wedge \ldots \wedge p_n$, то як значення функції $\mu_{q,p}(t)$ можна взяти кількість членів цієї кон'юнкції, що приймають значення «істина». У загальному випадку ефективно визначити функцію $\mu_{q,p}(t)$ важко.

***Покриття шляхів потоку керування***

Тестовий набір задовольняє критерієві покриття шляхів потоку керування, якщо його виконання хоча б один раз проходить по кожному можливому шляху в графі потоку керування який веде від точки входу до точки завершення роботи. Цей критерій сильніший критерію покриття галузей потоку керування. Кожен шлях являє собою послідовність переходів $R = \tau_1, \tau_2, \ldots, \tau_n$, де $\tau_i$ має вигляд $q_i \xrightarrow{p_i} q_{i+1}$, при 1≤*i* ≤*n*. Упорядкованою підмножиною шляху $\tau_1, \tau_2, \ldots, \tau_n$ назвемо послідовність $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ таку, що $1 \le i_1 < i_2 < \ldots < i_m \le n$. Помітимо, що в упорядкованій підмножині шляху кінцевий оператор переходу може не збігатися з початковим оператором наступних за ним переходу.

Нехай є два шляхи $R' = \tau'_1, \tau'_2, \ldots, \tau'_l$ і $R'' = \tau''_1, \tau''_2, \ldots, \tau''_k$, і нехай

$$\begin{cases} \tau'_{i_1} = \tau''_{j_1}, \\ \tau'_{i_2} = \tau''_{j_2}, \\ \quad \ldots \\ \tau'_{i_m} = \tau''_{j_m}, \end{cases}$$

причому $1 \le i_1 < i_2 < \ldots < i_m \le l$ і $1 \le j_1 < j_2 < \ldots < j_m \le k$. Тоді шляхи $R'$ і $R''$ мають загальну упорядковану підмножину розміру *m*.

Позначимо через *length(R)* довжину шляху *R*, а через $common(R', R'')$ - максимальний розмір загальної упорядкованої підмножини шляхів $R'$ і $R''$. Визначимо оцінну функцію для критерію покриття шляхів потоку керування в такий спосіб:

$$-m_R(t) = length(R) + length(path(t)) - 2 \cdot common(R, path(t)).$$

Тут *path(t)* - це шлях, по якому проходить керування при виконанні тесту *t*. Значення в правій частині дорівнює кількості переходів у шляхах *R* і *path(t)*, що не входять у максимальну загальну упорядковану підмножину цих шляхів. Воно дорівнює 0 тоді і тільки тоді, коли шляхи *R* і *path(t)* збігаються.

На закінчення відзначимо, що застосування генетичних алгоритмів для генерації тестів висуває додаткові вимоги до використовуваних критеріїв повноти тестового покриття. Це викликано тим, що критерій повноти використовується не тільки для оцінки якості згенерованих тестів, але і безпосередньо в процесі генерації для оцінки близькості отриманих тестів до

потрібних результатів. Таким чином, потрібно мати оцінну функцію, що дозволяє виміряти цю близькість, визначити, наскільки перспективними є вже побудовані тести з погляду їхнього використання як основу для побудови нових тестів. Крім того, потрібно мати на увазі, що тривіальні рішення - функції виду «покрито - 0, не покрито - 1» - працюють дуже погано. Для критеріїв, зв'язаних з покриттям тих або інших шляхів у коді програми, удається побудувати досить зручні оцінні функції, засновані на кількості непокритих дуг у шляху, який потрібно покрити.

# Література

1. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. — 1-е. — М.: Гаряча лінія - Телекому, 2001. — С. 382. — ISBN 5-93517-031-ПРО
2. В. А. Терехов, Д. В. Єфімов, И. Ю. Тюкин Нейромережні системи керування. — 1-е. — Высшая школа, 2002. — С. 184. — ISBN 5-06-004094-1
3. Ф. Уоссермен. Нейрокомп'ютерна техніка. Теорія і практика. — 1-е. — М.: Мир, 1992. — С. 240. — ISBN 5-03-002115-9
4. Саймон Хайкин. Нейроні мережі: повний курс = Neural Networks: A Comprehensive Foundation. — 2-е. — М.: «Вильямс», 2006. — С. 1104. — ISBN 0-13-273350-1
5. Роберт Каллан. Основні концепції нейроних мереж = The Essence of Neural Networks First Edition. — 1-е. — «Вильямс», 2001. — С. 288. — ISBN 5-8459-0210-X
6. Л.Н. Ясницкий. Введення в штучний інтелект. — 1-е. — Издательский центр "Академия", 2005. — С. 176. — ISBN 5-7695-1958-4
7. Г. К. Вороновский, К. В. Махотило, С. Н. Петрашев, С. А. Сергєєв. Генетичні алгоритми, штучні нейроні мережі і проблеми віртуальної реальності. — Замовне. — Х.: ОСНОВА, 1997. — С. 112. — ISBN 5-7768-0293-8
8. Д. Рутковская, М. Пилиньский, Л. Рутковский. Нейронные сети, генетические алгоритмы и нечеткие системы —1-е. — М.: Гарячая линия – Телеком, 2007. — С. 384. — ISBN 5-93517-103-1

Науково- практичне видання

**Кононюк Анатолій Юхимович**

**Нейроні мережі і генетичні алгоритми**

Авторська редакція

---

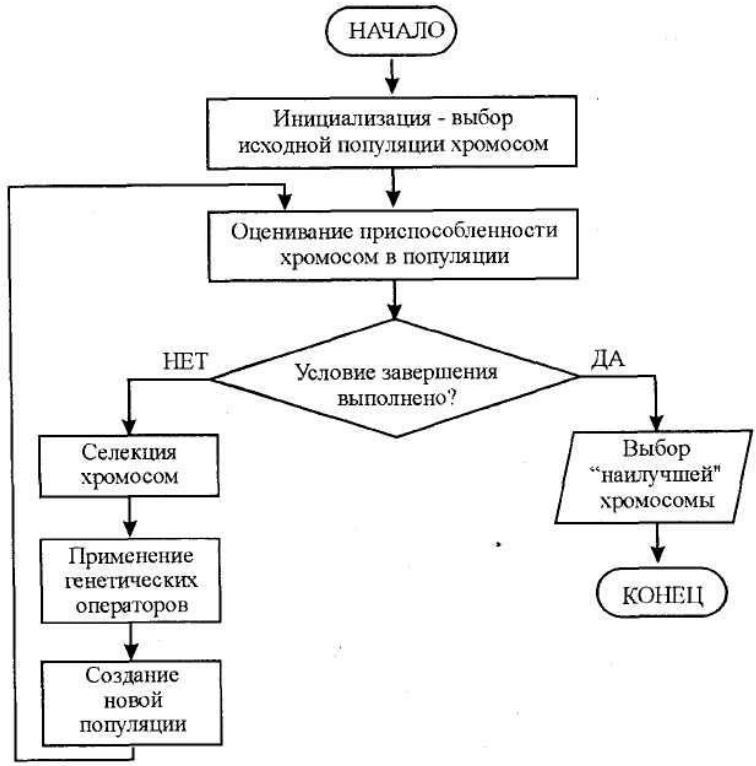
Видавництво "Корнійчук", 04116 , Київ-116, В.Василевської 15/15
відоцтво про внесения до Державного реєстру суб'єктів
видавничої справи №424 від 18.04.2001

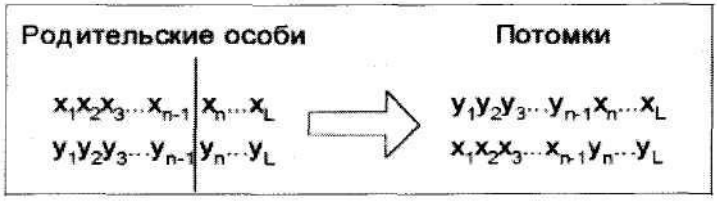
---

Підп. до друку Формат 60х48 1/16
Папір офсетний Тиражування
Наклад 300 прим. Замовлення №

---

**СПД Пачковський В.М.**
**Т. 237-60-26**


НАЧАЛО
Инициализация - выбор исходной популяции хромосом
Оценивание приспособленности хромосом в популяции
Условие завершения выполнено?
НЕТ
ДА
Селекция хромосом
Применение генетических операторов
Создание новой популяции
Выбор "наилучшей" хромосомы
КОНЕЦ


Родительские особи
Потомки
$x_1x_2x_3\cdots x_{n-1}|x_n\cdots x_L$
$y_1y_2y_3\cdots y_{n-1}|y_n\cdots y_L$
$y_1y_2y_3\cdots y_{n-1}x_n\cdots x_L$
$x_1x_2x_3\cdots x_{n-1}y_n\cdots y_L$