ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ'ЯЗКУ

О.П. Гожий

ВСТУП В НЕЙРОННІ МЕРЕЖІ

Методичні вказівки

Одеса

ДУІТЗ

2024

УДК _51

Автор:
Гожий О.П. д.т.н., проф. кафедри Комп'ютерних наук Державного університету інтелектуальних технологій і зв'язку;

Рецензенти:
Калініна І.О., д.т.н., доц. кафедри інтелектуальних інформаційних систем Чорноморського національного університету ім. П.Могили.
Сідень С.В., к.т.н., в.о. зав. каф. радіоелектронних систем і технологій Державного університету інтелектуальних технологій і зв'язку.

Рекомендовано до друку Навчально-методичною Радою
Державного університету інтелектуальних технологій і зв'язку
(протокол №_3__ від _13 грудня_ 2024 р.)

Гожий О.П. Вступ в нейронні мережі : методичні вказівки. Одеса: ДУІТЗ, 2024. 49 с.

Методичні вказівки "Вступ в нейронні мережі" містять завдання та теоретичні відомості до кожної практичної роботи. Вказівки містять перелік питань, які мають бути засвоєні для успішного виконання робіт що має сприяти цілеспрямованому вивченню та поглибленому засвоєнню необхідного матеріалу.

Вступ

Нейронні мережі почали активно поширюватися з 80 років 20 сторіччя. Вони дозволяють вирішувати складні завдання обробки даних. Нейромережеві методи було названо однією з ключових технологій

японської національної програми «Обчислення у світі», яка у 1992 року змінила програму створення комп'ютерів п'ятого покоління.

Нейронні мережі названі так тому, що їхня архітектура певною мірою імітує побудову біологічної нервової тканини з нейронів у мозку людини. І хоча вивченню роботи людського мозку – тисячі років, перші спроби апаратного відтворення процесу мислення розпочалися з появою сучасної електроніки. Перший крок був зроблений у 1943 році з виходом статті нейрофізіолога *Уоррена Маккалоха* та математика *Уолтера Піттса* про роботу штучних нейронів та представлення моделі нейронної мережі на електричних схемах.

У 50-х роках з'явилися перші програмні моделі нейромереж. В 1959 *Бернард Відров і Марсіан Хофф* розробили модель MADALINE, вона діяла як адаптивний фільтр, що усуває луна в телефонних лініях. До речі, ця нейромережа досі перебуває у комерційній експлуатації. Але в той час потенціал нейронних мереж був сильно перебільшений через обмежені можливості електроніки, і незабаром фінансування робіт зі створення нейромереж було згорнуто. Період спаду тривав до початку 80-х років, коли почалася розвиватися сучасна обчислювальна техніка.

Біологічний нейрон

Нейронні мережі виникли як результат моделювання біологічних процесів. Те, що нервова система складається з багатьох окремих клітин, вчені довели на початку XX століття. Ці клітини назвали нейронами. Нейрони обмінюються інформацією з безліччю своїх сусідів, ближніх та далеких.

Кожен нейрон складається з *соми* та відростків — *аксонів* та *дендритів*. *Сома* - це клітина діаметром від 3 до 100 мкм, що містить ядро та інші клітинні складові, занурені в цитоплазму. *Аксони* – це передавачі сигналів, а *дендрити* – сильно розгалужені відростки – приймачі сигналів.

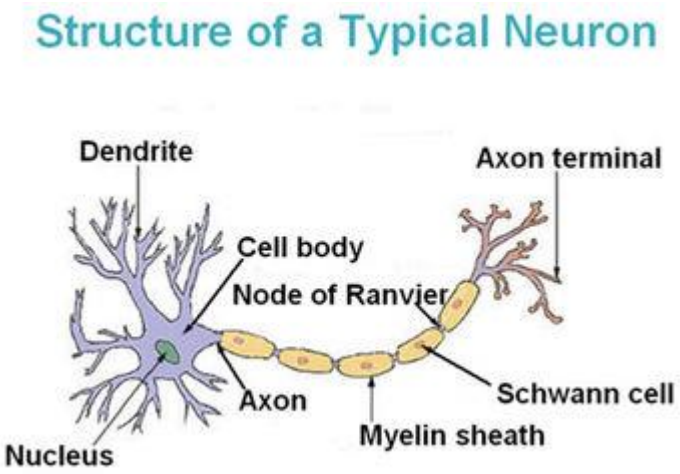


Рис.1 Біологічний нейрон

node - вузол,

sheath - оболонка

Schwann cell - клітини Шванна

http://en.wikipedia.org/wiki/Neuron

Аксони і *дендрити* розкидані по всьому тілу. Всього у людини близько 120 млрд. нейронів, причому переважна їхня кількість — де? - У головному мозку, в ньому переважна більшість нейронів. Повідомлення, які може надсилати нейрон, дуже прості. Він може знаходитися всього у двох положеннях: включеному, що передає сигнал, та вимкненому. Жодних проміжних положень йому не дано. І тому імпульси нейронів зорової системи нічим не відрізняються від імпульсів нервових клітин носа, шкіри чи вуха. Мозок сприймає їхні повідомлення як образи лише тому, що знає, звідки вони надійшли. Тому якщо постукати пальцем по оці, мозок розшифрує цей сигнал як світло. Звідси якраз і вираз „іскри з очей“.

Отримавши зоровий, звуковий, дотикальний сигнал, нейрон передає його в нервовий центр, а звідти після аналізу повідомлення посилається команда у відповідь до робочого органу. Біологи називають цей ланцюжок рефлекторною дугою. Найпростіший приклад - кінчик пальця відчув, що гаряче, і людина смикає руку.

Імпульс рухається *аксоном,* це електричний сигнал. Деякі аксони мають величезну довжину. *Аксон* у пальці жирафа ледве сягає 1/10 мм у

поперечнику, а проходить відстань у кілька метрів до закінчення у спинному мозку. Пучки аксонів складають ті нерви, які можна побачити неозброєним оком.

Від одного нейрона до іншого інформація передається через місце з'єднання, яке називається синапсом. *Синапс* - це область контакту з наступним нейроном, тут відбувається хімічна реакція, яка і передає імпульс дендриту. Тобто. по *аксону* йде електричний сигнал, а *синапс* — хімічна реакція. *Синапси* передають сигнал лише одному напрямку.

Нервові імпульси поширюються із швидкістю від 50 см/с до 120 м/с. 120 м/с – це 400 км/год! Найшвидше йдуть сигнали від скелетних м'язів, а сигнали болю зі швидкістю лише 1 м/с.

Штучний нейрон

Базовий модуль нейронних мереж – штучний нейрон – моделює основні функції природного нейрона.

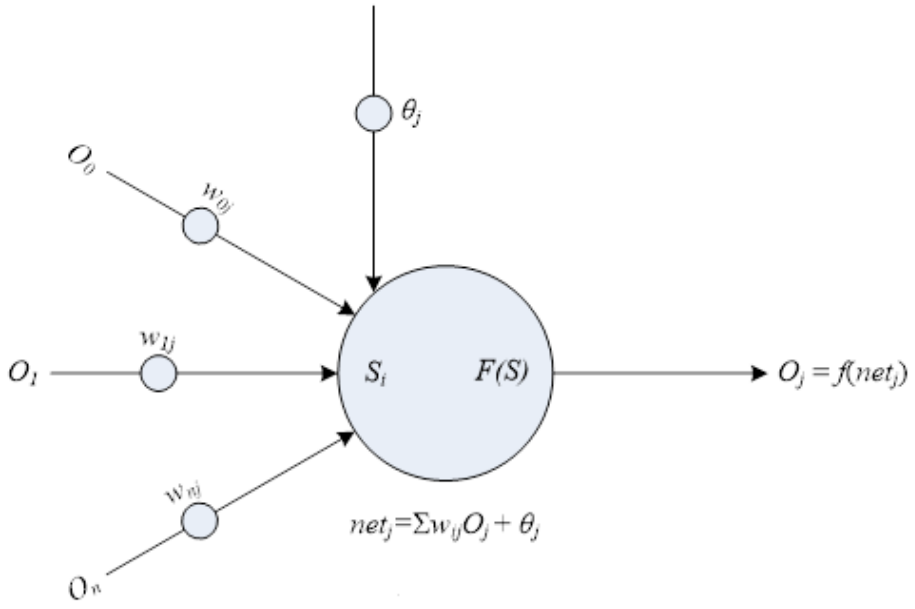


Рис.2 Схема штучного нейрона

Тут

x_0, x_1К - Вхідні сигнали (ніби з дендритів);

w_0, w_1К - Вагові коефіцієнти;

θ_j - вага зміщення, на вході одиниця.

Вхідні сигнали, зважені ваговими коефіцієнтами складаються, проходять через передатну функцію, яка генерує результат, який, власне, виводиться. (Як би *аксоном*).

Математична модель нейрона, її таки запропонували *Маккаллок і Піттс*, має вигляд:

$$\begin{cases} net_j = \sum_i w_{ij} O_i + \theta_j \\ O_j = f\left(net_j\right) \end{cases}$$

Де

net_j - вхідний сигнал мережі для j-го нейрона;

θ_j - вага зміщення;

w_{ij} - Вагове значення зв'язку;

O_i - Вихідний сигнал i-го нейрона;

$f(x)$ — функція виходу.

Для простоти зазвичай розглядають лише три типи функції виходу:

1) функція знака, або сигнум-функція: $f(x) = \mathrm{sign}(x) = \begin{cases} 1, & при\ x \geq 0; \\ 0, & при\ x < 0. \end{cases}$

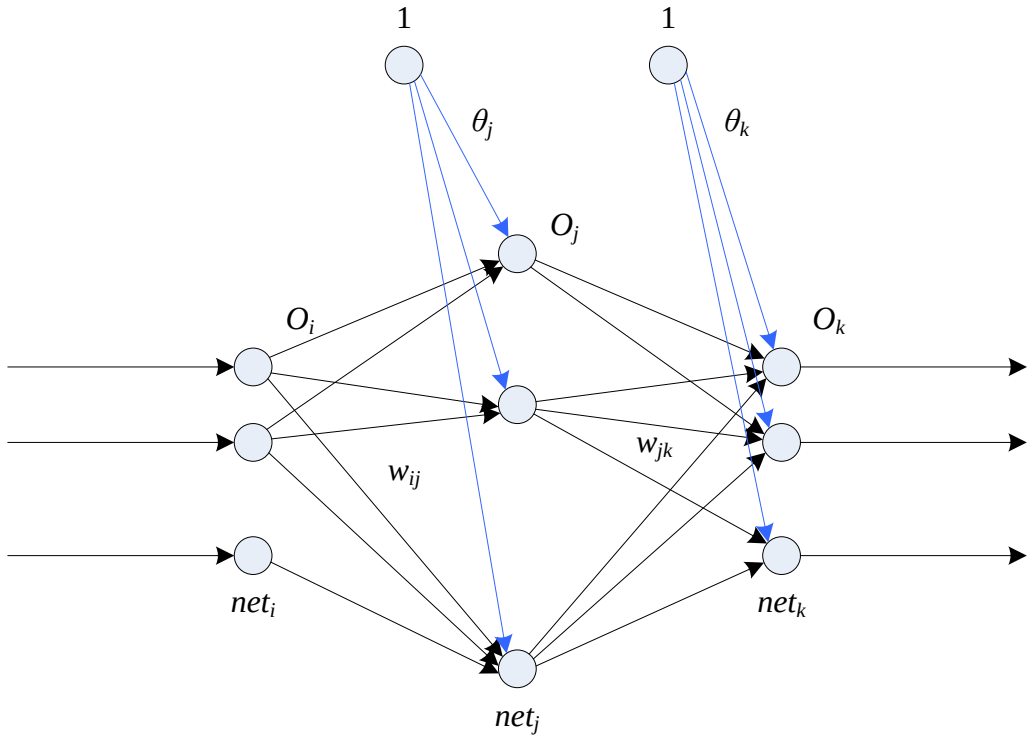
2) лінійная функція: $f(x) = x$

3) сигмоідальна функція: $f(x) = \mathrm{sigmoid}(x) = \dfrac{1}{1 + \exp\left(-\dfrac{x}{T}\right)}, \quad T > 0$

У 1974 році Вербос запропонував концептуальні основи підходу до побудови нейроних мереж, який отримав назву алгоритму зворотного розповсюдження помилки.

Алгоритм зворотного розповсюдження

Алгоритм зворотного поширення - *backpropagation algorithm* - ґрунтується на багатошаровій нейронній мережі. У ньому використовується щонайменше трьох шарів нейронів. Для простоти ми маємо тришарові мережі і будемо розглядати. Три шари: вхідний, прихований та вихідний.



Таким чином на вихідному шарі:

$$\begin{cases} net_k = \sum_j w_{jk} O_j + \theta_k \\ O_k = f\left(net_k\right) \end{cases}$$

На прихованому шарі:

$$\begin{cases} net_j = \sum_i w_{ij} O_i + \theta_j \\ O_j = f\left(net_j\right) \end{cases}$$

де $f\left(x\right) = \text{sigmoid}\left(x\right) = \dfrac{1}{1+\exp\left(-x\right)}$

Процедура навчання включає представлення набору пар вхідних та вихідних образів. Тобто. ми маємо показати нашій нейромережі, що має бути на виході, коли на вході якась комбінація сигналів. Спочатку нейронна мережа на основі вхідного образу створює власний вихідний образ, а потім порівнює його з цільовим вихідним чином. Якщо відмінностей між фактичним та цільовим образами немає, то навчання не відбувається. В

іншому випадку ваги зв'язків змінюються таким чином, щоб різниця зменшилася. Мірою відмінності при цьому є функція квадрата помилки, яка задається таким чином. Завдання алгоритму - досягти мінімуму цієї помилки, керуючи ваговими коефіцієнтами. Можна скористатися градієнтним методом якнайшвидшого спуску.

Більш детально алгоритм буде розглянуто в наступному розділі.

АЛГОРИТМИ НАВЧАННЯ ШНМ

Теоретичні відомості

Розглянемо навчання нейронних мереж на прикладі задачі прогнозування курсу дорогоцінних металів. Задача відноситься до классу задач прогнозування часових рядів. Справді, існує часовий ряд $\{t_1, t_2, \ldots, t_n, \ldots\}$ статистичних даних безпосередньо курсу дорогоцінного металу та факторів, що впливають на цей курс (у даній роботі таким фактором взято курс доллара). З нього вибирається кінцеве число членів $t_1, \ldots, t_n, \ldots$Потрібно скласти таку функцію $f(x_1, \ldots x_n)$, $k \le n$, щоб для неї виконувалося наступне:

$$\left| f(t_i, \ldots, t_{i+k-1}) - t_{i+k} \right| \text{ мало для усіх } 1 \le i \le n-k,$$

тобто функція f(її значення) є «гарним» (у деякому змісті) прогнозом значення t_{i+k} за значенням часового вікна $\{ t_1, \ldots, t_{i+k-1} \}$.

Оскільки критерій якості прогнозу може бути різним, то в даному випадку він не обмовляється. Кінцевою метою задачі є її використання для прогнозу невідомих значень $t_{n+1}, t_{n+2} \ldots$, оскільки передбачається, що якщо для великої кількості часових вікон знайдений «гарний» прогноз, то він залишається «гарним» і для невеликої кількості часових вікон, що виходять за границі відомих значень.

Використовувати знайдену функцію $f()$ передбачається так:

$$\hat{t}_{n+1} = f(t_{n-k+1},\ldots,t_n);$$

$$\hat{t}_{n+2} = f(t_{n-k+2},\ldots,t_n,\hat{t}_{n+1});$$

$$\hat{t}_{n+3} = f(t_{n-k+3},\ldots,t_n,\hat{t}_{n+1},\hat{t}_{n+2});$$

й так далі, де $\hat{t}_{n+1},\hat{t}_{n+2}$ - величини, прогнозовані за допомогою циклічної дії функції прогнозу на зміщувані на один крок часові вікна.

Розглянемо функцію прогнозу, яку будемо подавати наступним чином:

$y^L = f_0(x_1^0,\ldots,x_{n_0}^0)$, *де* $x_i^0 \in [0,1]$, $y \in [0,1]$

На рисунку 1 наведений алгоритм обчислення *y.* Кожний вузол графа приймає на вхід безліч вихідних значень попереднього шару графа та передає результат обчислень на наступний шар таким чином, що:

$a_k^L = \sum_{i=1}^{n_l} \omega_{ik}^{L-1} y_i^{L-1}$, де ω_{ik}^{L-1} - коефіцієнт зв'язку між *i*-м вузлом (*L-1*)-го шару чи з *k*-м вузлом *i*-го шару, а $y^{L-1} = f\left(a_i^{L-1}\right)$ – результат обчислення у вузлі номер *i* шару номер *L-1*.

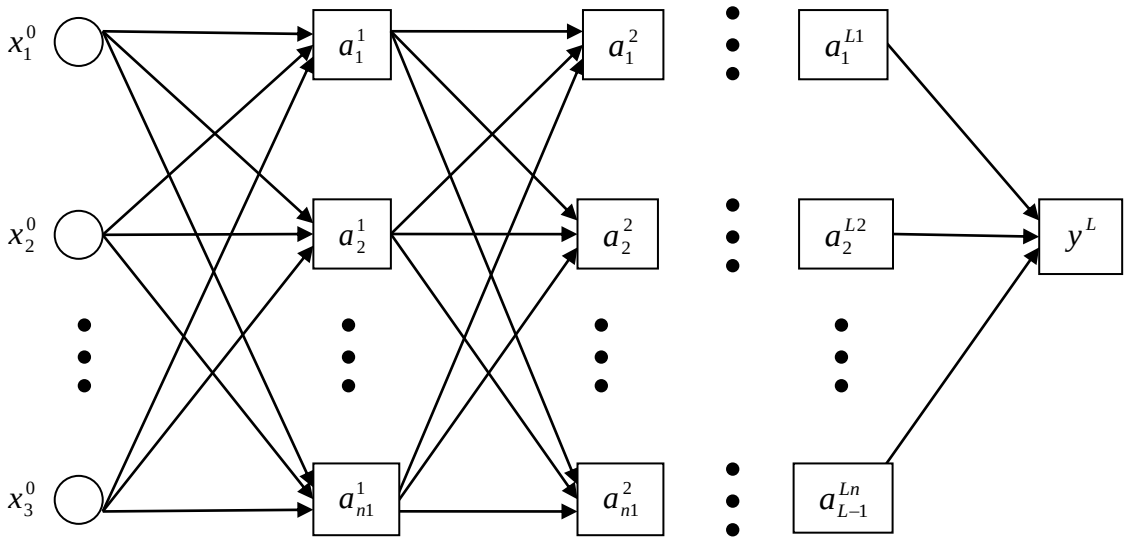


Рисунок 2.1 – Алгоритм обчислення функції прогнозу

Таким чином, нейронна мережа є однорідна композиція шарів елементарних обчислювачів. У якості елементарного обчислювача можна брати наступні функції:

1) $f(x) = \frac{1}{2} + \frac{1}{\pi} arctg(\sigma x)$, $f(x) \in [0,1]$

2) $f(x) = \frac{1}{\pi} arctg(\sigma x)$, $f(x) \in [-1,1]$

3) $f(x) = \frac{1}{1+e^{-\sigma x}}$, $f(x) \in [0,1]$

4) $f(x) = \frac{2}{1+e^{-\sigma x}} - 1 = \frac{1-e^{-\sigma x}}{1+e^{-\sigma x}}$, $f(x) \in [-1,1]$

З боку тільки функції прогнозу абсолютно все рівно з якого інтервалу брати значення [0;1] чи [-1;1]. Звичним прийнято брати [0;1], але експерименти показують, що в деяких випадках [-1;1] є доцільнішим.

Зауваження 1. Параметр σ насправді ролі не грає, тому що його зміна приводить тільки до відповідного перемасштабування всіх коефіцієнтів зв'язку ω_{ik}^{L} та нічому більш (далі σ = 1).

Зауваження 2. Запис n_L і n^L означають одне й те саме – кількість вузлів у шарі L і застосовуються для зручності запису.

Зауваження 3. Якщо величини елемента часового ряду не лежать у $R(f)$ – безлічі значень $\{f(x) \mid x \in R\}$, то всі елементи ряду перетворяться застосуванням до них функції f, тобто $\{t_1, t_2, \ldots, t_n\} \rightarrow \{f(t_1), f(t_2), \ldots, f(t_n)\}$ і задача прогнозу звужується до знов отриманого часового ряду.

Оскільки величини елементів часового ряду можуть змінюватися безперервно (на відрізку), то й вся мережа повинна постійно знаходитися в безперервному режимі, щоб реалізувати безперервну за всіма аргументами функцію $f_0(.)$. Тому вважається небажаним входження вузла (елементарного обчислювача) до режиму «насичення», тобто потрапляння a_i^L до такої точки, де $f'(a_i^L)$ - перша похідна – досить мала.

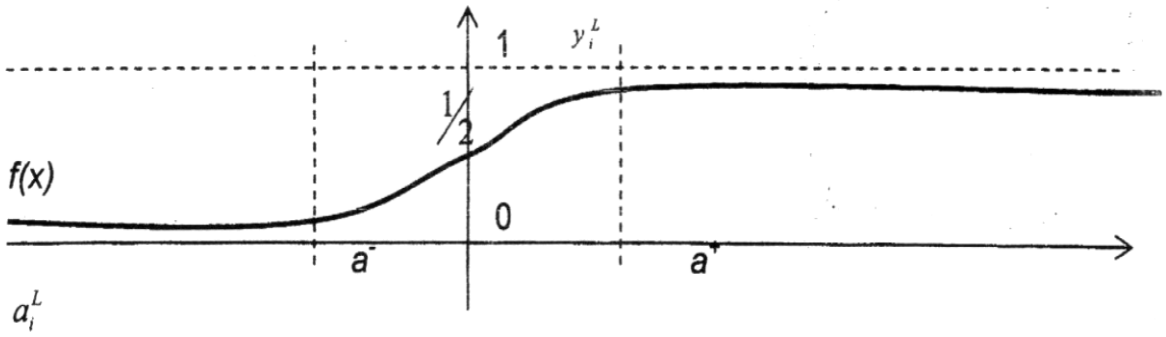


Рисунок 2.2 – Приклад елементарного обчислювача

Оскільки розглядаються цілком визначені функції, то це означає, що величина a_i^L не повинна сильно відхилятися від нуля.

Розглянемо обмеження $a^- \le a_i^L \le a^+$, $|a^-| = |a^+|$. Якщо $|a^+|$ мала, то *f(x)* досить точно апроксимується лінійною функцією.

Навчання мережі – це процес пошуку таких коефіцієнтів зв'язку, що нейронна мережа реалізує «гарну» функцію прогнозу. Оскільки мережа реалізує безперервну функцію, то вона може прогнозувати так само як деяких нормований ряд, так і перетворений, наприклад, коли всі значення зменшенні в k разів. Досить зручно елементарний обчислювач є функцією вигляду (4), оскільки в цьому випадку, якщо усі входи дорівнюють нулю, то й вихід також буде дорівнювати нулю. Разом з цією функцією потрібно дотримуватися ще однієї умови (для більшої надійності обчислень) – середня величина часового ряду повинна дорівнювати нулю. Наслідком цих тверджень є наступне: якщо величина a_i^L за модулем мала ($|a_i^L| \le 0{,}1$), то досить «гарним» наближенням елементарного обчислювача буде $f(x) = \frac{1}{2} + \frac{1}{4}x$, а елементарного обчислювача (4) - $f(x) = \frac{1}{2}x$.

2.2 Навчальний алгоритм генетичної селекції

Генетичні алгоритми навчання штучних нейронних мереж базуються на теоретичних досягненнях синтетичної теорії еволюції, яка враховує мікробіологічні механізми спадкування ознак у природних і штучних

популяціях організмів, а також на накопиченому людському досвіді в селекції тварин і рослин. При цьому механізмами зміни визначених ознак виступають схрещування (гібридизація) та мутації.

Методологічна основа генетичних алгоритмів ґрунтується на гіпотезі селекції, яка в самому загальному вигляді може буди сформульована таким чином: чим вища пристосованість особини, тим вища імовірність того, що в потомстві, отриманому з її участю, ознаки, які визначають пристосованість, будуть виражені ще більше.

Оскільки генетичні алгоритми мають справу з популяціями постійної чисельності, тут особливу актуальність нарівні з селекцією у батьків здобуває відбір на знищення. Найчастіше особини, які володіють низькою пристосованістю. Не тільки не беруть участь у генерації нового покоління, а вилучаються з популяції на черговому дискретному кроці еволюції.

Особини подаються у вигляді вектора. Координати векторів несуть змістовну «генетичну» інформацію. Подібно тому, як у природі схрещування організмів здійснюється на генетичному рівні у процедурі оптимізації координати нових спробних точок отримуються як результат маніпулювання координатами старих. При цьому тут присутні гіпотези селекції – у якості батьківських завжди виступають кращі, а не довільні особини з популяції потенційних рішень; безуспішні ж рішення відкидають на чинному кроці (можна вважати, що вони «помирають»).

Відмінною рисою генетичного алгоритму від інших методів навчання штучних нейронних мереж є те, що він дозволяє відшукати глобальний мінімум похибки при налаштуванні ваг, хоча з малою точністю.

У загальному випадку генетичний алгоритм відрізняється від інших чисельних методів оптимізації тим, що запозичає з біології:

- понятійний апарат;
- ідею колективного пошуку екстремуму за допомогою популяції особин;
- способи подання генетичної інформації;

- способи передачі генетичної інформації в черзі поколінь (генетичні оператори);
- ідею про пере важність розмноження найбільш пристосованих особин.

При розгляді генетичного алгоритму навчання штучної нейтронної мережі використовуються наступні поняття і позначення.

Особина – вектор вхідних і вихідних ваг нейронної мережі $W = (W_{inp}, W_{mid}^{j}, W_{out})$, де W_{inp} – вектор вхідних ваг нейронної мережі; W_{mid}^{j} - вектор ваг між j-м і (j + 1) – м прихованими шарами нейронної мережі (для багатошарової нейронної мережі); $j = \overline{1..k-1}$, де k – число прихованих шарів; W_{out} - вектор вихідних ваг нейронної мережі.

Розмірність W дорівнює $(N + 1)\ J_1 + J_1J_2 + .. + J_{k-1}J_k + J_kM$, де N – число входів НМ; J_j – число нейронів *j-го* прихованого шару; M – число виходів НМ.

Навчання нейронної мережі за допомогою генетичного алгоритму – ітеративний процес відшукання такої особини (тобто набору ваг нейронної мережі) за допомогою випадкової зміни компонентів, яка би задовольняла заданому критерію навчання.

Критерій навчання – сума квадратів відхилень отриманих за моделлю виходів і бажаних виходах.

$$e(W) = \frac{1}{2}\|d - y(W)\|^2 = \frac{1}{2}\sum_{k=1}^{M}(d_k - y_k(W))^2 .$$

Критерій зупинення навчання. Процес навчання припиняється при досягненні критерієм заданого порогу для будь-якої з шуканих особин. На практиці точність налаштування ваг за допомогою генетичного алгоритму невисока, й задана точність (порядку 0,001 чи вище) не досягається. Тому процес навчання припиняється після закінчення заданої кількості кроків еволюції. Краща особина на цей момент є шуканою.

У процесі навчання особини оцінюються за показником придатності. *Показник придатності (FI)* – величина, яка характеризує близькість особини до бажаного значення.

$GI = a – E(W)$,

де $E(W)$ – похибка налаштування мережі для даного набору ваг W; a – константа, яка відповідає ідеальному значенню критерію. Вона вибирається як кількість елементів у навчальній вибірці, оскільки, тому що дані при розрахунках нормуються, максимальна похибка від однієї точки складає 1.

Крок еволюції – одна ітерація.

Популяція розміру N – набір із N особин. Розмір популяції залишається постійним.

Особина-батько – одна з двох особин, які виходять у результаті процедури схрещування.

Схрещування – процедура імовірнісної заміни компонента чи блоків компонентів однієї особини-батька компонентами чи блоками іншої особини-батька з урахуванням критерію навчання, тобто заміна блоків відбувається лише в тому випадку, коли критерій навчання при цьому поліпшується. Процедура функціонує окремо за вхідними та вихідними вагами. Результатом даної процедури є дві особини-нащадки.

Батьківська пара для схрещування обирається з імовірністю, яка залежить від показника придатності.

Мутації – процедура зміни абсолютної величини випадково обраного компонента вектора ваг особини-нащадка на випадкову величину. На кожнім кроці еволюції мутують 10% компонент кожного вектора-нащадка. Обсяг мутації залежить від номера ітерації за експонентним законом.

*Обсяг мутацій = k * exp(-n * a).*

Навчальна послідовність – послідовність точок (вибірка реальних даних), на якій відбувається процес налаштування ваг НМ.

У роботі величина навчальної послідовності визначається при побудові моделі вхідних даних для кожної прогнозованої точки.

Перевірочна послідовність – послідовність точок (вибірка реальних даних), на якій перевіряється якість налаштування ваг НМ, тобто прогнозованих точок. Надалі будемо прогнозувати одну точку.

Генетичний алгоритм призначений для навчання нейронної мережі шляхом налаштування її вхідних і вихідних ваг. У результати навчання виходи нейронної мережі повинні збігатися з бажаними виходами (реальними даними) із заданим ступенем вірогідності (критерій навчання).

Процес навчання нейронної мережі за допомогою генетичного алгоритму являє собою ітеративний процес відшукання такої особина (тобто набору ваг нейронної мережі) за допомогою випадкової зміни компонентів, яка задовольняла би заданому критерію навчання. З цією метою на першому кроці еволюції генерується N (парна кількість) особин – популяція. На кожному дискретному кроці еволюції в результаті багаторазових змін визначених особин шляхом їхнього схрещування та мутації в популяцію крім N батьківських особин додається N особин-нащадків. З отриманих $2N$ особин відбирається тільки N кращих за величиною показника придатності. Процес перетворення особин продовжується до тих пір поки критерій навчання не буде задоволений для будь-якої з шуканих особин. На практиці точність налаштування ваг за допомогою генетичного алгоритму невисока, і задана точність (порядку 0,001 чи вище) не досягається. Тому процес навчання припиняється після закінчення заданої кількості кроків еволюції. Краща особина на цей момент є шуканою.

Процес навчання називається колективним, тому що на кожній його ітерації в навчання бере участь N особин. Крім того, процедура схрещування припускає наявність на кожнім кроці еволюції не менш за дві особини (парна кількість). Процес перетворення особин завершується пошуком глобального мінімуму (за вагами) похибки навчання.

Так само як і природний хромосомний матеріал являє собою лінійну послідовність різних комбінацій чотирьох нуклеотидів, вектора перемінних у генетичних алгоритмах також записують у вигляді ланцюжків символів, використовуючи двох-, трьох- і більш літерний алфавіт. В цьому розділі компоненти векторів можуть приймати будь-які дійсні значення.

Недоліки двох-, трьох-, … літерних алфавітів полягають у необхідності кодування інформації, яка в нашому розумінні означає перетворення дійсних чисел до, наприклад, бінарного еквіваленту. Незручність такого подання пов’язана з невизначеністю щодо абсолютних значень компонентів вектора, які у загальному випадку будь-які дійсні значення, тобто необхідно заздалегідь обмежувати величину компонентів вектора.

До векторів ваг з метою перетворення застосовуються наступні оператори: схрещування та мутації.

Схрещування – це оператор, який застосовується до двох особин-батьків. Результат його дії полягає в зміні місцями (чи заміні) компонентів чи блоків компонент цих векторів, у наслідку чого отримуємо два нащадки. У даній роботі заміна одного блоку іншим при отриманні нащадка відбувається тільки в тому випадку, якщо дана заміна призведе до поліпшення індексу придатності FI для даного нащадка.

Мутації – це оператор, який дозволяє змінювати окремі компоненти вектора шляхом впливу випадковими шумами (зміна величини компонента вектора на випадкову величину). В межах нашого опису ця випадкова величина є рівномірно розподіленою на інтервалі *[-k*exp(-n*a);+k*exp(-n*a)]*, де *a* – рівень мутацій, *n* – номер ітерації. За допомогою мутацій забезпечується мінливість потомства.

Процедуру навчання ШНМ за допомогою ГА можна зобразити на блок-схемі наступним чином:

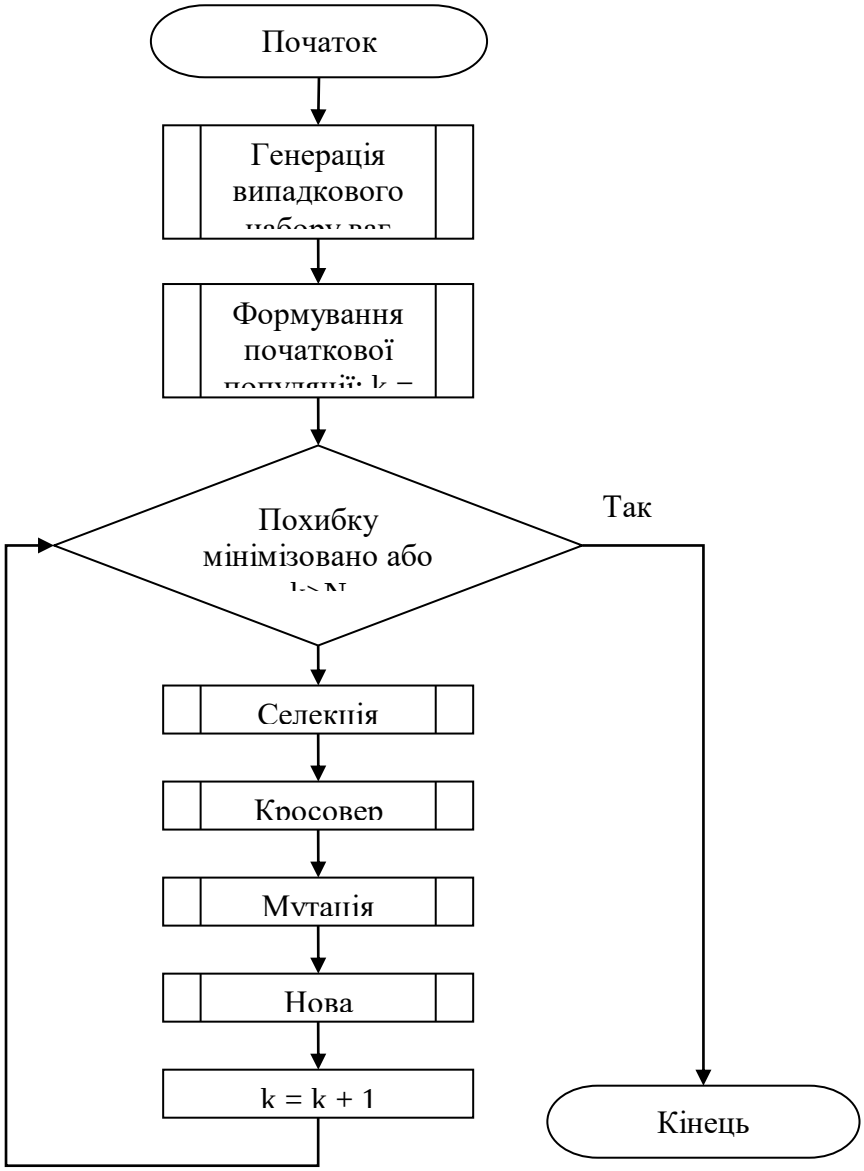


Рисунок 2.3 – Блок схема навчального алгоритму генетичної селекції

2.3 Навчальні алгоритм з використанням градієнтного спуску

Задачу навчання мережі у випадку застосування градієнтних алгоритмів розглядають як задачу мінімізації апріорно визначеної цільової функції $\xi(w)$. Градієнтні методи зв'язані з розкладанням цільової функції $\xi(w)$ в ряд Тейлора в найближчому окілі точки наявного рішення w. У випадку цільової функції від багатьох змінних ($w = [w_1, w_2, \ldots, w_n]^T$) таке представлення пов'язується з околом раніше визначеної точки (при старті алгоритму – це вихідна точка w_0) в напрямку x. Подібне розкладання описується універсальною формулою виду:

$$\xi(w+x) = \xi(w) + [p(w)]^T x + \frac{1}{2} x^T H(w) x + \ldots \quad (1)$$

Де $p(w) = \nabla\xi = \left[\frac{\delta\xi}{\delta w_1}, \frac{\delta\xi}{\delta w_2}, \ldots, \frac{\delta\xi}{\delta w_n}\right]^T$ – це вектор градієнту, а симетрична квадратна матриця

$$H(w) = \begin{bmatrix} \frac{\delta^2\xi}{\delta w_1 \delta w_1} & \ldots & \frac{\delta^2\xi}{\delta w_1 \delta w_n} \\ \ldots & \ldots & \ldots \\ \frac{\delta^2\xi}{\delta w_{1n} \delta w_1} & \ldots & \frac{\delta^2\xi}{\delta w_n \delta w_{1n}} \end{bmatrix}$$

являється матрицею похідних другого порядку і називається гессіаном.

У виразі (1) x грає роль направляючого вектора, що залежить від фактичних значень вектора w. На практиці частіше всього розраховуються три перших члени ряду, а наступні просто ігнорується.

Для спрощення опису значення змінних одержані в k-му циклі будемо записувати з нижнім індексом k. Точкою рішення $w = w_k$ будемо вважати точку, в якій досягається мінімум цільової функції $\xi(w)$ та $p(w_k)=0$, а гессіан $H(w_k)$ являється невід'ємно визначеним. При виконані цих умов функція в будь-якій точці, що лежить в окілі w_k, буде мати більше значення ніж в точці w_k, тому точка w_k являється рішенням, що відповідає критерію мінімізації цільової функції.

У процесі пошуку мінімального значення цільової функції напрямок пошуку p та крок h підбираються таким чином, щоб для кожної наступної точки $w_{k+1} = w_k + \eta_k x_k$ виконувалася умова $\xi(w_{k+1}) < \xi(w_k)$. Пошук мінімуму виконується до тих пір, поки норма градієнту не опуститься нижче апріорі заданого значення допустимої похибки, або поки не буде перевищено максимальної кількості ітерацій. Універсальний оптимізаційний алгоритм навчання нейронної мережі можна представити в наступному вигляді (будемо вважати, що початкове значення вектора, що оптимізується відоме і складає $w_k = w_0$):

Крок 1: Перевірка зходжуванності і оптимальності поточного значення w_k. Якщо точка w_k відповідає градієнтним умовам зупинки процесу – завершення обчислень. У протилежному випадку перейти до кроку 2.

Крок 2: Визначення вектора напрямку оптимізації x_k для точки w_k.

Крок 3: Вибір величини кроку η_k в напрямку x_k , при якому виконується умова $\xi(w_k + \eta_k x_k) < \xi(w_k)$.

Крок 4: Визначення нового рішення $w_{k+1} = w_k + \eta_k x_k$, а також відповідних йому значень $\xi(w)$ та $p(w_k)$, а якщо треба, то і $H(w_k)$ та повернення до кроку 1.

2.3.1 Алгоритм спряжених градієнтів

Цей алгоритм відрізняється від загального алгоритму градієнтного спуску тим, що при виборі напрямку мінімізації не використовується інформація про гессіан. Напрям пошуку x_k вибирається таким чином, щоб він був ортогональним та спряженим до всіх попередніх напрямків $x_0, x_1, \ldots, x_{k-1}$. Множина векторів x_i, $i = 0, 1, \ldots, k$, буде взаємно спряженим відносно матриці ***A***, якщо $x_i^T A p_j = 0, i \neq j$ (1)

Вектор, що задовольняє задані умови має вигляд:

$$x_{k+1} = x_k + \alpha_k p_k \quad (2)$$

де $p_k = p(w_k)$ є фактичним значенням вектору градієнта.

Із формули (2) слідує, що новий напрямок мінімізації залежить тільки від значення градієнту в точці рішення w_k та від попереднього напрямку пошуку p_k, помноженого на коефіцієнт спряження α_k.

Покроково алгоритм спряжених градієнтів виглядає так:

Крок 1: Задати початковий вектор $x^{(0)}$ та $\varepsilon > 0$ (рівень допустимих похибок).

Крок 2: Обчислити вектор $p_0 = \xi^{(0)}$, $k = 0$ (номер ітерації).

Крок 3: Обчислити скаляр $\alpha_k = \frac{(\xi^{(k)}, p^{(k)})}{(\xi^{(k)}, Ap^{(k)})}$

Крок 4: Обчислити вектор $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$

Крок 5: Обчислити: $\xi^{(k+1)} = \xi^{(k)} - \alpha_k A p^{(k)}$

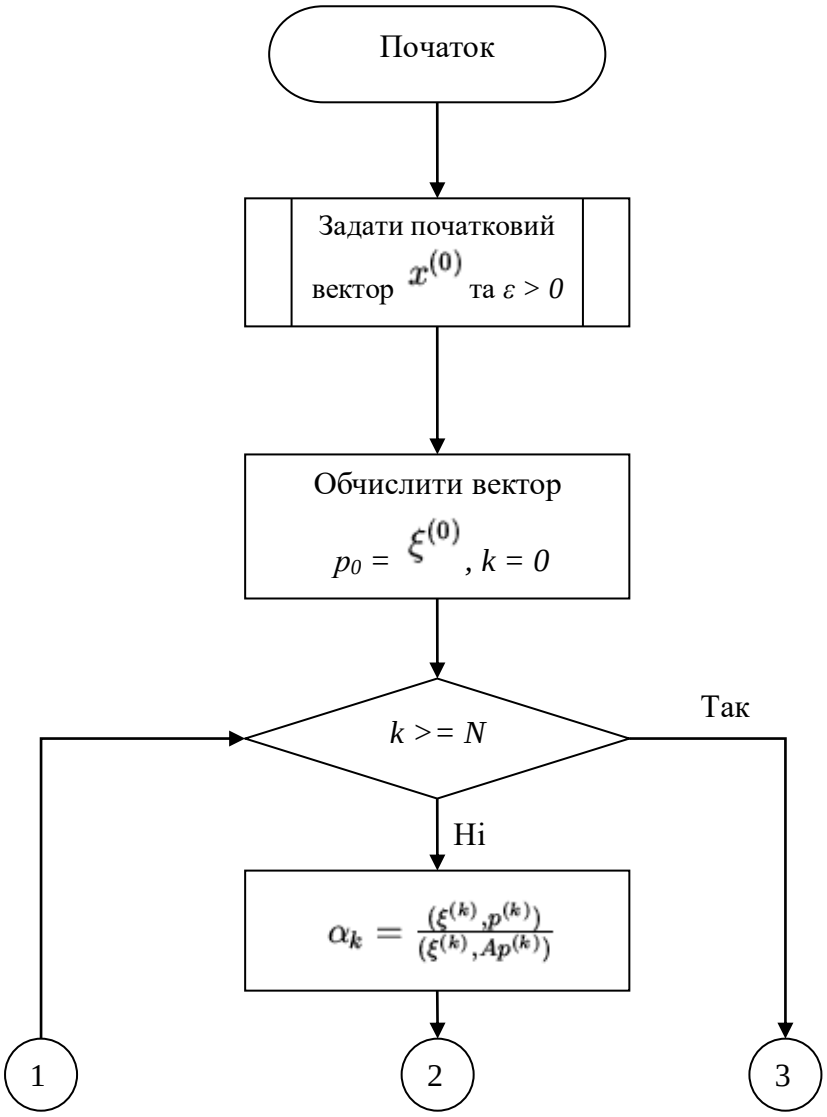
Крок 6: Перевірити виконання умови $||\xi^{(k+1)}|| \leq \varepsilon$. Якщо умова виконується – завершити алгоритм.

Крок 7: Обчислити скаляр $\beta_k = \frac{(\xi^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})}$

Крок 8: Обчислити $p^{(k+1)} = \xi^{(k+1)} - \beta_k p^{(k)}$

Крок 9: Встановити $k := k + 1$ і повернутися до кроку 3.

Зобразимо алгоритм навчання ШНМ методом спряжених алгоритмів на блок-схемі:


Початок
Задати початковий вектор $x^{(0)}$ та $\varepsilon > 0$
Обчислити вектор $p_0 = \xi^{(0)}$, $k = 0$
$k >= N$
Так
Ні
$\alpha_k = \frac{(\xi^{(k)}, p^{(k)})}{(\xi^{(k)}, Ap^{(k)})}$
1
2
3

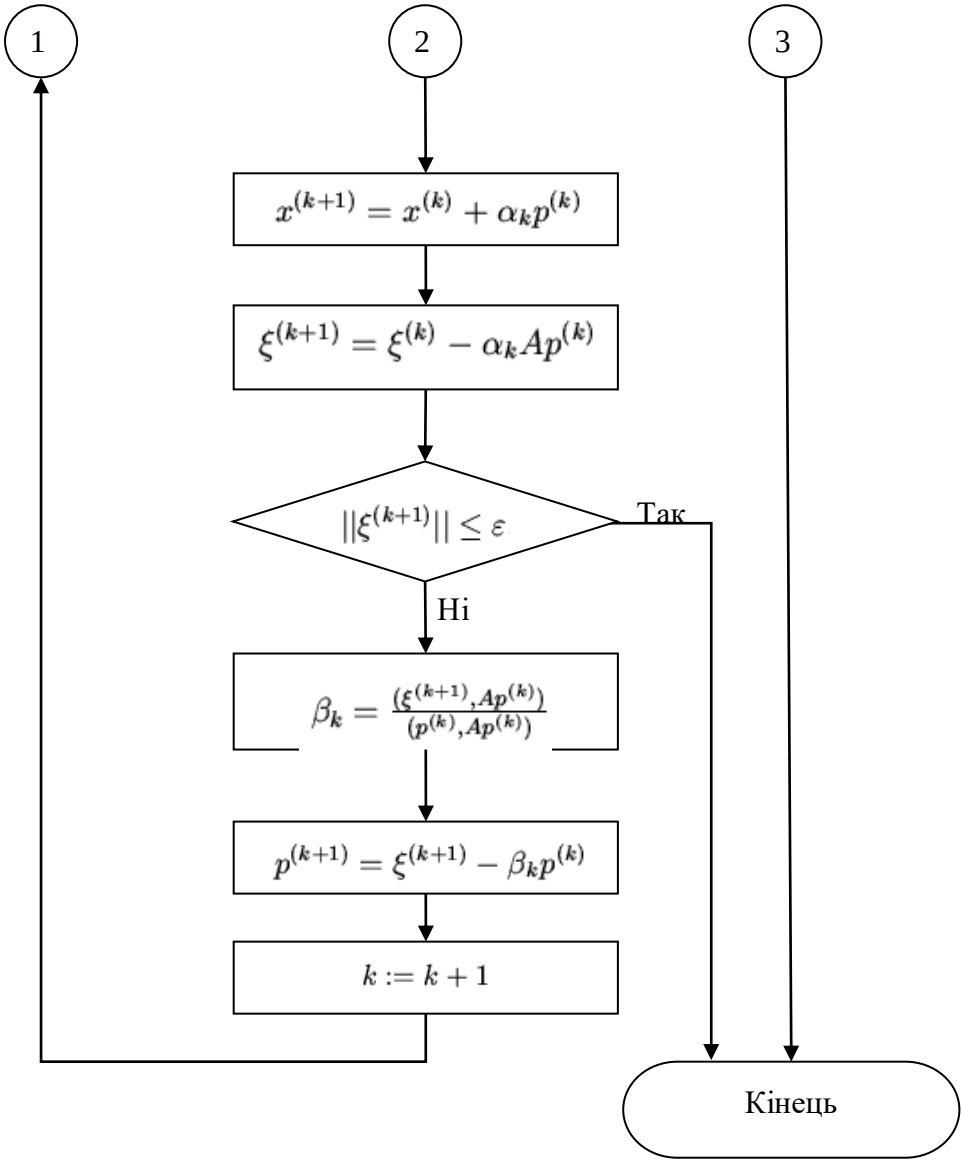


Рисунок 2.4 – Блок схема алгоритму спряжених градієнтів

2.3.2 Навчальний алгоритм зворотного поширення помилки

В загальному вигляді алгоритм зворотного поширення помилки являє собою наступну послідовність кроків:

Крок 1: Ініціювати ваги малими випадковими величинами.

Крок 2: Якщо умова зупинки не виконується, виконати кроки 3-10.

Крок 3: Для кожної навчальної пари виконати кроки 4-9.

Прямий прохід:

Крок 4: Кожен вхідний нейрон x_i *=1…n* приймає вхідний сигнал і поширює його до всіх нейронів прихованого шару.

Крок 5: Кожен нейрон прихованого шару v_j *=1…q* підсумовує свої зважені вхідні сигнали: $h_j = \sum_i^n w_{ij} x_i$, застосовує до одержаної суми функцію активації, формуючи вихідний сигнал: $v_j = f(h_j)$, котрий надсилається до всіх нейронів вихідного шару.

Крок 6: Кожен вихідний нейрон y_k, *k =1…m* підсумовує зважені сигнали: $h_k = \sum_j^q w_{jk} v_j$, формуючи після застосування функції активації вихідний сигнал мережі: $y_k = f'(h_k)$.

Зворотне поширення помилки:

Крок 7: Кожен вихідний нейрон співставляє своє значення виходу з потрібною цільовою функцією і вираховує: $\delta_k = (t_k - y_k) f'(h_k)$, після чого визначається корегуючий член ваг: $\Delta w_{jk} = \eta \delta_k v_j$, а параметри δ_k надсилаються в нейрони прихованого шару.

Крок 8: Кожен нейрон прихованого шару v_j підсумовує свої δ-входи від нейронів вихідного шару: $h_k = \sum_k^m \delta_k w_{jk}$, результат помножують на похідну від функції активації для визначення δ_j: $\delta_j = f'(h_j) \sum_k^m \delta_k w_{jk}$

та вираховується корегуючи член: $\Delta w_{ij} = \eta \delta_k w_{jk}$.

Корегування ваг:

Крок 9: Ваги між прихованим та вихідним шарами модифікуються так:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

Аналогічно корегуються ваги між вхідним та прихованим шарами:

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

Крок 10: Перевіряється умова зупинки: мінімізація похибки між потрібним та реальним виходом мережі.

За допомогою блок-схеми поданий алгоритм можна зобразити наступним чином:

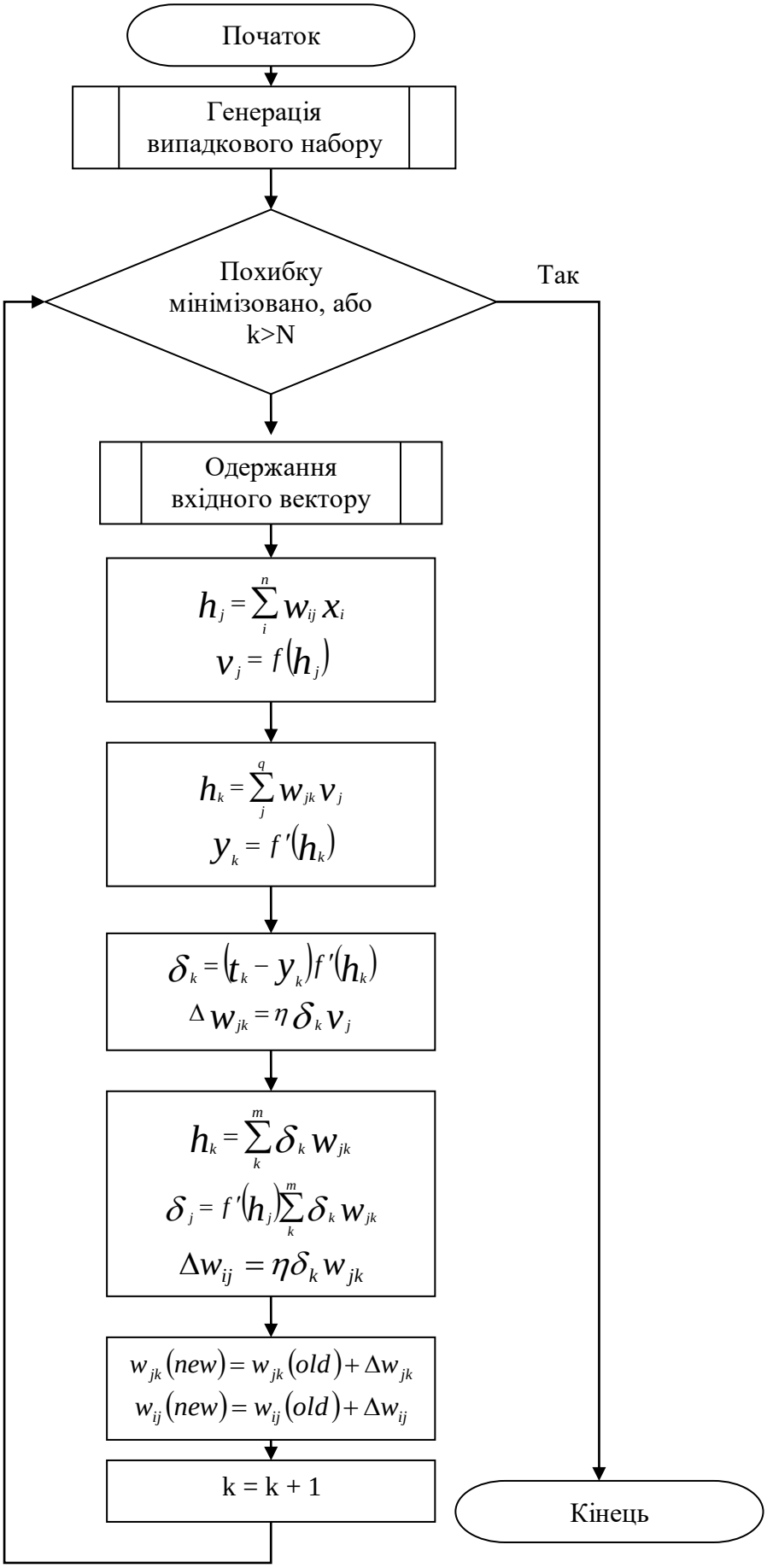


Рисунок 2.5 – Блок схема алгоритму зворотного поширення помилки

2.4 Порівняльний аналіз алгоритмів навчання штучної нейронної мережі

Порівняльний аналіз обраних для дослідження алгоритмів проводиться по таким критеріям, як час виконання алгоритму, час виконання програмної реалізації алгоритмів, швидкість сходження алгоритмів та точність результату програми при заданій кількості ітерацій.

2.4.1 Дослідження часу виконання обраних алгоритмів

Час виконання алгоритму являє собою кількість кроків, які виконуються в даному алгоритмі від початку і до кінця. Для підрахування кількості кроків досліджуваних в даній роботі алгоритмів, зручно скористатися приведеними вище блок-схемами (рисунки 2.3, 2.4 та 2.5).

Результати підрахунків приведено у таблиці 2.1:

Таблиця 2.1 – Час виконання алгоритмів навчання ШНМ

Назва алгоритму	Час виконання алгоритму (кроки)
Алгоритм генетичної селекції	8
Алгоритм спряжених градієнтів	11
Алгоритм зворотного поширення помилки	16

Кількість кроків характеризує собою ніщо інше, як складність реалізації алгоритму, що є зараз досить важливим фактором, адже часто робочий час програміста коштує дорожче, ніж робочий час комп’ютера, на відміну від минулих років.

2.4.2 Дослідження часової складності програми

На швидкість виконання програми в цілому впливають такі фактори:

- введення інформації в систему;

- якість скомпільованого коду програми;
- часова складність програми.

Оскільки програмні реалізації всіх трьох алгоритмів навчання ШНМ оперують з одним і тим самим набором вхідних даних, використовуються для навчання однієї й тієї ж структури ШНМ, та скомпільовані одним компілятором, то першими двома факторами можна знехтувати. Таким чином, нас цікавить лише часова складність програмної реалізації кожного з досліджуваних алгоритмів навчання ШНМ.

Часову складність програми визначають за допомогою так званої О-нотації. Асимптотична нотація великого О, відома також як нотація Ландау - розповсюджена математична нотація для формального запису асимптотичної поведінки функцій. Широко вживається в теорії складності обчислень, інформатиці та математиці.

Позначення

Нехай задані дві числові функції *f(x)* та *g(x)*. Тоді говорять, що f(x) є O(g(x)) тоді, коли виконується нерівність $|f(x)| \leq M\ |g(x)|$ для деяких x_0 та M таких, що $M > 0$ та $x > x_0$.

Функції f(x) та g(x) називають *функціями одного порядку*, якщо f(x) = O(g(x)) та g(x) є O(f(x)) для $x \to x_0$;

Часто для позначення того факту, що *f(x)* є *O(g(x))* використовується запис *f(x) = O(g(x))*, хоч це не зовсім вірно і в деяких випадках може призвести до плутанини.

Деякі важливі класи відношень

У таблиці 2.2 наведений список класів функцій, які часто зустрічаються в аналізі алгоритмів. Тут $n \to \infty$, *c* -- константа. Функції, які зростають повільніше, наведені першими.

Таблиця 2.2 – Часто вживані функції зростання складності програми

нотація	**назва**
O(1)	константа
O(log *n*)	логарифмічне
O([log *n*]^c)	полілогарифмічне
O(*n*)	лінійне
O(*n* · log *n*)	лінеарітмічне
O(n^2)	квадратичне
O(n^c)	поліноміальне
O(c^n)	експоненціальне
O(*n!*)	факторіальне

Для практичного використання О-нотації слід привести два основні правила – правило суми та правило добутку.

Правило суми: Нехай $T_1(n)$ та $T_2(n)$ – час виконання двох програмних фрагментів P_1 та P_2, T_1 має ступінь росту $O(f(n))$, а T_2 - $O(g(n))$. Тоді:

$T_1(n) + T_2(n)$ має ступінь росту $O(max(f(n), g(n)))$.

Правило добутку: Нехай $T_1(n)$ та $T_2(n)$ – час виконання двох програмних фрагментів P_1 та P_2, T_1 має ступінь росту $O(f(n))$, а T_2 - $O(g(n))$. Тоді:

$T_1(n) * T_2(n)$ має ступінь росту $O(f(n) * g(n))$.

Із правила добутку випливає, що $O(c * f(n)) = O(f(n))$, якщо c – невід'ємна константа.

Оскільки всі досліджувані в даній роботі алгоритми оброблюють один і той же набір вхідних даних, цю величину можна прийняти константною й такою, що не відіграє суттєвої ролі у встановлені часової складності програмної реалізації кожного з вище приведених алгоритмів.

Таким чином із величин, що мають значення при виконанні алгоритму навчання ШНМ: набір навчальних даних, набір ваг та кількість ітерацій навчання залишається лише останні дві. Отже, кількість ітерацій (k) і розмір вектору вагів (w) будуть виконувати роль міри об'єму вхідних даних.

Обчислимо часову складність першого із розглянутих алгоритмів – навчального алгоритму генетичної селекції:

Такі операції як селекція, кросовер, мутація та генерація нової популяції – є циклічними, для всього набору вагів, тому час кожної такої операції буде дорівнювати $O(w)$.

Як ми знаємо з правила сум $O(w) + O(w) = O(w)$. Таким чином, сумарний час виконання всіх операцій в середині циклу від $[k = 0; k <= N]$ буде дорівнювати $O(w)$.

Сам цикл навчання буде виконуватися $N - 1$ разів, тому, користуючись правилом добутку можемо обчислити загальний час виконання програми, що реалізує навчальний алгоритм генетичної селекції:

$O_{заг} = O(N - 1) * O(w) = Nw - w$.

Ми могли б знехтувати величиною w, якби мали на меті використовувати даний алгоритм лише для ШНМ з незначним за розміром вектором вагів. Однак для вирішення багатьох задач (і задачі прогнозування в тому числі) доводиться використовувати доволі складні структури нейронних мереж, тому відкидання числа w, як такого, що не має суттєвого впливу на час навчання може бути недоцільним. Отже, підсумовуючи все вищесказане, одержимо, що час виконання програми, яка реалізує навчальний алгоритм генетичної селекції буде дорівнювати $O_{ГА} = Nw - w$.

Такий же результат одержимо і у випадку із алгоритмом спряжених градієнтів, оскільки подібно до ГА, цей алгоритм виконує свої операції відразу для всього вектору вхідних вагів і також проходить через $N - 1$ ітерацій.

Тому, можемо відразу визначити, що $O_{СГ} = Nw - w$.

Із алгоритмом зворотного поширення похибки ситуація є дещо інакшою. Він також виконується $N - 1$ разів, однак операції всередині основного циклу виконуються пошарово, тобто не w разів, як у попередніх двох випадках, а w/m, де m – кількість шарів вектору вагів.

Таким чином, як і раніше використовуючи правило суми та правило добутку, одержимо: $O_{ЗПП} = Nw/m - w/m$.

Ми не можемо співвідносити кількість ітерацій головного циклу алгоритму (будь-якого з вище означених) і розмір вектору вагів, оскільки не знаємо, наскільки ці величини відрізняється. Завжди залишається невелика ймовірність того, що вдалий набір ваг буде віднайдено ще при випадковій генерації, і в такому випадку кількість ітерацій не буде відігравати ніякої ролі. Тому, в даній роботі ніяких співвідношень та спрощень щодо величин N та w не виконується.

Результати обчислень часу виконання програм, що реалізують досліджувані алгоритми навчання ШНМ приведено у таблиці 2.3:

Таблиця 2.3 – Часова складність досліджуваних алгоритмів

Програмна реалізація алгоритму:	Час виконання програми
генетичної селекції	$O_{ГА} = Nw - w$
спряжених градієнтів	$O_{ГА} = Nw - w$
зворотного поширення помилки	$O_{ЗПП} = Nw/m - w/m$

Отже, можна сказати, що кожен із приведених алгоритмів є лінійним по своїй суті, а наскільки важливими є параметри N, w та m можна сказати лише при розгляді конкретної структури ШНМ.

2.4.3 Дослідження точності результату роботи програмної реалізації алгоритмів навчання ШНМ при заданій кількості ітерацій

Вибір структури мережі проводився за допомогою програми STATISTICA Neural Networks.

Навчання ШНМ алгоритмом генетичної селекції виконувалося з використанням програмного забезпечення GeneHunter.

Навчання ШНМ алгоритмами спряжених градієнтів та зворотного поширення помилки проводилося з використанням програми STATISTICA Neural Networks.

Обрану структуру нейронної мережі, яка буде ідентичною для всіх трьох алгоритмів, показано на рисунку

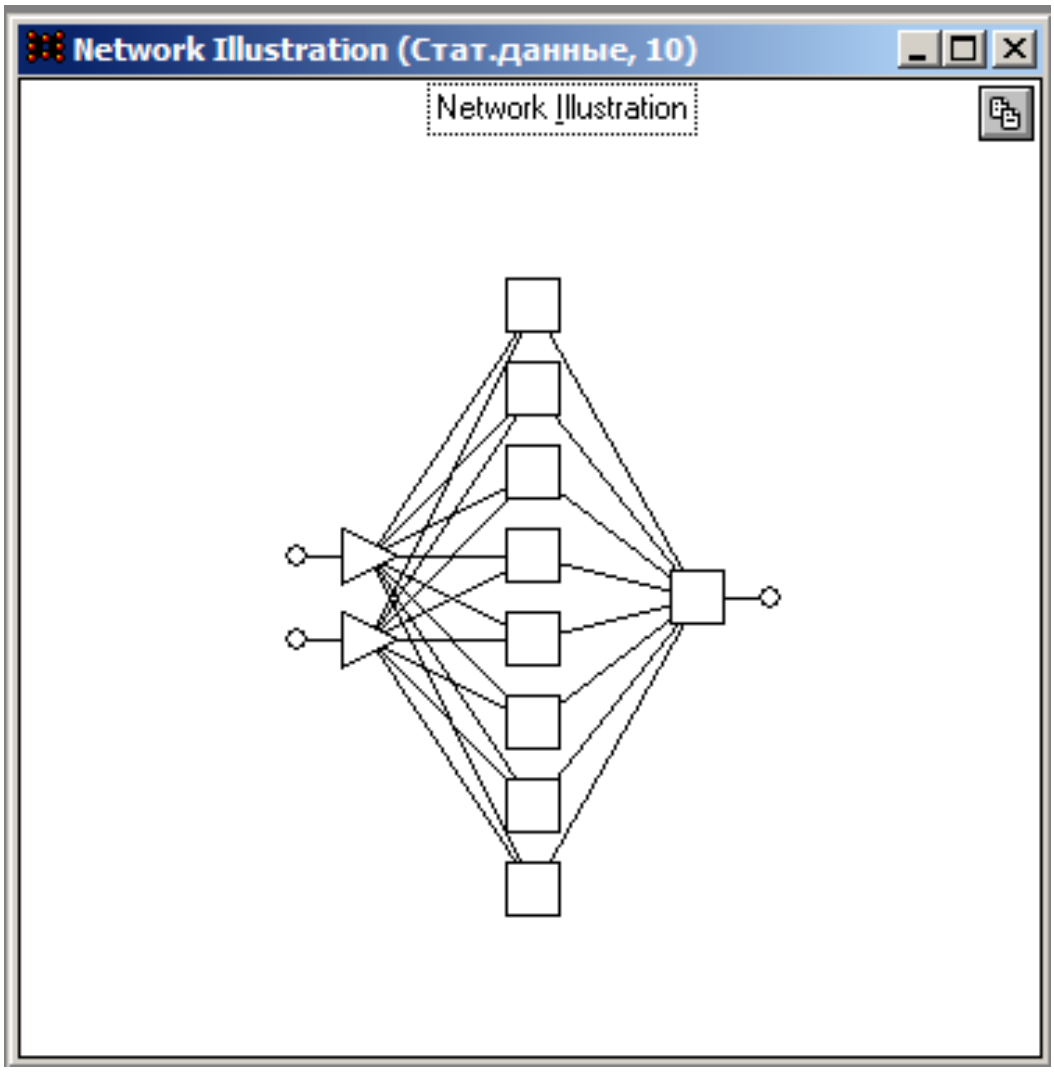


Рисунок 2.6 – Структура штучної нейронної мережі

Результати експериментальних досліджень приведено у таблиці 2.4:

Таблиця 2.4 – Середня похибка прогнозування досліджуваних алгоритмів при заданій кількості ітерацій

Алгоритм	Кількість епох*	Середня похибка
Генетичний алгоритм	1000	0.1607812
Спряжених градієнтів	1000	0.1403673**
Зворотного поширення помилки	1000	0.1549992

* Епохою вважається одне повне пред'явлення всіх навчальних даних.

** Навчання доводилося перезапускати кілька разів через потрапляння алгоритму до локального мінімуму.

2.4.4 Зведена порівняльна таблиця алгоритмів навчання штучної нейронної мережі

Таблиця 2.5 – Зведена порівняльна таблиця алгоритмів навчання ШНМ

Алгоритм/ Фактор	**Алгоритм генетичної селекції**	**Алгоритм спряжених градієнтів**	**Алгоритм зворотного поширення помилки**
Складність алгоритму (кроки)	8	11	16
Час виконання програми (О-нотація)	$O_{ГА} = Nw - w$	$O_{ГА} = Nw - w$	$O_{ЗПП} = Nw/m - w/m$
Похибка (10000 епох)	0.1287812	0.1203673	0.1249992
Фактична тривалість навчання (для 10000 епох) (с)	29 с	24 с. з урахуванням затримки на пере запуск алгоритму при локальному	30 с

		мінімумі – 52 с.	

Слід зробити деякі пояснення до даної таблиці: як можна побачити, наявна деяка невідповідність у співвідношеннях теоретичного і фактичного часу виконання програмних реалізацій досліджуваних алгоритмів навчання ШНМ.

Це пояснюється тим, що з огляду на досить невеликий розмір вхідного шару ШНМ (що пояснюється відсутністю достатніх статистичних даних) структура мережі є доволі примітивною, а тому і величина вектору вагів є незначною.

Однак, слід зазначити, що при ускладненні структури мережі, величина вагового вектору може стати фактором, що матиме суттєвий вплив на час навчання ШНМ.

Взагалі ж, виходячи з приведеної таблиці, можна зробити висновок, що оптимальним вибором для навчання ШНМ в задачі прогнозування курсу дорогоцінних металів є алгоритм зворотного поширення помилки з огляду на оптимальне співвідношення часу навчання і точності прогнозування.

Такий параметр як складність алгоритму відіграє менш важливу роль, оскільки у такій задачі, як прогнозування курсу дорогоцінних металів, валют та ін. головним фактором є точність прогнозу, оскільки ця точність прямо пропорційна одержуваним прибутками компанії-замовника.

ПРАКТИЧНА РОБОТА №1

Бінарний перцептрон

Мета: *Вивчити принципи функціонування і навчання перцептрона.*

Теоретичні відомості.

Нейронна мережа (*Neural network*) є сукупністю простих елементів (*units*), які функціонально базуються на нейронах. Біологічний нейрон зображений на рис. 1.1. Сигнали між нейронами передаються вздовж аксонів. Сигнали, які підсилюються або послаблюються синапсами, нейрон одержує через дендріти, які підсилюються, або послаблюються синапсами. Штучні нейрони, відомі як *Threshold Logic Unit* (TLU) і запропоновані Маккалоком і Піттсом *(McCulloch and Pitts*) у 1943 році. Нехай маємо вектори множину входів $\boldsymbol{X} = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ і вагових коефіцієнтів $\boldsymbol{W}$ - $\{w_1, w_2, \ldots, w_i, \ldots, w_n\}$. Входи-сигнали набувають значень лише "0" або "1".

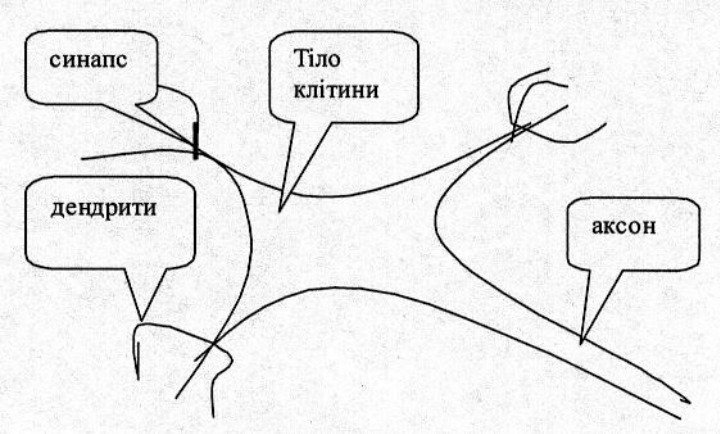


Рис. 1.1. Біологічний нейрон

Активацію $\boldsymbol{a}$ розраховуємо за формулою

$$a = \sum w_i x_i = (w, x). \qquad (1.1)$$

Вихід $\boldsymbol{Y}$ обчислюється за формулою

$$Y = \begin{cases} 1, \text{якщо } a \geq \theta, \\ 0, \text{якщо } a < \theta, \end{cases} \qquad (1.2)$$

де θ - порогове значення (*threshold*) найчастіше буває нулем (рис. 1.2). Більш загальний елемент нейронної мережі був введений Розенблаттом у 1962 році і названий ***перцептроном***. Він складається із TLU, на входи яких

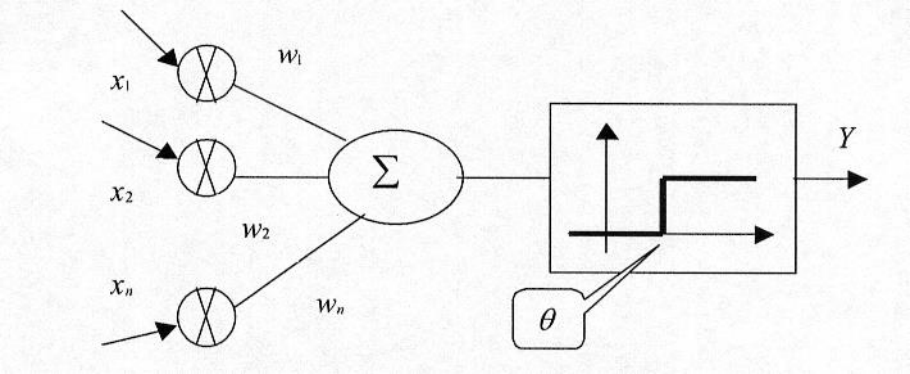


Рис. 1.2. Штучний нейрон

поступає множина входів асоціативних елементів (рис. 1.3).

Алгоритм навчання перцептрона.

1. Подати на вхід образ ***X***.

2. Кожна компонента $\boldsymbol{X} = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ множиться на відповідну компоненту вектора вагових коефіцієнтів $W = \{w_1, w_2, \ldots, w_n\}$. Знаходимо суму цих добутків

$$a = XW = \sum_{i=1}^{n} w_i x_i. \qquad (1.3.)$$

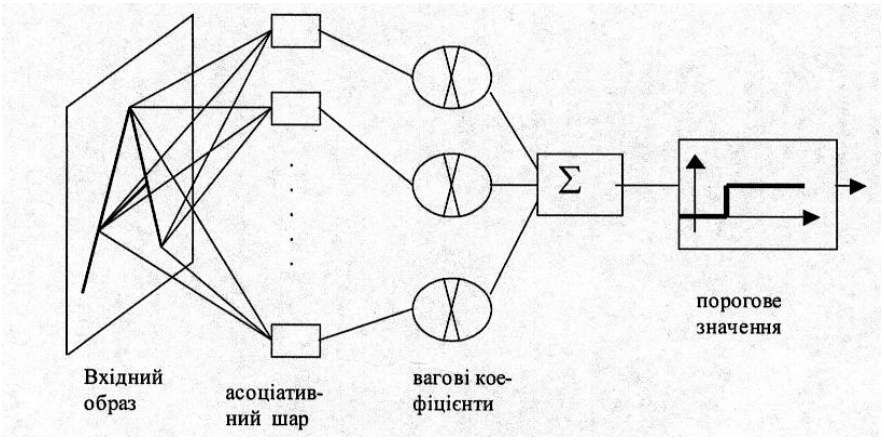


Рис. 1.3. Бінарний перцептрон

3. Якщо ***a*** перевищує порогове значення θ $(a > \theta)$, то вихід нейрона ***Y*** дорівнює *одиниці*, в іншому випадку він - *нуль*.

Якщо значення ***Y*** вірне, то нічого не змінюється і переходимо до кроку 1 у випадку, якщо на вході є інші образи. Якщо ні, то кінець.

Якщо значення ***Y*** не вірне, то підраховуємо різницю між обчисленим значенням ***Y*** і правильним значенням ***T***

$$\delta = T - Y. \qquad (1.4.)$$

4. Обчислюємо

$$\Delta_i = \eta\delta \cdot x_i \qquad (1.5.)$$

де η - коефіцієнт швидкості навчання, який знаходиться, в загальному випадку, в проміжку (0; 0,1).

5. Знаходимо

$$w_i(n+1) = w_i(n) + \Delta_i, i = \overline{1,n} \qquad (1.6)$$

де $w_i(n+1)$ - значення i-го вагового коефіцієнта після корекції, $w_i(n)$ - до корекції.

6. Якщо на вході є інші образи, то переходимо до кроку 1, якщо ні - то кінець.

Завдання до практичної роботи.

1. Програмно реалізувати наведений алгоритм.

2. Використовуючи дані таблиці 1.1 і результат пункту 1, побудувати таблицю 1.2.

Алгоритм реалізувати таким чином: на вхід поступово подати всі образи, коригуючи паралельно вагові коефіцієнти. Якщо для всіх образів δ = 0, то кінець, якщо ні, то на вхід знову ж таки подати всі образи, але із скоригованими ваговими коефіцієнтами.

3. Оформити звіт.

Таблиця 1.1- Початкові дані

№ варіанта	**Логічна функція**	**Можливі значення вагових коефіцієнтів**		
1	$x_1 \vee x_2 \wedge x_3$	0,2	0,4	0,6
2	$(x_1 \wedge x_2) \vee x_3$	0,2	0,04	0,08
3	$(x_1 \to x_2) \vee x_3$	0,05	0,1	0,3
4	$x_1 \vee x_2 \vee x_3$	0,07	0,1	0,3
5	$(\neg x_1 \vee x_2) \wedge x_3$	0,02	0,05	0,4
6	$x_1 \to (x_2 \vee x_3)$	0,1	0,5	0,7
7	$x_1 \wedge (x_2 \to x_3)$	0,2	0,02	0,002
8	$\neg x_1 \wedge (x_2 \to x_3)$	0,3	0,4	0,5
9	$x_1 \vee (\neg x_2 \to x_3)$	0,7	0,01	0,06
10	$(x_1 \wedge \neg x_2) \to x_3$	0,3	0,01	0,02
11	$(x_1 \to x_2) \wedge x_3$	0,01	0,04	0,5
12	$x_1 \vee (x_2 \wedge x_3)$	0,1	0,6	0,5
13	$\neg x_1 \vee x_2 \vee x_3$	0,1	0,03	0,09
14	$(x_1 \vee \neg x_2) \wedge x_3$	0,08	0,2	0,3
15	$x_1 \to (x_2 \wedge x_3)$	0,3	0,5	0,7
16	$x_1 \vee (x_2 \to x_3)$	0,2	0,03	0,4
17	$\neg x_1 \vee (x_2 \to x_3)$	0,7	0,2	0,06
18	$x_1 \wedge (\neg x_2 \to x_3)$	0,01	0,02	0,03
19	$(x_1 \vee \neg x_2) \to x_3$	0,5	0,4	0,03
20	$(x_1 \vee x_2) \wedge x_3$	0,7	0,06	0,1
21	$(x_1 \to \neg x_2) \wedge x_3$	0,01	0,5	0,02
22	$\neg x_1 \vee (\neg x_2 \wedge x_3)$	0,3	0,04	0,5
23	$(\neg x_1 \vee x_2) \wedge \neg x_3$	0,8	0,09	0,01
24	$x_1 \to (\neg x_2 \wedge \neg x_3)$	0,3	0,01	0,7
25	$\neg x_1 \vee (x_2 \to \neg x_3)$	0,2	0,03	0,04

26	$(x_1 \vee \neg x_2) \wedge \neg x_3$	0,1	0,7	0,03
27	$(\neg x_1 \vee x_2) \rightarrow \neg x_3$	0,5	0,8	0,02

Таблиця 1.2 – Таблиця подання результатів роботи програми.

w_1	w_2	w_3	θ	x_1	x_2	x_3	a	Y	T	$\eta(T-Y)$	δw_1	δw_2	δw_3	$\delta\theta$

Контрольні запитання

1. Коротка історична довідка про розвиток теорії нейронних мереж.
2. Будова біологічного нейрона.
3. Штучний нейрон та активаційні функції.
4. Будова та алгоритм функціонування перцептрона.
5. Проблема лінійного поділу досліджуваної області.

Практична робота №2

Алгоритм зворотного розповсюдження помилки для навчання нейронної мережі.

Мета роботи:

Вивчити основні принципи функціонування нейронних мереж, реалізувати алгоритм зворотного розповсюдження помилки для навчання мережі, дослідити можливості застосування нейронних мереж для вирішення задач теорії штучного інтелекту.

Завдання:

Скласти програму, яка реалізує функціонування нейронної мережі (рис.1) для задачі розпізнавання (класифікації) чисел. Представити операцію навчання мережі на прикладах для навчання (рис.3). Протестувати програму для контрольних прикладів (рис.4).

Передбачити для користувача можливість зміни:

1) початкового діапазону розподілу ваги для зв'язків нейронної мережі $A[a_1, a_2]$;
2) кількості прихованих шарів нейронної мережі t;
3) кількості нейронів кожного прихованого шару q_t, $i=[1..t]$;
4) коефіцієнту навчання η;
5) коефіцієнту інерції α.

Дослідити вплив вище перелічених параметрів на швидкість навчання мережі та якість розпізнавання образів.

Теоретичні відомості:

Задача полягає у розпізнаванні (класифікації) числа, яке представлене бітовою матрицею розміру 9x7. Вихідний вектор мережі має містити інформацію про належність числа до одного з 10 класів (числа від 0 до 9). Наприклад, якщо на вхід мережі буде подано матрицю, що відповідає числу 2, то у вихідному векторі значення другого біту буде дорівнювати 1. Для вирішення задачі обрати нейронну мережу з 63 вхідними нейронами, 6

нейронами прихованого шару і 10 вихідними нейронами (рис.1). Нейрони прихованого шару поєднані з усіма вхідними нейронами, а зовнішнього шару – з усіма нейронами прихованого.

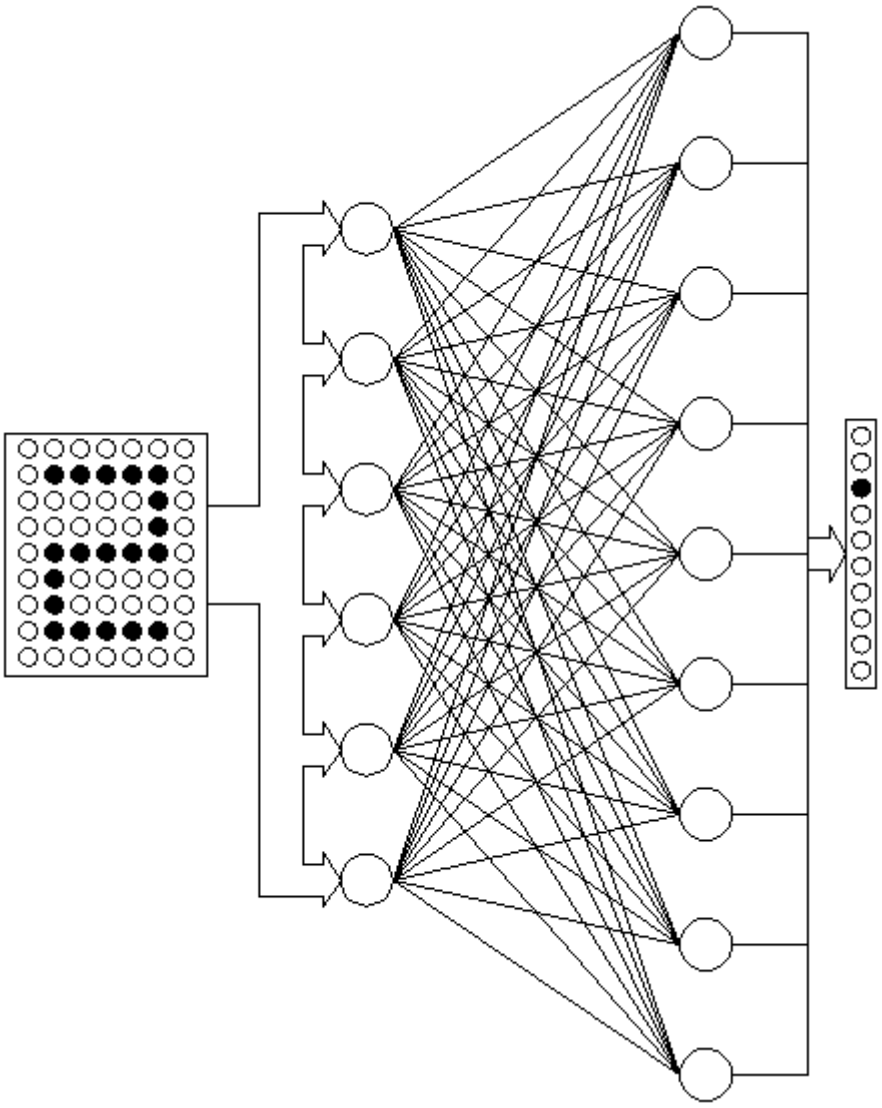


Рис.1. Загальний вигляд нейронної мережі.

Відповідність пікселів матриці нейронам вхідного шару наведена на рис.2.

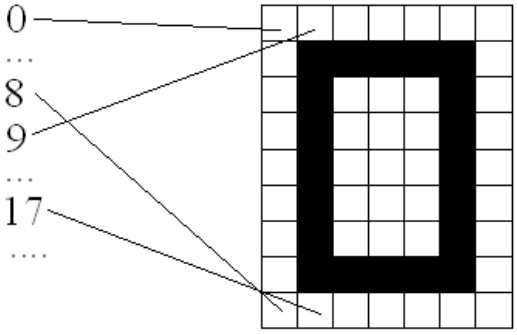


Рис.2. Відповідність пікселів нейронам вхідного шару

Числа для навчання нейронної мережі показано на рис.3.

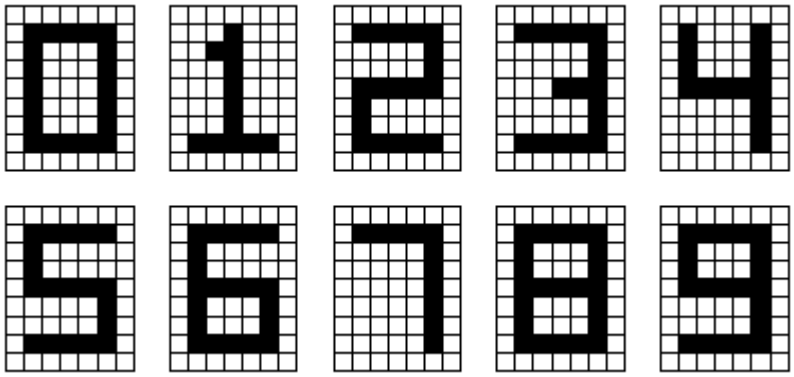


Рис.3. Числа для навчання нейронної мережі.

Алгоритм зворотного розповсюдження помилки для навчання нейтронної мережі

Суть алгоритму зводиться до послідовного подання на вхід мережі еталонних значень (рис.3), визначення вихідного вектору мережі (прямий прохід), і корегування коефіцієнтів ваги для зв'язків нейронів на основі визначення різниці між еталонним значенням вихідного вектору та отриманим (зворотній прохід). Корегування відбувається на основі норми навчання η та коефіцієнта інерції α. Навчання мережі припиняється при досягненні збіжності між еталонним та отриманим значенням вихідного вектора для всіх еталонних прикладів.

При реалізації алгоритму норму навчання обрати на рівні 0.3, а коефіцієнт інерції 0.7. Значення кожного біту вихідного вектора визначати за формулою:

$$out_j = \begin{cases} 1, o_j > 0.9; \\ 0, o_j < 0.1 \end{cases}$$

Послідовність кроків алгоритму:

1. Прочитати перший вхідний зразок(еталон) і встановити вихідний зразок, що йому відповідає; Converge = TRUE.

2. Задати значення для кожного з вхідних нейронів значення вхідного зразка.
3. Для елементів першого прихованого шару обчислити сумарний вхід та вихід $net_j = w_0 + \sum_{i=1}^{n} x_i w_{i,j}$, $o_j = \frac{1}{1+\exp(-net_j)}$
4. Повторити крок 3 для всіх наступних прихованих шарів та вихідного шару нейронів.
5. Якщо різниця між значенням виходу мережі і вихідним зразком не є допустимою, Converge = Converge&FALSE.
6. Для кожного вихідного елементу визначити його помилку $\delta_j = (t_j - o_j) o_j (1 - o_j)$
7. Для останнього прихованого шару визначити помилку кожного елемента $\delta_k = o_k (1 - o_k) \sum_{j=1}^{n} \delta_j w_{k,j}$
8. Повторити крок 7 для всіх інших шарів
9. Для всіх шарів перерахувати значення ваг кожного елемента $\Delta w_{i,j}(n+1) = \eta(\delta_j o_i) + \alpha \cdot \Delta w_{i,j}(n)$,

   $w_{i,j} = w_{i,j} + \Delta w_{i,j}$
10. Якщо вхідний зразок був не останнім, прочитати наступний і перейти до кроку 2.
11. Якщо Converge == FALSE, перейти до кроку 1.
12. Кінець

Перелік позначень:

i, j, k – індекси;

x_i - і-ий вхідний сигнал нейрону;

$w_{i,j}$ - ваговий коефіцієнт з'єднання і-го та j-го нейронів;

net_j - сумарне вхідне значення нейрону j;

o_j - вихідне значення нейрону j;

$\frac{1}{1+\exp(-net_j)}$ - функція активації;

δ_j - значення помилки для j-го нейрону;

t_j - еталонне значення j-го нейрону зовнішнього шару;

$\Delta w_{i,j}(n)$ - помилка для вагового коефіцієнта зв'язку між нейронами і та j на n кроці обчислень

η - норма навчання;

α - коефіцієнт інерції;

Converge – ознака припинення навчання.

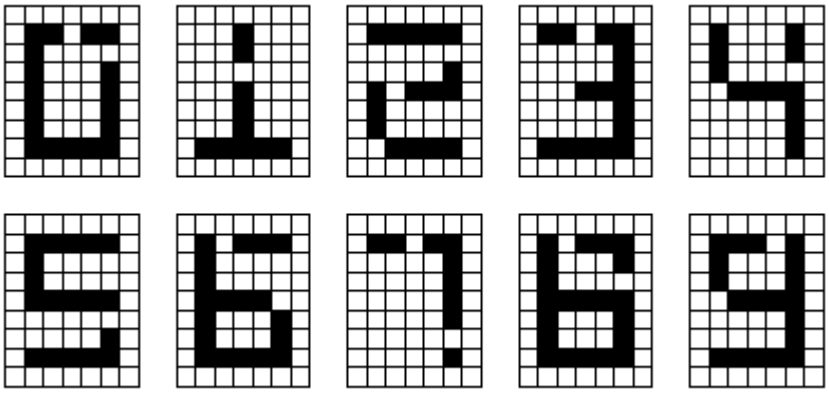


Рис.4. Приклади для тестування нейронної мережі

В звіті представити:

1. Завдання
2. Теоретичні відомості
3. Результати навчання
4. Результати розпізнавання
5. Текст програми
6. Висновки

ПРАКТИЧНА РОБОТА № 3

Навчання багатошарової нейронної мережі алгоритмом зворотнього поширенням похибки (back propagation)

Мета: *Вивчити один із методів функціонування нейронних мереж і принципи його застосування.*

Короткі теоретичні відомості. Нейронна мережа має вигляд (рис. 2.1): Кожен шар нейронної мережі має різну кількість нейронів: **A - *p*, S - *l*, R - *k***. Співвідношення між ***p*, *l*** і ***k*** визначимо пізніше. Відомо, що гіперболічний тангенс обчислюють за формулою:

$$y = thx = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (2.1)$$

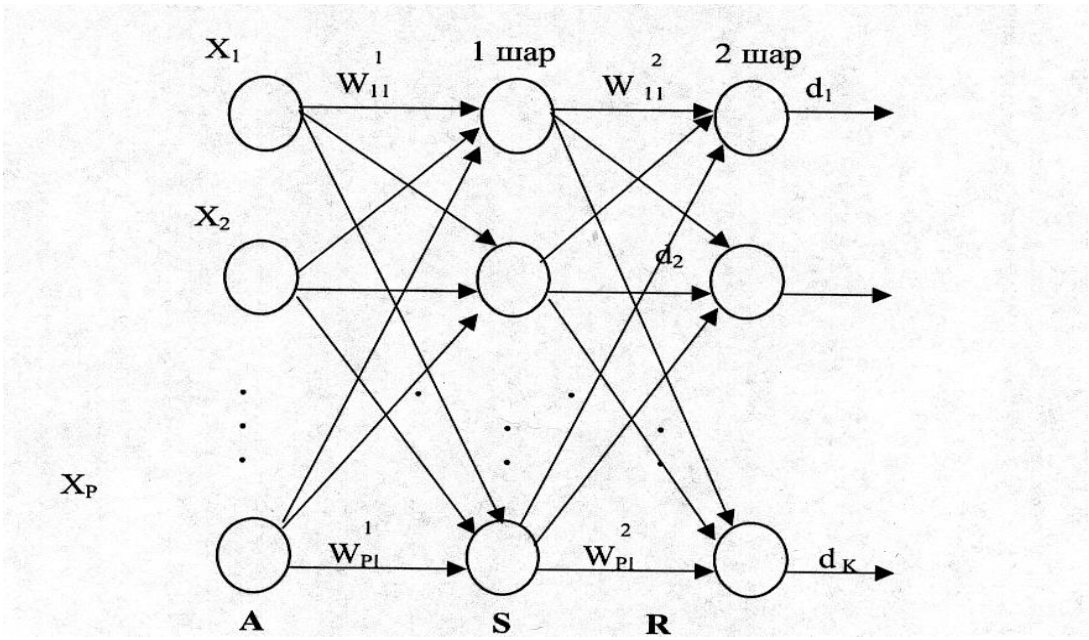


Рис. 2.1. Структура нейронної мережі

Обчислимо $th'x = \frac{1}{ch^2 x}$, де *chx* - гіперболічний косинус. Очевидно, що $(thx)' = 1 - (thx)^2$. В якості початкових даних маємо таблицю 2.1.

Таблиця 2.1 - Початкові дані для навчання нейронної мережі

X_1	x_1^1	x_1^2	x_1^3	. . .	x_1^N
X_2	x_2^1	x_2^2	x_2^3	. . .	x_2^N
• • •	. . .	. . .	. . .	. . .	. . .
X_p	x_p^1	x_p^2	x_p^3	. . .	x_p^N
d_1	d_1^1	d_1^2	d_1^3	. . .	d_1^N
d_2	d_2^1	d_2^2	d_2^3	. . .	d_2^N
• • •	. . .	. . .	. . .	. . .	. . .
d_k	d_k^1	d_k^2	d_k^3	. . .	d_k^N

N - кількість точок спостереження, одержаних в результаті експерименту або статистичних випробувань. Вектор $\vec{X}$ - входи, $\vec{D}$ - бажані або реальні виходи. $\vec{X} = (X_1, X_2, ..., X_p)$, $D = (d_1, d_2, ..., d_k)$. Цільова функція, яку необхідно мінімізувати:

$$E(\omega) = \frac{1}{2}\left[\sum_{k=1}^{N}\sum_{j=1}^{l}(y_{kj}^1 - d_{kj}^1) + \sum_{k=1}^{N}\sum_{j=1}^{l}(y_{kj}^2 - d_{kj}^2)\right]^2 = \frac{1}{2}\sum_{k=1}^{N}\left(\sum_{j=1}^{l}(y_{kj}^1 - d_{kj}^1) + \sum_{j=1}^{l}(y_{kj}^2 - d_{kj}^2)\right)^2 \quad (2.2)$$

Очевидно, що значення d_{kj}^1 (задані виходи першого шару) невідомі. Тому обмежимося виходами шару **R**. Формула (2.2) трансформується в (2.3):

$$E(\omega) = \frac{1}{2}\sum_{k=1}^{N}\sum_{j=1}^{k}(y_{kj}^2 - d_{kj}^2). \quad (2.3)$$

Маємо задачу:

$$E(\omega) \to \min \quad (2.4)$$

Функцію будемо мінімізувати методом градієнтного спуску, що означає настройку вагових коефіцієнтів наступним чином :

$$\Delta\omega_{ij}^{(n)} = -\eta \frac{\partial E}{\partial \omega_{ij}},\ n = \overline{1,2} \qquad (2.5)$$

$\omega_{ij}^{(n)}$ - з'єднує *i*-й нейрон (*n* - 1) шару з *j*-м нейроном *n*-го шару, $0 < \eta < 1$ - коефіцієнт швидкості навчання. Відомо, що

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{dS_j} \cdot \frac{\partial S_j}{\partial \omega_{ij}}\ , \qquad (2.6)$$

де y_j - вихід нейрона, S_j - зважена сума його вхідних сигналів (аргумент активаційної функції). Тут підійде класичний сигмоїд ($y = \frac{1}{1 + e^{\sum \omega_i x_i}}$), або гіперболічний тангенс. Третій множник $\frac{\partial S_j}{\partial \overline{\omega}_{ij}} = y_i^{(n-1)}$ - виходу нейрона попереднього шару. Перший множник в (2.6) легко розкласти наступним чином

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{dS_k} \cdot \frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{dS_k}. \qquad (2.7)$$

В (2.7) сума шукається серед нейронів (***N*** + 1) шару. Введемо нову зміну :

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{dS_j}. \qquad (2.8)$$

Одержимо рекурсивну формул:

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} \cdot \omega_{jk}^{(n+1)} \right] \cdot \frac{dy_j}{dS_j}. \qquad (2.9)$$

що дає змогу обчислити $\delta_j^{(n)}$, знаючи $\delta_j^{(n+1)}$. А для вихідного шару

$$\delta_e^{(n)} = (y_e^{(n)} - d_e) \cdot \frac{dy_e}{dS_e}, \qquad (2.10)$$

для випадку гіперболічного тангенса

$$\delta_e^{(n)} = (y_e^{(n)} - d_e) \cdot (1 - S_e^2) \qquad (2.11)$$

або у випадку сигмоїда

$$\delta_e^{(n)} = (y_e^{(n)} - d_e) \cdot (1 - S_e) \cdot S_e \qquad (2.12)$$

Тоді (2.5) має вигляд

$$\Delta \varpi_{ij}^{(n)} = -\eta \delta_j^{(n)} y_i^{(n-1)} \qquad (2.13)$$

Для того, щоб надати процесу корекції вагів деякої інерційності, що згладжує різкі стрибки при переміщенні по поверхні цільової функції, (2.13) доповнюється значенням зміни ваги на попередній ітерації

$$\Delta \varpi_{ij}^{(n)}(t) = -\eta \cdot (\mu \cdot \Delta \varpi_{ij}^{(n)} \cdot (t-1) + (1-\mu) \cdot \delta_j^{(n)} y_i^{(n-1)}), \qquad (2.14)$$

де μ - коефіцієнт інерційності, t - номер поточної ітерації.

<u>Алгоритм навчання мережі, що зображена на рис. 2.1 із *p*=3, *l*=3, *k*=2.</u>

Подаємо на вхід образ і розраховуємо значення виходів

$$S_j^{(n)} = \sum_{i=0}^{3} y_i^{(n-1)} \cdot \omega_{ij}^{(n)} \qquad (2.15)$$

де 3 - число нейронів в шарі **A**, враховуючи нульовий нейрон із постійним вихідним значенням **1**, що задає зміщення, $y_i^{(n-1)} = x_{ij}^{(n)}$ - i-й вхід нейрона j шару n. В нашому випадку

$$\begin{aligned} S_1^1 &= w_{01}^1 + w_{11}^1 y_1^0 + w_{21}^1 y_2^0 + w_{31}^1 y_3^0, \\ S_2^1 &= w_{02}^1 + w_{12}^1 y_1^0 + w_{22}^1 y_2^0 + w_{32}^1 y_3^0, \\ S_3^1 &= w_{03}^1 + w_{13}^1 y_1^0 + w_{23}^1 y_2^0 + w_{33}^1 y_3^0. \end{aligned} \qquad (2.16)$$

Обчислити:

$$y_1^1 = \frac{1}{1+e^{-S1}}, \quad y_2^1 = \frac{1}{1+e^{-S2}}, \quad y_3^1 = \frac{1}{1+e^{-S3}}. \qquad (2.17)$$

Одинички вгорі замінити на двійки. Знайдемо y_i^2, i =1,2 . Покласти

$$y_1^0 = x_1^1, \quad y_2^0 = x_2^1, \quad y_3^0 = x_3^1. \qquad (2.18)$$

2. Розрахувати для вихідного шару: за формулою (2.12). В нашому випадку

$$\delta_1^{(2)} = (y_1^2 - d_1^1)(1-S_1^2)S_1^2, \qquad (2.19)$$

$$\delta_2^{(2)} = (y_2^2 - d_2^1)(1-S_2^2)S_2^2. \qquad (2.20)$$

Розрахувати за формулами (2.13 або 2.14) зміни вагів. В нашому випадку за (2.13):

$$\Delta w_{11}^2 = -\eta \delta_1^{(2)} y_1^1, \quad \Delta w_{12}^2 = -\eta \delta_{21}^{(2)} y_1^1, \quad \Delta w_{21}^2 = -\eta \delta_1^{(2)} y_2^1, \quad \Delta w_{22}^2 = -\eta \delta_2^{(2)} y_2^1,$$
$$\Delta w_{31}^2 = -\eta \delta_1^{(2)} y_3^1, \quad \Delta w_{32}^2 = -\eta \delta_{21}^{(2)} y_3^1.$$

Розрахувати за формулами (2.9) і (2.13), або (2.14) $\delta^{(n)}$ і $\Delta w^{(n)}$ для всіх інших шарів.

4. Скорегувати всі ваги нейронної мережі

$$w_{ij}^{(n)}(f) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t). \qquad (2.21)$$

5. Якщо помилка мережі істотна, то перейти на 1. Інакше кінець.

Завдання до практичної роботи.

1. Програмно реалізувати наведений алгоритм із ***p***=3, ***l***=3, ***k***=2.

2. Використовуючи дані таблиці 2.2, написати програмний фрагмент для обчислення 20 значень функції по заданих значеннях аргументів та їх варіаціях з кроком ± 1 і знайти середнє значення.

3. Подати задані значення аргументів на вхід нейронної мережі. В якості виходу d_1 вважати обчислене значення функції, в якості d_2 - 1, якщо одержане значення d_1 більше середнього значення - **0**, якщо менше.

4. Задати контрольні приклади та оцінити швидкість і точність алгоритму.

5. В результаті навчання і тестування мережі необхідно побудувати графік залежності ймовірності (частотності) правильної відповіді від:

5.1. Кількості нейронів в проміжному шарі при заданій кількості вхідних нейронів.

5.2. Кількості вхідних нейронів при заданій кількості нейронів в проміжному шарі.

5.3. Кількості прикладів в навчальній вибірці при заданій кількості нейронів у вхідному і проміжному шарах.

5.4. Порога нейронів при фіксованих значеннях інших параметрів.

Таблиця 2.2 - Початкові дані

№ варіанта	x_1	x_2	x_3	Функція
1	1	2	3	$x_1^2 - x_2^2 + x_3^2$
2	5	4	3	$\sin x_1 + \sin x_2 - \sin x_3$
3	9	8	7	$tgx_1 + \sin x_2 - \sin x_3$

4	8	5	3	$\sin x_1 + \sin x_2 - \cos x_3$
5	5	6	7	$tgx_1 + \sin x_2 - tgx_3$
6	1	8	7	$\sin x_1 + tgx_2 - tgx_3$
7	2	5	4	$\cos x_1 + \cos x_2 - \sin x_3$
8	5	7	8	$\ln\cos x_1 + tgx_2 - ctgx_3$
9	2	3	6	$2^{x_i} + \cos x_2 - \sin x_3$
10	7	4	5	$\sin x_2^2 + x_2^2 - tgx_3$
11	2	1	3	$x_1^2 + x_2^2 - x_3^2$
12	3	4	5	$\sin x_1 - \sin x_2 + \sin x_3$
13	5	6	7	$tgx_1 - \sin x_2 + \sin x_3$
14	6	4	2	$\sin x_1 - \sin x_2 + \cos x_3$
15	7	6	5	$tgx_1 - \sin x_2 + tgx_3$
16	5	7	9	$\sin x_1 - tgx_2 + tgx_3$
17	2	4	6	$\cos x_1 - \cos x_2 + \sin x_3$
18	5	7	9	$\ln\cos x_1 - tgx_2 + ctgx_3$
19	2	4	2	$2^{x_i} - \cos x_2 + \sin x_3$
20	7	5	3	$\sin x_2^2 - x_2^2 + tgx_3$
21	1	2	1	$x_1^2 - x_2^2 - x_3^2$
22	3	4	3	$\sin x_1 - \sin x_2 - \sin x_3$
23	7	5	6	$tgx_1 - \sin x_2 - \sin x_3$
24	3	5	7	$\sin x_1 - \sin x_2 - \cos x_3$
25	5	6	5	$tgx_1 - \sin x_2 - tgx_3$
26	1	3	5	$\sin x_1 - tgx_2 - tgx_3$
27	2	3	4	$\cos x_1 - \cos x_2 - \sin x_3$

Контрольні запитання

1. В чому полягає зміст алгоритму зворотнього поширення похибки?

2. За яким оптимізаційним методом функціонує алгоритм?
3. Яке значення для алгоритму має функція похибки?
4. Для розв'язку яких задач застосовується даний алгоритм?
5. Які переваги та недоліки *back propagation*?
6. Чим відрізняється навчання з учителем від навчання без учителя?
7. Які існують методи навчання нейронних мереж?
8. До якого класу навчання відноситься навчання методом оберненого поширення похибки - до навчання без учителя чи з учителем?
9. Чому порядок пред'явлення прикладів в навчальній вибірці може впливати на якість навчання?
10. Як впливає зменшення кількості вхідних нейронів на функціонування мережі?
11. Які властивості повинна мати функція активації при використанні алгоритма оберненого поширення похибки?

Література

1. Тимощук П. В. Штучні нейронні мережі : навч. посіб. Львів: Видавництво Львівської політехніки, 2011, 444 с.
2. Руденко О.П., Бодянський Є.В. Штучні нейронні мережі: навч. посібник. Харків : Тов. “Компанія СНІТ”, 2014, 404 с.
3. Куссуль Н.М, Шелестов А.Ю.,Лавренюк А.М. Інтелектуальні обчислення : навчальний посібник. Київ: Наукова думка, 2006, 314 с.
4. Зайченко Ю.П. Основи проектування інтелектуальних систем. навчальний посібник. Київ.: Видавничий Дім “Слово”, 2011, 352 с.
5. Ямпольський Л. С., Лісовиченко О. І., Олійник В.В. Нейротехнології та нейрокомп’ютерні системи . Київ: «Дорадо» , 2016, 576 с.