Міністерство освіти і науки України
Донбаська державна машинобудівна академія

Кафедра Інноваційних технологій управління

Основи сучасних теорій моделювання процесів (ч.2)

конспект лекцій для студентів спеціальності
131-Прикладна механіка другого освітньо-кваліфікаційного рівня

Краматорськ 2021

ЗМІСТ

ВСТУП

Розв'язання завдань розпізнавання образів, кількісного та якісного прогнозування, керування складними об'єктами і процесами вимагає створення та використання інтелектуальних сенсорних систем. Одним з найпотужніших базисів для побудови таких систем є штучні нейронні мережі.

Метою другої частини конспекта є систематизований виклад основних понять, моделей і методів побудови штучних нейронних мереж, які можуть використовуватися для побудови інтелектуальних систем при вирішенні практичних завдань розпізнавання образів, прийняття рішень, класифікації та прогнозування, технічного і біомедичного діагностування.

У першому розділі розглянуто основні поняття нейроінформатики. Описано загальну структуру та принципи перетворення інформації біологічних нейронів. Розглянуто основні елементи для створення штучних нейронів. Наведено класифікацію та описано властивості нейромереж. Надано загальне уявлення про синтез нейромереж. Розглянуто основні методи оптимізації в задачах синтезу нейромереж.

Другий розділ присвячено нейронним мережам прямого поширення сигналу. Описано одношаровий персептрон та метод навчання Уідроу-Хоффа. Розглянуто структуру та описано функціонування багатошарових нейромереж (БНМ). Наведено загальне уявлення про навчання БНМ. Розглянуто метод зворотного поширення помилки. Наведено опис структури та функціонування радіально-базисних нейромереж, а також методів їхнього навчання.

У третьому розділі розглянуто нейронні мережі зі зворотними зв'язками. Описано структуру та принципи функціонування нейромереж Хопфілда. Значну увагу приділено розгляду методів навчання нейромереж Хопфілда. Зокрема описано ефект рознасичення синаптичної матриці ваг для збільшення обсягу пам'яті мереж Хопфілда. Описано структуру та функціонування нейромережі Елмана.

Четвертий розділ присвячено нейронним мережам із боковими зв'язками. Наведено опис нейронних мереж Кохонена, а також розглянуто методи їхнього навчання.

У п'ятому розділі наведено короткий огляд базових парадигм глибоких нейронних мереж.

Шостий розділ містить огляд програмних засобів для моделювання нейронних мереж: Python, Matlab та Statistica Neural Networks.

Для спрощення **самостійного опрацювання** та кращого засвоєння матеріалу книги наприкінці кожного розділу наведено контрольні питання, а також практичні та тестові завдання. Також, передбачається вивчення пакета NeuroPro 0.25 в лабораторному практикумі

.

РОЗДІЛ 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ТА ВЛАСТИВОСТІ НЕЙРОННИХ МЕРЕЖ

Нейроінформатика – це розділ штучного інтелекту, який вивчає принципи подання та обробки інформації у штучних нейронних мережах.

Штучні нейронні мережі (НМ) – математичні моделі, а також їх програмні або апаратні реалізації, побудовані за принципами подання й обробки інформації у *біологічних нейронних мережах* – мережах нервових кліток живого організму.

1.1 Біологічний нейрон

Нервова система людини складається з клітин, які називаються *нейронами*, і має надзвичайну складність: близько 10^{11} нейронів беруть участь у близько 10^{15} передавальних *зв'язках*, що мають довжину метр і більше. Кожен нейрон має багато якостей, спільних з іншими клітинами, але його унікальною здатністю є прийом, обробка і передача електрохімічних сигналів по нервовим шляхам, що утворюють комунікаційну систему мозку.

На рис. 1.1 показана спрощена схема *біологічного нейрона*. *Дендрити* (деревоподібні відростки на входах нейрона) йдуть від *тіла нейрона* до інших нейронів, де вони приймають сигнали в *точках з'єднання*, які називаються *синапсами*.

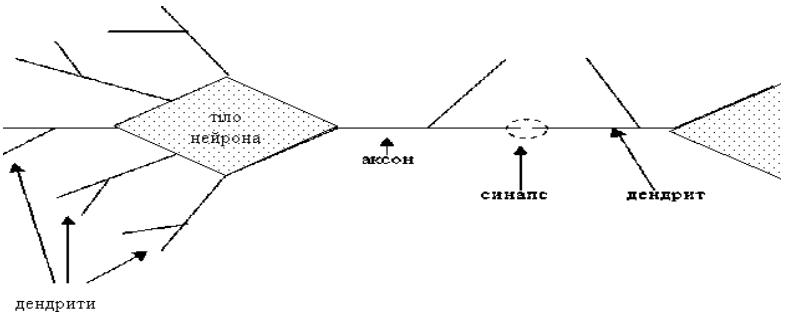


Рисунок 1.1 – Спрощена схема біологічного нейрона

Прийняті синапсом *вхідні сигнали* підводяться до тіла нейрона. Тут вони підсумовуються, причому одні входи прагнуть збудити нейрон, інші – перешкодити його збудженню. Коли сумарне *збудження* в тілі нейрона перевищує деякий *поріг*, нейрон збуджується, посилаючи по *аксону* (відросток на виході нейрона) сигнал іншим нейронам.

У цієї основної функціональної схеми багато ускладнень і виключень, проте більшість штучних НМ моделюють лише ці прості властивості.

1.2 Моделі штучних нейроелементів

Штучний нейрон (*формальний нейрон*, *нейроподібний елемент*) – це примітивний обчислювальний пристрій (або його модель), що має кілька входів і один вихід, і є основним обчислювальним елементом НМ.

Схему штучного нейрона зображено на рис. 1.2.

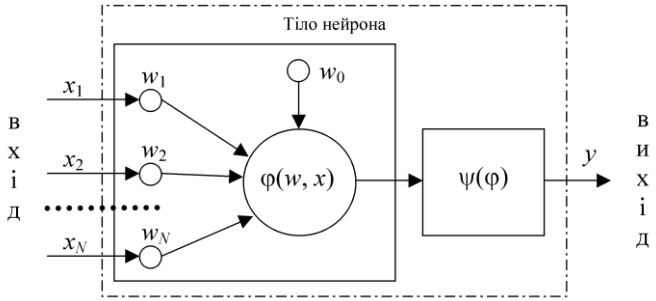


Рисунок 1.2 – Формальний нейрон

На вхід одношарового нейрона надходить *вхідний вектор* – набір вхідних сигналів $x=\{x_j\}$, $j = 1, 2, ..., N$, де N – кількість входів.

Кожний вхідний сигнал x_j зважується (масштабується певним чином) відносно зіставленої йому *ваги зв'язку* (*вагового коефіцієнта*) w_j, яка моделює перетворення сигналу у синапсі (міжнейронному контакті).

Дискримінантна (*вагова*, *постсинаптична*) *функція* нейрона φ поєднує зважені вхідні сигнали, отримуючи *постсинаптичний потенціал* та подає його значення до *функції активації*

(*передатної функції*) ψ, яка видає скалярне значення, що видається на *виході нейрона*. Таким чином, формальний нейрон реалізує скалярну функцію векторного аргументу, моделюючи перетворення вхідних сигналів у синапсах та тілі біологічного нейрона.

Отже, *математична модель функціонування штучного нейрона* описується співвідношенням: $y = \psi(\varphi(w, x))$, де x – вектор вхідних аргументів (сигналів); y – значення на виході нейрона; ψ – функція активації; φ – дискримінантна функція; $w=\{w_j\}$ – вектор, що містить значення вагових коефіцієнтів w_j і значення *зсуву* (*порогове значення*) w_0.

Набір вагових коефіцієнтів нейрона w моделює його *пам'ять*. Тому нейрони можна розглядати як запам'ятовуючі пристрої. У той же час нейрони можуть розглядатися як примітивні процесори, що здійснюють обчислення значення функції активації на основі значення дискримінантної функції вхідних сигналів і ваг.

Дискримінантні функції, як правило, використовують такі:

– *зважена сума*: $\varphi(w,x) = \sum_{j=1}^{N} w_j x_j + w_0$;

– *зважений добуток*: $\varphi(w,x) = \prod_{j=1}^{N} w_j x_j = w_0 \prod_{j=1}^{N} x_j$;

– *евклідова відстань*: $\varphi(w,x) = \sum_{j=1}^{N} \left(w_j - x_j\right)^2$.

Функція активації $\psi(x)$, де x – аргумент функції активації, повинна бути обмеженою (на інтервалі значень $x \in (-\infty, a]$ приймати значення y^0, на інтервалі $x \in [b,+\infty)$ – приймати значення y^1, на інтервалі $x \in (a, b)$ приймати значення y: $y^0 \le y \le y^1$, де y^0 та y^1 – деякі постійні мінімальне та максимальне значення, a та b – деякі константи, причому: $a \le b$) і монотонною (на інтервалі $x \in (a, b)$ $\Delta\psi(x) = \psi(x+\Delta x) - \psi(x)$) і не повинна змінювати знак при $\Delta x > 0$ та x, $x+\Delta x \in (a, b)$.

Як функція активації, як правило, застосовуються такі функції:

– *лінійна* приймає значення в діапазоні $(-\infty;+\infty)$:

$$\psi(x) = cx,$$

де c – константа, як правило, c =1;

– *лінійна біполярна з насиченням:*

$$\psi(x) = \begin{cases} 1, x > a_2; \\ Kx, a_1 \le x \le a_2; \\ -1, x < a_1, \end{cases}$$

де K, a_1, a_2 – константи;

– *лінійна уніполярна з насиченням:*

$$f(x) = \begin{cases} 1, x \ge \dfrac{1}{2a}; \\ ax + 0{,}5, |x| < \dfrac{1}{2a}; \\ 0, x \le -\dfrac{1}{2a}, \end{cases}$$

де a – константа.

– *порогова (Хевісайда)* приймає значення в діапазоні [0;1]:

$$\psi(x) = \begin{cases} 1, x \ge 0; \\ 0, x < 0; \end{cases}$$

– *порогова знакова (біполярна)* приймає значення в діапазоні [–1; 1]:

$$\psi(x) = \begin{cases} 1, x \ge 0, \\ -1, x < 0; \end{cases}$$

– *сигмоїдна логістична (уніполярна)* приймає значення в діапазоні (0;1):

$$\psi(x) = \frac{1}{1 + e^{-cx}},$$

де c – константа, як правило, $c = 1$.

– *гіперболічний тангенс* (*біполярна*):

$$\psi(x) = \tanh(Kx) = \frac{e^{Kx} - e^{-Kx}}{e^{Kx} + e^{-Kx}},$$

де K – константа.

– *косинусоїдальна з насиченням (уніполярна):*

$$\psi(x) = \begin{cases} 1, x \geq \frac{\pi}{2}; \\ \frac{1}{2}\left(1 + \cos\left(x - \frac{\pi}{2}\right)\right), |x| < \frac{\pi}{2}; \\ 0, x \leq -\frac{\pi}{2}; \end{cases}$$

– *синусоїдальна з насиченням* (*біполярна*):

$$\psi(x) = \begin{cases} 1, x \geq a; \\ \sin x, |x| < a; \\ -1, x \leq -a, \end{cases}$$

де a – константа.

– *радіально-базисна* (*Гауса*) приймає значення в діапазоні $(0; +\infty)$:

$$\psi(x) = e^{-cx^2},$$

де c – константа;

– *winner-take-all* (WTA – *переможець отримує усе*):

$$\psi^{(\mu,i)}(\{\varphi^{(\mu,j)}\}) = \begin{cases} 1, \varphi^{(\mu,i)} \leq \min_j\{\varphi^{(\mu,j)}\}; \\ 0, \text{інакше.} \end{cases}$$

Незважаючи на те, що лінійні функції є найбільш простими, їх застосування обмежене, в основному, найпростішими НМ, що не містять прихованих шарів, в яких, крім того, існує лінійна залежність між вхідними і вихідними змінними. Такі мережі мають обмежені можливості. Багатошарова ж лінійна мережа може бути замінена еквівалентною одношаровою.

Проте використання лінійних активаційних функцій не позбавлене сенсу, у багатошарових НМ для розширення можливостей мережі застосовують нелінійні функції активації.

При побудові НМ часто доводиться працювати як з активаційною функцією, так і з її першою похідною. У цих випадках необхідним є використання як активаційної монотонної диференційованої та обмеженої функції. Особливо важливу роль відіграють такі функції при моделюванні нелінійних залежностей між вхідними і вихідними змінними. Це так звані *сигмоїдальні функції*. Функція $f(\bullet)$ називається сигмоїдальною, якщо вона є монотонно зростаючою, диференційованою і задовольняє умові:

$$\lim_{\lambda \to -\infty} f(\lambda) = k_1, \lim_{\lambda \to \infty} f(\lambda) = k_2, k_1 < k_2 .$$

Сигмоїдальну функцію за аналогією з електронними системами можна вважати нелінійною підсилювальною характеристикою штучного нейрона. Центральна область такої функції, що має великий коефіцієнт підсилення, вирішує проблему обробки слабких сигналів, а області з падаючим посиленням на позитивному і від'ємному кінцях слугують для обробки сильних збуджень. Таким чином, нейрон функціонує з великим посиленням у широкому діапазоні рівнів вхідного сигналу.

Функція *Softmax* (нормована експоненційна функція) – це узагальнення логістичної функції, що "стискує" K-вимірний вектор $\mathbf{z}$ із довільним значеннями компонент до K-вимірного вектора $\sigma(\mathbf{z})$ з дійсними значеннями компонент в області [0, 1] що в сумі дають одиницю:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1,...,K.$$

Функція *ReLU* (Rectified Linear Units – ненасичуваний лінійний модуль) визначається формулою:

$$\psi(\varphi) = \max(0, \varphi).$$

1.3 Класифікація та види моделей нейромереж

У загальному випадку *нейронна мережа* являє собою сукупність нейронів, зв'язаних певним чином. Основними відмінностями нейромережевих моделей є способи зв'язку

нейронів між собою, функції нейроелементів, а також механізми та напрямки розповсюдження сигналів по мережі.

Конкретний вид виконуваного НМ перетворення інформації обумовлюється характеристиками нейроподібних елементів і особливостями архітектури мережі: топологією міжнейронних зв'язків, вибором певних підмножин нейроподібних елементів для введення і виведення інформації, способами навчання мережі, наявністю або відсутністю конкуренції між нейронами, напрямком і способами управління і синхронізації передачі інформації між нейронами.

Класифікація НМ:

– за *типом вхідної інформації* виділяють: *аналогові* (використовують інформацію у формі дійсних чисел) та *бінарні* (виконавчі) НМ (оперують з інформацією, описаною в двійковому вигляді);

– за *типом функції активації нейронів* виділяють мережі: *неперервні* (аналогові, дійсні, диференційовані – мережі, кожен елемент яких реалізує неперервно диференційовану функцію), *дискретні* (бінарні, порогові, недиференційовані – мережі, кожен елемент яких реалізує недиференційовану функцію), *дискретно-неперервні* (містять елементи з диференційованими і недиференційованими функціями);

– *за типом графа міжнейронних зв'язків* виділяють: *мережі без циклів* (*ациклічні мережі*); *мережі з циклами* (поділяються на *рівноважні мережі з циклами* і *мережі з обмеженими циклами*);

– *за типом структур нейронів* виділяють: *гомогенні* (*однорідні*) НМ (складаються з нейронів одного типу з єдиною функцією активації) та *гетерогенні* (*неоднорідні*) НМ (містять нейрони з різними функціями активації);

– *за кількістю шарів нейронів* виділяють: *одношарові* НМ (містять один шар нейронів) та *багатошарові* НМ (БНМ – містять більше одного шару взаємопов'язаних нейронів);

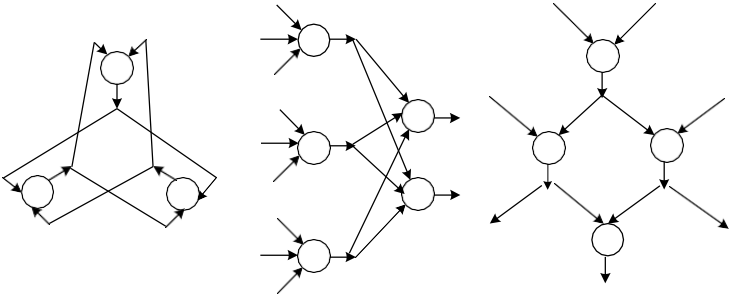
– *за типом дискримінантної функції* виділяють мережі; *зважені* (із вагами зв'язків) та *без ваг зв'язків*;

– *за принципом синтезу* виділяють: *мережі, що навчаються* (графи міжнейронних зв'язків та ваги входів змінюються при виконанні методу навчання) та мережі, що конструюються (кількість і

тип нейронів, граф міжнейронних зв'язків, ваги входів нейронів визначаються при створенні НМ, виходячи з розв'язуваної задачі);

– *за топологією зв'язків* виділяють (див. рис. 1.3): *повнозв'язні* (*інтерактивні*) мережі (усі нейрони пов'язані за принципом «кожний з кожним»: кожен нейрон передає свій вихідний сигнал іншим нейронам, включаючи самого себе, і вихідні сигнали мережі можуть бути всі або деякі вихідні сигнали нейронів після декількох тактів функціонування мережі; всі вхідні сигнали подаються всім нейронам), *неповнозв'язні* (*шаруваті*, *багатошарові*, *ієрархічні*) мережі (мають зазвичай шарувату організацію. У них різняться латеральні (бічні) зв'язки, які охоплюють нейрони одного шару, і проекційні (аферентні), що з'єднують шари нейронів. Частина нейронів має додаткові зовнішні входи, які утворюють рецепторне поле. Нейрони першого шару отримують вхідні сигнали, перетворять їх і через точки галуження передають нейронам наступного шару і так далі до останнього шару, який видає вихідні сигнали. У цьому випадку інформація рухається по висхідній від вхідного до вихідного прошарку і кожен наступний шар забезпечує як би більш високий рівень її обробки, ніж попередній. Число нейронів у кожному шарі може бути будь-яким і ніяк заздалегідь не пов'язано з кількістю нейронів в інших шарах) *ієрархічно інтерактивні* мережі (шари різного рівня пов'язані двосторонніми зв'язками і наявні зв'язки між елементами одного шару, але структура зв'язків не є однорідною, як в повністю інтерактивній мережі); *слабозв'язні* (*з локальними зв'язками*) мережі (нейрони розташовуються у вузлах прямокутної або гексагональної решітки, кожен нейрон зв'язаний з чотирма (окіл фон Неймана), шістьма (окіл Голео) або вісьмома (окіл Мура) своїми найближчими сусідами);

– *за характером зв'язків* виділяють: *мережі прямого поширення* (*мережі без зворотних зв'язків*, feedforward – сигнал по мережі проходить тільки в одному напрямку: від входу до виходу) та *рекурентні мережі* (*із зворотними зв'язками*, *зі зворотним поширенням інформації*, feedforward / feedback – характеризуються як прямим, так і зворотним поширенням інформації між шарами НМ). Серед рекурентних мереж, у свою чергу, виділяють мережі: *релаксаційні* (циркуляція інформації відбувається до тих пір, поки не перестануть змінюватися вихідні



а *б* *в*

Рисунок 1.3 – НМ з різною топологією зв'язків:

а – повнозв'язна мережа; *б* – шарувата мережа; *в* – слабозв'язана мережа

значення НМ – стан рівноваги) та *шаруваті мережі із зворотними зв'язками* (у них відсутній процес релаксації). Серед шаруватих мереж із зворотними зв'язками виділяють: *шарувато-циклічні мережі* (відрізняються тим, що шари замкнуті у кільце: останній шар передає свої вихідні сигнали першому; всі верстви рівноправні і можуть, як отримувати вхідні сигнали, так і видавати вихідні), *шарувато-повнозв'язні мережі* (складаються з шарів, кожен з яких представляє собою повнозв'язну мережу, а сигнали передаються як від шару до шару, так і всередині шару; в кожному шарі цикл роботи розпадається на три частини: прийом сигналів з попереднього шару, обмін сигналами всередині шару, вироблення вихідного сигналу і передача до подальшого шару), *повнозв'язно-шаруваті* (за своєю структурою аналогічні шарувато-повнозв'язним, але функціонуючи по-іншому: у них не розділяються фази обміну всередині шару і передачі наступного шару; на кожному такті нейрони всіх шарів приймають сигнали від нейронів як свого шару, так і наступних), *рециркуляційні мережі* (характеризуються прямим $y = f(x)$, і зворотним $x = f(y)$ перетворенням інформації; в рециркуляційних мережах навчання проводиться без вчителя, тобто вони самоорганізуються у процесі роботи) та *мережі з боковими зворотними зв'язками* (laterally connected);

– *за характером поділу зв'язків* виділяють: *немонотонні мережі* (не дозволяють визначити, як вплине зміна будь-якого

внутрішнього параметра мережі на вихідний сигнал) та *монотонні мережі* (кожен шар мережі крім останнього поділяється на два блоки: збудливий і гальмуючий. При цьому всі зв'язки в мережі влаштовані так, що елементи збудливої частині шару збуджують елементи збудливою частини наступного шару і гальмують елементи наступного шару. Аналогічно, які гальмуючі елементи збуджують гальмуючі елементи і гальмують збуджуючі елементи наступного шару (назви «гальмуючий» і «збуджуючий» відносяться до впливу елементів обох частин на вихідні елементи). Для нейронів монотонних мереж необхідна монотонна залежність вихідного сигналу нейрона від параметрів вхідних сигналів. Відзначимо, що для мереж з сигмоїдними елементами вимога монотонності означає, що ваги всіх зв'язків повинні бути не негативні);

– *за типом методу навчання* (характером налаштування синапсів) виділяють: *мережі з динамічними зв'язками і ітеративним навчанням, заснованому на принципі корекції помилок* (Базуються на запропонованій фізіологами моделі повторення шляхів нервового збудження. Основний метод навчання – зворотне поширення помилки широко використовується в сучасних нейрокомп'ютерів. Головною його перевагою є асимптотична збіжність процесу навчання, гарантуюча потенційну досяжність необхідної точності реакції НМ. Недолік методу полягає в необхідності багаторазового повторення ітерацій навчання на всьому обсязі запам'ятовуваних даних. Велика витрата часу й обчислювальних ресурсів при навчанні обмежували можливість застосування в адаптивних системах реального часу методів, заснованих на методі зворотного поширення), *мережі з фіксованими зв'язками і неітеративним навчанням* (Вони використовують методи прямого обчислення значення матриці зв'язків, які забезпечують запам'ятовування інформації без повторення, що прискорює процес навчання у порівнянні з ітеративними методами. Однак вони характеризується надмірною кількістю нейронів і низьким обсягом інформації, що запам'ятовується. Тому вони поки не отримали широкого поширення і застосовуються, в основному, в дослідницьких проектах);

– *за характером навчання* виділяють: *мережі з контрольованим* (*піднаглядним*, *з вчителем*, *з супервізором*) *навчанням* (коли відомим є вихідний простір рішень НМ; при навчанні порівнюють заздалегідь відомий вихід з отриманими значеннями) та *мережі з неконтрольованим* (*без нагляду*, *без вчителя*, *без супервізора*) *навчанням* (коли НМ формує вихідний простір рішень тільки на основі вхідних впливів, навчається, не знаючи заздалегідь правильних вихідних значень, але групуючи «близькі» вхідні вектори так, щоб вони формували один і той же вихід мережі; неспостережне навчання використовується, зокрема, при вирішенні задачі кластеризації) та *мережі зі змішаним навчанням* (коли частина ваг визначається при спостеріганні, а частина – при неспостережному навчанні; навчання здійснюється шляхом пред'явлення прикладів, що складаються з наборів вхідних даних у сукупності з відповідними результатами, при спостережному навчанні і без останніх при неспостережному);

– *за методом навчання* виділяють мережі: з неітеративним навчанням, з методом зворотного поширення помилки, з конкурентним навчанням, з використанням правила Хебба, з гібридним навчанням (в них застосовуються різні методи навчання) та інші;

– *за типом часу функціонування* виділяють мережі: *з неперервним часом* (аналогові мережі), *з дискретним асинхронним часом* (у кожен момент часу лише один нейрон змінює свій стан) та *з дискретним часом, що функціонують синхронно* (стан змінюється відразу у цілої групи нейронів);

– *у залежності від врахування попереднього стану мережі* виділяють мережі: *статичні* (не мають у своїй структурі ні зворотних зв'язків, ні динамічних елементів, а вихід залежить від заданої множини на вході і не залежить від попередніх станів мережі), *нечіткі* (поєднують в собі БНМ і нечітку систему, їх перший шар нейронів реалізує етап введення нечіткості (фаззіфікації), другий шар відображає сукупність нечітких правил, третій шар реалізує дефаззіфікацію – функцію приведення до чіткості), *нетрадиційні* НМ;

– *за типом розв'язуваних задач* виділяють мережі для: обробки та фільтрації даних, категоризації і таксономії даних, пошуку закономірностей у даних, заповнення прогалин у таблицях даних,

візуалізації та картографування даних, розпізнавання (класифікації) образів, непараметричної апроксимації залежностей за точковими даними, побудови баз даних великої ємності з швидким асоціативним пошуком інформації за неповними вхідними даними, розробки пристроїв асоціативної пам'яті, побудови експертних систем, яких навчають, на прикладах, здобуття знань з даних, адаптивного управління складними об'єктами і процесами, трудомістких задач оптимізації (типу задач про комівояжера і т. п.), шифрування, дешифрування, стиснення інформації, перекладу тексту;

– *за областю застосування* виділяють мережі для: розпізнавання сигналів, мови, зображень і тексту, технічної та біомедичної діагностики, моделювання залежностей в природничих науках і техніці, соціально-економічного прогнозування, управління в техніці та економіці, побудови інформаційно-пошукових систем, криптографії.

Для використання на практиці НМ реалізують у вигляді ***нейрокомп'ютера*** – обчислювальної системи, архітектура якої спеціалізована на виконанні операцій, адекватних структурі НМ.

Нейрокомп'ютери якісно відрізняються від усіх попередніх поколінь ЕОМ тим, що в них відсутні заздалегідь створені методичні програми і що вони, аналогічно людському мозку, здатні навчатися на окремих прикладах. У звичайних ЕОМ елементи схем з'єднані послідовно, кожен елемент з'єднаний тільки з двома-трьома елементами, так що сигнал обробляється поетапно, крок за кроком. Однак у НМ елементи мають множину паралельних з'єднань, причому кожен елемент з'єднаний майже з кожним. Через це вхідний сигнал поширюється по всій мережі, і всі елементи мережі працюють паралельно, реалізуючи, як кажуть, масовано-паралельні обчислення. Цим пояснюється можливість вирішувати складні обчислювальні задачі в реальному часі, справлятися з непередбаченими ситуаціями і навіть синтезувати знання з даних майже без участі людини.

Виділяють три *рівні нейрокомп'ютерів*:

– рівень 1 – імітаційна модель НМ на звичайній ЕОМ;

– рівень 2 – плата або приставка до персонального комп'ютера;

– рівень 3 – повнофункціональний нейрокомп'ютер на мікрочіпах.

1.4 Властивості штучних нейромереж

Переваги НМ і нейрокомп'ютерів полягають у тому, що вони дають стандартний спосіб рішення багатьох нестандартних завдань (неважливо, що спеціалізована машина краще вирішить один клас задач, важливіше, що один нейрокомп'ютер вирішить і цю задачу, і другу, і третю – і не треба кожен раз проектувати спеціалізовану ЕОМ – нейрокомп'ютер зробить все сам і майже не гірше); замість програмування використовується навчання (нейрокомп'ютер вчиться – потрібно лише формувати навчальні задачники, праця програміста, розпорядчого машині всі деталі роботи, заміщається працею вчителя, що створює «освітнє середовище», до якого пристосовується нейрокомп'ютер); нейрокомп'ютери особливо ефективні там, де необхідно подобу людської інтуїції і важко створити явний метод.

Основними ***властивостями нейромереж*** є:

– *однорідність* нейроелементів і базових операцій, а також технологічна простота різних способів їх фізичної реалізації: НМ за аналогією з мозком будуються з множини простих уніфікованих типових елементів (нейронів), що виконують елементарні дії (множення, додавання, обчислення найпростішої нелінійної функції) і з'єднаних між собою різними зв'язками;

– *можливість реалізації нелінійних відображень* шляхом використання нелінійних функцій активації нейронів (це важливо для вирішення завдань управління з істотними нелінійностями, для яких традиційні підходи поки не дають практично реалізованих рішень);

– *пластичність* – обумовлює складність поведінки НМ, яка розглядається як результат взаємодії багатьох елементів, кожен з яких обмежує дію інших і сам обмежується іншими на шляху до формування глобальної спостережуваною поведінки. Розрізняють *нейронну пластичність* (як пластичні елементи розглядаються нейрони), а також *синаптичну пластичність* (модифікація сили синаптичного зв'язку між нейронами). Оскільки кількість синапсів, як правило, на кілька порядків перевищує кількість нейронів, то мережу з пластичними синапсами буде володіти більшими можливостями, ніж мережа еквівалентного розміру з пластичними нейронами;

– *адаптивність* – здатність до адаптації (*адаптація* – сукупність пристосувальних, реакцій на зміну зовнішнього середовища, що протікають на різних ієрархічних рівнях, та організації сигналів зовнішнього середовища, що впливають на організм. НМ є адаптивними системами, оскільки можуть пристосовуватися до змін зовнішнього середовища за допомогою зміни своєї структури і значень параметрів);

– *здатність до навчання*: НМ здатні удосконалювати свою роботу (навчатися шляхом адаптації), використовуючи приклади для налаштування на вирішення певної задачі;

– *узагальнення* – здатність інтегрувати окремі дані для визначення закономірностей та пролонгації результатів, що дозволяє після навчання на одних даних застосовувати отримані знання для інших даних. Для НМ під властивістю узагальнення розуміється здатність генерувати правильні виходи для вхідних сигналів, які не були враховані в процесі навчання. Однак НМ не завжди може узагальнювати дані. Якщо НМ була перетренована, тобто вона запам'ятала тренувальні дані як таблицю і видає коректні дані лише при точній вказівці «адреси» в ній, то в цьому випадку мережа не зможе коректно інтерполювати вхідні дані між тренувальними. При низькому рівні узагальнення перетворення, здійснюване НМ, недостатньо гладко. *Гладкість* перетворення в НМ безпосередньо пов'язана з необхідністю вибрати найпростішу модель в умовах відсутності апріорної інформації. У даному випадку під найпростішою мається на увазі найбільш гладка функція, що найкращим чином апроксимує перетворення, яке розглядається. Вимога гладкості забезпечує гарні інтерполяційні властивості НМ і гарантує їм мінімальну складність структури, що прямо впливає на обсяг обчислень, які виконуються при навчанні. Іншою вимогою є достатня *репрезентативність* тренувального набору (навчальної вибірки) даних: НМ тим краще узагальнює, чим щільніше і рівномірніше розташовані тренувальні дані у вхідному просторі. Якщо тестові дані завжди будуть надаватися між близько розташованими тренувальними шаблонами, то мережа зможе проводити коректне узагальнення, інтерполюючи вхідні дані;

– *універсальна апроксимація*: НМ здійснюють наближення функцій багатьох змінних за допомогою лінійних операцій і суперпозицій функцій однієї змінної. Радянські вчені

О. М. Колмогоров і В. І. Арнольд показали, що будь-яку неперервну функцію N змінних можна отримати за допомогою операцій додавання, множення і суперпозиції з неперервних функцій однієї змінної. На основі цього доведено ряд теорем про апроксимації неперервних функцій багатьох змінних НМ з використанням довільної неперервної функції однієї змінної. НМ дозволяють з будь-якою точністю обчислювати довільну неперервну функцію. Отже, з їх допомогою можна як завгодно точно апроксимувати функціонування будь-якої неперервної системи. У 1987 році Р. Хехт-Нільсеном була запропонована теорема, яка доводить зображуваність функції багатьох змінних досить загального вигляду за допомогою НМ з прямими повними зв'язками, N нейронами вхідного шару, $(2N+1)$ нейронами прихованого шару з наперед відомими обмеженими функціями активації (наприклад, сигмоїдальними) і N_M нейронами вихідного шару з невідомими функціями активації. З теореми Хехт-Нільсена випливає зображуваність будь-якої багатовимірної функції декількох змінних за допомогою НМ фіксованого розмірності. Таким чином, НМ є універсальними структурами, що дозволяють реалізувати будь-який обчислювальний метод;

– *самоорганізація* являє собою процес динамічної перебудови системи з метою адаптації до зовнішнього середовища. Самоорганізація на рівні організму відбувається за допомогою навчання, в результаті якого динамічно перебудовуються нейронні структури головного мозку. Самоорганізація НМ називається також здатністю до навчання: НМ можуть автономно «вивчати» статичні і динамічні властивості модельованого об'єкта на основі результатів вимірювань, які проводилися в минулому, а потім діяти таким чином, щоб прийняти найкраще рішення при невідомому стані зовнішнього середовища. Звичайні комп'ютери повинні бути попередньо запрограмовані, щоб мати можливість обробляти дані, вони не можуть працювати за межами рішень, що задаються програмою. Таким чином, інженерія знань не може бути повною мірою реалізована на звичайних комп'ютерах, так як вони не можуть приймати рішення в нових зовнішніх умовах;

– *асоціативна пам'ять*: НМ розглядається як мультистабільна система, стійкий стан якої (атрактор) відповідає запам'ятованим

станам. Якщо зовнішній вплив встановлює мережу у стан, близький до одного з атракторів, то, будучи надана сама собі, мережа за одну або декілька ітерацій конвергенції перейде у стан атрактора. Таким чином, якщо початковий стан відображає неповний або спотворений набір раніше запам'ятованих даних, то після конвергенції весь цей набір даних буде відновлений. На цьому базується властивість асоціативної пам'яті, що дозволяє застосовувати нейрокомп'ютери для розпізнавання образів, прогнозування складних процесів і т. п. Обсяг асоціативної пам'яті НМ залежить від кількості стійких станів, положення яких визначається значенням матриці зв'язків, отриманих в результаті навчання. Число таких станів не може бути як завгодно великим. Якщо число запам'ятованих станів перевищує деяку межу, відбувається насичення пам'яті і з'являються неправдиві стійкі стани, що спотворюють вміст асоціативної пам'яті;

– *розподіленість пам'яті* – інформація зберігається за багатьма адресами, розподілених таким чином, що кожен елемент даних представляється шаблоном активності, розподіленим по багатьом обчислювальним елементам, і кожен обчислювальний елемент бере участь в уявленні багатьох різних елементів даних. У звичайних комп'ютерах реалізується локальна пам'ять, або локальне подання, в якому використовується один обчислювальний елемент для кожного елемента даних. На основі розподіленої архітектури подання інформація в НМ може дробитися і оброблятися по частинах;

– *ізоморфізм* НМ топології навчальних даних і структурній інформації про модельовану залежність. В основі принципів побудови штучних НМ закладені: *наївний ізоморфізм Кьолера*, який вимагає, щоб подання події в НМ, мало просторову структуру, що нагадує структуру відображуваного реального об'єкта, а також *ізоморфізм Гріна,* що допускає, щоб подання події в НМ мало таку логічну структуру (не обов'язково фізично реалізовану в просторовій формі), яка дозволяла б розбивати це подання, розчленовувати його і знову відновлювати за допомогою відповідних процедур або процесів направленої уваги так, що встановлюється зв'язок між частинами, поверхні або аспектами явищ реального світу. Відомо, що існують ділянки мозку –

«сенсорні проекційні поля», – в яких справді зберігається просторова структура образів. Однак залишається невідомим, наскільки далеко має поширюватися код уявлень в сенсі просторового ізоморфізму, щоб у ньому враховувалися організуючі властивості досвіду. Показано, що в ряді завдань необхідне структурне подання може бути отримано за допомогою адаптивних процесів і не потрібно, щоб уявлення були повністю ізоморфні піднаглядним явищам;

– *паралельна архітектура, масовий паралелізм і розподіленість обчислень* – обробка інформації в НМ виконується кількома процесорними елементами, в той час як у звичайних комп'ютерах, що мають тільки один центральний процесор, інформація обробляється послідовно, крок за кроком;

– *ієрархічна організація структури, цілісність і дробимість елементів НМ*: багато завдань штучного інтелекту неможливо вирішити без використання ієрархічних структур, що дозволяють будувати моделі складних об'єктів з простіших; робота ієрархічної структури вимагає, щоб інформаційний елемент у кожному ієрархічному рівні вів себе як єдине ціле, але при переході з рівня на рівень допускав дроблення, причому при переході з верхнього ієрархічного рівня на нижній це дроблення відповідає виділенню складових його елементів, а при переході з нижнього рівня на верхній воно відповідає включенню певної частини цього елемента в більш складний об'єкт;

– *попередня організація в навчанні*: незважаючи на те, що НМ мають більш широкі можливості до адаптації і навчання на прикладах порівняно з традиційними методами штучного інтелекту, вирішити на їх основі складні задачі штучного інтелекту за рахунок одного лише навчання, мабуть, неможливо. Необхідно, щоб вихідна структура НМ була попередньо організована. Для виконання цієї умови необхідно мати достатньо розвинені інструментальні засоби, що дозволяють формувати структуру НМ до початку її навчання;

– *функціональна блочність* – полягає у побудові архітектури НМ на стандартизованих функціональних блоках. Для вирішення задач штучного інтелекту необхідно будувати НМ, які мають велику кількість нейронів, тому базовими елементами для інструментальних засобів передорганізації НМ повинні бути

великі функціональні блоки, внутрішня організація і властивості яких визначені заздалегідь і відомі розробнику;

– *поліалгоритмічність* – властивість, що дозволяє одній і тій самій НМ одночасно обробляти вхідну інформацію за різними методами. Поліалгоритмічність пов'язана з наявністю в мережі різних вихідних нейронів, з'єднаних з нейронами інших ієрархічних рівнів. При цьому ланцюжки, по яких збудження передається від входу до вихідних нейронів, мають різну структурну конфігурацію, яка залежно від сполучень схем проміжних нейронів, що включають збуджуючі або гальмують синапси, відповідає різним математичним формулам;

– *логічна прозорість*: логічно прозорою називають НМ, яка вирішує задачу зрозумілим для людини способом, для якої легко сформулювати словесний опис у вигляді явного методу. Така мережа повинна мати мінімальну структурну складністю і при цьому задовольняти вимогам (перевагам) користувача і (або) вимогам методу автоматизованого здобуття знань до виду результуючої мережі;

– *здатність отримувати знання з даних*: в процесі навчання НМ засвоює (апроксимує) найбільш загальні закономірності, присутні в навчальних даних, витягуючи тим самим неявні для людини знання з даних. Труднощі використання таких мереж складаються в нашій обмеженій здатності пояснити, як НМ вирішує завдання: внутрішні уявлення НМ, що виходять в результаті навчання, настільки складні, що їх неможливо проаналізувати, за винятком найпростіших випадків. Спрощення НМ, що досягається контрастуванням, дозволяє отримувати логічно прозорі НМ, на основі структури яких можна побудувати вербальний опис методу отримання відповіді за наявною нейромережевої моделі – витягти явні знання з даних;

– *точність вирішення задач* НМ визначається помилкою розпізнавання (апроксимації залежності) для навчальних даних, а також помилкою розпізнавання (апроксимації залежності) для тестових даних;

– *ефективність (якість) вирішення завдань* НМ визначається точністю (помилкою) рішення задачі для навчальних і тестових

даних, простотою, логічної прозорістю і швидкодією отриманої нейромережевої моделі, а також витратами на побудову нейромережевої моделі (вимоги до апаратних засобів, ітераційність і витрати часу методу навчання);

– *варіативність моделей апроксимованої залежності*: для однієї і тієї ж таблиці даних і різних мереж або однієї мережі, але з різною початковою генерацією вихідних значень набору параметрів, що настроюються, після навчання, спрощення за єдиною схемою та вербалізацією можуть вийти різні НМ (нейромережеві моделі) і, відповідно, кілька методів вирішення однієї і тієї ж задачі. За кінцевою таблицею даних завжди будується кілька методів (моделей) рішення. Далі методи починають перевірятися і конкурувати між собою. Комбінуючи фрагменти декількох методів (моделей), можна сконструювати нову теорію. У силу цього неєдиність одержуваного знання (варіативність, неєдиність нейромережевої моделі) не є недоліком;

– *складність НМ* визначається кількістю і топологією міжнейронних зв'язків, складністю виконуваних перетворень і кількістю нейроелементів;

– *надійність та стійкість мережі до відмов окремих елементів*, складових НМ, проявляється в тому, що відмова одного або декількох нейроелементів мережі не призводить до відмови всієї НМ і не може істотно впливати на роботу мережі в цілому.

1.5 Загальне уявлення про синтез нейромереж

Нехай задана *вибірка даних* $<x, y>$, $x = \{x^s\}$, $y = \{y^s\}$, $s=1, 2, ..., S$, $x^s=\{x^s_j\}$, $j=1, 2, ..., N$, $y^s=\{y^s_p\}$, $p = 1, 2, \ldots, M$, де $<x^s, y^s>$ – s-й *екземпляр*, x^s – набір значень *вхідних ознак*, що характеризує s-й екземпляр, y^s – набір значень *вихідних ознак*, зіставлений s-му екземпляру, s – номер екземпляра, j – номер вхідної ознаки, p – номер вихідної ознаки, S – кількість екземплярів, N – кількість вхідних (описових) ознак, M – кількість вихідних (цільових) ознак.

Тоді ***задача синтезу нейромережевої моделі*** за заданою вибіркою полягає в *ідентифікації її структури і параметрів*: НМ = $<C, W, DF, TF>$, таким чином, щоб значення критерію

оптимальності ξ нейромоделі НМ було мінімальним: $\xi(\text{НМ}, X, Y) \to \min$, де C – матриця, що визначає наявність синаптичних зв'язків між елементами мережі (рецепторами, нейронами); $W = \{B, W(C)\}$; $W(C)$ – матриця вагових коефіцієнтів, відповідних зв'язків, присутніх в мережі НМ; $B = B(C)$ – вектор зміщень (порогів) нейронів мережі; $DF = DF(C)$ – вектор дискримінантних функцій нейроелементів; $TF = TF(C)$ – вектор функцій активації нейронів мережі; $\xi(\text{НМ}, X, Y)$ – критерій, що визначає ефективність використання нейромережевої моделі НМ для апроксимації залежності між набором вхідних параметрів X і відповідним йому вектором значень вихідного параметра Y.

Як правило, в якості *критерію оптимальності нейромоделі* використовується *квадратичний критерій*:

$$\xi = \frac{1}{2}\sum_{s=1}^{S}\sum_{p=1}^{M}\left(y_p^s - \text{НМ}_p^s\left(x^s\right)\right)^2,$$

де НМ_p^s – розрахункове значення на p-му виході нейромоделі НМ при подані на її входи екземпляра x^s.

Процес побудови нейромереж на основі відомої вибірки навчальних даних може бути поданий у вигляді послідовності етапів:

– вибір системи інформативних ознак;

– структурний синтез;

– параметричний синтез;

– оптимізація і спрощення побудованої нейромоделі.

Розглянемо основні задачі етапів процесу побудови НМ.

Вибір системи інформативних ознак: з заданого набору ознак виділяють піднабір ознак меншого обсягу, які несуть у собі достатню для побудови нейромоделі інформацію. Виконання даного етапу забезпечує зменшення кількості входів мережі, зменшення розмірності навчальних і тестових даних, зменшення кількості ваг мережі, спрощення структури мережі, підвищення її узагальнювальних властивостей та інтерпретабельності у порівнянні з використанням великого за розміром первісного набору ознак;

Структурний синтез – етап, на якому формується топологія зв'язків, вибираються нейрони, що надалі визначають принцип

функціонування мережі та її ефективність для розв'язання досліджуваної задачі.

Так, нейромережі, що мають невелику кількість нейронів і лінійні функції активації, як правило, через свої обмежені апроксимаційні здібності не дозволяють вирішувати реальні практичні завдання. У той же час вибір надлишкової кількості нейронів у мережі призводить до проблеми перенавчання та втрати апроксимаційних властивостей нейромоделі.

Оскільки для вирішення більшості практичних задач розпізнавання і оцінювання можуть бути використані нейромережі прямого поширення з нейроелементами, що використовують зважену суму як дискримінантну функцію, вибір виду вагових функцій при структурному синтезі нейромоделей, як правило, не здійснюють.

Головною метою дослідника на етапі структурного синтезу є визначення загального вигляду структури шуканого зв'язку, представленого у вигляді нейромережевої моделі, вихідного параметра і вектора керуючих змінних.

Задача структурного синтезу нейромоделі полягає в пошуку структури моделі виду НМ = <*C*, *TF*>, для якої $\xi(\text{НМ}, X, Y) \rightarrow \min$, де $C = C(L, A)$ – матриця, що визначає наявність синаптичних зв'язків між елементами мережі (рецепторами, нейронами); A – максимально допустима кількість нейронів у мережі.

На даний час не існує стандартних методів, які утворювали б сувору теоретичну базу для вирішення задачі структурного синтезу нейромоделей. Будучи вузловим у процесі синтезу нейромоделей реальних об'єктів, цей етап не має суворих і закінчених математичних рекомендацій щодо його реалізації. Тому його реалізація вимагає спільної роботи фахівця у відповідній предметній області та аналітика, спрямованої на якомога більш глибоке проникнення у фізичний механізм досліджуваного зв'язку.

У ряді випадків структуру нейромоделі можна вибрати апріорі, виходячи з фізичних, хімічних та емпіричних міркувань про характер взаємодії досліджуваного об'єкта з зовнішнім середовищем. Однак для складних об'єктів, процесів і систем, що характеризуються багатомірністю і нелінійністю, загальних апріорних міркувань може виявитися недостатньо.

У цьому випадку завдання вибору структури зводиться до пошуку математичними методами оптимізації за апостеріорними даними з використанням наявної додаткової інформації. Використання таких методів передбачає формування певних гіпотез про топологію мережі, які, як правило, перевіряють на основі критеріїв , що базуються на наступних ідеях:

– досягнення компромісу між складністю моделі і точністю її оцінювання;

– пошук моделі, найбільш стійкої до варіювання складу вибіркових даних, на основі яких вона оцінюється.

Існуючі *методи автоматичного пошуку оптимальної структури нейромережевих моделей*, як правило, використовують *жадібну стратегію пошуку*. Так конструктивні методи починають пошук з мінімально можливою архітектури мережі (нейромережа з мінімальною кількістю шарів, нейронів і міжнейронних зв'язків) і послідовно на кожній ітерації додають нові шари, нейрони і міжнейронні зв'язки. При використанні деструктивних методів на початковій ітерації оцінюється ефективність нейромоделі, що містить максимально допустиму кількість шарів, нейронів і міжнейронних зв'язків, потім у процесі пошуку структура такої моделі скорочується до найбільш прийнятною.

Однак такі методи внаслідок застосування жадібної стратегії досліджують незначну частину простору всіх можливих структур нейромоделей і схильні до потрапляння в локальні оптимуми.

Параметричний синтез: відбувається навчання нейромережевої моделі, тобто підбираються такі значення вагових коефіцієнтів мережі, а іноді і параметрів функцій активації, при яких мережа найбільш ефективним чином дозволяє вирішувати поставлену задачу.

Задача параметричного синтезу нейромоделі заданої структури полягає в пошуку такого набору значень вагових коефіцієнтів і зміщень (порогів), при якому досягається мінімум критерію помилки нейромоделі. Навчання НМ полягає в зміні значень синаптичних ваг в результаті послідовного пред'явлення екземплярів навчальної вибірки.

Розрізняють дві ***концепції навчання нейромереж***: навчання з учителем і навчання без учителя.

Навчання без учителя передбачає використання для налаштування вагових коефіцієнтів мережі тільки значення вхідних ознак екземплярів навчальної вибірки. При цьому реальні значення вихідного параметра невідомі, а процес навчання полягає в об'єднанні близько розташованих екземплярів у кластери. Таким чином, екземпляри, розташовані досить близько одне до одного в просторі ознак, будуть віднесені до одного класу при їх класифікації за допомогою налаштованої мережі.

Навчання з учителем передбачає, що навчальна вибірка містить значення вхідних ознак, що характеризують даний об'єкт, і значення вихідного параметра. При такому вигляді навчання мінімізується сума квадратів різниці між модельними та реальними виходами досліджуваного об'єкта.

У загальному випадку навчання нейронних мереж зводиться до *задачі багатовимірної нелінійної оптимізації*, розв'язуваної, як правило, за допомогою *градієнтних* або *стохастичних* (включаючи еволюційні) методів.

Оптимізація побудованої нейромоделі. На можливість застосування нейромережевих моделей на практиці істотний вплив роблять складність побудованої нейромережі та швидкість обчислення значення цільового параметра по набору даних, що не входить в навчальну вибірку. Тому актуальним є спрощення структури синтезованої нейромоделі.

Завдання оптимізації побудованої нейромоделі полягає в пошуку таких нових значень $C' \subseteq C$, W', DF', TF', при яких досягаються оптимальні значення заданих критеріїв оптимальності $\xi_1, \xi_2, \ldots, \xi_K$, враховують основні характеристики нейромоделі, де K – кількість цільових критеріїв.

В даний час існує два основні *підходи до оптимізації структури* синтезованою нейромережевої моделі:

– методи видалення зв'язків;

– методи видалення нейронів.

Методи видалення зв'язків зменшують кількість елементів матриці ваг нейромережі, спрощуючи таким чином її структуру. При використанні таких методів видаляються зв'язки, які мають

найменші значення синаптичних ваг або надають найменший вплив на ефективність функціонування нейромережевої моделі. Виділяють такі методи видалення зв'язків: метод обнуління найменших ваг, метод оптимального руйнування нейромережі, метод оптимальної розбудови мережі.

Методи видалення нейронів спрощують структуру нейромоделі шляхом виключення з неї нейронів, видалення яких не призводить до значного збільшення помилки мережі. При їх використанні додатково необхідно оцінювати значимість нейронів шляхом обчислення помилки спрощеної моделі. До таких методів відносять: метод видалення нейронів з урахуванням їх важливості і метод видалення нейронів з використанням вартісної функції.

Проте існуючі підходи спрощення нейромоделей, як правило, передбачають використання штрафних функцій, що в багатьох випадках не дозволяє отримувати оптимальну структуру нейромоделі, а іноді призводить до неможливості збіжності процесу спрощення нейромоделі або до видалення значущих зв'язків або нейронів.

1.6 Методи оптимізації в задачах синтезу нейромереж

Для побудови інтелектуальних моделей, як правило, застосовуються оптимізаційні методи, сутність яких полягає в пошуку екстремуму *цільового критерію*, в якості якого звичайно використовується середньоквадратична похибка.

Градієнтні методи – це методи *багатовимірної нелінійної безумовної оптимізації*, що використовують обчислення похідних цільового функціоналу при корекції керованих параметрів для знаходження екстремуму цільової функції.

Нехай задано правило обчислення деякої функції $y = f(w, x)$, де w та x – складені вектори-стовпці змінних і відомі реалізації функції y при відомих значеннях x. Потрібно оцінити невідомі параметри w, таким чином, щоб значення функції y при цих параметрах незначно відрізнялися від відомих реалізацій y^* при однакових заданих значеннях x, тобто необхідно мінімізувати деяку цільову функцію $F(w)$, де w – складений вектор-стовпець змінних, що керуються, за кінцеву кількість ітерацій.

Представимо функцію $F(w)$ у вигляді інтегрального критерію якості

$$F(w) = \frac{1}{k} \sum_{m=1}^{k} Q(\varepsilon(w,m)),$$

де k – номер поточної ітерації, m=1, 2,..., k – номера попередніх ітерацій, а $Q(\varepsilon(w, k))$ – деякий миттєвий критерій якості, що залежить від вектора помилки $Q(\varepsilon(w, m))$: $\varepsilon(w, m) = y(m) - y^*(m)$ і який має вигляд квадратичної форми: $Q(\varepsilon(w, m)) = \varepsilon^T(w, m) R\varepsilon(w, m)$, де R – позитивно визначена матриця.

Градієнтні методи мінімізації інтегрального критерію якості засновані на використанні градієнта цільової функції $F(w)$. Ці методи носять ітеративний характер, оскільки компоненти градієнта виявляються нелінійними функціями.

Усі далі розглянуті градієнтні методи засновані на *ітераційній процедурі*, що реалізована відповідно до формули:

$$w_{k+1}=w_k+\alpha_k p(w_k),$$

де w_k, w_{k+1} – поточне і нове наближення значень керованих змінних до оптимального рішення, відповідно, α_k – *крок збіжності*, $p(w_k)$ – *напрямок пошуку* в N-вимірному просторі керованих змінних. Спосіб визначення $p(w_k)$ і α_k на кожній ітерації залежить від особливостей конкретного методу.

Метод Коші (*метод найшвидшого спуску*) полягає в реалізації правила:

$$w_k = w_{k-1} - \gamma \nabla_w Q(\varepsilon(w_{k-1}, k)) = w_{k-1} - \gamma \frac{\partial Q(\varepsilon(w_{k-1}, k))}{\partial w}.$$

Позначивши поточний градієнт $g = \frac{\partial Q}{\partial w}$, одержимо: $w_{k+1}=w_k-\gamma_k g_k$, де γ_k – швидкість навчання.

Величина γ або покладається постійною, і тоді звичайно послідовність w_k сходиться в околицю оптимального значення w, або вона є спадаючою функцією часу так, як це робиться в стохастичних методах оптимізації й адаптації.

Дана процедура може виконуватися доти, доки значення керованих змінних не стабілізуються, або доки помилка не зменшиться до прийнятного значення.

Слід зазначити, що дану процедуру характеризує повільна швидкість збіжності і можливість потрапляння в локальні мінімуми цільового функціонала.

Метод Ньютона відносно до задачі мінімізації цільової функції $F(w)$ має вигляд:

$$w_k = w_{k-1} - \left(\sum_{m=1}^{k} \frac{\partial}{\partial w} \left(\frac{\partial Q(m)}{\partial w} \right)^{\mathrm{T}} \right)^{-1} \gamma \nabla_w Q\left(\varepsilon\left(w_{k-1}, k\right)\right).$$

Обчислення градієнта $\nabla_w Q(\varepsilon(w_{k-1}, k))$ може бути виконано за допомогою методу Коші.

У результаті задача зводиться до обчислення матриці других похідних $\sum_{m=1}^{k} \frac{\partial}{\partial w} \left(\frac{\partial Q(m)}{\partial w} \right)^{\mathrm{T}}$ мінімізованого функціонала.

Метод Ньютона за певних умов має істотно більшу швидкість збіжності, ніж метод Коші.

Методи спряжених градієнтів утворюють підклас методів, що збігаються квадратично. Методи спряжених градієнтів використовують метод Коші для обчислення похідних цільової функції щодо керувальних змінних.

Всі методи спряжених градієнтів на першій ітерації починають роботу з пошуку у найкрутішому напрямку спуску – протилежному градієнту цільової функції: $p_0 = -g_0$. Це напрямок, у якому цільова функція зменшується найбільш швидко. Однак зазначимо, що це не обов'язково забезпечує найшвидшу збіжність.

Після цього виконується лінійний пошук, щоб визначити оптимальну відстань для руху у поточному напрямку пошуку: $w_{k+1}=w_k+\alpha_k p_k$.

Наступний напрям пошуку визначається так, щоб він був спряженим з попередніми напрямками. Пошук відбувається за спряженим напрямком градієнта, щоб визначити розмір кроку, що мінімізує цільову функцію. Загальна процедура для визначення нового напрямку пошуку поєднує новий найкрутіший напрямок спуску з попереднім напрямком пошуку:

$$p_{k+1} = -g_k + \beta_k p_{k-1}.$$

У більшості методів спряжених градієнтів розмір кроку корегується при кожній ітерації, на відміну від інших методів, де

швидкість навчання використовується для визначення розміру кроку.

Різні версії методів спряжених градієнтів відрізняються способом, за яким обчислюється константа β:

– у *методі Флетчера-Рівса* це правило має вигляд:

$$\beta_k = \frac{g_k^{\mathrm{T}} g_k}{g_{k-1}^{\mathrm{T}} g_{k-1}},$$

де β – відношення квадрата норми поточного градієнта до квадрата норми попереднього градієнта;

– у *методі Полака-Ріб'єра* це правило має вигляд:

$$\beta_k = \frac{\Delta g^{\mathrm{T}}{}_{k-1}\, g_k}{g_{k-1}^{\mathrm{T}} g_{k-1}},$$

де β – внутрішній добуток попередньої зміни в градієнті і поточного градієнта, розділений на квадрат норми попереднього градієнта.

Квазіньютонівські методи подібні методам спряжених градієнтів, оскільки також засновані на властивостях квадратичних функцій. Відповідно до викладених вище методів пошуку рішення здійснюється за системою спряжених напрямків, тоді як квазіньютонівські методи мають позитивні риси методу Ньютона, однак використовують тільки перші похідні. В усіх методах зазначеного класу побудова векторів напрямків пошуку здійснюється за допомогою формули:

$$w_{k+1}=w_k-\alpha_k A_k g_k,$$

де A_k – матриця порядку $N\times N$, що називається *метрикою*. Методи пошуку уздовж напрямків, обумовлених цією формулою, називаються *методами змінної метрики*, оскільки матриця A змінюється на кожній ітерації.

Градієнт g_k може бути знайдений за допомогою методу Коші і, отже, квазіньютонівські методи полягають у знаходженні матриці A_k.

Метод Левенберга-Марквардта є одним з найпопулярніших та ефективних квазіньютонівських методів. В ньому використовується метод Коші, щоб обчислити якобіан J цільової функції щодо керованих змінних. Керовані змінні змінюються відповідно до коригувального правила, що у матричній формі має вигляд:

$$H=J^{\mathrm{T}}J,\ g=J^{\mathrm{T}}e,$$

$$w_{k+1} = w_k - [H_k + \eta I]^{-1} g_k,$$

де J – якобіан, e – вектор помилок, η – скаляр, I – одинична матриця.

На початковій стадії пошуку параметру η_0 приписується велике значення. Якщо після першого кроку отримана точка з меншим значенням цільової функції, то слід встановити: $\eta_{k+1}=\eta^{(-)}\eta_k$, в іншому випадку – встановити: $\eta_{k+1}=\eta^{(+)}\eta_k$ і знову реалізувати попередній крок. Тут $\eta^{(-)}$ та $\eta^{(+)}$ деякі константи, такі що: $\eta^{(-)} < 1$, $\eta^{(+)}>1$.

Градієнтні методи оптимізації передбачають обчислення значень цільової функції, що ускладнює, а в деяких випадках і унеможливлює їх застосування на практиці. Еволюційні методи, на відміну від градієнтних, не вимагають диференційованості цільової функції та не накладають обмежень на її вигляд.

Еволюційний (генетичний) пошук – група багатовимірних, стохастичних, евристичних оптимізаційних методів, заснованих на ідеї еволюції за допомогою природного відбору. Методи генетичного пошуку отримані в процесі узагальнення та імітації в штучних системах таких властивостей живої природи, як природний відбір, пристосовність до змінюваних умов середовища, спадкоємність нащадками життєво важливих властивостей від батьків і т. ін.

Формально *методи генетичного пошуку* можуть бути описані у вигляді такої функції: ГМ = ГМ (P_0, N, L, f, Ω, Ψ, ϑ, T), де $P_0 = \{H_1^0, H_2^0, \ldots, H_N^0\}$ – *початкова популяція* – множина рішень задачі, поданих у вигляді *хромосом;* $H_j^0 = \{h_{1j}^0, h_{2j}^0, \ldots, h_{Lj}^0\}$ – j-та хромосома (рішення) популяції P_0 – набір значень незалежних змінних, поданих у вигляді *генів*; h_{ij}^0 – i-й ген j-ї хромосоми популяції P_0 – значення i-го оптимізованого параметру задачі, що входить в j-те рішення; N – кількість хромосом в популяції (*розмір популяції*); L – *довжина хромосом* (кількість генів); f – цільова функція (*фітнес-функція*, функція пристосованості, функція здоров'я, функція придатності); Ω – *оператор відбору*; Ψ – *оператор схрещування*; ϑ – *оператор мутації*; T – критерії зупинення.

На кожній ітерації генетичного пошуку метод працює не з єдиним рішенням, а із деякою множиною рішень (сукупністю хромосом), за рахунок чого забезпечується паралельність пошуку.

При цьому кожна нова множина рішень залежить лише від попередньої і, в загальному випадку, є кращою за попередню.

Оскільки генетичні методи в процесі пошуку використовують деяке *кодування множини параметрів* замість самих параметрів, то вони можуть ефективно застосовуватися для вирішення задач дискретної оптимізації, визначених як на числових множинах, так і на скінчених множинах довільної природи.

Для роботи генетичних методів як інформація про функцію, що оптимізується, використовуються її значення в точках простору пошуку і не потрібно обчислювати похідні або інші характеристики. Тому дані методи можуть бути застосовані до широкого класу функцій, зокрема до тих, що не мають аналітичного опису. Таким чином, методи генетичного пошуку є достатньо гнучкими і можуть бути застосовані до широкого кола задач, в тому числі до задач, для розв'язування яких не існує загальновідомих методів.

Сутність генетичного пошуку полягає в циклічній заміні однієї популяції наступною, більш пристосованою. Таким чином, популяція існує не тільки в просторі, але і в часі. Часто можна вважати, що вся *популяція* складається в просторі і в часі з дискретних поколінь (генерацій, епох) $P_0, P_1, P_2, ..., P_T$.

Покоління P_{t+1} – це сукупність *особин* (хромосом, рішень), *батьки* яких належать поколінню P_t. Покоління P_0 є початковою популяцією. Процес формування покоління P_0 називається ініціалізацією. Кожне покоління є результатом циклу роботи генетичного методу.

Кожна хромосома (особина, точка в просторі пошуку) оцінюється *мірою її пристосованості* відповідно до того, наскільки є гарним відповідне їй рішення задачі. Пристосованість визначається як обчислена цільова функція (фітнес-функція) для кожної з хромосом.

Правила *відбору* (*селекції*) прагнуть залишити лише ті точки-рішення, де досягається оптимум цільової функції.

Найбільш пристосовані особини дістають можливість відтворювати нащадків за допомогою перехресного *схрещування* з іншими особинами популяції. Це призводить до появи нових особин, які поєднують в собі деякі характеристики, успадковані ними від батьків. Найменш пристосовані особини з меншою

ймовірністю зможуть відтворити нащадків, внаслідок чого ті властивості, якими вони володіли, поступово зникатимуть з популяції в процесі еволюції.

Таким чином, з покоління в покоління, гарні характеристики розповсюджуються по всій популяції. Комбінація гарних характеристик від різних батьків іноді може призводити до появи суперпристосованого нащадка (або *мутанта*), чия пристосованість більша, ніж пристосованість будь-якого з його батьків. Схрещування найбільш пристосованих особин призводить до того, що досліджуються найбільш перспективні ділянки простору пошуку. Зрештою, популяція збігатиметься до оптимального рішення задачі.

Після схрещування інколи відбуваються *мутації* – спонтанні зміни в генах, які випадковим чином розкидають точки по всій множині пошуку.

Процес збіжності за рахунок відбору повинен бути більш виражений порівняно з розкидом точок за рахунок мутації і інвертування, інакше збіжність до екстремумів не матиме місця. Збіжність за рахунок відбору не повинна бути занадто швидкою, інакше всі точки можуть зібратися поблизу локального екстремуму, а інший, можливо глобальний, так і не буде знайдено.

Після отримання нащадків за допомогою схрещування та мутації розмір популяції збільшується. Для подальших перетворень число хромосом поточної популяції зменшується до заданого розміру популяції.

Подальша робота генетичного методу є ітераційним процесом застосування *генетичних операторів* до особин наступного покоління. Генетичні оператори необхідні, щоб застосувати принципи спадковості і мінливості до популяції. Такі оператори володіють властивістю ймовірності, тобто вони не обов'язково застосовуються до всіх особин, що вносить додатковий елемент невизначеності в процес пошуку рішення. Невизначеність не має на увазі негативного чинника, а є своєрідною мірою свободи роботи генетичного методу.

Схему роботи узагальненого генетичного методу наведено на рис. 1.4.

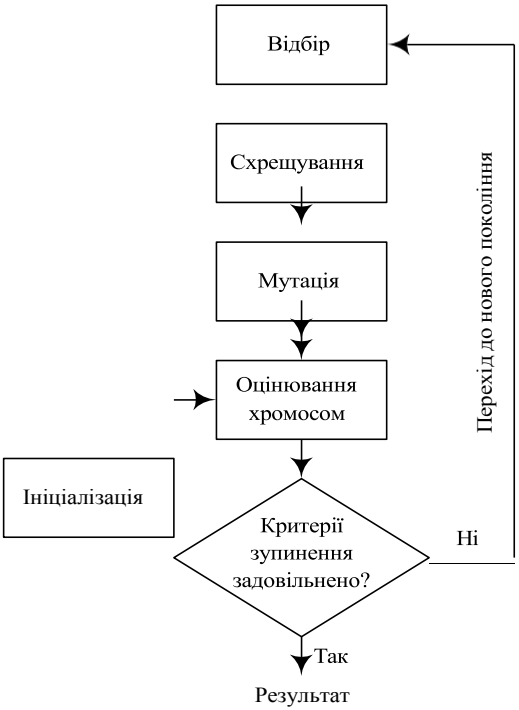


Рисунок 1.4 – Схема роботи узагальненого генетичного метода

Узагальнений метод генетичного пошуку можна записати таким чином.

Крок 1. Встановити лічильник ітерацій (часу): $t = 0$. Виконати ініціалізацію (initialization) початкової популяції особин: $P_t = \{H_1, H_2, \ldots, H_N\}$.

Крок 2. Оцінити особини поточної популяції (evaluating) шляхом обчислення їхніх фітнес-функцій $f(H_j)$, $j = 1, 2, \ldots, N$.

Крок 3. Перевірити умови закінчення пошуку (termination criteria). Як такі умови можуть бути використані: досягнення максимально допустимого часу функціонування методу, числа ітерацій, значення функції пристосованості і т. ін. Якщо критерії закінчення пошуку задовільнено, тоді виконати перехід до кроку 12.

Крок 4. Збільшити лічильник ітерацій (часу): $t = t + 1$.

Крок 5. Вибрати частину популяції (батьківські особини) для схрещування (selection of parents) P'.

Крок 6. Сформувати батьківські пари (mating) з особин, що відібрані на попередньому кроці.

Крок 7. Схрестити (crossover) вибрані батьківські особини.

Крок 8. Застосувати оператор мутації (mutation) до особин P'.

Крок 9. Обчислити нову функцію пристосованості $f(H_j)$ особин, отриманих в результаті схрещування та мутації.

Крок 10. Сформувати нове покоління шляхом вибору особин, що вижили, виходячи з рівня їх пристосованості (replacing, selection of survivors).

Крок 11. Перейти до кроку 3.

Крок 12. Зупинення.

Генетичні методи є більш ефективним інструментом пошуку в порівнянні з класичними методами оптимізації за таких умов:

– досліджуваний простір пошуку є великим, негладким (існують точки розриву) і неунімодальним (є декілька оптимумів);

– цільова функція пошуку може мати шуми;

– задача не вимагає знаходження надточного глобального оптимуму. Тобто необхідно достатньо швидко знайти прийнятне рішення, що досить часто має місце в реальних задачах.

Таким чином, генетичний пошук може успішно використовуватися для вирішення комбінаторних задач, а також для пошуку оптимальних значень полімодальних функцій.

До *переваг генетичних методів* відносять:

– відсутність необхідності в специфічних знаннях про вирішувану задачу. Проте у випадку, якщо додаткова інформація про досліджувану систему, об'єкт або процес є відомою, то вона може бути використана в процесі пошуку;

– концептуальна простота та прозорість реалізації;

– можливість розпаралелювання;

– простота кодування вхідної і вихідної інформації. Некритичність до виду параметрів досліджуваних систем (можливість використання експертної, емпіричної, довідкової та іншої інформації про об'єкт, поданої різними типами даних);

– можливість застосування до великого кола задач без внесення серйозних змін у внутрішню структуру методу;

– можливість адаптивності параметрів генетичного пошуку до особливостей вирішуваної задачі;

– менша ймовірність попадання і зациклення в локальному оптимумі, яка досягається за рахунок використання популяційного підходу;

– можливість застосування в методі інших пошукових процедур.

До *недоліків генетичного пошуку* відносять:

– високу ітеративність;

– сильну залежність ефективності генетичного пошуку від його параметрів (розмір популяції, початкова точка пошуку, імовірнісні характеристики генетичних операторів і т. ін.);

– *епістазис* – внутрішню залежність між змінними (генами), закодованими в хромосомі. Якщо гени не пов'язані один з одним, то вважається, що епістазис малий або не існує. У випадку, якщо гени є взаємозалежними, то епістазис може створювати проблеми для генетичного пошуку, спричинені тим, що при схрещуванні ланцюжки взаємозалежних генів будуть зруйновані, що призводить до появи низько пристосованих нащадків. Вирішення проблеми епістазису полягає в тому, щоб зберігати в хромосомі взаємозалежні гени, розташовуючи їх близько один до одного. При групуванні залежних генів істотно знижується ймовірність того, що вони будуть зруйновані при застосуванні схрещування;

– передчасна збіжність, пов'язана з недостатньою різноманітністю хромосом в популяції. Найпоширенішою причиною передчасної збіжності є недостатній розмір популяції. Таким чином, вирішенням такої проблеми може бути збільшення кількості хромосом в популяції.

Враховуючи особливості, переваги і недоліки оптимізаційних методів, можна надати такі *рекомендації щодо вибору пошукового методу* при вирішенні практичних задач:

– якщо простір пошуку є дискретним і невеликим за розміром, то можна скористатися методом повного перебору, який знайде найкраще рішення. Генетичний метод, на відміну від методу повного перебору, може з більшою ймовірністю зійтися до локального оптимуму, а не до глобального. Проте генетичний пошук швидше знайде субоптимальне рішення, що знаходиться недалеко від дійсного оптимуму;

– якщо цільова функція в пошуковому просторі є гладкою і унімодальною, то будь-який градієнтний метод (наприклад, метод найшвидшого спуску) буде ефективнішим, ніж генетичний пошук;

– якщо про простір пошуку відома деяка додаткова інформація (наприклад, для задачі комівояжера), то методи пошуку, що використовують апріорні відомості про пошуковий простір, часто

перевершують будь-який універсальний метод, у тому числі й генетичний метод;

– при достатньо складному рельєфі цільової функції методи пошуку, що працюють із єдиним рішенням на кожній ітерації (наприклад, простий метод спуску), можуть зациклитися в локальному рішенні. Генетичні методи працюють з набором із декількох рішень, тому вони мають менше шансів зійтися до локального оптимуму і надійно функціонують на багатоекстремальних поверхнях.

? 1.7 Контрольні питання

1. Біологічні нейрони. Математичні моделі нейроелементів.
2. У чому полягають переваги еволюційних методів?
3. Властивості штучних нейромереж.
4. Для чого призначений оператор мутації?
5. Загальна характеристика та принципи побудови нейромереж.
6. Загальне уявлення про навчання нейромереж.
7. Структурний синтез нейромереж.
8. Параметричний синтез нейромереж.
9. Класифікація та види моделей нейромереж.
10. Наведіть особливості методів спряжених градієнтів.
11. Наведіть особливості, переваги та недоліки градієнтних методів оптимізації.
12. Наведіть послідовність виконання узагальненого еволюційного пошуку.
13. Назвіть особливості еволюційних методів.
14. Особливості багатопараметричного пошуку.
15. Поняття: нейрон, нейромережа, нейрокомп'ютер.
16. Поняття: синапс, вага (ваговий коефіцієнт), поріг, дискримінантна (вагова) функція, функція активації, бажаний і реальний вихід нейромережі.
17. Опишіть квазіньютонівські методи.
18. Порівняйте методи еволюційного пошуку з іншими методами оптимізації.
19. Які методи відносять до еволюційних?
20. Порівняйте методи спряжених градієнтів.
21. Порівняйте стратегії створення початкової популяції.

22. Поясніть призначення методу Левенберга-Марквардта. Порівняйте його з іншими градієнтними методами.

23. Поясніть схему роботи методу Ньютона.

24. Проаналізуйте метод Коші.

25. Проаналізуйте узагальнену схему роботи еволюційних методів.

26. Проаналізуйте умови ефективного використання методів еволюційного пошуку.

27. У чому подібність і відмінність біологічних і формальних нейронів?

28. Характеристики нейромереж.

29. Що таке фітнес-функція?

30. Яким чином відбувається формування нового покоління при еволюційному пошуку?

31. Які критерії зупинення використовуються при еволюційному пошуку?

32. Які недоліки еволюційного пошуку та в чому вони полягають?

1.8 Практичні завдання

Завдання 1. Побудувати за графіки функцій активації:

а) логістичний сигмоїд (функція logsig);

б) тангенційний сигмоїд (функція tansig);

в) лінійна (функція purelin);

г) лінійна уніполярна з насиченням (функція satlin);

д) радіально-базисна (функція radbas);

е) порогова (функція hardlim).

Завдання 2. Побудувати графіки перших похідних функцій активації:

а) логістичний сигмоїд (функція dlogsig);

б) тангенційний сигмоїд (функція dtansig);

в) лінійна (функція dpurelin);

г) лінійна уніполярна з насиченням (функція dsatlin);

д) радіально-базисна (функція dradbas);

е) порогова (функція dhardlim).

Завдання 3. Проаналізуйте, чи пов’язані (і якщо так, то яким чином, а якщо ні, то чому) такі властивості нейромереж, як:

а) адаптивність і функціональна блочність;

б) асоціативна пам’ять і універсальна апроксимація;

в) варіативність і узагальнення;
г) ефективність і точність;
д) здатність до навчання і складність;
е) здатність отримувати знання з даних і самоорганізація;
ж) ієрархічність і розподіленість пам'яті;
з) ізоморфізм і попередня організація;
и) логічна прозорість і поліалгоритмічність;
к) надійність і пластичність;
л) нелінійність і паралельна архітектура;
м) однорідність і надійність;
н) паралельна архітектура і адаптивність;
о) пластичність і асоціативна пам'ять;
п) поліалгоритмічність і варіативність ;
р) попередня організація і ефективність;
с) розподіленість пам'яті і здатність до навчання;
т) самоорганізація і здатність отримувати знання з даних;
у) складність і ієрархічність ;
ф) точність і ізоморфізм;
х) узагальнення і логічна прозорість;
ц) універсальна апроксимація і надійність;
ч) функціональна блочність і нелінійність.

Завдання 4. Підготувати реферат на одну з таких тем.

1. Історія розвитку нейроінформатики.
2. Фізичні моделі нейронів.
3. Фізичні моделі нейромереж.
4. Сучасні програмні засоби для моделювання нейромереж.
5. Сучасні апаратні засоби для моделювання нейромереж.
6. Еволюційні методи в задачах синтезу нейромереж.

РОЗДІЛ 2

НЕЙРОННІ МЕРЕЖІ ПРЯМОГО ПОШИРЕННЯ

Нейронні мережі прямого поширення сигналу – це нейронні мережі, де нейрони розділено на групи – *шари*, а зв'язки між нейронами організовано таким чином, що перший (*вхідний*) *шар* сприймає зовнішній сигнал, обробляє його і передає значення з виходів своїх нейронів наступному шару, при цьому всередині кожного шару зв'язки між нейронами відсутні, а на входи нейронів *схованих шарів* (усіх інших шарів, крім першого), поступають значення з відповідних виходів нейронів попереднього шару. Останній шар мережі називають *вихідним*. Значення з виходів його нейронів утворюють загальний вихідний сигнал НМ.

2.1 Одношаровий персептрон. Метод Уідроу-Хоффа

Одношаровий персептрон є одним з найпростіших варіантів НМ і містить лише один нейрон (див. рис. 1.2). Будучи самостійною моделлю НМ з одного боку, одношаровий персептрон (формальний нейрон) є основним конструкційним елементом для більшості моделей НМ, з іншого боку.

Одношаровий персептрон використовує як дискримінантну функцію зважену суму, а як активаційну – порогову або сигмоїдну, рідше – лінійну функцію. У залежності від типу функції активації розрізняють *дискретні персептрони*, що використовують порогову функцію активації, і *дійсні персептрони*, що використовують дійсні функції активації, наприклад, сигмоїдну функцію.

Метод навчання одношарового дискретного персептрона має наступний вигляд.

Крок 1. Вагам w_i, $i = 1, 2, ..., N$, та порогу w_0 присвоюються випадкові значення.

Крок 2. Пред'являються черговий екземпляр $x^p=\{x_{1p}, ..., x_{Lp}\}$ з навчальної множини, $p = 1, 2, ..., m$, де m – кількість екземплярів у навчальній вибірці, і бажаний вихід y^{s*}.

Крок 3. Обчислюється реальне значення на виході персептрона $y = \psi(\varphi(w, x))$, де $w = \{w_0, w_1, w_2, ..., w_L\}$ – набір ваг, що утворюють пам'ять нейрона.

Крок 4. Коригуються ваги персептрона:

$$w_i = w_i + \alpha \ (y^{s*} - y^s) \ x_{ip},$$

$$i = 0, 1, ..., L, \ x_0 = 1,$$

де α – позитивний коригувальний приріст (крок навчання).

Крок 5. Якщо досягнуто збіжність, то процедура корекції ваг закінчується, інакше – перехід до кроку 2.

Відповідно до даного методу спочатку здійснюється ініціалізація параметрів персептрона випадковими значеннями. Потім по черзі пред'являються образи з відомою класифікацією, вибрані з навчальної множини, і коригуються ваги відповідно до формул кроків 3 та 4.

Величина корекції визначається позитивним коригувальним приростом, конкретне значення якого вибирається досить великим, щоб швидше проводилася корекція ваг, і в той же час досить малим, щоб не допустити надмірного зростання значень ваг.

Процедура навчання триває до тих пір, поки не буде досягнута збіжність, тобто поки не будуть отримані ваги, що забезпечують правильну класифікацію для всіх образів з навчальної множини.

У тому випадку, коли навчальні вибірки розділити гіперплощиною неможливо, для навчання персептрона можна використовувати метод *Уідроу-Хоффа*, здатний мінімізувати середньоквадратичну помилку між бажаними і реальними виходами мережі для навчальних даних. Цей метод також можна застосовувати для *навчання одношарового дійсного персептрона*. Метод Уідроу-Хоффа можна записати в тому ж вигляді, що й вищеописаний метод, припускаючи, що у вузлах персептрона нелінійні елементи відсутні, а коригувальний приріст в процесі ітерацій поступово зменшується до нуля.

Якщо для вирішення задачі розпізнавання образів використовується дискретний персептрон, вирішальне правило відносить вхідний образ до класу K^1, якщо вихід персептрона дорівнює 1, і до класу K^0 – в іншому випадку. У разі, якщо для

вирішення задач розпізнавання образів використовується дійсний персептрон, вирішальне правило відносить вхідний образ до класу K^1, якщо вихід мережі більше 0,5; і до класу K^0 – в іншому випадку.

Незважаючи на досить обмежені здібності кожного нейрона окремо, їх об'єднання в мережу дає можливість вирішувати більш складні завдання, непідвладні кожному окремому нейрону.

2.2 Багатошарова нейронна мережа

Багатошарова нейронна мережа (БНМ) прямого поширення (*багатошаровий персептрон*) складається з формальних нейронів і характеризується наступними параметрами і властивостями: M – кількість шарів мережі, N_μ – кількість нейронів у μ-му шарі, зв'язки між нейронами у шарі відсутні. Виходи нейронів μ-го шару, μ=1,2,..., M–1, надходять на входи нейронів тільки наступного μ+1-го шару. Зовнішній векторний сигнал x надходить на входи нейронів тільки першого шару, виходи нейронів останнього M-го шару утворюють вектор виходів мережі $y^{(M)}$. Схема мережі показана на рис. 2.1.

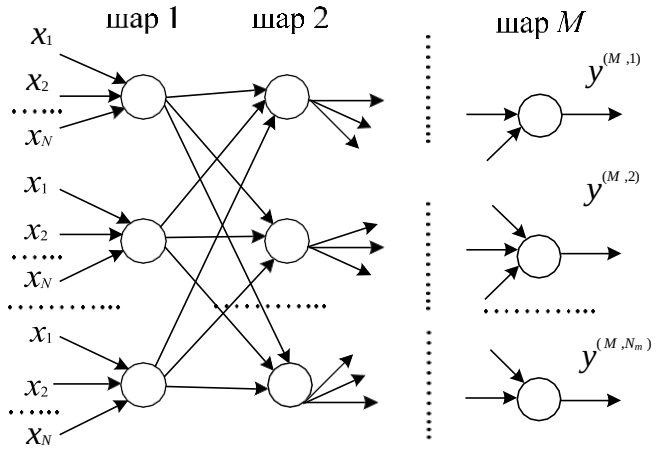


Рисунок 2.1 – Схема БНМ

Кожен i-й нейрон μ-го шару ((μ, i)-й нейрон) перетворює вхідний вектор $x^{(\mu,\ i)}$ у вихідну скалярну величину $y^{(\mu,\ i)}$. Це

перетворення складається з двох етапів: спочатку обчислюється дискримінантна функція $\varphi^{(\mu,i)}(w^{(\mu,i)}, x^{(\mu,i)})$, яка далі перетворюється на вихідну величину $y^{(\mu,\ i)} = \psi^{(\mu,i)}(\varphi^{(\mu,i)}(w^{(\mu,i)}, x^{(\mu,i)}))$, де $w^{(\mu,\ i)}=(w_0^{(\mu,\ i)}, w_1^{(\mu,\ i)}, \ldots, w_{N\mu}^{(\mu,\ i)})^{\mathrm{T}}$ – вектор вагових коефіцієнтів нейрона, $x_j^{(\mu,\ i)}$ – j-та компонента N_μ – вимірного вхідного вектора $x^{(\mu,\ i)}$.

Якість рішення, одержуваного БНМ, буде істотно залежати від кількості шарів, кількості нейронів у кожному шарі і кількості зв'язків між шарами.

Загальна кількість ваг і порогів у БНМ визначається формулою:

$$N_w = \sum_{\mu=1}^{M} N_\mu (N_{\mu-1} + 1), N_0 = N.$$

Для різних класів задач, що вирішуються за допомогою НМ, запропоновані евристичні оцінки кількості шарів і нейронів.

Для забезпечення узагальнювальних властивостей необхідно, щоб кількість ваг і порогів БНМ $N_w << NS$, де N – розмірність вхідного сигналу (кількість ознак), S – кількість екземплярів у навчальній вибірці.

Виходячи для простоти з того, що розмірність виходу БНМ дорівнює одному (тобто мережа має на останньому шарі тільки один нейрон), а кількість шарів може становити два або три (менше двох не дозволить вирішувати нелінійні задачі, а більше трьох шарів непотрібно, бо відомо, що тришарова БНМ здатна апроксимувати будь-яку обчислювану функцію), а також приймаючи $N \approx S$, отримаємо:

– для двошарової БНМ:

$$N_w = N_1(N+1) + (N_1 + 1) = N_1(N+2) + 1 << NS,$$

$$N_1 << \frac{NS-1}{N+2} \Rightarrow N_1 << N;$$

– для тришарової БНМ:

$$N_w = N_1(N+1) + N_2(N_1+1) + (N_2+1) = N_1(N+1) + N_2N_1 + 2N_2 + 1 << NS.$$

Приймемо $N_2 = 0{,}5N_1$, а $N_1 N \approx N_1^2$ тоді для тришарової БНМ отримаємо: $1{,}5N_1 N_1 + 2N_1 << NN - 1 \Rightarrow N_1 << N$.

Відомі й інші підходи. Наприклад, для оцінки кількості нейронів у прихованих шарах однорідних двошарових БНМ з сігмоїдними функціями активації можна використовувати формулу для оцінки необхідного числа синаптичних ваг N_w:

$$\frac{N_M S}{1+\log_2 S} \le N_w \le N_M \left(\frac{S}{N_M}+1\right)(N+N_M+1)+N_M,$$

де N_M – розмірність вихідного сигналу.

Оцінивши необхідну кількість ваг, можна розрахувати $N_{н.}$ – кількість нейронів у прихованих шарах. Наприклад, для двошарової мережі вона складе: $N_н = N_w/(N+N_M)$.

Відомі й інші формули для оцінки, наприклад:

$$2(N+N_н+N_M) \le S \le 10(N+N_н+N_M);\ 0{,}1S–N–N_M \le N_н \le 0{,}5S–N–N_M.$$

2.3 Навчання БНМ. Метод зворотного поширення помилки

Процес *навчання БНМ* здійснюється у результаті мінімізації *цільової функції* – певного критерію якості $E(Q(\varepsilon, s))$, що характеризує інтегральну міру близькості виходів мережі $y^{(M)}$ і вказівок вчителя $y^* = \{y^{s*}\}$, $s = 1,2, ..., S$, де s – номер поточного екземпляра навчаючої вибірки, S – кількість екземплярів у навчальній вибірці, $Q(\varepsilon, s)$ – миттєвий критерій якості, що залежить від вектора помилки мережі для i-ї вихідної змінної s-го екземпляра $\varepsilon(s,i,w) = y^{(M,\,i)} - y_i^s$, $i = 1,2, ..., N_M$, де w – множина ваг мережі, $y^{(M,\,i)}$ та y_i^{s*} – відповідно, розраховане мережею і бажане значення i-ї вихідної змінної для s-го екземпляра навчаючої вибірки.

Як миттєвий критерій якості найбільш часто використовують суму квадратів помилки

$$Q(\varepsilon, s) = \sum_{i=1}^{N_M} \varepsilon(s,i,w)^2$$

або суму модулів помилки для s-го екземпляра вибірки

$$Q(\varepsilon, s) = \sum_{i=1}^{N_M} \left|\varepsilon(s,i,w)\right|.$$

Інтегральну міру близькості E визначають за формулою:

$$E = \frac{1}{\upsilon} \sum_{s=1}^{S} Q(\varepsilon, s),$$

де υ– деякий коефіцієнт (якщо υ =1, то E визначає сумарну миттєву помилку; якщо $\upsilon = S$, то E визначає середню миттєву помилку).

Для кожного вхідного вектора x^s з навчальної множини повинен бути визначений вектор бажаних виходів мережі y^{s*}. Якщо навчальна БНМ використовується в якості класифікатора, то зазвичай бажані виходи мають низький рівень (0 або менше 0,1), крім виходу вузла, відповідного класу, до якого належить x; цей вихід у даному випадку має високий рівень (1 або більше 0,9).

Для початку навчання ваги БНМ заданої структури встановлюють рівними випадковим числам.

Для мінімізації цільової функції навчання вирішують завдання багатомірної нелінійної оптимізації, для чого традиційно використовують *градієнтні методи*. Ці методи носять ітеративний характер, оскільки компоненти градієнта виявляються нелінійними функціями. Градієнтні методи засновані на ітераційній процедурі корекції значень ваг, реалізованої у відповідності з формулою:

$$w_{k+1} = w_k + \alpha_k p(w_k),\ k = 0,1,2, \ldots$$

де w_k , w_{k+1} – поточне і нове наближення значень ваг і порогів НМ до оптимального рішення, відповідно; α_k – крок збіжності; $p(w_k)$ – напрям пошуку в багатовимірному просторі ваг. Спосіб визначення $p(w_k)$ та α_k на кожній ітерації залежить від особливостей конкретного методу.

Оскільки градієнтні методи вимагають обчислення похідних цільових функції для визначення напрямку пошуку, для розрахунку частинних похідних цільової функції за вагами мережі використовують ***метод зворотного поширення помилки*** (error backpropagation method), який був розроблений як

узагальнення методу Уідроу-Хоффа для БНМ з нелінійними диференційовними функціями активації.

Метод зворотного поширення помилки для настроювання ваг НМ використовує градієнт функції помилки таким чином, щоб мінімізувати помилку на навчальній вибірці. Для цього мережі послідовно надаються вхідні вектори (екземпляри) з навчальної вибірки, і, починаючи з останнього шару, обчислюються *градієнти функції помилки* для кожного нейрона:

$$\frac{\partial E}{\partial w_j^{(\mu,i)}} = \frac{\partial E}{\partial \psi^{(\mu,i)}} \frac{\partial \psi^{(\mu,i)}}{\partial \phi^{(\mu,i)}} \frac{\partial \phi^{(\mu,i)}}{\partial w_j^{(\mu,i)}},$$

де $\frac{\partial \psi^{(\mu,i)}}{\partial \phi^{(\mu,i)}} = \psi'^{(\mu,i)}\left(\phi^{(\mu,i)}\right)$.

Якщо в якості дискримінантної функції нейронів мережі використовується зважена сума, то $\frac{\partial \phi^{(\mu,i)}}{\partial w_j^{(\mu,i)}} = x_j^{(\mu,i)} = \psi^{(\mu-1,j)}$.

Для нейронів вихідного шару: $\frac{\partial E}{\partial \psi^{(M,i)}} = y_i^s - \psi^{(M,i)}$,

для нейронів інших шарів:

$$\frac{\partial E}{\partial \psi^{(\mu,i)}} = \sum_j \frac{\partial E}{\partial \psi^{(\mu+1,j)}} \frac{\partial \psi^{(\mu+1,j)}}{\partial \phi^{(\mu+1,j)}} \frac{\partial \phi^{(\mu+1,j)}}{\partial \psi^{(\mu,i)}},$$

де $\frac{\partial \phi^{(\mu+1,j)}}{\partial \psi^{(\mu,i)}} = w_j^{(\mu,i)}$.

У методі зворотного поширення помилки ваги і пороги змінюються у напрямку, протилежному градієнту. Тому базовий метод зворотного поширення часто називають *методом градієнтного спуску* (gradient descent). Базовий метод зворотного поширення помилки складається з таких кроків.

Крок 1. Задати навчальну вибірку екземплярів x і зіставлених ним значень цільового параметру y^*. Ініціалізувати всі ваги і пороги мережі. Задати значення прийнятного рівня помилки ξ. Встановити покажчик поточного екземпляра вибірки $s = 1$.

Крок 2. Подати на вхід БНМ s-й екземпляр з навчальної вибірки $x^s = \{x^s_{*i}\}$, а на вихід – зіставлений йому бажаний вихідний вектор $y^{s*} = \{y^{s*}_j\}$, $s = 1,2,..., S$; $i = 1,2,..., N$; $j = 1,2,..., N_M$.

Крок 3. Рухаючись від першого шару до останнього, розрахувати значення на виходах нейронів і фактичний вихід мережі $y^{(M)} = \{y^{(M,\ i)}\}$, $i = 1, 2, ..., N_M$, при подачі на вхід мережі екземпляра x^s.

Крок 4. Розрахувати помилку мережі E. Якщо $E > \xi$ (умова закінчення навчання для s-го екземпляра не виконується), тоді перейти на крок 5; в іншому випадку – перейти на крок 6.

Крок 5. Рухаючись від вихідного шару до першого шару мережі провести настроювання ваг і порогів за формулою $w_{k+1} = w_k + \alpha_k p(w_k)$, визначивши напрямок пошуку в багатовимірному просторі ваг за формулою:

$$p(w^{(\mu,i)}_{j,k}) = -g^{(\mu,i)}_{j,k},$$

де $g^{(\mu,i)}_{j,k} = \dfrac{\partial E}{\partial w^{(\mu,i)}_j}$ – градієнт помилки за j-ю вагою i-го нейрона μ-го шару БНМ на k-й ітерації навчання, який визначається за формулою:

$$g^{(M,i)}_{j,k} = \psi'^{(M,i)}_k\left(\varphi^{(M,i)}_k\right)\left(y^s_i - \psi^{(M,i)}_k\left(\varphi^{(M,i)}_k\right)\right)\psi^{(M-1,j)}_k\left(\varphi^{(M-1,j)}_k\right),$$

а для нейронів інших шарів за рекурентною формулою:

$$g^{(\mu,i)}_{p,k} = \psi'^{(\mu,i)}_k\left(\varphi^{(\mu,i)}_k\right)\left(\sum_{j=1}^{N_{\mu+1}} g^{(\mu+1,j)}_{i,k} w^{(\mu+1,j)}_{i,k}\right) x^{(\mu,i)}_{p,k},$$

$$\mu = M-1, ...,1;\ p = 0,..., N_{\mu-1}.$$

Якщо у мережі використовуються нейрони з сигмоїдною функцією активації, то для них:

$$\psi'^{(\mu,i)} = \left(\frac{1}{1+e^{-\varphi^{(\mu,i)}(w^{\mu,i}, x^{\mu,i})}} \right)' = \frac{e^{-\varphi^{(\mu,i)}(w^{(\mu,i)}, x^{(\mu,i)})}}{\left(1+e^{-\varphi^{(\mu,i)}(w^{(\mu,i)}, x^{(\mu,i)})}\right)^2} =$$

$$= \frac{1}{1+e^{-\varphi^{(\mu)}(w^{,i}, x^{,i})}} \left(1 - \frac{1}{1+e^{-\varphi^{(\mu)}(w^{,i}, x^{,i})}}\right) = \psi^{(\mu,i)}(1-\psi^{(\mu,i)}).$$

Перейти на крок 4.

Крок 6. Встановити: $s = s + 1$. Якщо $s > S$, то перейти на крок 7, у протилежному випадку – перейти на крок 2.

Крок 7. Визначити помилку мережі E. Якщо вона є прийнятною ($E \leq \xi$), то завершити пошук і видати як результат роботи значення ваг і порогів w, у протилежному випадку – встановити $s = 1$ та перейти до кроку 2.

Як додаткові *критерії завершення пошуку* у методі зворотного поширення помилки можуть використовуватися досягнення ліміту часу або ліміту кількості циклів корекції ваг (*епох навчання*) або досягнення заданого мінімального значення градієнта цільової функції.

Стандартний метод зворотного поширення часто збігається дуже повільно, рухаючись вздовж плоских ділянок поверхні помилки. Тому на практиці його часто використовують лише для розрахунку приватних похідних цільової функції помилки за вагами НМ, а пошук в просторі ваг НМ роблять на основі більш швидких градієнтних методів.

Основним недоліком традиційно використовуваних градієнтних методів навчання БНМ є обумовлена ітераційною корекцію ваг низька швидкість збіжності навчання, яка серйозно обмежує практичне застосування нейронного керування. Іншим не менш важливим недоліком даних методів є невизначеність у виборі початкової точки пошуку, параметрів архітектури та топології мережі.

2.4 Радіально-базисні нейромережі

Радіально-базисні нейромережі (РБНМ – Radial Basis Function Net, RBFN) були запропоновані для апроксимації функцій багатьох змінних. За допомогою радіально-базисних функцій можна як завгодно точно апроксимувати задану функцію. Як і багатошаровий

персептрон, радіально-базисна мережа є універсальним апроксиматором.

Математичну основу РБНМ складає метод потенційних функцій, розроблений М. А. Айзерманом, Е М. Браверманом і Л. І. Розоноером, що дозволяє подати певну функцію $y(x)$ у вигляді суперпозиції потенційних або базисних функцій $f_i(x)$:

$$y(x) = \sum_{i=1}^{N} a_i f_i(x) = a^{\mathrm{T}} f(x),$$

де $a_i = (a_1, a_2, ..., a_N)^T$ – вектор підлягаючих визначенню параметрів; $f(x) = (f_1(x), f_2(x), \ldots, f_N(x))^T$ – вектор базисних функцій.

В РБНМ в якості *базисних функцій* обираються деякі функції відстані між векторами $f_i(x) = f(\| x - c_i \|)$.

Вектори c_i називають *центрами базисних функцій*. Функції $f(x)$ вибираються невід'ємними і зростаючими при збільшенні $\|x - c_i\|$. В якості міри близькості векторів x та c_i вибираються, як правило, такі метрики:

– *евклідова метрика*: $\| x - c_i \| = \sqrt{\sum_{j=1}^{N} (x_j - c_{ij})^2}$;

– *манхеттенська метрика*: $\| x - c_i \| = \sum_{i=1}^{N} | x_j - c_{ij} |$,

де $| x_j - c_{ij} | = (x_j - c_{ij}) \operatorname{sign}(x_j - c_{ij})$, $\operatorname{sign}(x_i - c_{ij}) = \begin{cases} 1, (x_i - c_{ij}) > 0; \\ 0, (x_i - c_{ij}) = 0; \\ -1, (x_i - c_{ij}) < 0. \end{cases}$

Структурно РБНМ є окремим випадком двошарової БНМ, нейрони першого шару якої здійснюють локалізовану реакцію на вхідний стимул і використовують як дискримінантну функцію Евклідову відстань, а як функцію активації – радіально-базисну

(ядерну) функцію Гауса, а нейрони другого шару формують зважену лінійну комбінацію з радіально-базисних функцій, обчислених нейронами першого шару (використовують як дискримінантну функцію зважену суму, а як функцію активації – лінійну функцію). Нейрони другого шару РБНМ відповідають вихідним класам, у той час як нейрони першого шару відповідають кластерам, на які розбивається вхідний простір ознак. Кількість кластерів заздалегідь, як правило, невідома. На практиці її визначають на основі методів кластер-аналізу або задають рівним двом і далі нарощують, навчаючи щоразу мережу заново, поки помилка мережі не досягне прийнятного значення.

2.5 Навчання радіально-базисних нейромереж

РБНМ характеризується трьома типами параметрів:

– лінійні вагові параметри вихідного шару (входять в опис мережі лінійно);

– центри кластерів – нелінійні (входять в опис нелінійно) параметри прихованого шару;

– відхилення (радіуси базисних функцій) – нелінійні параметри прихованого шару.

Навчання РБНМ, що складається у визначенні цих параметрів, може зводиться до одного з таких варіантів:

1) задаються центри і відхилення, а обчислюються тільки ваги вихідного шару;

2) визначаються шляхом самонавчання (за допомогою методів кластеризації) центри і відхилення, а для корекції ваг вихідного шару використовується навчання з учителем. Тобто методом кластеризації формується фіксована множина центрів кластерів. Потім мінімізацією помилки E отримують асоціації центрів кластерів з виходом;

3) визначаються всі параметри мережі за допомогою навчання з учителем, наприклад за допомогою методу зворотного поширення помилки.

Перші два варіанти застосовуються в мережах, що використовують базисні функції з жорстко заданим радіусом (відхиленням). Третій варіант, будучи найбільш складним і

трудомістким в реалізації, передбачає використання будь-яких базисних функцій.

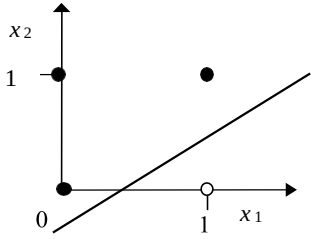
2.6 Приклади виконання завдань

Приклад 1. Чи можна навчити дискретний одношаровий персептрон обчислювати значення функції: y = not (x_1 and not (x_2))?

Спочатку обчислимо таблицю істинності для функції y.

x_1	x_2	not x_2	x_1 and not (x_2)	y
0	0	1	0	1
0	1	0	0	1
1	0	1	1	0
1	1	0	0	1

Зобразимо графічно функцію *y*.



На рисунку закрашені круги відповідають значенням y=1, а білий круг – значенню y=0. Як видно з рисунку, білий круг можна відділити однією гіперплощиною (у проекції – прямою лінією) від чорних кругів, тобто задача є лінійно роздільною.

Тому одношаровий дискретний персептрон можна навчити безпомилково обчислювати значення цієї функції.

Приклад 2. Яке буде значення на виході y одношарового персептрона із пороговою функцією активації, дискримінантною функцією зважена сума та двома зовнішніми входами при поданні на його входи сигналів x_1=1 та x_2= 0,6 при відомих значеннях ваг входів w_1 = 1 та w_2 = –1 і порога w_0 = 0?

Одношаровий персептрон складається усього з одного нейрона. Функціонування штучного нейрона описується формулою: $y = \psi(\varphi(w, x))$.

Спочатку визначимо значення дискримінантної функції:

$$\varphi(w, x) = \sum_{j=1}^{N} w_j x_j + w_0 = 1 \cdot 1 + (-1) \cdot 0{,}6 + 0 = 0{,}4.$$

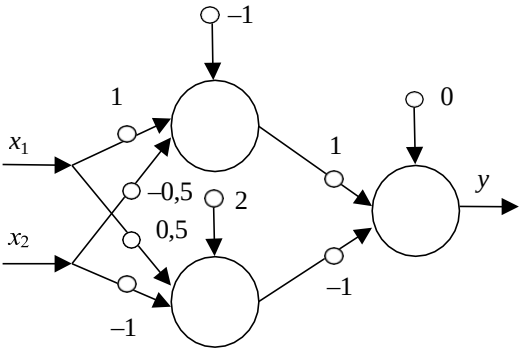
Після чого розрахуємо значення функції активації:

$$\psi(x) = \begin{cases} 1, x \geq 0; \\ 0, x < 0; \end{cases}$$

Підставляючи x=0,4, отримаємо: $\psi(x) = 1$.

Таким чином, y = 1.

Приклад 3. Нехай задано БНМ, структуру і значення параметрів якої зображено на рисунку. Для екземпляра $x=(1\ 2)^{\mathrm{T}}$ обчислити значення на виході БНМ, виходячи з того, що усі її нейрони використовують дискримінантну функцію зважена сума та функцію активації – порогову.



Спочатку для кожного нейрона першого шару визначимо значення на його виході:

– для першого нейрона першого шару:

$$\varphi^{(1,1)}(w^{(1,1)}, x^{(1,1)}) = 1 \cdot 1 + (-0{,}5) \cdot 2 - 1 = -1; \psi^{(1,1)}(\varphi^{(1,1)}) = 0;$$

– для другого нейрона першого шару:

$$\varphi^{(1,2)}(w^{(1,2)}, x^{(1,2)}) = 0{,}5 \cdot 1 + (-1) \cdot 2 + 2 = 0{,}5; \psi^{(1,2)}(\varphi^{(1,2)}) = 1.$$

Потім визначимо значення на виході єдиного нейрона другого шару БНМ:

$$\varphi^{(2,1)}(w^{(2,1)}, x^{(2,1)}) = 1 \cdot 0 + (-1) \cdot 1 + 0 = -1; \psi^{(2,1)}(\varphi^{(2,1)}) = 0.$$

Отже отримаємо значення на виході БНМ: y = 0.

? 2.7 Контрольні питання

1. Багатошаровий персептрон: модель і принципи побудови архітектури.
2. Градієнтні методи навчання багатошарових нейромереж.
3. Дайте порівняльну характеристику відомих Вам методів навчання багатошарових нейромереж.
4. Застосування кластер-аналізу при навчанні радіально-базисних нейромереж.
5. Критерії завершення пошуку у методі зворотного поширення помилки.
6. Математична основа РБНМ.
7. Метод зворотного поширення помилки.
8. Метод Уідроу-Хоффа.
9. Методи навчання радіально-базисних нейромереж.
10. Моделі та принципи синтезу архітектури радіально-базисних нейромереж.
11. Можливості і властивості одношарових персептронів.
12. Навчання радіально-базисних нейромереж.
13. Навчання одношарового персептрона.
14. Недоліки методу зворотного поширення помилки.
15. Опишіть функціонування багатошарового персептрона.
16. Опишіть функціонування одношарового персептрона.
17. Опишіть функціонування РБНМ.
18. Переваги і недоліки радіально-базисних мереж.
19. Порівняння моделей та методів навчання нейромереж прямого поширення.
20. Призначення шарів РБНМ.
21. Структура РБНМ.
22. У чому подібність і відмінність радіально-базисних нейромереж і багатошарових нейромереж?
23. У чому полягає відмінність вхідного, вихідного та схованих шарів у БНМ?
24. Чи впливає величина кроку навчання на час навчання одношарового персептрона, багатошарової мережі? Відповідь обґрунтуйте теоретично та доведіть експериментально.

25. Чи впливає вид функції активації нейрона на тривалість навчання і роботи персептрона, величину помилки навчання і класифікації (оцінювання) і, якщо впливає, то яким чином?

26. Чи впливає кількість використаних ознак на швидкість навчання персептрона? Відповідь обґрунтуйте теоретично та доведіть експериментально.

27. Чи впливає обсяг навчальної вибірки на швидкість навчання нейромереж?

28. Чи впливає репрезентативність навчальної вибірки на точність класифікації екземплярів тестової вибірки?

29. Чи впливає репрезентативність тестової вибірки на точність класифікації екземплярів тестової вибірки?

30. Чи впливає репрезентативність тестової вибірки на точність навчання персептрона по навчальній вибірці?

31. Чи доцільно застосовувати одношаровий персептрон для класифікації складно (нелінійно) роздільних образів? Обґрунтуйте і доведіть відповідь. Наведіть приклади.

32. Чи доцільно застосовувати радіально-базисних нейромереж для класифікації складно (нелінійно) роздільних образів?

33. Чи завжди збігається метод зворотного поширення помилки для багатошарового персептрона?

34. Чи завжди збігаються методи навчання одношарового персептрона?

35. Чи завжди збігаються методи навчання радіально-базисних нейромереж?

36. Чи залежить якість навчання нейромереж від якості та обсягу навчальної вибірки?

37. Чи можливе використання неградієнтних методів багатовимірної безумовної оптимізації для настроювання ваг багатошарових нейромереж, і, якщо можливе, то наскільки це доцільно робити?

38. Що таке крок навчання?

39. Що таке нейронні мережі прямого поширення сигналу?

40. Що таке шар нейронів?

41. Як визначається цільова функція навчання у методі зворотного поширення помилки?

42. Як визначити кількість шарів, нейронів та ваг у БНМ?

43. Як визначити часткову похідну цільової функції за керованою змінною в методі зворотного поширення помилки?

44. Який з градієнтних методів навчання на основі зворотного поширення помилки Ви рекомендували б для застосування, якщо критерієм оптимальності навчання НМ є: а) швидкість навчання, б) простота обчислень, в) універсальність?

45. Які задачі можна вирішувати на основі багатошарових персептронів, а які не можна? Обґрунтуйте і доведіть відповідь. Приведіть приклади.

46. Які задачі можна вирішувати на основі одношарових персептронів, а які не можна?

47. Які задачі можна вирішувати на основі радіально-базисних нейромереж, а які не можна? Обґрунтуйте і доведіть відповідь. Наведіть приклади.

48. Які методи навчання багатошарових нейромереж Вам відомі?

49. Які функції активації нейронів найчастіше використовують у нейромережах прямого поширення?

2.8 Практичні завдання

Завдання 1. Чи може дійсний багатошаровий персептрон безпомилково або із незначною помилкою навчитися моделювати функцію $y = x_1^2 + 0{,}5x_1 + x_2$, якщо число його шарів дорівнює: а) 1; б) 2; в) 3? Відповіді обґрунтуйте і, якщо можливо, поясніть рисунками.

Завдання 2. Чи можна навчити дискретний багатошаровий персептрон безпомилково або із незначною помилкою обчислювати значення функцій: 1) $y = x_1$ and x_2; 2) $y = x_1$ xor x_2; 3) y = not ((not x_1) and x_2), а дійсний багатошаровий персептрон – обчислювати значення функцій: 1) $y = 3x_1 - 0{,}05x_2$; 2) $y = \sin(x_1) + 0{,}3x_2$; 3) $y = 0{,}5x_1 + 2x_2 - 2{,}5(x_1 - x_2) + 5{,}5$; 4) $y = x_1 / (x_2 \sin(\pi))$? Відповіді обґрунтуйте і, якщо можливо, поясніть рисунками.

Завдання 3. Чи можна навчити дискретний одношаровий персептрон обчислювати значення функцій: $y = (x_1$ and $x_2)$, $y = (x_1$ xor $x_2)$, y = not ((not x_1) and x_2), а дійсний одношаровий персептрон – обчислювати значення функцій: 1) $y = 3x_1 - 0{,}05x_2$;

2) $y = \sin(x_1) + 0{,}3x_2$; 3) $y = 0{,}5x_1 + 2x_2 - 2{,}5(x_1 - x_2) + 5{,}5$? Відповіді обґрунтуйте і, якщо можливо, поясніть рисунками.

Завдання 4. Чи можна навчити радіально-базисну нейромережу обчислювати значення функцій: 1) $y = x_1$ and x_2; 2) $y = x_1$ xor x_2; 3) $y =$ not ((not x_1) and x_2), а дійсний багатошаровий персептрон – обчислювати значення функцій: 1) $y = 3x_1 - 0{,}05x_2$; 2) $y = \sin(x_1) + 0{,}3x_2$; 3) $y = 0{,}5x_1 + 2x_2 - 2{,}5(x_1 - x_2) + 5{,}5$; 4) $y = x_1 / (x_2 \sin(\pi))$? Відповіді обґрунтуйте і, якщо можливо, поясніть рисунками.

Завдання 5. Яке буде значення на виході y одношарового персептрона із двома зовнішніми входами при поданні на його входи сигналів $x_1 = -1$ та $x_2 = 0{,}3$ при відомих значеннях ваг входів $w_1 = -1$ та $w_2 = 2$ і порога $w_0 = 1$, якщо він має:

а) дискримінантну функцію зважена сума та функцію активації – лінійну?

б) дискримінантну функцію зважена сума та функцію активації – порогову?

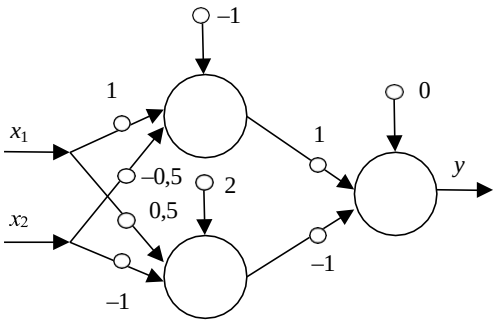
в) дискримінантну функцію зважена сума та функцію активації – сигмоїдну?

г) дискримінантну функцію евклідова відстань та функцію активації – радіально-базисну (Гауса)?

д) дискримінантну функцію евклідова відстань та функцію активації – порогову?

е) дискримінантну функцію евклідова відстань та функцію активації – сигмоїдну?

Завдання 6. Нехай задано БНМ, структуру і значення параметрів якої зображено на рисунку.



Для екземпляра $x = (-1\ 3)^T$ обчислити значення на виході БНМ, виходячи з того, що усі її нейрони використовують дискримінантну функцію зважена сума та функцію активації – лінійну.

Завдання 7. Підготуйте реферат на одну з таких тем.

1. Методи навчання багатошарових нейромереж прямого поширення сигналу.
2. Каскадно-кореляційні мережі прямого поширення.
3. Апаратні моделі нейромереж прямого поширення.
4. Історичні моделі і методи навчання персептронів 60-х років XX сторіччя.
5. Порівняльна характеристика нейромереж прямого поширення.
6. Застосування нейромереж прямого поширення.

Завдання 8. Емуляція та навчання одношарового персептрона.

1. Задати функції активації та дискримінантні функції нейрона, кількість входів одношарового персептрона, навчальну і контрольну вибірки (набори пар значень входів і бажаних виходів персептрона), значення максимальної припустимої помилки навчання, значення параметра збіжності навчання.

2. Написати і відлагодити програму на алгоритмічній мові високого рівня, що моделює формальний нейрон (одношаровий персептрон) і реалізує алгоритми навчання дискретного і дійсного персептронів. Передбачити в програмі відображення на екран і запис у файл на диску поточного стану параметрів персептрона і результатів його роботи: матриці ваг, помилки класифікації (оцінювання), значень на входах і виході персептрона, а також часу навчання і класифікації на основі персептрона.

3. Для вибірки даних здійснити навчання персептрона і зберегти у файлі на диску результати його роботи для навчальної і контрольної вибірок.

4. Кілька разів змінити вид функції активації нейрона і виконати п. 3.

5. Результати виконання п. п. 3–4 занести у таблицю, стовпці якої повинні мати назви: вид функції активації, метод навчання,

час навчання, час класифікації (оцінювання) для навченого персептрона (для навчальної і контрольної вибірок), помилка класифікації (оцінювання) для навчальної і контрольної вибірок. Проаналізувати отримані результати і зробити висновки про те, як впливає вид функції активації формального нейрона на час навчання і класифікації (оцінювання), а також величину помилки навчання і класифікації (оцінювання) одношарового персептрона.

6. На основі розробленої програми, змінюючи значення кроку навчання, для персептрона з заданою функцією активації дослідити, як впливає величина кроку навчання на час навчання. Побудувати графік залежності часу навчання персептрона від величини кроку навчання.

 *Завдання 9.* Емуляція та навчання багатошарового персептрона.

1. Задати число шарів і число нейронів у кожному шарі, вид функції активації нейронів, кількість входів і виходів персептрона, набір пар значень входів і бажаних виходів персептрона, максимально припустиму помилка навчання; крок навчання.

2. Розробити на алгоритмічній мові програмування високого рівня програму, що моделює багатошаровий персептрон та здатна навчати його на основі методу зворотного поширення помилки.

3. Здійснити навчання і моделювання багатошарового персептрона для заданих даних.

4. Кілька разів змінити вид функції активації нейрона і виконати п. 3.

5. Результати моделювання занести у таблицю, стовпці якої повинні мати назви: вид функції активації нейронів, помилка навчання, помилка розпізнавання, час навчання, час класифікації персептрона.

6. Проаналізувати отримані результати і зробити висновки про те, як впливають вид функції активації на час навчання і класифікації (оцінювання), а також величину помилки навчання / класифікації багатошарового персептрона.

 *Завдання 10.* Емуляція та навчання радіально-базисної нейромережі.

1. Задати навчальну і контрольну вибірки даних.

2. Розробити програму на алгоритмічній мові високого рівня, що реалізує навчання і моделювання РБНМ.

3. Здійснити навчання і моделювання РБНМ для навчальних і тестових даних.

4. Результати моделювання занести в таблицю, стовпці якої повинні мати назви: помилка навчання, помилка розпізнавання, час навчання, час класифікації.

5. Проаналізувати отримані результати.

РОЗДІЛ 3

НЕЙРОННІ МЕРЕЖІ ЗІ ЗВОРОТНИМИ ЗВ'ЯЗКАМИ

Нейромережі зі зворотними зв'язками – це нейромережі, де окрім *прямих зв'язків* (значення з виходу нейрону попереднього шару подається на вхід нейронів наступного шару) наявні *зворотні зв'язки* (*зв'язки нейронів самих з собою*, коли значення з виходу нейрона подається на його вхід, та (або) зв'язки наступного шару з попереднім, коли значення з виходів нейронів наступного шару подаються на входи нейронів попереднього шару).

До класу цих мереж відносять нейромережі Хопфілда та Елмана, що розглядаються у даному розділі.

3.1 Нейромережа Хопфілда

НМ Хопфілда (псевдоінверсна НМ) задається четвіркою $net=(N, w, \theta, x)$, де N – кількість нейронів у мережі, $\theta = \{\theta_1,\theta_2,...,\theta_N\}$ – вектор зовнішніх впливів (рис. 3.1).

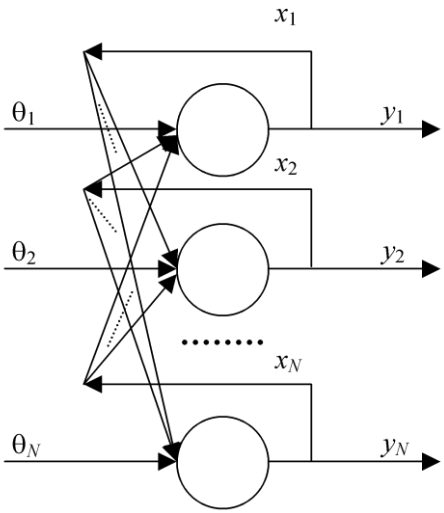


Рисунок 3.1 – Нейромережа Хопфілда

Кількість входів і виходів мережі становить N. Нейрони зв'язані за принципом «усі з усіма», це значить, що у мережі є $N \times N$ зв'язків. Також кожний нейрон має один вхід, на який подається зовнішній сигнал. Зв'язок між i-м та j-м нейронами позначається як w_{ij}. Величина w_{ij} називається *вагою зв'язку* і може бути нулем, позитивним чи негативним числом. Ваги зв'язків задаються *матрицею зв'язків* $w = \{w_{ij}\}$, $i, j = 1, .., N$. У моделі Хопфілда *зв'язки симетричні*, тобто $w_{ij} = w_{ji}$. Нейрони не мають власний зворотних зв'язків самих з собою ($w_{ii}=0$).

Стан мережі визначається *вектором станів нейронів* $x=\{x_1, ..., x_N\}$. Нейрон розглядається як двостабільний пороговий елемент. Стан x_i нейрона i може мати два значення 0 та 1 або –1 та 1. Нейрон i має зовнішній вхід θ, входи від інших нейронів x_j і один вихід, що дорівнює x_i. Постсинаптичний потенціал нейрона i визначається сумою зважених станів, зв'язаних з ним нейронів:

$$\varphi_i = \sum_{j=1}^{N} w_{ij} x_j + \theta_i .$$

У залежності від величини входу φ_i нейрон i змінює свій стан або залишається у попередньому стані відповідно до порогового правила $\varphi_i^{k+1} = \psi(\varphi_i^k)$, де k, $k+1$ – номери старого і нового станів нейрона i, а $\psi(x)$ – функція активації нейрона (порогова або сигмоїдна).

Мережа може *змінювати* свій *стан* синхронним або асинхронним способом.

У *синхронному* випадку всі нейрони одночасно змінюють свої стани. Аналітичний вираз переходу мережі зі стану x_k у x_{k+1} записується в матричній формі: $\varphi_k = wx_k + \theta_k$, $x_{k+1} = \psi(\varphi_k)$, де $x_k = \{x_1^k, x_2^k, ..., x_N^k\}$, $\varphi_k = \{\varphi_1^k, \varphi_2^k, ..., \varphi_N^k\}$ Функція ψ застосовується до вектора φ_k поелементно.

В *асинхронному* випадку кожен нейрон може змінювати свій стан випадково, при цьому він використовує інформацію про поновлені стани інших нейронів. Аналітичний запис переходу мережі зі стану x_k у x_{k+1} в асинхронному випадку, коли нейрон m змінює свій стан, має вигляд: $\varphi_m^k = w_m x_k + \theta_k$,

$x_{k+1}=\{x_1^k,...,\psi(\varphi_m^k),..., x_N^k\}$, де w_m – рядок матриці w з номером m.

Починаючи з *початкового стану* x_0 і працюючи синхронно або асинхронно, мережа генерує послідовність станів x_0, x_1, ..., x_k, що у сприятливих випадках закінчується *стійким станом*, у несприятливих випадках можуть виникнути коливання.

Основною операцією, що виконується НМ Хопфілда, є множення матриці на вектор (у синхронному випадку) або вектора на вектор (в асинхронному випадку) з наступним обчисленням нелінійної функції. Однак, завдяки масовості зв'язків великого числа нейронів при такій досить простій операції мережа має здатність вирішувати складні задачі.

3.2 Навчання нейромережі Хопфілда

Мережа Хопфілда з N нейронів може відновлювати образи розміру N, що запам'ятовані в мережі, за ключем, тобто за неповною або неточною інформацією про ці образи. Це дозволяє створювати на основі НМ блоки асоціативної пам'яті в обчислювальних системах. Еталонні образи кодуються словами довжиною N, що складаються з 1 та –1 (0 та 1). Перехід нейрона зі стану 1 у –1 (та навпаки) будемо вважати східчастим.

Роботу НМ, що містить нейрони, стан яких визначається дією зовнішніх стимулів, виходи якої є реакцією на стимул, можна розглядати як пофрагментну класифікацію діючих стимулів. Множина передсинаптичних потенціалів, що відповідає позитивній реакції нейрона $x_i(t) = 1$, створює компактну область навколо вектора w_{ij}.

Ця область не порожня лише при виконанні умови:

$$\theta_i(t) \le \left(N \sum_{j=1}^{N} w_{ij}^2 \right)^{0,5}.$$

Комбінуючи значення вагових коефіцієнтів w_{ij} і порогів θ_i можна створювати НМ, здатні до класифікації образів без обмежень. Знаходження значень величин w_{ij} та $\theta_i(t)$, що забезпечують необхідні реакції на заданій множині стимулів, складає задачу навчання НМ.

Асоціювання (чи розпізнавання) образа досягається мережею шляхом еволюції з початкового стану, що відповідає введеному образу, у кінцевий стан, яким є асоціація образа з запам'ятованим раніше образом.

Основна задача побудови мережі складається в наділенні її розпізнавальними властивостями, які виявляються у тому, що при подачі на вхід мережі зашумленої версії образа, мережа на виході здатна відновлювати оригінал із шуму. Тобто, маючи S образів-еталонів необхідно знайти таку матрицю зв'язків w, що змушувала б мережу виявляти розпізнавальні властивості, щодо цих еталонів. Знаходження такої матриці складає процес навчання НМ, а правило обчислення матриці w є навчальним правилом.

Продуктивність НМ визначається кількістю еталонів S, що можуть бути запам'ятовані і розпізнані мережею, що складається з N нейронів, і тим, наскільки добре відбувається розпізнавання, тобто відділення еталонів від шуму при даній кількості еталонів S.

Помітним кроком на шляху до збільшення продуктивності псевдоінверсних НМ була пропозиція Хопфілда розглядати їх з енергетичної точки зору. Процес розпізнавання можна представити як «зсув» мережі в мінімуми деякої *енергетичної функції* E у просторі станів. Запропоноване Хопфілдом визначення цієї функції має вигляд:

$$E(t) = -0{,}5\, x(t)(wx(t)^{\mathrm{T}} - \theta(t)),$$

де $x(t)$ – вектор стану системи, $^{\mathrm{T}}$ – знак транспонування, $\theta(t)$ – зовнішнє поле, що визначає поріг чутливості, w – оператор, що враховує відстань між станами системи. Дану функцію можна спростити, якщо вважати зовнішні сигнали відсутніми: $E(t) = -0{,}5\, x(t)(wx(t)^{\mathrm{T}})$.

Проекційний метод навчання (*псевдоінверсний метод*) НМ Хопфілда полягає у тому, що ми здійснюємо проекцію навчальної множини на множину ваг НМ, що дозволяє навчати мережу один раз перед використанням, і не вимагає виконання ітеративного процесу корекції ваг, як, наприклад, в методах навчання персептронів.

Нехай є S еталонних образів $x_в$, k = 1, ..., S. Поклавши θ_i=0, i=1, 2, ..., N, можемо записати вираз для знаходження ваг мережі:

$$w_{ij} = \begin{cases} \sum_{k=1}^{S} x_i^k x_j^k, i \neq j; \\ 0, i = j. \end{cases}$$

Починаючи роботу в стані x_0, мережа може потрапити в одне з наступних трьох положень:

– прийти у стійкий стан, що відповідає еталонному образу, який за хеммінговою відстанню (числом компонентів, за якими розрізняються два вектори) є найбільш близьким до x_0;

– прийти у стійкий стан, що не відповідає ніякому еталону, тобто до помилкового образа;

– виявитися втягнутою у коливальний процес.

Результати моделювання мережі показують, що мережа працює добре, тобто без помилок відновлює еталонні образи з випадкових, якщо до неї записується не більш, ніж 0,15N еталонних образів.

Для більшості задач можливості одноразового внесення інформації в асоціативну пам’ять виявляється недостатньо, оскільки потрібно в процесі роботи додавати у пам’ять нову інформацію – проекційний метод виявляється непридатним. Для вирішення таких задач використовують *ітеративний проекційний метод навчання*.

Нехай ми маємо навчену для S екземплярів НМ Хопфілда, тоді для внесення у пам’ять S+1 екземпляра, значення ваг можна установити за формулою:

$$w_{ij}^{S+1} = w_{ij}^{S} + \frac{(x_i^S - \sigma_i)(x_j^S - \sigma_j)}{\sum_{j=1}^{N} x_j^{S+1}(x_j^{S+1} - \sigma_j)},$$

де $\sigma_i = \sum_{j=1}^{N} w_{ij}^S x^{S+1}$.

Для спрощення обчислювальної процедури запропоновані й інші правила налаштовування ваг, серед яких особливо варто виділити:

$$w_{ij}^{S+1} = w_{ij}^{S} + \frac{(x_i^{S+1} - \sigma_i)x_j^{S+1}}{N}.$$

У 1995 році Д. О. Городничий, вивчаючи НМ Хопфілда з перенасиченою пам'яттю, знайшов, що при зменшенні ваги зв'язків, які замикають вихід нейрона на його вхід, НМ здатна згадувати образи, що здавалися безнадійно загубленими. Цей «*ефект рознасичення*» був покладений в основу розробки нового методу підвищення обсягу асоціативної пам'яті НМ Хопфілда, що дозволило майже в два рази збільшити теоретичну границю для обсягу пам'яті та зробило можливим регулювати кількість образів, що запам'ятовуються.

Теоретична межа обсягу пам'яті для псевдоінверсного навчального правила складає 50% від кількості нейронів мережі, але практично вона є недосяжною. При навчанні на основі псевдоінверсного навчального правила поведінка мережі істотно залежить від рівня зворотного зв'язку нейронів. Зменшення зворотного зв'язку буде сприяти поліпшенню характеристики навчального правила НМ.

Теоретичний аналіз цієї методики і її експериментальна перевірка шляхом програмного моделювання, показують, що обсяг пам'яті модифікованої псевдоінверсної НМ може не тільки досягати теоретичної границі 50% від кількості нейронів, але і значно перевищувати її. Запропоновану методику назвали рознасиченням мережі, а мережу, побудовану за цією методикою, – рознасиченою псевдоінверсною НМ.

Для псевдоінверсного навчального правила при збільшенні заповнення пам'яті НМ (S/N) діагональні елементи синаптичної матриці починають домінувати над іншими її елементами, а зі збільшенням ваги діагональних елементів зменшується імовірність влучення мережі в локальні мінімуми. Ці два факти дозволили запропонувати модифікацію псевдоінверсного навчального правила.

Після того, як значення синаптичної матриці знайдені, усі діагональні елементи цієї матриці необхідно частково зменшити за правилом: $w^*_{ii} = Dw_{ii}$, $0 < D < 1$.

Таке скорочення послаблює рівень негативного зворотного зв'язку нейронів, що приводить до визначеної дестабілізації поведінки НМ. Зменшуючи діагональні елементи матриці, ми зменшуємо величину співвідношення w_{ii} / w_{ij}, що дає ефект, схожий на скорочення кількості запам'ятованих еталонів, тобто скорочення зменшення насичення пам'яті мережі. Тому псевдоінверсне навчальне правило зі скороченими зворотними зв'язками називають *рознасиченим псевдоінверсним правилом*. Повну НМ, побудовану за цим правилом назвали *рознасиченою мережею*. Синаптична матриця w^* для рознасиченого псевдоінверсного правила може бути визначена як: $w^* = xx - (1 - D) I$, де D – коефіцієнт рознасичення, $0< D <1$, оптимальне значення коефіцієнта D лежить у межах 0,1–0,2; I – одинична матриця.

При рознасиченні після навчання НМ діагональні елементи синаптичної матриці збільшуються на позитивний коефіцієнт $D < 1$. При випробуванні такої мережі, постсинаптичний потенціал кожного нейрона одержує приріст: $d_i = (D-1)w_{ii}x_i$.

Оскільки $D < 1$, а $w_{ii} > 0$, збільшення має знак, протилежний виходу нейрона, і діє як дестабілізуючий фактор. Критичний обсяг пам'яті мережі при деформації збільшується, і подвоюється при $D = 0$.

Експериментально підтверджена можливість значного (у 2–3 рази) збільшення обсягу асоціативної пам'яті НМ при ослабленні діагоналі в 3–5 разів (D = 0,2–0,3). Подальше зменшення ваги діагональних елементів синаптичної матриці при паралельній організації нейрообчислень збільшує ризик утрати стабільності мережі.

3.3 Нейромережа Елмана

Нейронна мережа Елмана – це рекурентна мережа, яку можна отримати з багатошарового персептрона введенням зворотних зв'язків від виходів внутрішніх нейронів. Це дозволяє врахувати передісторію процесів, що спостерігаються, і

нагромадити інформацію для вироблення правильної стратегії керування.

Найпростіша мережа Елмана складається з одного прихованого й одного контекстного шару нейронів (рис. 3.2).

Зворотні зв'язки містять *елементи затримки сигналу* й утворюють *контекстний шар* нейронів. У мережі Елмана кількість нейронів контекстного і прихованого шарів збігається. Крім того, тут немає власних зворотних зв'язків у нейронів контекстного шару.

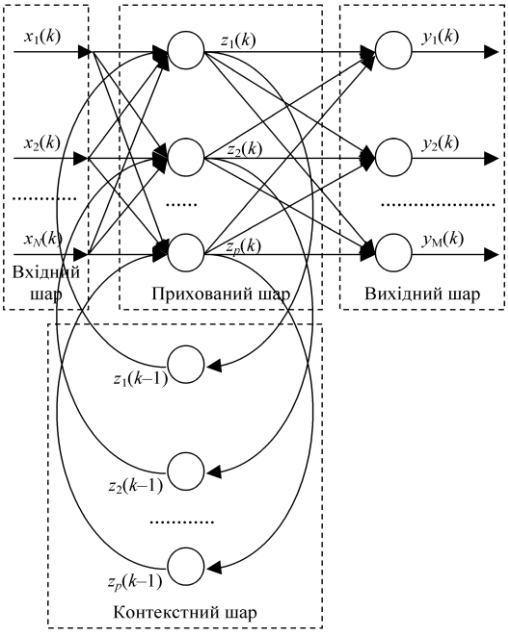


Рисунок 3.2 – Мережа Елмана

Надходження на вхід мережі першого екземпляра активує нейрони всіх шарів: прихованого, вихідного і контекстного. При цьому нейрони контекстного шару перейдуть у новий стан, що відповідає стану нейронів прихованого шару, копіюючи тим самим (запам'ятовуючи) інформацію. З виходів нейронів прихованого шару сигнал передається на входи

нейронів вихідного шару, які формують вихідний сигнал мережі. При надходженні наступного екземпляра стан нейронів контекстного шару відповідає попередньому екземпляру.

Основним завданням нейронів прихованого шару є вироблення бажаного вихідного сигналу шляхом порівняння нового екземпляра, що надійшов на входи, та екземпляра, який був запам'ятований у контекстному шарі.

Якщо відкинути сигнали зворотних зв'язків, що надходять на контекстний шар, то вийде мережа прямого поширення, у якої контекстний шар є додатковим вхідним. При цьому розширений вхідний вектор складається з вхідного вектора і вектора станів нейронів контекстного шару, обумовленого на кожному такті функцією переходів.

Оскільки мережа Елмана описуються нелінійними рівняннями, для настроювання параметрів цих мереж використовують методи нелінійної оптимізації, серед яких найчастіше застосовують градієнтні методи.

3.4 Приклади виконання завдань

Завдання 1. Чи можна навчити бінарну НМ Хопфілда відновлювати за ключем 1001**** запам'ятований набір кодових слів: 10011001, 01010010, 10010001, 10011011?

Оскільки кодове слово 01010010 не відповідає ключу, то виключимо його з подальшого розгляду.

Випишемо ті кодові слова, що відповідають заданому ключу (ключ виділимо жирним шрифтом).

10011001

10010001

10011011

Оскільки заданому ключу у запам'ятованій послідовності відповідає більше одного образу (кодового слова), то за заданим ключем не можна буде видати тільки один єдиний результат, оскільки мережа буде знаходитися у коливальному процесі

переходячи між запам'ятованими кодовими словами як стабільними станами.

? 3.5 Контрольні питання

1. Дайте визначення понять: повнозв'язна мережа, мережа зі зворотними зв'язками, функція обчислювальної енергії.
2. Псевдоінверсне навчальне правило.
3. Проективний метод настроювання ваг.
4. Ефект рознасичення (ефект Городничого).
5. Бінарні повнозв'язні нейромережі Хопфілда.
6. Чи дозволяє модель і традиційні методи навчання мереж Хопфілда побудувати на її основі асоціативний запам'ятовуючий пристрій, здатний запам'ятовувати стільки образів, скільки нейронів у мережі? Відповідь поясніть.
7. Які задачі можна вирішувати на основі бінарних НМ Хопфілда, а які не можна? Обґрунтуйте і доведіть відповідь. Наведіть приклади.
8. Чи доцільно застосовувати бінарні мережі Хопфілда для класифікації складно (нелінійно) роздільних образів?
9. Чи завжди збігаються проекційні методи навчання мережі Хопфілда?
10. Метод рознасичення синаптичної матриці мережі Хопфілда.
11. Нейромережа Елмана.
12. Чи доцільно застосовувати мережі Хопфілда при вирішенні задач, для рішення яких може використовуватися одношаровий персептрон?
13. Застосування нейромереж для асоціативного пошуку інформації.
14. Мережі Хопфілда у задачах комбінаторної оптимізації.
15. Порівняння мереж Хопфілда та Елмана.
16. Порівняйте можливості бінарних мереж Хопфілда і дискретного одношарового персептрона.
17. Порівняйте можливості бінарних мереж Хопфілда і багатошарового персептрона.
18. Порівняйте можливості бінарних мереж Хопфілда і радіально-базисних мереж.

19. Використання мереж зі зворотними зв'язками при вирішенні практичних задач.

20. Розпізнавання зображень букв тексту за допомогою мереж Хопфілда.

21. Доцільність та обмеження практичного використання мереж зі зворотними зв'язками.

22. Порівняйте мережі прямого поширення та мережі зі зворотними зв'язками.

23. Обмеження методів синтезу мереж Хопфілда та засоби їх подолання.

24. Розпізнавання образів: постановка задачі та можливість застосування мереж зі зворотними зв'язками.

25. Асоціативний пошук інформації на основі мереж Хопфілда.

3.6 Практичні завдання

Завдання 1. Чи можна навчити бінарну НМ Хопфілда відновлювати за ключем 1101**** запам'ятовані набори кодових слів: а) 11011001, 01010010, 11001111, б) 11011001, 11011010, 01011111, в) 11011001, 11010010, 11011111? Відповідь поясніть.

Завдання 2. Емуляція та навчання нейромережі Хопфілда.

1. Задати кількість входів нейронів і розмір мережі (число нейронів), набір пар значень входів і бажаних виходів НМ Хопфілда.

2. Написати програму на високорівневій алгоритмічній мові програмування, що моделює НМ Хопфілда і реалізує методи її навчання: проекційний (псевдоінверсний) алгоритм та ітераційний проекційний алгоритм, а також рознасичене псевдоінверсне правило. Передбачити в програмі відображення на екран і запис у файл на диску поточного стану параметрів НМ і результатів її роботи: матрицю ваг, число помилкових рішень розпізнавання, значень на входах і виходах НМ Хопфілда, тривалість навчання і роботи НМ.

3. Для відповідних варіанту вхідних даних зробити навчання НМ кожним з методів і зберегти у файлі на диску результати її роботи.

4. Результати занести у таблицю, стовпці якої повинні мати назви: метод навчання, час навчання, час класифікації, число помилкових рішень при розпізнаванні.

5. Проаналізувати отримані результати і дати порівняльну характеристику методів навчання НМ Хопфілда.

Завдання 3. Побудувати мережу Елмана для даних, що задаються формулою: $y_i = 2\sin(3x_i - x_{i-1})$, де $i = 1, 2, ..., 100$, $y_0 = 0$.

РОЗДІЛ 4
НЕЙРОННІ МЕРЕЖІ З ЛАТЕРАЛЬНИМИ ЗВ'ЯЗКАМИ

До *нейронних мереж із латеральними* (*боковими*) *зв'язками* відносять шаруваті нейромережі прямого поширення сигналу, у яких допускається наявність зв'язків між нейронами усередині одного шару.

Найбільш відомими мережами із латеральними зв'язками є мережі Кохонена.

4.1 Нейронна мережа Кохонена SOM

Карта Кохонена, що самоорганізується, (Kohonen Self-organizing Map – *SOM*) є НМ із латеральними зв'язками і відноситься до класифікаторів, для навчання яких використовуються вибірки образів із заздалегідь не заданою класифікацією.

Схему мережі SOM зображено на рис. 4.1. Вона складається з одного шару нейронів, які використовують як дискримінантну – функцію відстані, а як активаційну – функцію winner-take-all.

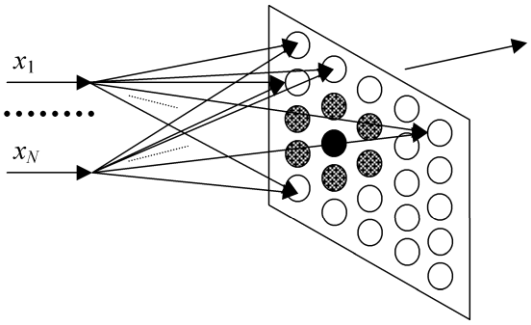


Рисунок 4.1 – Схема нейромережі SOM

Задачею мережі SOM є визначення належності вхідного вектора ознак *s*-го екземпляра вибірки $x^s = \{x^s_1, x^s_2, ..., x^s_N\}^T$ до одного з L можливих кластерів, поданих векторними центрами $w_j = \{w_{j1}, w_{j2}, ..., w_{jN}\}^T$, $j = 1, 2, ..., L$, де ^{T} – символ транспонування.

Позначимо *i*-ту компоненту вхідного вектора x^s у момент часу t як $x^s_i(t)$, а вагу *i*-го входу *j*-го вузла у момент часу t як $w_{ij}(t)$.

Якщо вузли SOM є лінійними, а вага i-го входу j-го вузла дорівнює w_{ij}, $i = 1, 2, ..., N$, $j = 1, 2, ..., L$, тоді при відповідних значеннях порогів кожен i-й вихід мережі з точністю до несуттєвих постійних буде дорівнює евклідовій відстані d_j між пред'явленим вхідним вектором $x^s{}_i$ і j-м центром кластера.

Вважається, що вектор x^s належить до j-го кластера, якщо відстань d_j для j-го центра кластера w_j мінімальна, тобто якщо $d_j \le d_k$ для кожного $k \ne j$.

При навчанні НМ пред'являються вхідні вектори без указівки бажаних виходів і корегуються ваги відповідно до методу, що запропонував Т. Кохонен. *Метод Кохонена*, що формує SOM, вимагає, щоб біля кожного вузла було визначене поле *NE*, розмір якого з часом постійно зменшується.

Крок 1. Ініціюються ваги входів вузлів малими випадковими значеннями. Встановлюється початковий розмір поля *NE*.

Крок 2. Пред'являється новий вхідний вектор x^s.

Крок 3. Обчислюється відстань (метрика) d_j між вхідним вектором і кожним вихідним вузлом j:

$$d_j = \sum_{i=1}^{N} (x^s(t) - w_{ji}(t))^2 .$$

Крок 4. Визначається вузол j^* з мінімальною відстанню d_j.

Крок 5. Корегуються ваги входів вузлів, що знаходяться в полі $NE_{j^*}(t)$ вузла j^*, таким чином, щоб нові значення ваг були:

$$w_{ji}(t+1) = w_{ji}(t) + \eta(t)(x_i^s - w_{ji}(t)) , \quad j \in NE_{j^*}(t), \quad i = 1,2,...,N.$$

При цьому коригувальний приріст $\eta(t)$ ($0 < \eta(t) < 1$) повинний спадати зі зростанням t.

Крок 6. Якщо збіжність не досягнута, то перейти до кроку 2.

Збіжність вважається досягнутою, якщо ваги стабілізувалися і коригувальний приріст η у кроці 5 знизився до нуля.

Якщо кількість вхідних векторів у навчальній множині є великою стосовно обраного числа кластерів, то після навчання ваги мережі будуть визначати центри кластерів, розподілені в просторі входів таким чином, що функція щільності цих центрів буде апроксимувати функцію щільності ймовірності вхідних

векторів. Крім того, ваги будуть організовані таким чином, що топологічно близькі вузли будуть відповідати фізично близьким (за евклідовою відстанню) вхідним векторам.

Важливо відзначити, що при класифікації за допомогою SOM, номер вузла, до якого віднесений екземпляр, і фактичний номер його класу в загальному випадку не збігаються – розділяючи екземпляри, SOM робить суб'єктивну класифікацію, що не має того реального фактичного змісту, яким ми наділяємо класи.

Результати класифікації SOM можуть бути наділені фактичним смислом шляхом постановки у відповідність номеру кожного вузла SOM номера того фактичного класу, до якого відноситься більша частина екземплярів навчальної вибірки, віднесених SOM до даного вузла.

4.2 Нейронна мережа Кохонена LVQ

Квантування навчальних векторів (Learning Vector Quantization – LVQ) є продовженням і розвитком ідеї SOM.

Нейронна мережа LVQ (рис. 4.2) складається з двох послідовно з'єднаних шарів нейронів: конкуруючого шару (SOM) і лінійного шару. Обидва шари НМ LVQ містять по одному конкуруючому та одному лінійному нейрону на кожен підклас (кластер) та цільовий клас. Позначимо S^1 – кількість підкласів (кластерів), S^2– кількість цільових класів (S^1 завжди буде більше, ніж S^2).

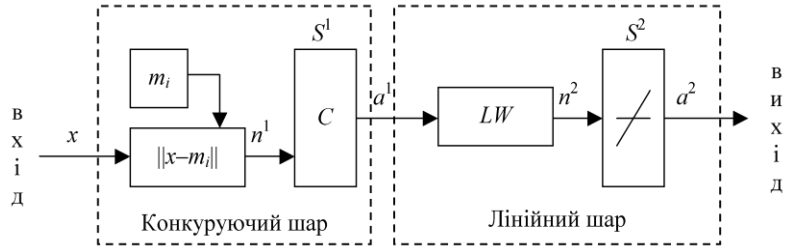


Рисунок 4.2 – Схема нейронної мережі LVQ у векторній формі

Конкуруючий шар являє собою мережу SOM і здійснює поділ вхідних векторів x на класи (кластери), виділяючи центри зосередження вхідних векторів (центри, еталони кластерів) m_i.

Для цього визначаються відстані $n^1=\|x - m_i\|$ між вхідними векторами x і початковими значеннями центрів зосередження векторів m_i, $i = 1, 2, \ldots, q$, де q – кількість вхідних векторів.

Лінійний шар містить нейрони, що використовують дискримінантну функцію зважена сума та лінійну функцію активації. Він перетворює клас вхідного вектора, визначений конкуруючим шаром – підклас a^1, у клас, визначений користувачем – цільовий клас a^2, шляхом множення a^1 на значення ваг LW лінійних нейронів, що встановлюються рівними 1, якщо цільовий клас і підклас збігаються і 0 – у протилежному випадку. Відповідні добутки $n^2=a^1LW$ подаються на виходи всіх лінійних нейронів, утворюючи бінарний вектор a^2, всі елементи якого рівні 0, за винятком елемента, що відповідає цільовому класу (цей елемент дорівнює 1).

4.3 Методи навчання мережі LVQ

В основі методів LVQ лежить механізм навчання шару конкуруючих нейронів (конкуруючого шару), контрольований вчителем.

Конкуруючий шар може автоматично навчатися класифікувати вхідні вектори. Поділ класів, що визначає карта Кохонена, заснований тільки на відстані між вхідними векторами. Якщо два вхідних вектори дуже близькі, то конкуруючий шар, дуже ймовірно, віднесе їх до одного класу. Однак, поділ на класи, вироблений конкуруючим шаром, як правило, не збігається з тим, що визначає вчитель. Виникає задача розробки механізму, який би дозволяв карті Кохонена здійснювати класифікацію, близьку до заданої вчителем. Методом, що дозволяє вирішувати поставлену задачу, є LVQ.

Метод навчання LVQ1. Нехай деяка кількість векторів з вільними параметрами (центрів кластерів) m_i поміщена у вхідний простір для апроксимації різних областей вхідного вектора x їх квантованими значеннями. Кожному класу значень x призначається кілька векторів з вільними параметрами, і потім приймається рішення про віднесення x до того класу, до якого належить найближчий вектор m_i. Нехай індекс c визначає найближчий до x вектор m_i, позначений далі як m_c:

$$c = \underset{i}{\operatorname{argmin}} \left\{ \left\| x - \mathrm{m}_i \right\| \right\} .$$

Значення для m_i, що мінімізують помилку класифікації, можуть бути знайдені як асимптотичні значення в наступному процесі навчання. Нехай $x(t)$ – екземпляр, поданий на входи, $y(t)$ – номер класу, до якого відноситься екземпляр $x(t)$, $m_i(t)$ – подання послідовності m_i, дискретизованої за часом, $Y_i(t)$ – номер класу, до якого відноситься $m_i(t)$. Починаючи з правильно визначених початкових значень, основний процес методу LVQ1 визначають правилами:

$$m_c(t + 1) = m_c(t) + \alpha(t)[x(t) - m_c(t)], y(t)=Y_c(t);$$

$$m_c(t + 1) = m_c(t) - \alpha(t)[x(t) - m_c(t)], y(t) \neq Y_c(t);$$

$$m_i(t + 1) = m_i(t) , \forall i \neq c.$$

Тут $0 < \alpha(t) < 1$, $\alpha(t)$ може бути константою або монотонно зменшуватися з часом. Для методу LVQ1 рекомендується, щоб первісне значення α було менше 0,1.

Метод навчання LVQ2 вирішує задачу класифікації ідентично методу LVQ1. Однак у процесі навчання LVQ2 два вектори з вільними параметрами m_i і m_j, що є найближчими сусідами x, модифікуються одночасно. При цьому вони повинні належати до різних класів. Крім того, x повинний знаходитися в зоні значень, що називається «вікном», яке визначене навколо середини площини, утвореної векторами m_i і m_j. Нехай d_i і d_j – евклідові відстані x від m_i і m_j, тоді x визначено потрапить у «вікно» відносної ширини w, якщо

$$\min\left(\frac{d_i}{d_j} , \frac{d_j}{d_i} \right) > s ,$$

де $s = \dfrac{1-w}{1+w}$.

Рекомендується, щоб значення відносної ширини «вікна» w знаходилися в межах від 0,2 до 0,3.

Метод навчання LVQ2 має вид:

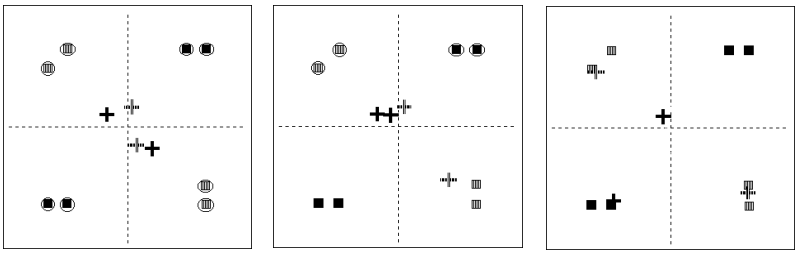
$$m_i(t + 1) = m_i(t) - \alpha(t)[x(t) - m_i(t)],$$

$$m_j(t + 1) = m_j(t) + \alpha(t)[x(t) - m_j(t)],$$

де m_i і m_j – два найближчих до x вектора з вільними параметрами,

причому x і m_j належать до одного класу, у той час як x і m_i належать до різних класів, крім того, x повинний потрапляти в «вікно».

На рис. 4.3 квадратами позначені значення ознак екземплярів, а хрестами – значення ваг НМ LVQ. Штрихування фігур позначає їхню приналежність до класу (суцільна – клас 1, пунктирна – клас 2). Помилково класифіковані екземпляри виділені колами. На рис. 4.3, *а* показане початкове розташування ваг НМ LVQ – всі екземпляри класифіковані невірно. На рис. 4.3, *б* для тих же екземплярів показане положення ваг НМ LVQ після однієї ітерації навчання за допомогою методу LVQ2 – одночасно модифіковані значення двох векторів, кількість неправильно класифікованих екземплярів зменшилася. На рис. 4.3, *в* показане положення ваг НМ LVQ після 10 ітерацій навчання – усі екземпляри класифіковані вірно.



а *б* *в*

Рисунок 4.3 – Процес квантування навчальних векторів:

а – початкове положення; *б* – після першої ітерації;
в – після десяти ітерацій навчання

Метод LVQ3. Метод навчання LVQ2 заснований на ідеї диференціального зсуву меж рішення відносно Байєсовських меж, при цьому не враховується те, що може відбутися з положенням m_i у випадку досить тривалої роботи методу. Отже, необхідно внести зміни, що гарантували б, що m_i хоча б грубо продовжить апроксимацію розподілу класів. При поєднанні цих ідей, ми одержуємо поліпшений метод – LVQ3:

$$m_i(t+1) = m_i(t) - \alpha(t)[x(t) - m_i(t)],$$

$$m_j(t+1) = m_j(t) + \alpha(t)[x(t) - m_j(t)],$$

Для $k \in \{i, j\}$, якщо x, m_i і m_j належать тому самому класу:

$$m_k(t+1) = m_k(t) + \varepsilon\alpha(t)[x(t) - m_k(t)],$$

де m_i і m_j – два найближчих до x вектора з вільними параметрами, причому x і m_j належать до тому самому класу, у той час як x і m_i належать різним класам, крім того, x повинний попадати в «вікно».

У результаті ряду експериментів було встановлено, що значення ε повинні знаходитися між 0,1 і 0,5. Оптимальне значення ε, можливо, залежить від розміру найменшого вікна.

Цей метод є таким, що самостабілізується, тобто оптимальне розміщення m_i не змінюється при тривалому навчанні.

Метод навчання OLVQ1 (Optimized-learning-rate LVQ1) являє собою метод LVQ1, модифікований таким чином, щоб кожному m_i була призначена індивідуальна швидкість навчання $\alpha_i(t)$. Таким чином, ми одержуємо наступний дискретизований за часом процес навчання.

Нехай c визначається рівнянням:

$$c = \arg\min_i \left\{ \| x - m_i \| \right\} .$$

Тоді встановити:

– $m_c(t + 1) = m_c(t) + \alpha_c(t)[x(t) - m_c(t)]$, якщо x класифікований правильно;

– $m_c(t + 1) = m_c(t) - \alpha_c(t)[x(t) - m_c(t)]$, якщо x класифікований неправильно;

– $m_c(t + 1) = m_i(t)$, $\forall i \neq c$.

Розглянемо спосіб визначення оптимального $\alpha_i(t)$ для найбільш швидкої збіжності OLVQ1. Виразимо розглянуті рівняння у формі:

$$m_c(t + 1) = [1 - s(t)\alpha_c(t)]\, m_c(t) + s(t)\alpha_c(t)\, x(t),$$

де $s(t) = +1$, якщо класифікація правильна і $s(t) = -1$, якщо класифікація неправильна.

Відзначено, що $m_c(t)$ статистично не залежить від $x(t)$, і статистична точність отриманих значень векторів з вільними параметрами оптимальна, якщо результати виправлень зроблені в різний час.

Помітимо, що $m_c(t + 1)$ містить слід $x(t)$ через останній член в останньому рівнянні і прослідкує попередні $x(t')$, $t' = 1, 2, ..., t-1$ через $m_c(t)$.

Абсолютна величина останнього сліду $x(t)$ масштабується коефіцієнтом $\alpha_c(t)$ і, у свою чергу, слід $x(t - 1)$ масштабується коефіцієнтом $[1 - s(t)\, \alpha_c(t)]\, \alpha_c(t - 1)$.

Тепер передбачимо, щоб обоє ці масштабування були ідентичні

і застосуємо дана умова для всіх *t*. Тоді «сліди» усіх попередніх *x*, зібрані до часу *t*, будуть наприкінці масштабуватися одним числом, і, отже, «оптимальні» значення $\alpha_i(t)$ визначаються рекурсивно:

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 + s(t)\alpha_c(t)}.$$

На практиці можна переконатися, що це правило забезпечує швидку збіжність.

Однак, помітимо, що $\alpha_c(t)$ може також збільшуватися, і важливо, щоб значення $\alpha_c(t)$ не перевищувало 1. Початкові значення $\alpha_i(t)$ можуть бути обрані досить високими (наприклад, 0,3), завдяки чому навчання значне прискорюється (особливо на початку) і наближені асимптотичні значення m_i знаходяться досить швидко.

Варто звернути увагу на те, що розглянуте вираження не застосовне для LVQ2, тому що α_i, у середньому, не буде зменшуватися і процес не буде сходитися.

4.4 Приклади виконання завдань

Приклад 1. Нехай задано розпізнаваний екземпляр $x = (2{,}3\ 1\ -2\ 0)^T$, поданий на входи мережі SOM, яка має три нейрони. Ваги нейронів дорівнюють: $w^{(1,1)} = (0\ 3\ -2\ 1)^T$, $w^{(1,2)} = (-1\ 8\ 5\ 3)^T$ та $w^{(1,3)} = (1\ 1\ -1\ 1)^T$, а пороги вважаються відсутніми. Визначити значення на виходах нейронів SOM.

Спочатку визначимо для нейронів значення дискримінантних функцій:

$$\varphi^{(1,1)}(w^{(1,1)}, x^{(1,1)}) = (0-2{,}3)^2 + (3-1)^2 + (-2-(-2))^2 + (1-0)^2 = 10{,}29;$$

$$\varphi^{(1,2)}(w^{(1,2)}, x^{(1,2)}) = (-1-2{,}3)^2 + (8-1)^2 + (5-(-2))^2 + (3-0)^2 = 117{,}89;$$

$$\varphi^{(1,3)}(w^{(1,3)}, x^{(1,3)}) = (1-2{,}3)^2 + (1-1)^2 + (-1-(-2))^2 + (1-0)^2 = 3{,}69.$$

Після цього для нейронів визначимо значення функції активації «переможець забирає усе»: $\psi^{(1,1)}=0$, $\psi^{(1,2)}=0$, $\psi^{(1,3)}=1$.

Приклад 2. Нехай задано розпізнаваний екземпляр $x = (2\ -1\ 1)^T$, поданий на входи мережі LVQ, яка має два нейрони на першому

шарі та два нейрони на другому шарі. Ваги нейронів першого шару дорівнюють: $w^{(1,1)}=(3\ -2\ 1)^{\mathrm{T}}$ та $w^{(1,2)}=(8\ 5\ 3)^{\mathrm{T}}$, а пороги – одиниці (або вважаються відсутніми). Ваги нейронів другого шару дорівнюють: $w^{(2,1)}=(1\ 0)^{\mathrm{T}}$ та $w^{(1,2)}=(0\ 1)^{\mathrm{T}}$, а пороги – нулю. Визначити номер класу, до якого має віднести мережа розпізнаваний екземпляр x.

Спочатку визначимо для нейронів першого шару значення дискримінантних функцій:

$$\varphi^{(1,1)}(w^{(1,1)}, x^{(1,1)}) = (3-2)^2 + (-2-(-1))^2 + (1-1)^2 = 2;$$

$$\varphi^{(1,2)}(w^{(1,2)}, x^{(1,2)}) = (8-2)^2 + (5-(-1))^2 + (3-1)^2 = 76.$$

Після чого для нейронів першого шару визначимо значення функції активації «переможець забирає усе»: $\psi^{(1,1)}=1$, $\psi^{(1,2)}=0$.

Далі визначимо для нейронів другого шару значення дискримінантних функцій (а через те, що їхні функції є лінійними, то фактично і значення на виходах нейронів):

$$y^{(2,1)} = \varphi^{(2,1)}(w^{(2,1)}, x^{(2,1)}) = 1\cdot 1 + 0\cdot 0 = 1;$$

$$y^{(2,2)} = \varphi^{(2,2)}(w^{(2,2)}, x^{(2,2)}) = 0\cdot 1 + 1\cdot 0 = 0.$$

? 4.5 Контрольні питання

1. У чому полягає відмінність мереж Кохонена між собою та у порівнянні з іншими нейромережами?
2. Запропонуйте способи урахування апріорної інформації про значимість ознак при навчанні SOM.
3. Запропонуйте стратегію (метод) визначення необхідної кількості нейронів конкуруючого шару LVQ.
4. Обмеження мереж LVQ.
5. Опишіть структуру мережі LVQ.
6. Опишіть структуру мережі SOM.
7. Порівняйте мережі Кохонена з бінарною мережею Хопфілда.
8. Порівняйте мережі Кохонена з мережею Елмана.
9. Порівняйте мережі Кохонена з персептронами.
10. Порівняйте можливості SOM, мереж Хопфілда та одношарового дискретного персептрона.
11. У чому полягає метод навчання LVQ1?
12. У чому полягає метод навчання LVQ2?
13. У чому полягає метод навчання LVQ3?

14. У чому полягає метод навчання OLVQ?
15. У чому полягає навчання мережі Кохонена LVQ?
16. Чи доцільно використовувати SOM для попереднього аналізу даних у задачах діагностики і прогнозування?
17. Як впливають вид і параметри метрики на точність визначення центрів зосередження екземплярів. Відповідь поясніть і проілюструйте.
18. Які задачі можна вирішувати на основі мереж SOM і LVQ, а які не можна? Обґрунтуйте і доведіть відповідь. Наведіть приклади.
19. Які метрики можна використовувати для побудови мереж SOM?

4.6 Практичні завдання

Завдання 1. Підготувати реферат на одну з таких тем.

1. Застосування мереж Кохонена у технічній діагностиці.
2. Застосування мереж Кохонена у медичній діагностиці.
3. Застосування мереж SOM для кластер-аналізу.
4. Застосування мереж SOM у картографії.
5. Застосування мереж SOM для створення пошукових систем.
6. Застосування мереж Кохонена для розпізнавання мови.
7. Застосування мереж SOM для планування експерименту.
8. Нейронні мережі з латеральним гальмуванням.

Завдання 2. Нехай задано розпізнаваний екземпляр $x = (3\ -4\ 1\ 2\ 3)^T$, поданий на входи мережі SOM, яка має п'ять нейронів. Ваги нейронів дорівнюють: $w^{(1,1)}=(0\ 3\ -2\ 1\ 5)^T$, $w^{(1,2)}=(-1\ 8\ 5\ 3\ 3)^T$ та $w^{(1,3)}=(1\ 1\ -1\ 1\ 2)^T$, $w^{(1,4)}=(0\ 7\ -1\ 2\ 5)^T$ та $w^{(1,5)}=(-2\ 9\ 4\ 1\ 2)^T$, а пороги вважаються відсутніми. Визначити значення на виходах нейронів SOM.

Завдання 3. Нехай задано розпізнаваний екземпляр $x = (1\ 3\ 2\ 3)^T$, поданий на входи мережі LVQ, яка має чотири нейрони на першому шарі та два нейрони на другому шарі. Ваги нейронів першого шару дорівнюють: $w^{(1,1)}=(1\ 2\ 1\ 1)^T$, $w^{(1,2)}=(2\ 3\ 2\ 3)^T$, $w^{(1,3)}=(3\ 3\ 3\ 3)^T$ та $w^{(1,4)}=(1\ -1\ 2\ 3)^T$, а пороги вважаються відсутніми. Ваги нейронів другого шару дорівнюють: $w^{(2,1)}=(1\ 0\ 0\ 1)^T$ та $w^{(1,2)}=(0\ 1\ 1$

$0)^T$, а пороги – нулю. Визначити номер класу, до якого має віднести мережа розпізнаваний екземпляр *x*.

 *Завдання 4*. Емуляція та навчання мереж Кохонена SOM.

1. Задати число класів, кількість входів нейронів і розмір мережі (число нейронів), набір пар значень входів і бажаних виходів, а також вид метрики SOM.

2. Написати на алгоритмічній мові програмування високого рівня програму, що моделює SOM і реалізує алгоритми її навчання. Передбачити в програмі відображення на екран і запис у файл на диску поточного стану параметрів НМ і результатів їхньої роботи: матриці ваг, значення на входах і виходах, тривалість їхнього навчання і роботи.

3. Для заданих даних здійснити навчання SOM і зберегти у файлі на диску результати її роботи.

4. Виконати п. 3, варіюючи по черзі вид / параметри метрики SOM. Результати занести в таблицю, стовпці якої повинні мати назви: вид метрики, помилка поділу на класи (число помилкових рішень про віднесення екземплярів до вузлів, до яких вони не мають фактичної близькості). Проаналізувати отримані результати і зробити висновки про те, як впливає вид метрики на число помилкових рішень.

5. Проаналізувати результати виконання п.3–п.4.

 *Завдання 5*. Емуляція та навчання мереж Кохонена LVQ.

1. Задати число класів, кількість входів нейронів і розмір мережі (число нейронів), набір пар значень входів і бажаних виходів, а також вид метрики.

2. Написати на алгоритмічній мові програмування високого рівня програму, що моделює LVQ і реалізує методи її навчання. Передбачити в програмі відображення на екран і запис у файл на диску поточного стану параметрів НМ і результатів їхньої роботи: матриці ваг, значення на входах і виходах, тривалість їхнього навчання і роботи.

3. Для заданих даних здійснити навчання LVQ і зберегти у файлі на диску результати її роботи.

4. Виконати п. 3, варіюючи по черзі вид / параметри метрики. Результати занести в таблицю, стовпці якої повинні мати назви: вид

метрики, помилка поділу на класи (число помилкових рішень про віднесення екземплярів до вузлів, до яких вони не мають фактичної близькості). Проаналізувати отримані результати і зробити висновки про те, як впливає вид метрики на число помилкових рішень.

5. Проаналізувати результати виконання п. 3–4.

РОЗДІЛ 5
ГЛИБИННІ НЕЙРОННІ МЕРЕЖІ

5.1 Глибинне навчання та глибинні нейромережі

Глибинне навчання (deep learning) – це розділ машинного навчання, спрямований на побудову ієрархічних моделей шляхом використання високорівневих масових абстракцій даних на основі глибинного графу з багатьма обробними шарами, які здійснюють лінійні або нелінійні перетворення, тобто це – прогресуючий високорівневий витяг ознак з сирих (необроблених) первісних вхідних даних.

У центрі глибинного навчання є глибинні нейронні мережі та методи їхньої побудови.

Глибинна нейронна мережа (ГНМ, deep neural network, DNN) – це різновид ШНМ, що має багато шарів обробки даних, яка перетворює вхідні дані у вихідні, ієрархічно виділяючи та агрегуючи ознаки, підвищуючи рівень абстракції даних в напрямку від входів до виходів.

Порівняно з мілкими НМ, ГНМ за рахунок збільшення кількості нейроелементів та зв'язків отримують більшу обчислювальну потужність і здатність моделювати більш складні залежності, а за рахунок спеціалізації шарів та високої ієрархічності обробки даних стають більш зручними для сприйняття та аналізу людиною. При цьому спеціалізація шарів обробки даних у ГНМ робить їх більш пристосованими до інтеграцію в мережеву модель апріорної інформації про предметну область. Проте, як правило, конкретні парадигми ГНМ мають більш обмежене застосування у конкретних задачах (наприклад, деякі архітектури можуть застосовуватися лише для розпізнавання зображень і не придатні для інших задач).

5.2 Згорткові нейромережі

Згорткові нейронні мережі (ЗНМ, convolutional neural network, CNN, ConvNet) – це глибинні ШНМ прямого поширення. Фактично вони є різновидом БНМ, адаптованої до обробки зображень.

Згорткові мережі (рис. 5.1) інспіровані організацією зорової кори тварин, де окремі нейрони кори реагують на стимули лише в обмеженій області зорового поля (*рецептивному полі*). Рецептивні поля різних нейронів частково перекриваються таким чином, що вони покривають усе зорове поле.

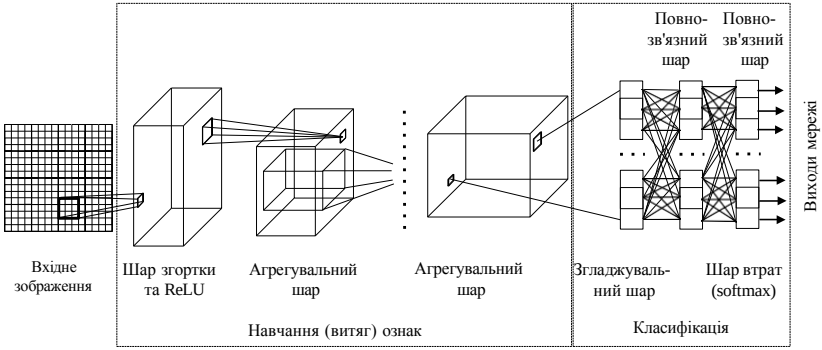


Рисунок 5.1 – Схема ЗНМ

ЗНМ складається з шарів входу та виходу, а також із декількох прихованих шарів. На відміну загальної парадигми БНМ, *приховані шари* ЗНМ зазвичай є спеціалізованими і складаються зі згорткових шарів, агрегувальних шарів, повноз'єднаних шарів та шарів нормалізації.

Згортковий шар (convolutional layer) застосовує до входу операцію згортки, передаючи результат до наступного шару:

$$\mathbf{z}^l = h^{l-1} * W^l,$$

де $\mathbf{z}^l$ – постсинаптичний (передактиваційний) рівень збудження нейронів *l*-го шару НМ, h^{l-1} – активаційний рівень нейронів (*l*–1)-го шару НМ, W^l – налагоджуваний параметр *l*-го шару НМ, * – дискретний оператор згортки.

Згортка імітує реакцію окремого нейрона на зоровий стимул. Кожен згортковий нейрон обробляє дані лише для свого рецептивного поля.

Параметри шару складаються з набору фільтрів для навчання (або ядер), які мають невеличке рецептивне поле, але простягаються на всю глибину вхідної ємності. Протягом прямого проходу кожен фільтр здійснює згортку за шириною та висотою вхідної ємності, обчислюючи скалярний добуток даних фільтру та входу, і формуючи двовимірну карту збудження цього фільтру. В результаті мережа навчається, які фільтри активуються, коли вона виявляє певний конкретний тип ознаки у певному просторовому положенні у вході. Складання карт збудження всіх фільтрів уздовж виміру глибини формує повну ємність виходу

згорткового шару. Таким чином, кожен запис в ємності виходу може також трактуватися як вихід нейрона, що реагує на невеличку область у вході, та має спільні параметри з нейронами тієї ж карти збудження.

Агрегувальний шар (pooling layer) ЗНМ призначений для локального або глобального агрегування та об'єднує виходи кластерів нейронів одного шару до одного нейрону наступного шару. Агрегування є різновидом нелінійного зниження дискретизації, ідея якого полягає у тому, що точне положення ознаки не так важливе, як її грубе положення відносно інших ознак. Агрегувальний шар слугує поступовому скороченню просторового розміру подання для зменшення кількості параметрів та об'єму обчислень у мережі, і відтак також для контролю перенавчання. В архітектурі ЗНМ є звичним періодично вставляти агрегувальний шар між послідовними згортковими шарами. Операція агрегування забезпечує ще один різновид інваріантності відносно паралельного перенесення. Агрегувальний шар діє незалежно на кожен зріз глибини входу, і зменшує його просторовий розмір.

Максимізаційне агрегування (max pooling) використовує максимальне значення з кожного з кластерів нейронів попереднього шару. Воно розділяє вхідне зображення на набір прямокутників без перекриттів, і для кожної такої підобласті виводить її максимум:

$$h^{l}_{x,y} = \max_{i.\,j}\{h^{l-1}_{(x+i)(y+j)}\}, i = 0,...,s,\ j = 0,...,s,$$

де $h^{l}_{x,y}$ – рівень активації нейрона з індексами (x, y) l-го шару НМ.

Агрегування областей інтересу (Region of Interest pooling, RoI pooling) – це різновид максимізаційного агрегування, в якому розмір виходу фіксовано, а прямокутник входу є параметром:

$$h^{l}_{x,y} = \frac{1}{s^2}\sum_{i=0}^{s}\sum_{j=0}^{s} h^{l-1}_{(x+i)(y+j)}.$$

Усереднювальне агрегування (average pooling) використовує усереднене значення з кожного з кластерів нейронів попереднього шару.

$$h^{l}_{x,y} = \frac{1}{s^2}\sum_{i=0}^{s}\sum_{j=0}^{s} h^{l-1}_{(x+i)(y+j)}.$$

L^2-нормове агрегування (L^2-norm pooling) базується на Евклідовій відстані:

$$h_{x,y}^{l} = \sqrt{\sum_{i=0}^{s}\sum_{j=0}^{s}\left(h_{(x+i)(y+j)}^{l-1}\right)^2}.$$

Шар зрізаних лінійних вузлів (Rectified Linear Units layer, ReLU) застосовує ненасичувальну передавальну функцію *ReLU*. Він посилює нелінійні властивості функції ухвалення рішення і мережі в цілому, не зачіпаючи рецептивних полів згорткового шару. Для посилення нелінійності застосовуються й інші функції, наприклад, насичувальні гіперболічний тангенс та сигмоїдна функція.

Згладжувальний шар (flatten layer) згортає просторові розмірності входу в розмірність каналу.

Повноз'єднаний шар (fully connected layer) з'єднує кожен нейрон одного шару з кожним нейроном наступного шару і забезпечує високорівневі міркування. Такий шар є різновидом шару традиційної БНМ:

$$\mathbf{z}^{l} = W^{l} h^{l-1}.$$

Шар втрат (loss layer) визначає, як навчання штрафує відхилення між передбаченими та справжніми мітками класів, і є, як правило, завершальним шаром. Для різних завдань у ньому можуть використовувати різні функції втрат. Нормовані експоненційні втрати (*softmax*) застосовуються для передбачення єдиного класу з *K* взаємно виключних класів. Сигмоїдні перехресно-ентропійні втрати застосовуються для передбачення *K* незалежних значень імовірності в проміжку [0, 1]. Евклідові втрати застосовуються для регресії до дійснозначних міток $(-\infty,+\infty)$.

Основними відмінами ЗНМ від БНМ є наявність у ЗНМ:

– *просторової організації (тривимірної ємності нейронів)*: згорткові шари ЗНМ мають нейрони, впорядковані у трьох вимірах: ширина, висота та глибина; нейрони всередині шару з'єднані лише з невеликою областю попереднього шару, що називається рецептивним полем; для формування архітектури ЗНМ складають різні типи шарів, як локально-, так і повноз'єднані;

– *локальної з'єднаності* між нейронами сусідніх шарів для забезпечення просторової локальності: навчені «фільтри» виробляють найсильніший відгук до просторово локального вхідного образу, а складання багатьох таких шарів веде до нелінійних фільтрів, що стають все глобальнішими (тобто, чутливими до більшої області піксельного простору), так що мережа спочатку створює подання дрібних деталей входу, а потім з них збирає подання більших областей;

– *спільних параметрів (ваг)*: для кожного рецептивного поля шару використовується один і той же фільтр (банк ваг), що це зменшує обсяг необхідної пам'яті та поліпшує продуктивність ЗНМ, бо кожен фільтр повторюється на всьому зоровому полі, а повторні вузли використовують спільну параметризацію (вектор ваг та упередженості) й формують карту ознак, що означає, що всі нейрони в заданому згортковому шарі реагують на одну й ту ж саму ознаку в межах свого рецептивного поля, повторювання вузлів таким чином дозволяє ознакам бути виявленими незалежно від їхнього положення в зоровому полі, забезпечуючи таким чином властивість інваріантності відносно зсуву;

– *шарів об'єднання,* у яких карти ознак поділяються на прямокутні підрегіони, а ознаки кожного прямокутника незалежно зменшуються до одного значення, зазвичай, беручи середнє або максимальне значення.

Разом ці властивості дозволяють ЗНМ досягати кращого узагальнення на задачах бачення. Спільне використання ваг різко зменшує кількість вільних параметрів, яких вчиться мережа, знижуючи таким чином вимоги до пам'яті для роботи мережі та уможливлюючи тренування більших, потужніших мереж.

Навчання ЗНМ здійснюють шляхом використання техніки зворотного поширення помилки.

5.3 Мережі довгої короткочасної пам'яті

Мережа довгої короткочасної пам'яті (ДКЧП, long short-term memory, LSTM) – це архітектура, що відноситься до парадигми рекурентних нейронних мереж (РНМ), але на відміну від традиційних РНМ, мережа ДКЧП добре підходить вирішення задач класифікації, обробки або передбачення часових рядів, коли між важливими подіями існують часові затримки невідомої тривалості. Відносна нечутливість до довжини прогалин дає

ДКЧП перевагу у численних застосуваннях над альтернативними РНМ, прихованими марковськими моделями та іншими методами навчання послідовностей.

Мережа ДКЧП складається з шарів, серед яких містяться вузли ДКЧП.

Вузол ДКЧП – це вузол РНМ, який виділяється запам'ятовуванням значень для довгих, або коротких проміжків часу. Це забезпечується завдяки тому, що вузол не використовує функції активації в межах своїх рекурентних складових. Таким чином, значення, що зберігається, не розплющується ітеративно з плином часу, і член градієнту або штраф не має схильності розмиватися, коли для його навчання застосовується зворотне поширення у часі.

Вузли ДКЧП часто втілюють у блоках, які містять декілька вузлів ДКЧП. Така конструкція є типовою для «глибинних» БНМ, і сприяє реалізаціям на паралельному апаратному забезпеченні.

Блоки ДКЧП містять три або чотири вентилі, які вони використовують для керування плином інформації до або з їхньої пам'яті. Ці вентилі реалізують із застосуванням логістичної функції для обчислення значень між 0 та 1. Для часткового дозволу або заборони плину інформації до або з цієї пам'яті застосовується множення на це значення.

Наприклад, вхідний вентиль керує мірою, до якої нове значення входить до пам'яті. Забувальний вентиль керує мірою, до якої значення залишається в пам'яті. А вихідний вентиль керує мірою, до якої значення в пам'яті використовується для обчислення активування виходу блоку. У деяких втіленнях вхідний та забувальний вентилі об'єднують в один. Ідея їхнього об'єднання полягає в тому, що час забувати настає тоді, коли з'являється нове значення, варте запам'ятовування.

Схема простого блоку ДКЧП зображена на рис. 5.2. Тут для блоку позначено: i – вхідні, o – вихідні та f – забувальні ворота. Кожне з цих воріт може розглядатися як нейрон БНМ: тобто вони обчислюють функцію активації зваженої суми, а i_t, o_t та f_t являють собою активації відповідно вхідних, вихідних та забувальних воріт у момент часу t. Три стрілки, що виходять з комірки пам'яті

c до трьох воріт i, o та f являють собою з'єднання комірок. Ці з'єднання фактично позначають внесок активації комірки пам'яті c у момент часу t–1, тобто внесок c_{t-1} (а не c_t, як може підказати рисунок). Іншими словами, ворота i, o та f обчислюють їх активацію на етапі часу t (тобто, відповідно, i_t, o_t та f_t, також враховуючи активацію комірки пам'яті c на етапі часу t–1, тобто c_{t-1}. Єдина стрілка зліва-направо, що виходить з комірки пам'яті, не є з'єднанням комірок і позначає c_t. Маленькі кола, що містять символ ×, являють собою поелементне множення між їхніми входами. Великі кола, що містять S-подібні криві, позначають застосування диференційованої функції (наприклад, сигмоїдної функції) до зваженої суми.

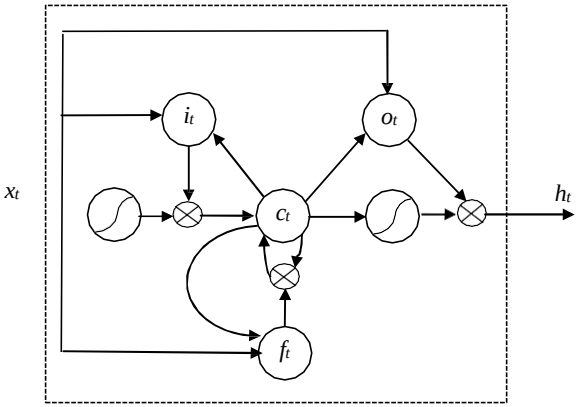


Рисунок 5.2 – Схема простого блоку ДКЧП

Нехай $\circ$ – позначає добуток Адамара (поелементний добуток елементів матриць), t – час, x_t – вхідний вектор, h_t – вихідний вектор, c_t – вектор стану комірки, $\tilde{c}_t$ – вектор активації входу комірки, параметри: W_q – матриця ваг зв'язків від входів до вентилів q), U_q – матриця уточнень (ваги зв'язків від вихідних вузлів до вентилів q), V_q – матриця ваг зв'язків від комірок пам'яті до вентилів q, b_q – вектор упередження (порогів), де нижній індекс q позначує: i – вхідний вентиль, o – вихідний вентиль, f – забувальний вентиль, c – комірка пам'яті; вектори вентилів: f_t – вектор забувального вентиля (вага пам'ятання старої

інформації), i_t – вектор вхідного вентиля (вага отримання нової інформації), o_t – вектор вихідного вентиля (кандидатність на вихід); функції активації: a_g – сигмоїдна функція, a_c – гіперболічний тангенс, a_h – гіперболічний тангенс (але практично рекомендується використовувати лінійну функцію: $a_h(x)=x$). У наведених нижче рівняннях кожна змінна курсивом у нижньому регістрі подає вектор, що має розмір, який дорівнює числу вузлів ДКЧП у блоці. Початкові значення: $c_0 =0$, $h_0 =0$.

Традиційна ДКЧП із забувальними вузлами описується як:

$$
\begin{aligned}
f_t &= a_g(W_f x_t + U_f h_{t-1} + b_f),\\
i_t &= a_g(W_i x_t + U_i h_{t-1} + b_i),\\
o_t &= a_g(W_o x_t + U_o h_{t-1} + b_o),\\
\tilde{c}_t &= a_c(W_c x_t + U_c h_{t-1} + b_c),\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t,\\
h_t &= o_t \circ a_h(c_t).
\end{aligned}
$$

Вічкова ДКЧП із забувальними вентилями описується як:

$$
\begin{aligned}
f_t &= a_g(W_f x_t + U_f c_{t-1} + b_f),\\
i_t &= a_g(W_i x_t + U_i c_{t-1} + b_i),\\
o_t &= a_g(W_o x_t + U_o c_{t-1} + b_o),\\
c_t &= f_t \circ c_{t-1} + i_t \circ a_c(W_c x_t + b_c),\\
h_t &= o_t \circ a_h(c_t).
\end{aligned}
$$

Згорткова ДКЧП описується як:

$$
\begin{aligned}
f_t &= a_g(W_f * x_t + U_f * h_{t-1} + V_f \circ c_{t-1} + b_f),\\
i_t &= a_g(W_i * x_t + U_i * h_{t-1} + V_i \circ c_{t-1} + b_i),\\
o_t &= a_g(W_o * x_t + U_o * h_{t-1} + V_o \circ c_{t-1} + b_o),\\
c_t &= f_t \circ c_{t-1} + i_t \circ a_c(W_c * x_t + U_c * h_{t-1} + b_c),\\
h_t &= o_t \circ a_h(c_t),
\end{aligned}
$$

де * – оператор згортки.

Єдині ваги, що є в блоці ДКЧП W та U, використовуються для спрямовування дії вентилів. Ці ваги застосовуються між значеннями, які надходять до блоку (включно з входовим вектором x_t та виходом з попереднього моменту часу h_{t-1} та кожним із вентилів. Отже, блок ДКЧП визначає, яким чином підтримувати свою пам’ять як функцію від цих значень, і

тренування ваг блока ДКЧП спричиняє його навчання такої функції, яка мінімізує втрати.

Тренування ДКЧП-мереж здійснюють на основі техніки зворотного поширення помилки у часі. Також використовують еволюційні методи.

5.4 Гібридні глибинні мережі

Утворити ГНМ можливо шляхом певного поєднання більш простих нейромереж. Розглянемо кілька найбільш популярних класів гібридних ГНМ.

Генеративні змагальні мережі (ГЗМ, generative adversarial networks, GANs) – це клас ГНМ, що використовуються в навчанні без учителя, реалізовані системою двох штучних нейронних мереж, які змагаються одна з одною в рамках гри з нульовою сумою.

Схему ГЗМ зображено на рис. 5.3.

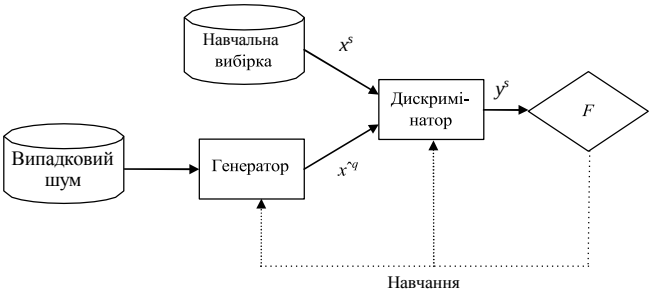


Рисунок 5.3 – Схема ГЗМ

Одна мережа – *генератор* генерує екземпляри-кандидати, а інша мережа – *дискримінатор* оцінює їх. Як правило, генератор навчається будувати відповідності з латентного (прихованого) простору до певного розподілу даних, тоді як дискримінаційна мережа розрізняє представників x^s справжнього розподілу даних (навчальної вибірки) та кандидатів $\hat{x}^q$, вироблених генератором. Метою генератора як тренувальної мережі є збільшення частоти помилок дискримінаційної мережі (тобто «обдурити» дискримінатор шляхом створення нових синтезованих екземплярів, які повинні походити на представників справжнього розподілу даних).

На практиці заздалегідь відомий набір даних використовують як початкові навчальні данні для дискримінатора. Навчання дискримінатора передбачає забезпечення його зразками з набору даних, доки він не досягне певного рівня точності F. Зазвичай генератор на початку отримує випадково відібрані дані із заздалегідь визначеного латентного простору. Після цього екземпляри, синтезовані генератором, оцінюються дискримінатором.

Генератор, як правило, є деконволюційною НМ, а дискримінатор – ЗНМ.

Для навчання обох мереж застосовують техніку зворотного поширення помилки.

Сіамська нейронна мережа (siamese neural network, twin neural network) – це різновид ГНМ, яка використовує однакові ваги при одночасній роботі з двома різними вхідними векторами для обчислення вихідних векторів, що підлягають порівнянню.

Часто один з вихідних векторів попередньо обчислюється, утворюючи таким чином базу, з якою порівнюють інший вихідний вектор. Це схоже на локально-чуттєве хешування (тобто порівняння замість багатомірних оригінальних екземплярів їхніх ґешів – свого роду "відбитків пальців" або описів-портретів невеликої розмірності).

Сіамська мережа складається з двох однакових НМ, кожна з яких приймає один з двох векторів вхідних даних. Виходи останніх шарів цих двох мереж подаються на контрастуючу функцію втрат, яка обчислює схожість між двома векторами вхідних даних. Узагальнену схему сіамської НМ зображено на рис. 5.4. Тут x^p, x^q – два різні екземпляри, подані на входи першої (net_1) та другої (net_2) НМ, відповідно. Мережі net_1 та net_2 на виходах видають значення y_1 та y_2, які подаються на відповідні входи блоку функції втрат L. Цей блок обчислює вихід мережі y.

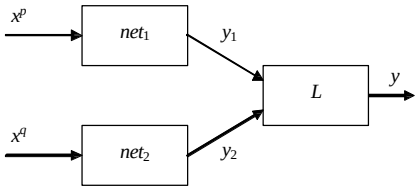


Рисунок 5.4 – Схема сіамської НМ

Навчання в сіамських мережах може бути здійснено з триплетними (трійковими) втратами або контрастними втратами. Для навчання за допомогою триплетних втрат базовий вектор порівнюється з позитивним вектором та негативним вектором. Негативний вектор змусить мережу навчатися, тоді як позитивний вектор буде діяти як регуляризатор. Для навчання за допомогою контрастних втрат необхідно запровадити зменшення ваги для регулювання ваг мережі або подібну операцію, наприклад, нормалізацію.

? 5.5 Контрольні питання

1. Що таке глибинне навчання?
2. Що таке глибинна нейронна мережа?
3. Порівняйте мілкі НМ та ГНМ.
4. Що таке згорткова нейронна мережа?
5. Наведіть схему згорткової нейромережі.
6. Опишіть структуру та функціонування згорткової нейромережі.
7. Для чого призначений та як функціонує згортковий шар?
8. Що таке згортка?
9. Для чого призначений та як функціонує агрегувальний шар?
10. Що таке максимізаційне агрегування (max pooling)?
11. Що таке агрегування областей інтересу (RoI pooling)?
12. Що таке усереднювальне агрегування (average pooling)?
13. Що таке L^2-нормове агрегування (L^2-norm pooling)?
14. Для чого призначений та як функціонує шар зрізаних лінійних вузлів (ReLU)?
15. Для чого призначений та як функціонує згладжувальний шар (flatten layer)?
16. Для чого призначений та як функціонує повноз'єднаний шар (fully connected layer)?
17. Для чого призначений та як функціонує шар втрат (loss layer)
18. Опишіть основні відмінності ЗНМ від БНМ.
19. Що таке шар об'єднання?
20. Що таке мережа довгої короткочасної пам'яті (LSTM)?

21. Що таке вузол ДКЧП?
22. Зобразіть та опишіть схему простого блоку ДКЧП?
23. Наведіть математичний опис функціонування ДКЧП із забувальними вузлами.
24. Наведіть математичний опис функціонування вічкової ДКЧП із забувальними вентилями.
25. Наведіть математичний опис функціонування згорткової ДКЧП.
26. Які Ви знаєте гібридні глибинні нейромережі?
27. Як утворюють гібридні глибинні нейромережі?
28. Що таке генеративна змагальна мережа (GAN)?
29. Зобразіть схему генеративної змагальної нейромережі.
30. Що таке мережа-генератор?
31. Що таке мережа-дискримінатор?
32. Що таке сіамська нейронна мережа?
33. Зобразіть схему сіамської нейромережі.
34. Як навчають глибинні нейромережі?

5.6 Практичні завдання

Завдання 1. Підготувати реферат на одну з таких тем.

1. Архітектури згорткових нейромереж.
2. Методи навчання згорткових нейромереж.
3. Методи навчання мереж LSTM.
4. Архітектури гібридних глибинних нейромереж.
5. Методи навчання змагальних глибинних нейромереж.
6. Застосування глибинних нейромереж.

Завдання 2. Емуляція та навчання глибинних мереж.

1. Обрати парадигму глибинної мережі, програмний засіб, що її реалізує та практичну задачу (навчальну вибірку). За допомогою обраного засобу побудувати нейромодель для відповідної задачі. Перевірити працездатність моделі на даних, що не використовувалися у навчанні. Зробити висновки щодо швидкості побудови, роботи, точності відповідної парадигми глибинних мереж.

2. Повторити п. 1 для іншої парадигми або програмного засобу. Порівняти отримані результати.

РОЗДІЛ 6

ПРОГРАМНІ ЗАСОБИ ДЛЯ МОДЕЛЮВАННЯ НЕЙРОМЕРЕЖ

6.1 Моделювання нейронних мереж у пакеті Matlab

Matlab являє собою математичний пакет, призначений для вирішення задач обчислювальної математики та побудови чисельних моделей складних об'єктів і процесів.

Пакет Matlab складається з інтерпретатора – модельного середовища, що має термінальний інтерфейс, ядра (набору найпростіших стандартних операцій, функцій і процедур для обчислень), а також бібліотек функцій (Toolbox).

Перевагами пакета є: багаті графічні можливості, великий набір математичних функцій, простота вбудованої мови Matlab, можливість автоматичного перетворення текстів програм мовою Matlab у тексти програм на мові Сі, а також те, що тексти бібліотечних функцій постачаються у вихідному виді.

До недоліків пакета варто віднести низьку швидкість роботи.

Для моделювання НМ за допомогою пакета Matlab необхідно встановити і використовувати бібліотеку *Deep Learning Toolbox* (раніше –*Neural Network Toolbox*). Бібліотека містить як функції для моделювання мілких (shallow networks), так і глибоких (deep networks) нейромереж.

Пакет Matlab, починаючи з версії 6.0, містить візуальний інтерфейсний модуль *nntool*, який входить до бібліотеки Neural Network Toolbox. Використання nntool дозволяє більш зручними засобами, ніж написання програми вручну, будувати нейромережеві моделі. Розглянемо деякі основні можливості та прийоми роботи із засобом nntool.

Після запуску Matlab. exe в командному вікні для початку роботи із nntool треба ввести: `nntool`. Після цього завантажиться засіб nntool (рис. 6.1).

На панелі Network and Data («Нейромережі та дані») користувач має натиснути кнопки для завдання вихідних даних для побудови нейромережевої моделі.

Кнопка New Data («Нові дані») викликає редактор для створення нових даних (рис. 6.2).

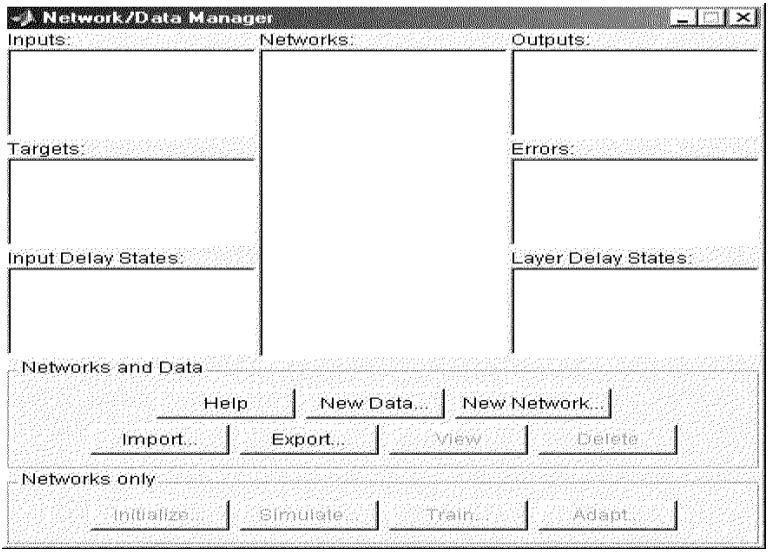


Рисунок 6.1 – Головна діалогова форма засобу nntool

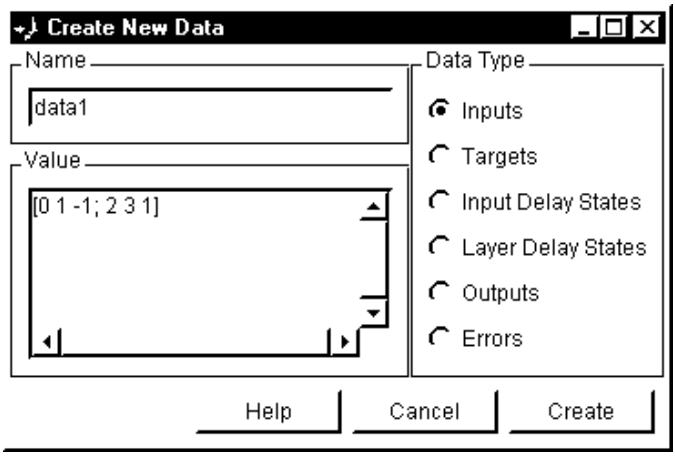


Рисунок 6.2 – Редактор даних засобу nntool

Поле Name («Ім’я») задає ім’я нової змінної середовища Matlab, до якої зберігається масив нових даних, що вводяться у полі Value («Значення»).

Панель Data Type («Тип даних») визначає призначення введених даних: Inputs – входи мережі, Targets – цільові значення виходів мережі, Input Delay States та Layer Delay States – описи затримок на входах та у шарах мережі, Outputs – фактичні значення на виходах мережі, Errors – помилки мережі.

Якщо дані вже існують у вигляді зовнішніх файлів або містяться у середовищі Matlab у вигляді змінних, вони можуть бути імпортовані за допомогою кнопки Import («Імпорт»). При цьому з'являється діалогова форма (рис. 6.3).

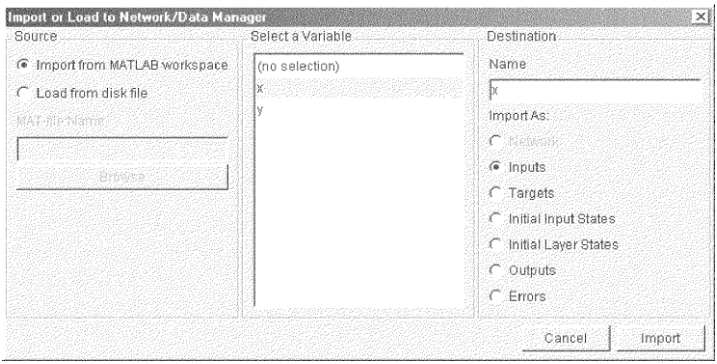


Рисунок 6.3 – Діалогова форма імпорту даних

Поле Source («Джерело») дозволяє обрати джерело введення даних: Import from Matlab workspace (імпорт даних із середовища Matlab) або Load from disk file (завантаження даних із файлу на диску). Кнопка Browse дозволяє обрати необхідний файл.

Поле Select a Variable («Вибір змінної») дозволяє вказати засобу nntool, яку змінну треба використовувати для імпорту даних.

Панель Destination («Приймач») дозволяє задати змінну для прийому даних, що імпортуються. Її ім'я вказується у полі Name («Ім'я»), а призначення (Import as) обирається із наведеного меню.

Кнопка Export («Експорт») головної діалогової форми дозволяє зберегти дані із середовища nntool у файлі на диску, або передати їх до середовища Matlab.

Кнопка «New Network» («Нова мережа») викликає діалогову форму для конструювання нейромережі та визначення її параметрів (рис. 6.4).

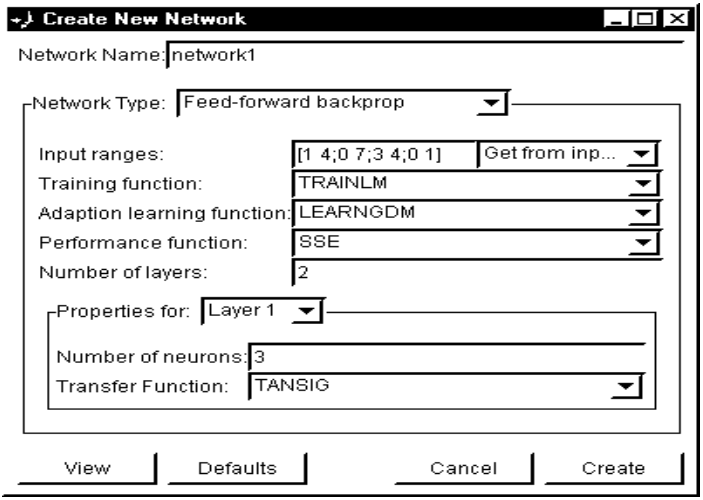


Рисунок 6.4 – Форма конструювання нейромережі

Поле Network Name («Ім'я мережі») визначає ім'я змінної, де зберігається мережа. Список вибору Network Type («Тип мережі») дозволяє обрати тип архітектури мережі (наприклад, Feed-forward backprop – багатошарова нейромережа прямого поширення), поля Training Function, Adaptation Learning Function, Performance Function та Number of Layers визначають, відповідно, тип методу навчання мережі, тип методу адаптації ваг мережі, цільову функцію та кількість шарів мережі.

Панель Properties for Layer K дозволяє задати властивості для нейронів K-го шару мережі. У полі Number of Neurons вказують кількість нейронів для поточного шару, а у полі Transfer Function – тип функції активації нейронів поточного шару мережі.

Кнопка «View» дозволяє отримати графічне зображення схеми побудованої нейромережі (рис. 6.5).

Кнопка «Delete» головної форми дозволяє видалити непотрібний елемент даних (змінну або мережу).

Кнопка «Help» дозволяє викликати довідкову службу Matlab з описом необхідних компонентів та поясненнями щодо їхнього використання.

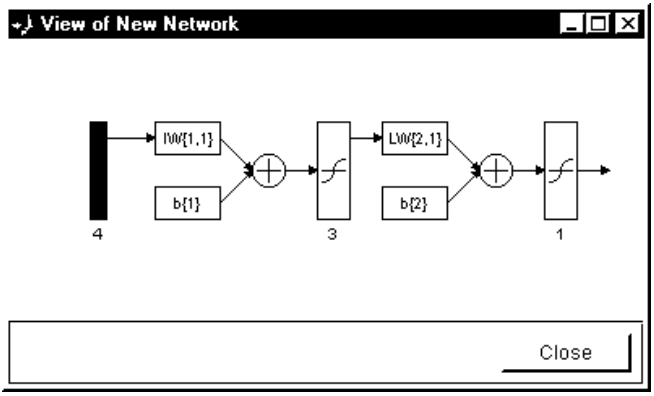


Рисунок 6.5 – Приклад зображення форми нейромережі, побудованої за допомогою засобу nntool

Після побудови нейромережі у нижній частині головної діалогової форми nntool стає доступною панель Networks only, що призначена для роботи із побудованою мережею (рис. 6.6).

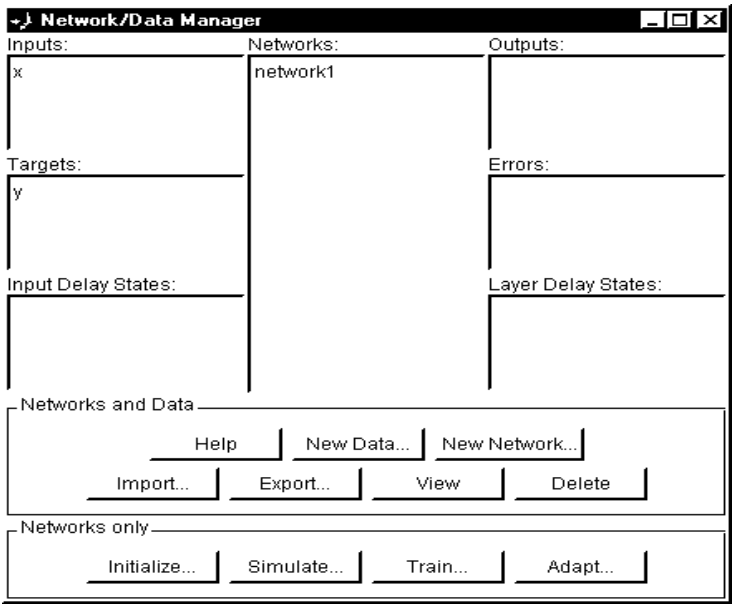


Рисунок 6.6 – Головна діалогова форма nntool після побудови мережі

При натисненні будь-якої з кнопок цієї панелі викликається діалогова форма Network («Мережа»), яка містить набір закладок-панелей для роботи з мережею (рис. 6.7–6.10).

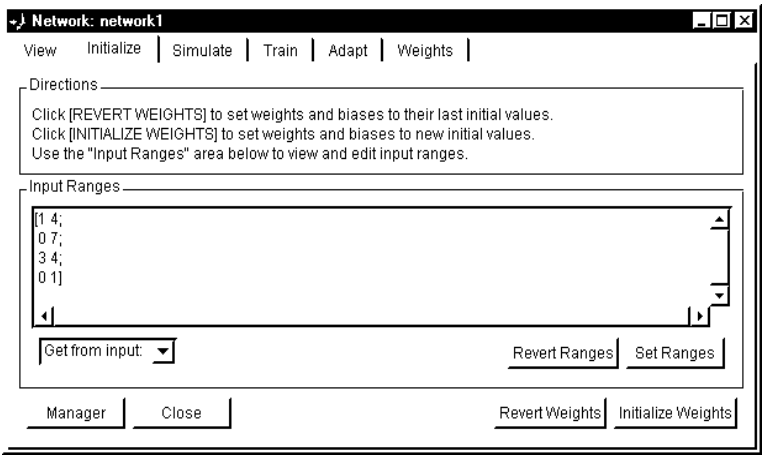


Рисунок 6.7 – Форма Network: закладка Initialize

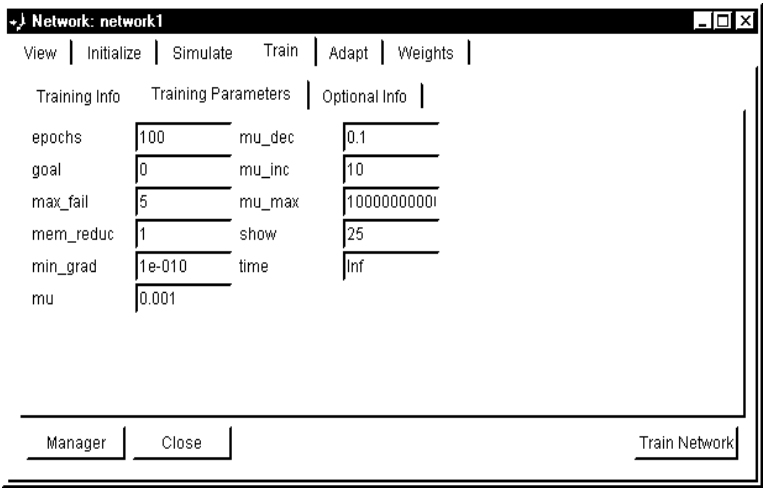


Рисунок 6.8 – Форма Network: закладка Train

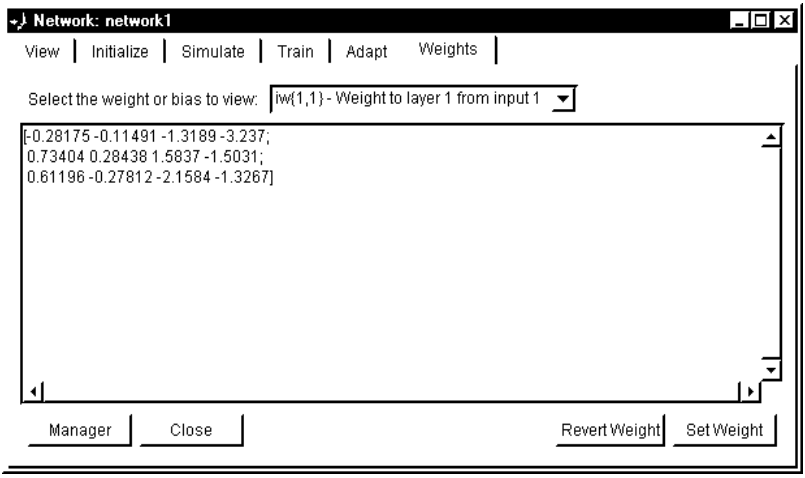


Рисунок 6.9 – Форма Network: закладка Weights

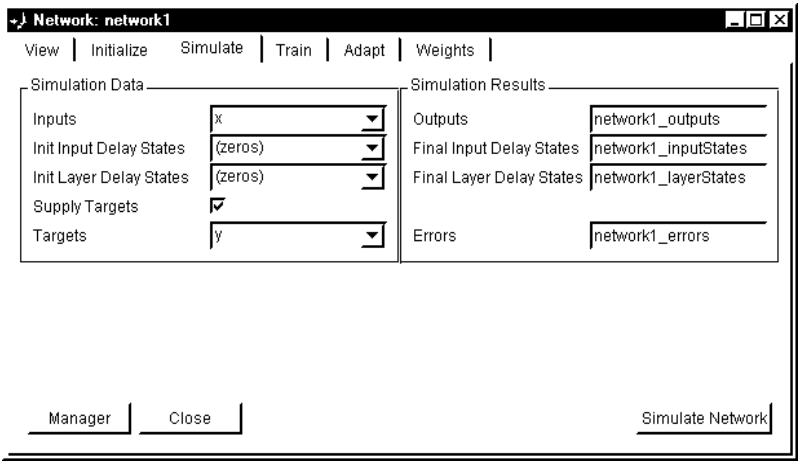


Рисунок 6.10 – Форма Network: закладка Simulate

Закладка Initialize («Ініціалізація») дозволяє задати межі, в яких змінюються вхідні дані та розрахувати на їхній основі початкові значення ваг мережі.

Ініціалізована мережа може бути навчена за допомогою закладки Train («Тренування, навчання»). Серед параметрів

навчання, доступних на цій закладці обов'язково необхідно задати: goal – максимально припустиме значення цільової функції, epochs – максимальна припустима кількість циклів навчання мережі, show – крок виведення на екран інформації про навчання мережі, задається у циклах навчання.

У процесі навчання середовище Matlab будує графік зміни значення цільової функції за епохами – циклами навчання (рис. 6.11).

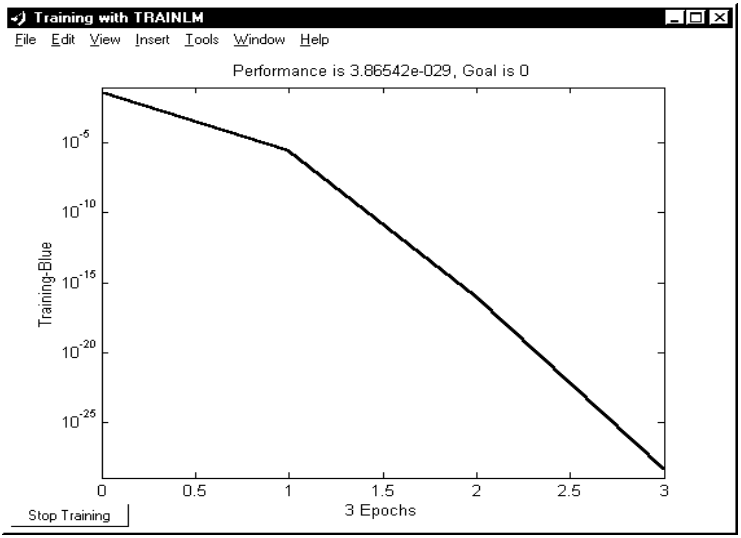


Рисунок 6.11 – Графік зміни значення цільової функції в процесі навчання

Ваги мережі, що була навчена, можна переглянути, використовуючи закладку Weights («Ваги»).

Після того, як мережа навчилася, її можна використовувати для розпізнавання за допомогою закладки Simulate («Моделювання»).

Нейронні мережі, створені за допомогою функцій Matlab, мають єдиний формат подання, як це показано у табл. 6.1.

Поряд із використанням засобу nntool пакет Matlab дозволяє використовувати функції для моделювання нейромереж у програмному та ручному режимах.

Таблиця 6.1 – Подання структури нейромережі у пакеті Matlab

Поле структури	Опис, значення поля
numInputs	кількість входів мережі
numLayers	кількість (схованих) шарів мережі без урахування вхідного шару
biasConnect, inputConnect, layerConnect, outputConnect, targetConnect	булеві масиви, що визначають зв'язки між структурними елементами мережі
numOutputs	кількість виходів мережі
numTargets	кількість цільових ознак
numInputDelays	кількість затримок вхідного шару
numLayerDelays	кількість затримок схованих шарів
inputs	входи мережі
layers	шари мережі
outputs	виходи мережі
targets	цільові ознаки
biases	масив порогів нейронів мережі
inputWeights	масив вагових коефіцієнтів вхідного шару мережі
layerWeights	масив вагових коефіцієнтів схованих шарів мережі
adaptFcn	ім'я функції адаптації нейронів
initFcn	ім'я функції ініціалізації мережі
performFcn	ім'я цільової функції навчання (помилки)
trainFcn	ім'я функції, що реалізує процес навчання
adaptParam	параметри адаптації мережі
initParam	параметри ініціалізації мережі
performParam	параметри цільової функції мережі
trainParam. epochs	максимально допустима кількість ітерацій навчання (епох)
trainParam.. goal	максимально допустиме значення цільової функції навчання
trainParam. max_fail	максимально допустима кількість відмов у процесі навчання мережі
trainParam. mem_reduc	коефіцієнт, що регулює (зменшує) використання пам'яті при навчанні нейромереж
trainParam. min_grad	мінімально допустиме значення градієнту цільової функції
trainParam. mu, trainParam. mu_dec, trainParam. mu_inc, trainParam. mu_max	параметри методу Левенберга-Марквардта
trainParam. show	кількість ітерацій, через яку будуть відображатися зміни стану процесу навчання мережі
trainParam. time	максимально допустимий час навчання мережі у секундах
IW	масив значень ваг вхідного (першого) шару
LW	масив значень ваг схованих шарів
b	масив значень порогів нейронів
userdata	дані користувача

Перелік найважливіших функцій наведено в табл. 6.2. Формат виклику функцій можна дізнатися, вводячи команду: `help ім'я_функції`.

Таблиця 6.2 – Основні функції, використовувані для моделювання та побудови нейромереж у пакеті Matlab

Група функцій	Ім'я функції	Призначення або результат роботи функції
1	2	3
Функції для аналізу	rrsurf	Поверхня помилки нейрона з єдиним входом
	maxlinlr	Максимальна швидкість навчання для лінійного нейрона
Функції відстані	boxdist	Відстань між двома векторами
	dist	Евклідова функція
	linkdist	Зв'язана функция
	mandist	Манхеттенська метрика
Функції навчання	learncon	Навчальна функція порогів
	learngd	Навчальна функція градієнтного спуску
	learngdm	Навчальна функція градієнтного спуску з урахуванням моментів
	learnh	Навчальна функція Хебба
	learnhd	Навчальна функція Хебба з урахуванням загасання
	learnis	Навчальна функція instar
	learnk	Навчальна функція Кохонена
	learnlv1	Навчальна функція LVQ1
	learnlv2	Навчальна функція LVQ2
	learnos	Навчальна функція outstar
	learnp	Навчальна функція ваг і порогів персептрона
	learnpn	Навчальна функція нормалізованих порогів і ваг персептрона
	learnsom	Навчальна функція SOM
	learnwh	Правило навчання Уідроу-Хоффа
Функції графіки	hintonw	Графік Хінтона для матриці ваг
	hintonwb	Графік Хінтона для матриці ваг і векторів порогів
	plotbr	Графік функціонування мережі при регулярному байєсовському тренуванні
	plotep	Зображення положень ваг і порогів на поверхні помилки
	plotes	Зображення поверхні помилок одиничного вхідного нейрона

Продовження таблиці 6.2

1	2	3
	plotpc	Зображення лінії класифікації у векторному просторі персептрона
	plotperf	Графічне подання функціонування мережі
	plotpv	Графічне подання вхідних цільових векторів
	plotsom	Графічне подання SOM
	plotv	Графічне подання векторів у виді ліній, що виходять з початку координат
	plotvec	Графічне подання векторів різними кольорами
Топологічні функції	gridtop	Топологічна функція у виді сіткового шару
	hextop	Топологічна функція у виді гексагонального шару
	randtop	Топологічна функція у виді випадкового шару
Лінійні функції пошуку	srchbac	Одновимірна мінімізація з використанням пошуку з поверненням
	srchbre	Одновимірна локалізація інтервалу з використанням методу Брента
	srchcha	Одновимірна мінімізація з використанням методу Караламбуса
	srchgol	Одновимірна мінімізація з використанням золотого перетину
	srchhyb	Одновимірна мінімізація з використанням гібридного бісекційного пошуку
Функції використання мережі	adapt	Адаптація мережі
	disp	Відображає властивості нейронної мережі
	display	Відображає імена змінних і властивості мережі
	init	Ініціалізація нейронної мережі
	sim	Моделювання нейронної мережі
	train	Тренування нейронної мережі
	nntool	Виклик графічного інтерфейсу користувача
	gensim	Генерація блоку Simulink для моделювання нейронної мережі
Функції створення нової мережі	network	Створення нейронної мережі користувача
	newc	Створення конкурентного шару
	newcf	Створення каскадної спрямованої мережі
	newelm	Створення мережі Елмана
	newff	Створення мережі прямого поширення
	newfftd	Створення прямого поширення з вхідними затримками
	newgrnn	Створення узагальненої регресійної нейронної мережі

Продовження таблиці 6.2

1	2	3
	newhop	Створення мережі Хопфілда
	newlin	Створення лінійного шару
	newlind	Конструювання лінійного шару
	newlvq	Створення мережі LVQ
	newp	Створення персептрона
	newpnn	Конструювання імовірнісної нейронної мережі
	newrb	Конструювання радіально-базисної мережі
	newrbe	Конструювання точної мережі з радіальними базисними функціями
	newsom	Створення мережі SOM
Цільові функції і їхні похідні	dmae	Похідна функції mae
	dmse	Похідна функції mse
	dmsereg	Похідна функції msereg
	dsse	Похідна функції sse
	mae	Середня абсолютна помилка
	mse	Середньоквадратична помилка
	msereg	Зважена сума середньоквадратичної помилки і середнього квадратів значень ваг і порогів
	sse	Сумарна квадратична помилка
Функції попередньої і пост- обробки	postmnmx	Ненормалізовані дані, що були нормалізовані за допомогою prenmmx
	postreg	Лінійний регресійний аналіз виходів мережі стосовно цільових значень навчального масиву
	poststd	Ненормовані дані, що були нормовані за допомогою функції prestd
	premnmx	Нормування даних у діапазоні від 1 до +1
	prepca	Аналіз головних компонентів для вхідних даних
	prestd	Нормування даних до одиничного стандартного відхилення і нульовому середній
	tramnmx	Перетворення даних з попередньо обчисленими мінімумом і максимумом
	trapca	Перетворення даних з використанням PCA-матриці (головних компонент), обчисленої за допомогою функції prepca
	trastd	Перетворення даних з використанням попередньо обчислених значень стандартного відхилення і середнього
Функції навчання (тренування)	trainb	Пакетне тренування з використанням правил навчання для ваг і порогів

Продовження таблиці 6.2

1	2	3
	trainbfg	Тренування мережі з використанням квазіньютонівського методу Бройдена-Флетчера-Гольдфарба-Шенно
	trainbr	Регуляризація Байєса
	trainc	Використання збільшень циклічного порядку
	traincgb	Метод зв'язаних градієнтів Пауелла-Біла
	traincgf	Метод зв'язаних градієнтів Флетчера-Пауелла
	traincgp	Метод зв'язаних градієнтів Полака-Рібьєра
	traingd	Метод градієнтного спуску
	traingda	Метод градієнтного спуску з адаптивним навчанням
	traingdm	Метод градієнтного спуску з урахуванням моментів
	traingdx	Метод градієнтного спуску з урахуванням моментів і адаптивним навчанням
	trainlm	Метод Левенберга-Марквардта
	trainoss	Одноступінчатий метод січних
	trainr	Метод випадкових збільшень
	trainrp	Метод пружного зворотного поширення
	trains	Метод послідовних збільшень
	trainscg	Метод шкальованих зв'язаних градієнтів
Вагові функції і їхні похідні	dnetprod	Обчислення похідної від входів мережі з перемножуванням входів
	dnetsum	Обчислення похідної від входів мережі з підсумовуванням входів
	netprod	Функція добутку входів
	netsum	Функція підсумовування входів
	ddotprod	Похідна скалярного добутку
	dist	Евклідова відстань
	dotprod	Вагова функція у виді скалярного добутку
	mandist	Вагова функція відстань Манхеттена
	negdist	Вагова функція негативна відстань
	normprod	Вагова функція нормований скалярний добуток
Функції активації і їхні похідні	dhardlim	Похідна східчастої функції активації
	dhardlms	Похідна симетричної східчастої функції активації
	dlogsig	Похідна сигмоидної (логістичної) функції активації
	dposlin	Похідна позитивної лінійної функції активації
	dpurelin	Похідна лінійної функції активації
	dradbas	Похідна радіально-базисної функції активації
	dsatlin	Похідна лінійної функції активації з насиченням

Продовження таблиці 6.2

1	2	3
	dsatlins	Похідна симетричної функції активації з насиченням
	dtansig	Похідна функції активації гіперболічний тангенс
	dtribas	Похідна трикутної функції активації
	compet	Конкуруюча функція активації
	hardlim	Східчаста функція активації
	hardlims	Східчаста симетрична функція активації
	logsig	Сигмоїдна (логістична) функція активації
	poslin	Позитивна лінійна функція активації
	purelin	Лінійна функція активації
	radbas	Радіально-базисна функція активації
	satlin	Лінійна функції активації з насиченням
	satlins	Симетрична лінійна функція активації, що насичується
	softmax	Функція активації, що зменшує діапазон вхідних значень
	tansig	Функція активації гіперболічний тангенс
	tribas	Трикутна функція активації
Допоміжні функції	calca	Обчислює виходи мережі й інші сигнали
	calca1	Обчислює сигнали мережі для одного кроку за часом
	calce	Обчислює помилки шарів
	calce1	Обчислює помилки шарів для одного кроку за часом
	calcgx	Обчислює градієнт ваг і порогів як єдиний вектор
	calcjejj	Обчислює Якобіан
	calcjx	Обчислює Якобіан ваг і порогів як одну матрицю
	calcpd	Обчислює затримані входи мережі
	calcperf	Обчислення виходів мережі, сигналів і функціонування
	formx	Формує один вектор з ваг і порогів
	getx	Повертає усі ваги і пороги мережі як один вектор
	setx	Установлює усі ваги і пороги мережі у виді одного вектора
	cell2mat	Поєднує масив елементів матриць в одну матрицю
	combvec	Створює всі комбінації векторів

Продовження таблиці 6.2

1	2	3
	con2seq	Перетворить вектори, що сходяться, у послідовні вектори
	concur	Створює вектори порогів, що сходяться
	ind2vec	Перетворення індексів у вектори
	mat2cell	Розбивка матриці на масив елементів матриць
	minmax	Обчислює мінімальні і максимальні значення рядків матриці
	normc	Нормує стовпці матриці
	normr	Нормує рядки матриці
	pnormc	Псевдо-нормування стовпців матриці
	quant	Дискретизація величини
	seq2con	Перетворення послідовних векторів у вектори, що сходяться
	sumsqr	Сума квадратів елементів матриці
	vec2ind	Перетворення векторів в індекси
Функції ініціалізації ваг і порогів	initcon	Функція ініціалізації порогів
	initzero	Ініціалізація з установкою нульових значень ваг і порогів
	midpoint	Ініціалізація з установкою середніх значень ваг
	randnc	Ініціалізація з установкою нормалізованих значень стовпців вагових матриць
	randnr	Ініціалізація з установкою нормалізованих значень рядків вагових функцій
	rands	Ініціалізація з установкою симетричних випадкових значень ваг і порогів
	revert	Повернення вагам і порогам значень, що відповідають попередньої ініціалізації
	initnw	Функція ініціалізації Нгуена-Уідроу
	initwb	Функція ініціалізації по вагам і порогам
	initlay	Функція пошарової ініціалізації мережі

Приклади використання функцій пакету Matlab наведено у підрозділі 6.4.

Поряд із використанням функцій модуля Neural Network Toolbox для автоматизації певних етапів синтезу НМ у пакеті Matlab можна використовувати функції інших модулів.

Особливе місце серед інших функцій посідають *функції вирішення задач оптимізації на основі еволюційного пошуку*, які містить бібліотека *Genetic Algorithm and Direct Search Toolbox* пакету Matlab.

Для використання генетичних методів, у середовищі Matlab передбачена функція [*x*, Fmin] = ga(@fitnessfun, n, options), яка знаходить мінімум Fmin функції fitnessfun, що має n параметрів, а також вектор x, при якому досягається мінімум цільової функції fitnessfun.

Параметри роботи функції ga задаються у змінній options, що являє собою структуру, у якій можуть бути задані: спосіб подання інформації у хромосомі, використовувані генетичні оператори відбору, схрещування й мутації, критерії зупинення й інші параметри генетичного пошуку. За допомогою структури options також можна вибрати графіки, які будуть відображати основні результати генетичного пошуку в процесі оптимізації цільової функції.

Для зміни значень параметрів функції ga передбачена функція gaoptimset, а для одержання поточних параметрів – функція gaoptimget.

Моделювання глибоких нейромереж у пакеті Matlab можна здійснити шляхом використання засобу *Deep Network Designer*. Він забезпечує дружній інтерфейс користувача для конструювання, візуалізації, навчання, редагування глибоких нейромереж. Також засіб дозволяє зберігати та завантажувати нейромоделі, копіювати, видаляти та редагувати їхні складові, а також аналізувати мережу, імпортувати дані, налаштовувати параметри мереж та методів навчання, слідкувати за точністю.

Для запуску засобу потрібно увести команду: `deepNetworkDesigner`.

Форми засобу DeepNetworkDesigner наведені на рис. 6.12–6.18.

Імпорт даних у Deep Network Designer для навчання здійснюють таким чином. Спочатку на вкладці "Дані" треба натиснути "Імпортувати дані". Можна імпортувати дані з папки з підпапками зображень для кожного класу або з imageDatastore в робочій області. Deep Network Designer надає вибір варіантів збільшення зображень. Можна ефективно збільшити обсяг навчальних даних, застосувавши до даних рандомізоване збільшення. Якщо треба збільшити дані, Deep Network Designer випадковим чином вносить зміни до даних на кожній епосі. Кожна епоха потім використовує дещо інший набір даних.

Імпорт тестових даних здійснюють, обравши папку або імпортуючи imageDatastore з робочої області.

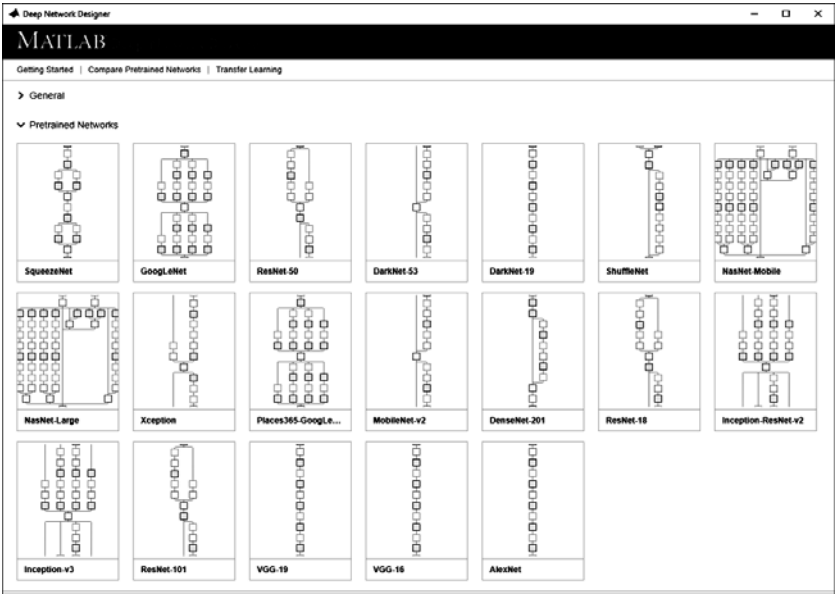


Рисунок 6.12 – Головна форма засобу DeepNetworkDesigner

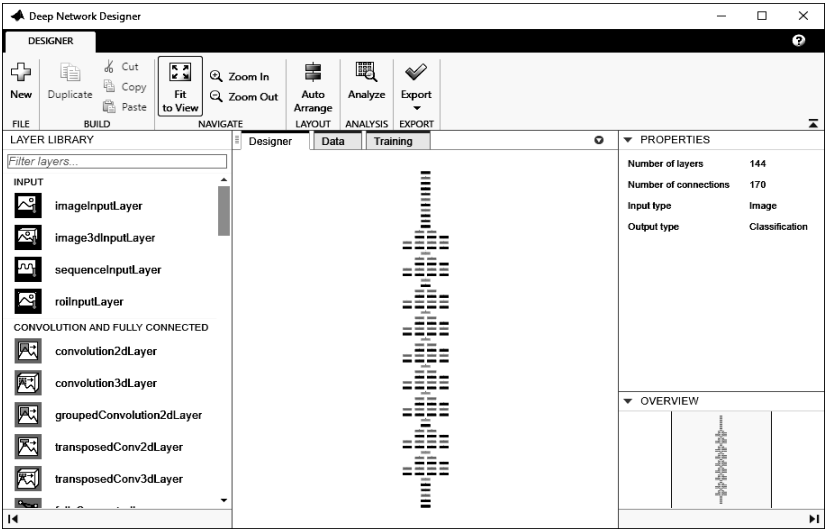


Рисунок 6.13 – Форма створення нейромереж засобу DeepNetworkDesigner

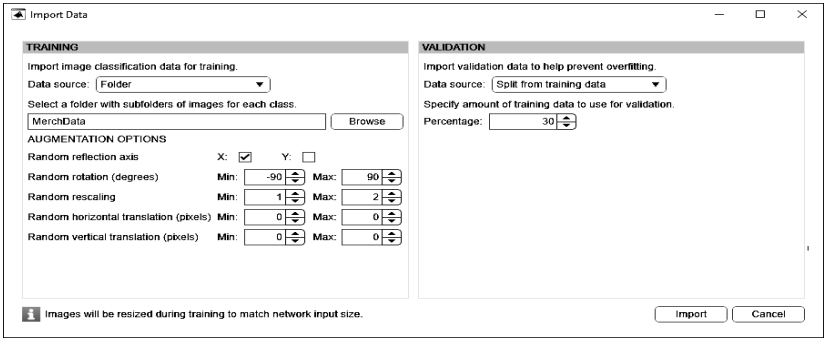


Рисунок 6.14 – Форма імпорту даних засобу DeepNetworkDesigner

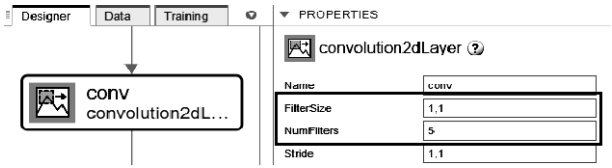


Рисунок 6.15 – Форма властивостей згорткового шару засобу DeepNetworkDesigner

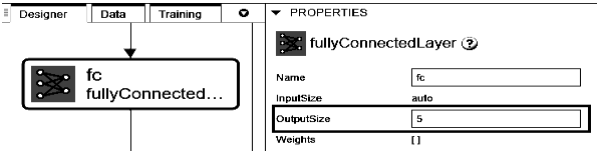


Рисунок 6.16 Форма властивостей повнозв'язного шару засобу DeepNetworkDesigner

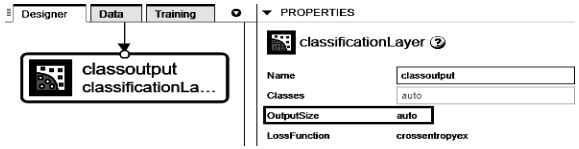


Рисунок 6.17 Форма властивостей вихідного шару засобу DeepNetworkDesigner

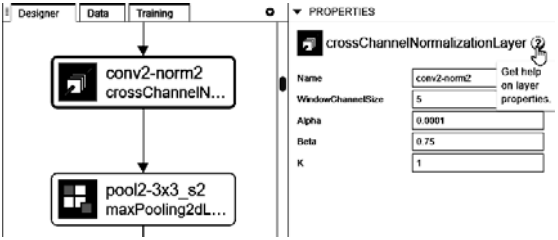


Рисунок 6.18 – Форма властивостей шару нормалізації

Також можна виділити тестові дані з навчальних даних. Обравши місце навчальних даних, вкажіть тестові дані та встановіть будь-які параметри збільшення, натисніть Import, щоб імпортувати набір даних.

Редагування попередньо навченої нейромережі для переданого навчання (передане навчання – це процес налагодження ваг вже попередньо навченої мережі для її настроювання на конкретну задачу за заданою навчальною вибіркою, який дозволяє швидше будувати прикладну нейромодель у порівнянні з навчанням мережі з випадковими значеннями початкових ваг) здійснюють таким чином.

Спочатку відкривають Deep Network Designer, завантажують попередньо навчену нейромережу, обираючи її на початковій сторінці Deep Network Designer (наприклад, можна обрати SqueezeNet). Після цього треба натиснути "Open" для завантаження мережі.

Для підготовки мережі до переданого навчання треба замінити останній шар, що навчається, та останній класифікуючий шар.

Якщо останній шар, що навчається, це двовимірний згортковий шар (наприклад, шар 'conv10' у SqueezeNet), то перетягніть новий convolutional2dLayer на робоче полотно, встановіть властивість NumFilters рівній новій кількість класів, а FilterSize встановіть як 1,1, видаліть останній convolutional2dLayer і замість цього підключіть новий шар.

Якщо останній шар, що навчається, це повнозв'язний шар (наприклад, у GoogLeNet), то перетягніть новий fullyConnectedLayer на робоче полотно та встановіть властивість OutputSize рівною новій кількості класів, видаліть останній fullyConnectedLayer та підключіть замість нього новий шар, після чого видаліть вихідний класифікуючий шар, потім перетягніть новий classificationLayer на полотно та підключіть його замість старого шару.

Щоб перевірити, чи готова мережа до навчання, на вкладці "Designer" натисніть "Analyze".

Для навчання мережі оберіть вкладку "Training".

Для отримання інформації про властивості шару на панелі "Designer" треба обрати шар для перегляду та редагування властивостей. Далі треба натиснути значок довідки поруч із назвою шару.

Навчання нейромережі на основі даних, імпортованих у Deep Network Designer, здійснюють таким чином. На вкладці "Training" треба обрати "Train". Якщо треба налаштувати параметри навчання, то обирають "Training Options".

Якщо необхідно навчати мережу на інших типах даних, то на вкладці "Designer" треба обрати "Export" для експорту початкової архітектури мережі. Після цього можливо програмно навчати мережу.

Експорт архітектури мережі, створеної в Deep Network Designer, у робочу область. Для експорту архітектури мережі з початковими вагами на вкладці "Designer" натисніть "Export". Для експорту архітектури мережі з навченими вагами на вкладці "Training" натисніть "Export".

Генерація тексту програми Matlab для створення архітектури мережі. Для відтворення шарів мережі на вкладці "Designer" оберіть "Export > Generate Code". Крім того, можливо відтворити мережу, включаючи будь-які параметри, що можуть навчатися, обравши "Export > Generate Code with Initial Parameters". Після генерації тексту програми можна виконати такі дії: відтворити шари мережі, створеної засобом Deep Network Designer (для цього треба запустити створений скрипт), навчати мережу (для цього треба запустити скрипт та подати шари до функції TrainNetwork), проаналізувати текст скрипту, щоб навчитися створювати та підключати шари програмно, змінити шари (для цього треба відредагувати текст скрипту), запустити скрипт та імпортувати мережу назад у Deep Network Designer для редагування.

Генерація тексту програми Matlab для навчання. Щоб відтворити імпорт даних та навчання, на вкладці «Training» оберіть Export > Generate Code for Training.

Основні функції для програмування глибоких мереж наведено у табл. 6.3.

Таблиця 6.3 – Основні функції для моделювання
глибоких нейромереж у пакеті Matlab

Назва функції	Призначення функції
1	2
trainingOptions	Параметри навчання нейромережі
trainNetwork	Навчання нейромережі
analyzeNetwork	Аналіз архітектури нейромережі
squeezenet	Створення згорткової нейромережі SqueezeNet
googlenet	Створення згорткової нейромережі GoogLeNet
inceptionv3	Згорткова нейромережа Inception-v3
densenet201	Згорткова нейромережа DenseNet-201
mobilenetv2	Згорткова нейромережа MobileNet-v2
resnet18	Згорткова нейромережа ResNet-18
resnet50	Згорткова нейромережа ResNet-50
resnet101	Згорткова нейромережа ResNet-101
xception	Згорткова нейромережа Xception
inceptionresnetv2	Попередньо навчена згорткова нейромережа Inception-ResNet-v2
nasnetlarge	Попередньо навчена згорткова нейромережа NASNet-Large
nasnetmobile	Попередньо навчена згорткова нейромережа NASNet-Mobile
shufflenet	Попередньо навчена згорткова нейромережа ShuffleNet
darknet19	Згорткова нейромережа DarkNet-19
darknet53	Згорткова нейромережа DarkNet-53
alexnet	Згорткова нейромережа AlexNet
vgg16	Згорткова нейромережа VGG-16
vgg19	Згорткова нейромережа VGG-19
imageInputLayer	Шар введення зображення
image3dInputLayer	Тривимірний шар введення зображення
convolution2dLayer	Двовимірний згортковий шар
convolution3dLayer	Тривимірний згортковий шар
groupedConvolution2dLayer	Двовимірний груповий згортковий шар
transposedConv2dLayer	Транспонований двовимірний згортковий шар
transposedConv3dLayer	Транспонований тривимірний згортковий шар
fullyConnectedLayer	Повністю зв’язаний шар
reluLayer	Шар ReLU
leakyReluLayer	Тікучий шар ReLU
clippedReluLayer	Обрізаний ректифікований шар ReLU
eluLayer	Шар ELU

Продовження табл. 6.3

1	2
tanhLayer	Шар гіперболічного тангенсу (tanh)
batchNormalizationLayer	Шар нормалізації пакетів
crossChannelNormalizationLayer	Шар нормалізації локальної канальної реакції
dropoutLayer	Шар випадання
crop2dLayer	Двовимірний збиральний шар
crop3dLayer	Тривимірний збиральний шар
averagePooling2dLayer	Двовимірний шар об'єднання усередненням
averagePooling3dLayer	Тривимірний шар об’єднання усередненням
globalAveragePooling2dLayer	Двовимірний глобальний шар об’єднання усередненням
globalAveragePooling3dLayer	Тривимірний глобальний шар об’єднання усередненням
globalMaxPooling2dLayer	Двовимірний глобальний шар об’єднання максимізацією
globalMaxPooling3dLayer	Тривимірний глобальний шар об’єднання максимізацією
maxPooling2dLayer	Двовимірний шар об’єднання максимізацією
maxPooling3dLayer	Тривимірний шар об’єднання максимізацією
maxUnpooling2dLayer	Двовимірний шар роз’єднання максимізацією
additionLayer	Шар додавання
concatenationLayer	Шар з’єднання
depthConcatenationLayer	Шар з’єднання в глибину
softmaxLayer	Шар Softmax
classificationLayer	Вихідний класифікуючий шар
regressionLayer	Вихідний регресійний шар
augmentedImageDatastore	Перетворення пакетів для збільшення даних зображень
imageDataAugmenter	Налаштування збільшення даних зображенньь
augment	Застосування ідентичних випадкових перетворень до множини зображень
layerGraph	Схема шарів мережі для глибокого навчання
plot	Побудова схеми шарів мережі
addLayers	Додавання шарів до схеми шарів
removeLayers	Видалення шарів зі схеми шарів
replaceLayer	Заміна шару у схемі шарів
connectLayers	З’єднання шарів у схемі шарів

Продовження табл. 6.3

1	2
disconnectLayers	Роз'єднання шарів у схемі шарів
DAGNetwork	Створення нейромережі Directed acyclic graph (DAG)
classify	Класифікація даних на основі навченої глибокої нейромережі
activations	Обчислення активацій шару глибокої нейромережі
predict	Передбачення виходів на основі навченої глибокої нейромережі
confusionchart	Створення діаграми матриці помилок для задач класифікації
sortClasses	Сортування класів діаграми матриці помилок

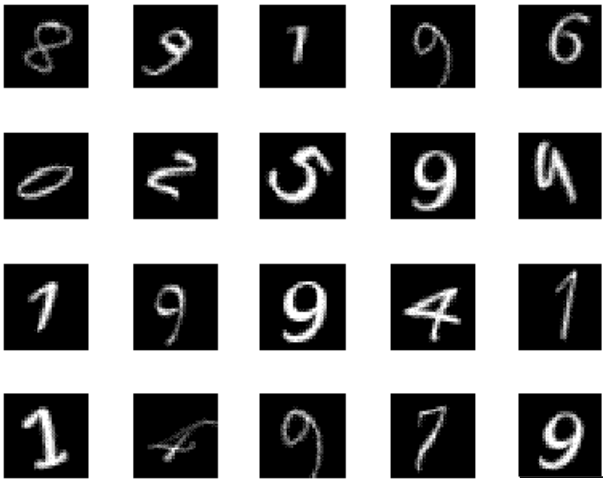
Розглянемо *створення простої згорткової нейронної мережі* з глибоким навчанням для класифікації (приклад TrainABasicConvolutionalNeuralNetworkForClassificationExample.m).

```
% Завантаження даних - вибірки зображень цифр
% як об'єкта ImageDatastore: imageDatastore – функція,
% що маркує зображення автоматично на основі імен папок
% та зберігає дані як об'єкт ImageDatastore, який дозволяе
% зберігати великі дані зображень, включаючи ті,
% що не уміщуються до пам'яті, та ефективно зчитує пакети
% зображень протягом навчання мережі.
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet', ...
'nndemos', 'nndatasets','DigitDataset');
digitData = imageDatastore(digitDatasetPath, ...
        'IncludeSubfolders',true,'LabelSource','foldernames');

% Показ деяких зображень з об'єкту datastore.
figure;
perm = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
    imshow(digitData.Files{perm(i)});
end

% Обчислення кількості зображень у кожному класі
% CountLabel - таблиця, що містить мітки кластів та
% номери зображень у кожному класі.
CountLabel = digitData.countEachLabel;

% Перевірка та друк розміру першого зображення в об'єкті digitData.
img = readimage(digitData,1);
size(img)

% Визначення навчальної та тестової вибірок
% Дані розподіляються на навчальні та тестові так,
% щоб кожний клас у навчальній вибірці мав по 750 зображень,
% а решта - потрапляють у тестову вибірку.
% Функція splitEachLabel розбиває зображення хранилища digitData
```

```
% на два нових хранилища зображень trainDigitData та testDigitData.
trainingNumFiles = 750;
rng(1) % For reproducibility
[trainDigitData,testDigitData] = splitEachLabel(digitData, ...
trainingNumFiles,'randomize');
% Визначення архітектури згорткової нейромережі:
% Image Input Layer - вхідний шар мережі - визначається
% розміром вхідного зображення: висота, ширина, та
% розмір каналу (1 - тони сірого, 3 - кольоровий);
% Convolutional Layer - згортковий шар мережі: перший аргумент
% filterSize - висота та ширина фільтрів, які використовує функція
% навчання підчас перегляду зображень (у прикладі число 5 позначає,
% що розмір фільтра - [5,5]), другий аргумент - кількість фільтрів,
% що є кількістю нейронів, пов’язаних з цим регіоном виходу
% (визначає кількість карт ознак);
% ReLU Layer - шар вузлів з нелійнійною функцією активації;
% Max-Pooling Layer - шар, що зменшує розмірність даних.
% Він повертає максимальні значення прямокутних регіонів входів,
% заданих першим аргументом, розмір прямокутного регіону - [2,2],
% необов’язковий аргумент Stride визначає розмір кроку, з яким
% функція навчання проходить по зображенню;
% Fully Connected Layer - повнозв'язний шар, кількість вузлів
% на останньому повнозв'язному шарі має дорівнювати кількості
% розпізнаваних класів;
% Softmax Layer - повнозв'язний шар, що використовує
% функцію активації softmax;
% Classification Layer - останній шар мережі,
% що видає клас розпізнаваного екземпляра.
layers = [imageInputLayer([28 28 1])
          convolution2dLayer(5,20)
          reluLayer
          maxPooling2dLayer(2,'Stride',2)
          fullyConnectedLayer(10)
          softmaxLayer
          classificationLayer()];

% Визначення параметрів навчання: 'sgdm' - метод навчання
% нейромережі - стохастичний градієнтний спуск з моментом,
% 'MaxEpochs' - максимальна кількість повних
% циклів навчання (епох),15 - значення MaxEpochs,
% 'InitialLearnRate' - початкове значення кроку навчання,
% 0.0001 – значення InitialLearnRate.
options = trainingOptions('sgdm','MaxEpochs', ...
15,'InitialLearnRate',0.0001);

% Навчання мережі на основі заданих параметрів
% та навчальної вибірки даних.
convnet = trainNetwork(trainDigitData,layers,options);

%% Розпізнавання зображень тестової вибірки.
YTest = classify(convnet,testDigitData);
TTest = testDigitData.Labels;

% Обчислення точності моделі.
accuracy = sum(YTest == TTest)/numel(TTest)
```

Результати роботи програми

```
TrainABasicConvolutionalNeuralNetworkForClassificationExample
ans =
 28 28
Training on single CPU.
Initializing image normalization.
|=========================================================================|
|Epoch |Iteration|Time Elapsed|Mini-batch|Mini-batch|Base Learning|
|      |         |  (seconds) |    Loss  | Accuracy |     Rate    |
|=========================================================================|
|    1 |      1  |     3.50   |  3.0845  |  13.28%  |  1.00e-04   |
|    1 |     50  |    61.74   |  1.0945  |  65.63%  |  1.00e-04   |
|    2 |    100  |   105.03   |  0.7281  |  74.22%  |  1.00e-04   |
|    3 |    150  |   143.27   |  0.4736  |  83.59%  |  1.00e-04   |
|    4 |    200  |   182.55   |  0.3092  |  91.41%  |  1.00e-04   |
|    5 |    250  |   244.61   |  0.2325  |  92.97%  |  1.00e-04   |
|    6 |    300  |   289.25   |  0.1540  |  97.66%  |  1.00e-04   |
|    7 |    350  |   327.57   |  0.1314  |  97.66%  |  1.00e-04   |
|    7 |    400  |   365.52   |  0.0943  |  96.09%  |  1.00e-04   |
|    8 |    450  |   404.55   |  0.0665  |  99.22%  |  1.00e-04   |
|    9 |    500  |   442.89   |  0.0460  |  99.22%  |  1.00e-04   |
|   10 |    550  |   482.56   |  0.0545  | 100.00%  |  1.00e-04   |
|   11 |    600  |   520.64   |  0.0658  |  99.22%  |  1.00e-04   |
|   12 |    650  |   559.68   |  0.0338  | 100.00%  |  1.00e-04   |
|   13 |    700  |   601.57   |  0.0342  | 100.00%  |  1.00e-04   |
|   13 |    750  |   639.85   |  0.0370  |  99.22%  |  1.00e-04   |
|   14 |    800  |   678.92   |  0.0264  | 100.00%  |  1.00e-04   |
|   15 |    850  |   717.38   |  0.0181  | 100.00%  |  1.00e-04   |
|   15 |    870  |   732.74   |  0.0235  | 100.00%  |  1.00e-04   |
|=========================================================================|
accuracy = 0.9808
```



Опис результатів роботи програми.

Після запуску скрипту TrainABasicConvolutionalNeuralNetwork-ForClassificationExample програма видає розмір зображення.

В окремому вікні видається зображення зі зразками вхідних образів.

Далі зазначається, що тренування виконувалося на одному процесорі. За наявності більшої кількості процесорів або графічної карти програма автоматично обере потрібну кількість процесорів, яку виведе на екран.

Після чого видається підтвердження того, що йде процес нормалізації зображень.

Далі послідовно видається підчас навчання таблиця, що характеризує процес навчання: Epoch – номер виконаної епохи, Iteration — номер поточної ітерації, Time Elapsed – обсяг витраченого часу, Mini-batch loss – втрати для мініпакету (підвибірки), точність мініпакету, базовий крок навчання (коригувальний приріст)

Після таблиці видається точність навчання для тестової вибірки.

Налаштування попередньо навченої згорткової нейромережі. При необхідності побудувати нейромодель за обмеженою вибіркою спостережень, наявною у користувача, може виявитися більш ефективною технологія transfer learning – передачі навчання, коли користувач отримує попередньо навчену на великому масиві даних мережу, а потім донавчає її на подібних спостереженнях, які він має.

```
% Завантаження попередньо навченої мережі у змінній net з файлу
% LettersClassificationNet.mat, що навчена на великій колекції
% зображень літер у тонах сірого розміром 28x28 пікселів.
% Мережа розподіляє літери на три класи: 'A', 'B' та 'C'.
load(fullfile(matlabroot,'examples','nnet',...
'LettersClassificationNet.mat'))

% Перевірка параметрів архітектури мережі, що містяться
% у властивості Layers об'єкта net.
net.Layers

% Завантаження навчальної вибірки даних зображень цифр як
% об’єкту ImageDatastore. Функція imageDatastore автоматично
% позначає зображення мітками класів, грунтуючись на іменах
% папок, та зберігає дані як об’єкт ImageDatastore, який
% дозволяє зберігати великі дані зображень, включаючи ті,
% що не уміщуються у пам’яті. Цей об’єкт також дозволяє
% ефективно зчитувати пакети зображень підчас навчання.
% Хранилище даних містить 10000 штучно синтезованих зображень
% цифр 0-9. Зображення згенеровані застосуванням випадкового
```

```
% перетворення до зображень цифр, створених з використанням
% різних гарнітур шрифтів. Кожне зображення має розмір
% 28x28 пікселів.
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet',...
'nndemos', 'nndatasets','DigitDataset');
digitData = imageDatastore(digitDatasetPath, ...
'IncludeSubfolders',true,'LabelSource','foldernames');

% Розбиття вибірки даних на навчальну та тестову вибірки
% таким чином, щоб кожне хранилище даних мало 50% зображень
% кожного класу. splitEachLabel розбиває файли зображень
% в digitData на два нових хранилища даних: trainDigitData
% та testDigitData.
[trainDigitData,testDigitData] = splitEachLabel(digitData,...
0.5,'randomize');

% Показ 20 випадкових зразків зображень навчальної вибірки
numImages = numel(trainDigitData.Files);
idx = randperm(numImages,20);
for i = 1:20
    subplot(4,5,i)
    I = readimage(trainDigitData, idx(i));
    imshow(I)
end

% Передача шарів до цільової мережі.
% Оскільки останні три шари попередньо навченої мережі net
% навчені для літер, то їх треба замінити та навчити для цифр.
% Тому здійснюється витяг усіх шарів, окрім трьох останніх,
% з попередньо навченої мережі
layersTransfer = net.Layers(1:end-3);

% Оскільки дані цифр мають 10 класів, то додамо новий
% повнозв'язний шар для 10 класів, шар softmax та
% вихідний шар класифікації.
% Для підвищення швидкості навчання тільки для нових шарів
% збільшимо кроки навчання для ваг та порогів повнозв'язного шару
numClasses = numel(categories(trainDigitData.Labels))
layers = [layersTransfer fullyConnectedLayer(numClasses, ...
'WeightLearnRateFactor',20,'BiasLearnRateFactor', ...
20) softmaxLayer classificationLayer];

% Якщо вхідні зображення відрізняються за розміром від
% вхідного шару зображень, тоді треба змінити розмір даних
% зображень. У цьому прикладі розміри співпадають.
% Визначення параметрів процесу навчання.
% Для передачі навчання треба зберегти ознаки з попередніх
% шарів попередньо навченої мережі (передача ваг шарів).
% Для цього встановимо InitialLearnRate як мале значення.
% Це мале значення кроку навчання буде уповільнювати
% навчання на переданих шарах.
% На попередньому кроці для повнозв'язного шару встановлені
% великі кроки навчання для пришвидшення навчання нових
% останніх шарів.
% Така комбінация призваеде до швидкого навчання тільки
% нових шарів, фіксуючи інші шари. Передача навчання
```

```
% не потребує багато циклів тренування (епох).
% Для пришвидшення навчання доцільно зменшити значення
% параметру 'MaxEpochs' при виклиці trainingOptions.
optionsTransfer = trainingOptions('sgdm', ...
    'MaxEpochs',5, ...
    'InitialLearnRate',0.0001);

% Донавчання мережі, що складається з переданих та нових шарів
netTransfer = trainNetwork(trainDigitData,layers,optionsTransfer);
% Обчислення точності класифікації - долі правильно
% розпізнаних зразків тестової вібірки
YPred = classify(netTransfer,testDigitData);
YTest = testDigitData.Labels;
accuracy = sum(YPred==YTest)/numel(YTest)

% Показ прикладів тестових зображень з їхніми мітками класів,
% розпізнаними мережею.
idx = 501:500:5000;
figure
for i = 1:numel(idx)
    subplot(3,3,i)
    I = readimage(testDigitData, idx(i));
    label = char(YTest(idx(i)));
    imshow(I)
    title(label)
end
```

Результати роботи програми.

```
ans =
 7x1 Layer array with layers:
  1 'imageinput' Image Input
           28x28x1 images with 'zerocenter' normalization
  2 'conv' Convolution
           20 5x5x1 convolutions with stride [1 1] and padding [0 0]
  3 'relu'   ReLU     ReLU
  4 'maxpool' Max Pooling
           2x2 max pooling with stride [2 2] and padding [0 0]
  5 'fc'   Fully Connected
            3 fully connected layer
  6 'softmax'  Softmax     softmax
  7 'classoutput' Classification Output
           crossentropyex with 'A', 'B', and 1 other classes
numClasses = 10
Training on single CPU.
Initializing image normalization.
|=========================================================================|
|Epoch|Iteration|Time Elapsed|Mini-batch|Mini-batch| Base Learning|
|     |         |   seconds) |   Loss   | Accuracy |     Rate     |
|=========================================================================|
|   1 |       1 |      1.57  |    9.4193|   11.72% |   1.00e-04   |
|   2 |      50 |     40.01  |    3.6911|   64.06% |   1.00e-04   |
|   3 |     100 |     78.50  |    1.7290|   82.81% |   1.00e-04   |
|   4 |     150 |     117.13 |    1.2480|   88.28% |   1.00e-04   |
|   5 |     195 |     151.77 |    0.6499|   89.84% |   1.00e-04   |
|=========================================================================|
accuracy =    0.8946
```

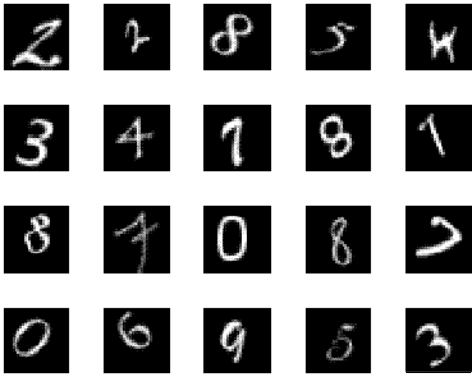


Figure 1

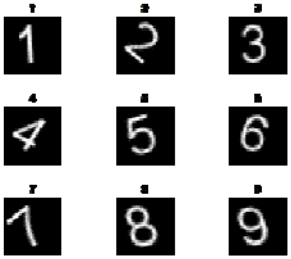


Figure 2

Опис результатів роботи програми. Після запуску програма видає значення параметрів архітектури мережі. Мережа містить сім шарів. Шари 2-4 знаходять ознаки. Шари 5-7 перетворюють ознаки у класи літер. 20 виходів (каналів) шару згорток відповідають вивченим ознакам. Останні три шари попередньо навченої мережі net налаштовуються за даними літер. Шляхом заміни останніх трьох шарів попередньо навченої мережі можливо навчити мережу класифікувати цифри.

Далі видається кількість класів numClasses = 10

Після чого повідомляється, що навчання здійснюється на одному процесорі.

Далі зазначається, що виконується нормалізація зображень.

Після чого у процесі навчання послідовно видається таблиця з характеристиками процесу навчання: Epoch – номер виконаної епохи, Iteration – номер поточної ітерації, Time Elapsed – обсяг витраченого часу, Mini-batch – втрати для мініпакету (підвибірки), точність мініпакету, базовий крок навчання (коригувальний приріст)

Після таблиці видається точність навчання для тестової вибірки.

На окремих рисунках видаються 20 випадкових зображень з навчальної вибірки (figure 1) та 9 забражень тестової вибірки зі співставленими ним мітками класів мережею (figure 2).

Навчання згорткової нейронної мережі регресії. Розглянемо приклад передбачення кутів повороту рукописних цифр за допомогою згорткової нейромережі.

```
% Завантаження навчальних даних як чотиривимірного масиву.
% Мережа навчатиметься на колекції штучно синтезованих
% рукописних символах (5000 зображень цифр з відповідними
% углами повороту).
[trainImages,~,trainAngles] = digitTrain4DArrayData;

% Демонстрація 20 випадкових зразків навчальних цифр.
numTrainImages = size(trainImages,4);
figure
idx = randperm(numTrainImages,20);
for i = 1:numel(idx)
    subplot(4,5,i)
    imshow(trainImages(:,:,:,idx(i)))
    drawnow
end

% Створення шарів мережі
% Перший шар визначає розмір та тип вхідних даних.
% Вхідні зображення мають розмір 28x28 точок та
% 1 канал у тонах сірого.
% Середні шари визначають ядро архітектури мережі.
% Створюється двовимірний згортковий шар з 25 фільтрами
% розміром 12, за яким йде шар ReLU.
% Останні шари визначають розмір та тип вихідних данихю.
% Для регресії повнозв'язаний шар має передувати
% регресійному шару, що розміщується у кінці мережі.
% Створюються повнозв'язний шар розміру 1 та регресійний шар.
% Далі усі шари поєднуються в масиві Layer.
layers = [ ...
    imageInputLayer([28 28 1])
    convolution2dLayer(12,25)
    reluLayer
    fullyConnectedLayer(1)
    regressionLayer];

% Навчання мережі
% Визначення параметрів навчання мережі.
% Установка начального значення кроку навчання у 0.001.
```

```
% Для зменшення часу навчання можна зменшити значення 'MaxEpochs'.
options = trainingOptions('sgdm','InitialLearnRate',0.001, ...
'MaxEpochs',15);

% Створення нейромережі за допомогою функції trainNetwork.
net = trainNetwork(trainImages,trainAngles,layers,options)

% Перевірка параметрів архітектури мережі,
% що містяться у властивості Layers об'єкту net.
net.Layers

% Перевірка якості мережі шляхом обчислення точності
% передбачення для тестових даних.
% Завантаження тестової множини цифр.
[testImages,~,testAngles] = digitTest4DArrayData;

% Функція predict передбачає кути поворотів тестових зображень.
predictedTestAngles = predict(net,testImages);

% Обчислення якості моделі.
% Обчислення помилки передбачення між передбаченими
% та дійсними кутами поворотів.
predictionError = testAngles - predictedTestAngles;

% Обчислення кількості передбачень у межах прийнятної
% помилки відносно дійсних кутів. Установка порогу
% у 10 градусів. Обчислення відсотку передбачень з цим порогом.
thr = 10;
numCorrect = sum(abs(predictionError) < thr);
numTestImages = size(testImages,4);
accuracy = numCorrect/numTestImages

% Обчислюємо середньоквадратичну помилку (RMSE) для вимірювання
% різниці між передбаченим та справжнім кутами повороту.
squares = predictionError.^2;
rmse = sqrt(mean(squares))

% Обчислення залишків.
residuals = testAngles - predictedTestAngles;

% Функція boxplot вимагає матрицю, де кожний стовпець відповідає
% залишкам кожного класу цифр.
% Тестові дані містять зображення цифр 0-9
% по 500 зразків кожного класу.
% Функція reshape групує залишки за класами цифр.
% Кожний стовпець residualMatrix відповідає залишкам
% для відповідної цифри.
residualMatrix = reshape(residuals,500,10);

% Зображення залишків для кожного класу цифр
% у вигляді діаграм "шухляда з вусами"
figure
boxplot(residualMatrix, ...
'Labels',{'0','1','2','3','4','5','6','7','8','9'})
xlabel('Digit Class')
ylabel('Degrees Error')
```

```
title('Residuals')

% Класи цифр з найвищою точністю мають
% середні, близькі до нуля, та незначні дисперсії.
% Корекція поворотів цифр
idx = randperm(numTestImages,49);
for i = 1:numel(idx)
    image = testImages(:,:,:,idx(i));
    predictedAngle = predictedTestAngles(idx(i));
    imagesRotated(:,:,:,i) = imro-
tate(image,predictedAngle,'bicubic','crop');
end

% Демонстрація оригінальних зображень цифр та
% їхніх відкоректованих поворотів.
figure
subplot(1,2,1)
montage(testImages(:,:,:,idx))
title('Original')
subplot(1,2,2)
montage(imagesRotated)
title('Corrected')
```

Результати роботи програми.

```
>> TrainAConvolutionalNeuralNetworkForRegressionExample
Training on single CPU.
Initializing image normalization.
|=========================================================================|
| Epoch |Iteration|Time Elapsed|Mini-batch|Mini-batch|Base Learning|
|       |         |  (seconds) |   Loss   |   RMSE   |     Rate     |
|=========================================================================|
|     1 |       1 |       3.71  | 392.5880 |    28.02 |    0.0010    |
|     2 |      50 |      96.96  |  96.0224 |    13.86 |    0.0010    |
|     3 |     100 |     185.28  |  55.5496 |    10.54 |    0.0010    |
|     4 |     150 |     270.46  |  64.4647 |    11.35 |    0.0010    |
|     6 |     200 |     352.72  |  36.2445 |     8.51 |    0.0010    |
|     7 |     250 |     433.93  |  53.0401 |    10.30 |    0.0010    |
|     8 |     300 |     513.46  |  35.7745 |     8.46 |    0.0010    |
|     9 |     350 |     592.11  |  34.9632 |     8.36 |    0.0010    |
|    11 |     400 |     670.20  |  24.9389 |     7.06 |    0.0010    |
|    12 |     450 |     748.11  |  33.0234 |     8.13 |    0.0010    |
|    13 |     500 |     830.36  |  27.1019 |     7.36 |    0.0010    |
|    15 |     550 |     915.09  |  18.1279 |     6.02 |    0.0010    |
|    15 |     585 |     968.85  |  18.9767 |     6.16 |    0.0010    |
|=========================================================================|
net =
    SeriesNetwork with properties:
    Layers: [5×1 nnet.cnn.layer.Layer]
ans =
 5x1 Layer array with layers:
1 'imageinput' Image Input
         28x28x1 images with 'zerocenter' normalization
2 'conv' Convolution
         25 12x12x1 convolutions with stride [1 1] and padding [0 0]
3 'relu' ReLU ReLU
```

```
4 'fc'    Fully Connected 1 fully connected layer
5 'regressionoutput' Regression Output
          mean-squared-error with response 'Response'
accuracy = 0.8488
rmse =
  single
  7.3413
```

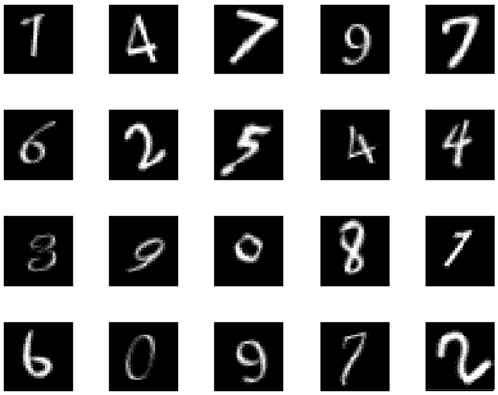


figure 1

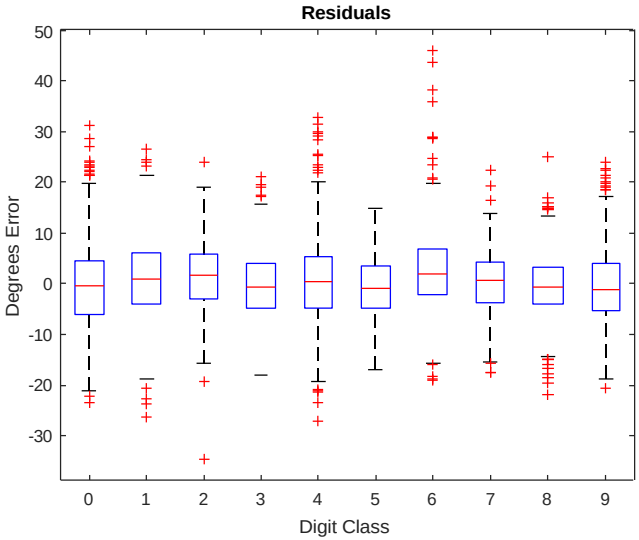


figure 2

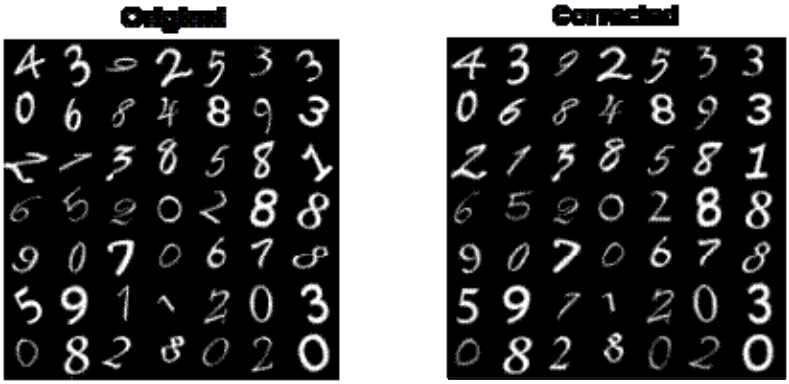


figure 3

Опис результатів роботи програми. Після запуску програми вона зазначає, що навчання відбувається на одному процесорі.

Далі повідомляється, що виконується нормалізація зображень.

Після чого у процесі навчання послідовно видається таблиця з характеристиками процесу навчання: Epoch – номер виконаної епохи, Iteration – номер поточної ітерації, Time Elapsed – обсяг витраченого часу, Mini-batch loss – втрати для мініпакету (підвибірки), Mini-batch RMSE – середньоквадратична помилка для мініпакету, Base Learning Rate – базовий крок навчання (коригувальний приріст).

Після таблиці видається опис мережі об’єкту net. Зазначається, що це – серійна мережа, що складається з 5x1 шарів:

1 'imageinput' – вхідне зображення (28x28x1 з нормалізацією 'zerocenter');

2 'conv' – згортковий (25 12x12x1 згорток з кроком [1 1] та заповненням [0 0]);

3 'relu' – ReLU;

4 'fc' –повнозв’язний шар;

5 'regressionoutput' – регресійний вихід (середньоквадратична помилка з відгуком 'Response').

Після чого видаються точність accuracy = 0.8488 та середньоквадратична помилка rmse = 7.3413

На окремих рисунках видаються приклади випадкових зображень з навчальної множини (figure 1), діаграма залишків за класами цифр

(figure 2) та порівняння випадкових зображень тестової множини та їхніх корекцій на передбачену величину кута повороту (figure 3).

Класифікація послідовностей даних на основі LSTM мереж. Для класифікації послідовностей даних у пакеті Matlab можливо використовувати мережі довгої короткочасної пам'яті (Long Short-Term Memory – LSTM).

Розглянемо приклад навчання LSTM-мережі розпізнаванню мовця за двома японськими голосними, вимовленими послідовно.

```
% Завантаження навчальних даних – масиву звуків японської мови
% X, що містить 270 послідовностей розмірності 12 різної довжини.
% Y – це категоріальний вектор з мітками "1","2",...,"9",
% що відповідають 9 особам, яки вимовляли звуки.
load JapaneseVowelsTrain

% Перегляд п'яти перших спостережень.
% Записи в X – це матриці з 12 рядків (один рядок на кожну ознаку)
% та змінною кількістю стовпців (один стовпець на кожний крок часу).
X(1:5)

% Візуалізація перших часових рядів
% (кожен вбудований графік відповідає ознаці)
figure
for i = 1:12
    subplot(12,1,13-i)
    plot(X{1}(i,:));
    ylabel(i)
    xticklabels('')
    yticklabels('')
    box off
end
title("Training Observation 1")
subplot(12,1,12)
xticklabels('auto')
xlabel("Time Step")

% Отримання довжини посідовності для кожного спостереження.
numObservations = numel(X);
for i=1:numObservations
    sequence = X{i};
    sequenceLengths(i) = size(sequence,2);
end

% Упорядковування даних за довжиною послідовностей.
% Під час тренувань навчальні дані розбиваються на міні-партії
% та набиває або обрізає послідовності так, щоб вони мали однакову
% довжину. Завеликі набивка або видалення даних можуть негативно
% вплинути на продуктивність мережі. Для запобігання цьому можна
% відсортувати навчальні дані за довжиною послідовності та обрати
% розмір міні-пакету так, щоб послідовності у міні-пакеті мали
% однакову довжину.
[sequenceLengths,idx] = sort(sequenceLengths);
```

```
X = X(idx);
Y = Y(idx);

% Перегляд упорядкованих довжин послідовностей
% на стовпчиковій діаграмі.
figure
bar(sequenceLengths)
ylim([0 30])
xlabel("Sequence")
ylabel("Length")
title("Sequence Lengths")

% Встановлення обмежень міні-пакету
% Розмір міні-пакету 27 поділяє навчальні дані рівномірно
% та зменшує кількість набивок в міні-пакетах.
miniBatchSize = 27;
miniBatchLocations = miniBatchSize+1:miniBatchSize:numObservations;
XLocations = repmat(miniBatchLocations,[2 1]);
YLocations = repmat([0;30],[1 9]);

% Зображення обмежень міні-пакетів з довжинами послідовностей.
% Зображення показує як послідовності розподіляються
% на міні-пакети.
hold on
line(XLocations,YLocations, 'Color','r', 'LineStyle','--')

% Визначення архітектури LSTM-мережі: розмірності вхідних даних
% inputSize, розмірності вихідних даних outputSize, режиму видачі
% (outputMode) – останній елемент послідовності 'last', кількості
% класів numClasses.
inputSize = 12;
outputSize = 100;
outputMode = 'last';
numClasses = 9;

% Визначення параметрів навчання: максимальної кількості
% епох навчання maxEpochs, розміру міні-пакету miniBatchSize
% та того, що не треба перетасовувати дані.
maxEpochs = 150;
miniBatchSize = 27;
shuffle = 'never';
options = trainingOptions('sgdm', 'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize,'Shuffle', shuffle);

% Навчання LSTM-мережі.
net = trainNetwork(X,Y,layers,options);

% Перевірка навченої мережі
% Завантаження тестової вибірки та класифікація послідовностей
% за мовцями.XTest – масив, що містить 370 послідовностей
% розмірністю 12 різної довжини. YTest – категоріальний вектор
% міток "1","2",..."9", що відповідають дев’яти мовцям.
load JapaneseVowelsTest
XTest(1:3)

% Візуалізуємо перші часові ряди на графіку.
```

```
% Кожен вбудований графік відповідає ознаці.
figure
for i = 1:12
    subplot(12,1,13-i)
    plot(XTest{1}(i,:))
    ylabel(i)
    xticklabels('')
    yticklabels('')
    box off
end
title("Test Observation 1")
subplot(12,1,12)
xticklabels('auto')
xlabel("Time Step")

% Мережа net навчена з використанням міні-пакетів
% однакової довжини. Пересвідчимося, що тестові дані мають
% таку ж довжину та організовані подібним чином.
% Упорядкуємо тестові дані за довжинлю послідовності.
numObservationsTest = numel(XTest);
for i=1:numObservationsTest
    sequence = XTest{i};
    sequenceLengthsTest(i) = size(sequence,2);
end
[sequenceLengthsTest,idx] = sort(sequenceLengthsTest);
XTest = XTest(idx);
YTest = YTest(idx);

% Класифікація тестових даних.
% Для зменшення кількості набивок, уведених процесом
% класифікації встановлюється розмір міні-пакету 27.

miniBatchSize = 27;
YPred = classify(net,XTest, 'MiniBatchSize',miniBatchSize);

% Обчислення точності класифікації
acc = sum(YPred == YTest)./numel(YTest)
```

Результати роботи програми.

```
ans = 5×1 cell array
    {12×20 double}
    {12×26 double}
    {12×22 double}
    {12×20 double}
    {12×21 double}

layers =
5x1 Layer array with layers:
1 ''   Sequence Input          Sequence input with 12 dimensions
2 ''   LSTM                   LSTM with 100 hidden units
3 ''   Fully Connected         9 fully connected layer
4 ''   Softmax                softmax
5 ''   Classification Output   crossentropyex

Training on single GPU.
```

```
|=========================================================================|
|Epoch|Iteration|Time Elapsed|Mini-batch|Mini-batch  |Base Learning|
|     |         |  (seconds) |   Loss   | Accuracy   |     Rate    |
|=========================================================================|
|    1 |    1    |    0.37    |  2.1973  |  0.00%     |  0.0100     |
|    5 |   50    |    1.36    |  2.2147  |  0.00%     |  0.0100     |
|   10 |  100    |    2.12    |  2.2042  |  0.00%     |  0.0100     |
|   15 |  150    |    2.89    |  2.1926  |  0.00%     |  0.0100     |
|   20 |  200    |    3.69    |  2.1652  |  14.81%    |  0.0100     |
|   25 |  250    |    4.46    |  2.0326  |  7.41%     |  0.0100     |
|   30 |  300    |    5.20    |  1.5640  |  3.70%     |  0.0100     |
|   35 |  350    |    5.96    |  1.1499  |  55.56%    |  0.0100     |
|   40 |  400    |    6.70    |  0.8349  |  62.96%    |  0.0100     |
|   45 |  450    |    7.44    |  0.7832  |  70.37%    |  0.0100     |
|   50 |  500    |    8.19    |  0.6772  |  74.07%    |  0.0100     |
|   55 |  550    |    8.93    |  0.5303  |  77.78%    |  0.0100     |
|   60 |  600    |    9.66    |  0.3584  |  88.89%    |  0.0100     |
|   65 |  650    |   10.44    |  0.2077  |  96.30%    |  0.0100     |
|   70 |  700    |   11.18    |  0.2477  |  88.89%    |  0.0100     |
|   75 |  750    |   11.99    |  0.0967  |  100.00%   |  0.0100     |
|   80 |  800    |   12.77    |  0.0551  |  100.00%   |  0.0100     |
|   85 |  850    |   13.56    |  0.1457  |  96.30%    |  0.0100     |
|   90 |  900    |   14.36    |  0.0581  |  100.00%   |  0.0100     |
|   95 |  950    |   15.14    |  0.0332  |  100.00%   |  0.0100     |
|  100 |  1000   |   15.89    |  0.0272  |  100.00%   |  0.0100     |
|  105 |  1050   |   16.64    |  0.0255  |  100.00%   |  0.0100     |
|  110 |  1100   |   17.42    |  0.0163  |  100.00%   |  0.0100     |
|  115 |  1150   |   18.18    |  0.0133  |  100.00%   |  0.0100     |
|  120 |  1200   |   18.94    |  0.0113  |  100.00%   |  0.0100     |
|  125 |  1250   |   19.70    |  0.0100  |  100.00%   |  0.0100     |
|  130 |  1300   |   20.48    |  0.0126  |  100.00%   |  0.0100     |
|  135 |  1350   |   21.32    |  0.1965  |  88.89%    |  0.0100     |
|  140 |  1400   |   22.12    |  0.0172  |  100.00%   |  0.0100     |
|  145 |  1450   |   22.91    |  0.0101  |  100.00%   |  0.0100     |
|  150 |  1500   |   23.69    |  0.0088  |  100.00%   |  0.0100     |
|=========================================================================|
ans = 3×1 cell array
    {12×19 double}
    {12×17 double}
    {12×19 double}
acc = 0.9432
```

Опис результатів роботи програми. Після запуску програми видається опис масиву з перших трьох спостережень та зображення графіків 12 ознак першого екземпляра навчальної вибірки (figure 1). Після цього на стовпчиковій діаграмі зображуються упорядковані довжини послідовностей (figure 2).

Далі після створення архітектури нейромережі видається її опис. Зазначається, що мережа складається з 5 шарів: 1 – вхідна послідовність (з 12 ознаками), 2 – LSTM зі 100 прихованими вузлами, 3 – повнозв'язний шар, 4 – шар Softmax, 5 – вихідний шар.

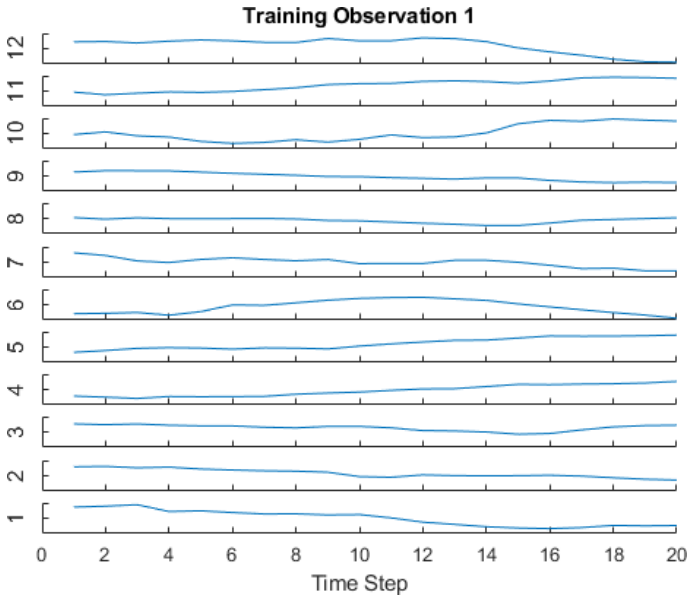


figure 1

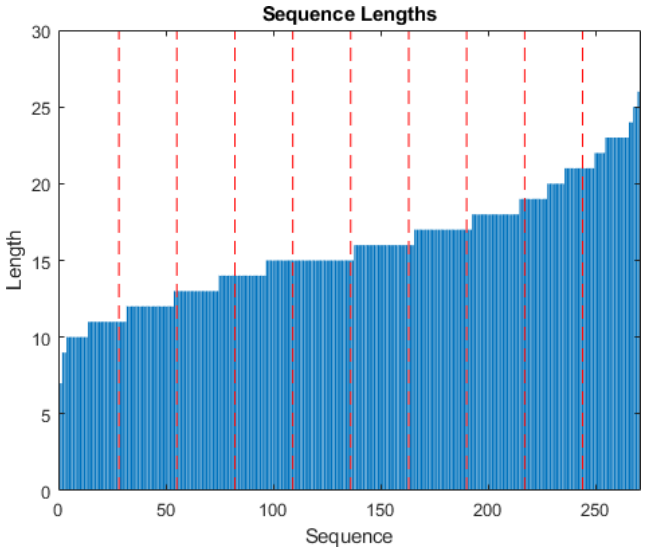


figure 2

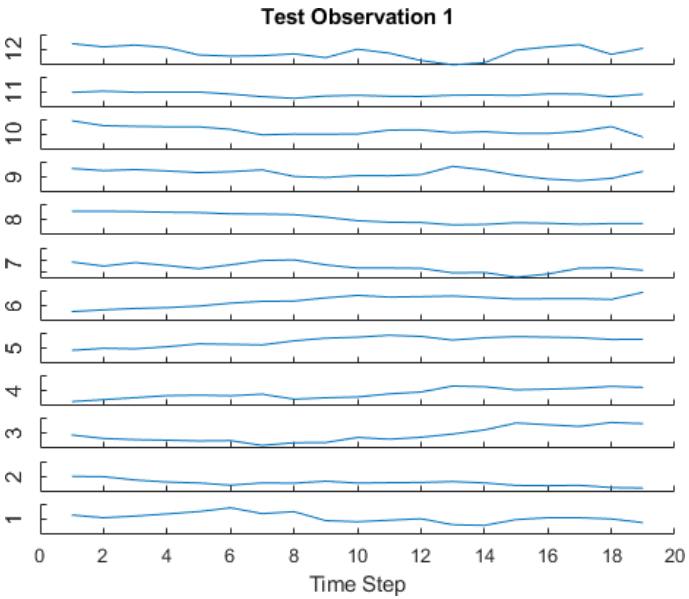


figure 3

Після чого описується процес навчання. Зазначається, що воно відбувається на одному графічному процесорі. Далі у процесі навчання послідовно видається таблиця з характеристиками процесу навчання: Epoch – номер виконаної епохи, Iteration – номер поточної ітерації, Time Elapsed – обсяг витраченого часу, Mini-batch loss – втрати для мініпакету (підвибірки), Mini-batch Accuracy – точність для мініпакету, Base Learning Rate – базовий крок навчання (коригувальний приріст).

Далі візуалізуються тестові послідовності (figure 3). Після чого видаються опис трьох елементів тестової вибірки та точність accuracy = 0.9432.

6.2 Моделювання нейронних мереж у пакеті Statistica Neural Networks

Statistica Neural Networks – це універсальний пакет нейромережевого аналізу фірми StatSoft. Він може працювати як самостійний додаток, а також у рамках системи *Statistica*. Пакет має ряд переваг.

Він містить найсучасніші і могутні методи навчання мережі (включаючи методи сполучених градієнтів та Левенберга-Марквардта); має можливість створювати складні, практично не обмежені в розмірах комбінації з мереж різних архітектур; дозволяє вибіркове навчання фрагментів нейронної мережі; забезпечує повний контроль над вибором функцій активації; містить систему *Network Wizard* – Нейро-Майстер , що допомагає користувачу приймати рішення і вибирати параметри.

Пакет має виняткову простоту у використанні та неперевершену аналітичну міць. Так, наприклад, *Automatic Network Designer* – Автоматичний Конструктор Мережі знайде найкращу архітектуру для НМ (вибір потрібної архітектури мережі – це важкий процес проб і помилок; Statistica Neural Networks зробить це за користувача).

Пакет містить могутні методи аналізу, у тому числі, унікальний інструмент нейро-генетичного відбору вхідних даних – *Neuro-Genetic Input Selection* (вибір потрібних вхідних змінних при розвідницькому аналізі даних – області, де найчастіше застосовуються нейромережі – займає багато часу; пакет Statistica Neural Networks здатний виконувати цю роботу за користувача).

Statistica Neural Networks може працювати у середовищі Statistica, але також може використовуватися і як самостійний додаток (є можливість уведення-виведення файлів даних системи Statistica; матриць вихідних даних і графіки).

Простий і зручний інтерфейс системи Statistica Neural Networks дозволяє швидко створювати нейромережеві додатки для вирішення користувальницьких задач.

Statistica Neural Networks цілком підтримує *інтерфейс прикладного програмування* – API, так що досвідчені користувачі (і розроблювачі користувальницьких систем «пошуку знань» і «видобутку даних») можуть використовувати могутні обчислювальні можливості модуля Statistica Neural Networks у своїх власних додатках. Інтерфейс прикладного програмування Statistica Neural Networks' API дозволяє вмонтувати ці додатки у вже наявну в систему, наприклад, зробити їх частиною більш широкого обчислювального середовища (це можуть бути процедури, розроблені окремо й

убудовані в комп'ютерну мережу фірми). API поставляється в двох варіантах. Скорочена версія API дозволяє запускати створені і навчені в середовищі Statistica Neural Networks нейронні мережі з інших програм-додатків, написаних на C, C++, Delphi або Visual Basic. Повний API надає користувачу доступ до всіх можливостей ядра пакета Statistica Neural Networks, включаючи створення, редагування, уведення/виведення даних і архітектур мереж, навчання мереж.

Пакет має багатовіконний Windows інтерфейс. Головне і дочірнє вікна пакета зображені на рис. 6.19.

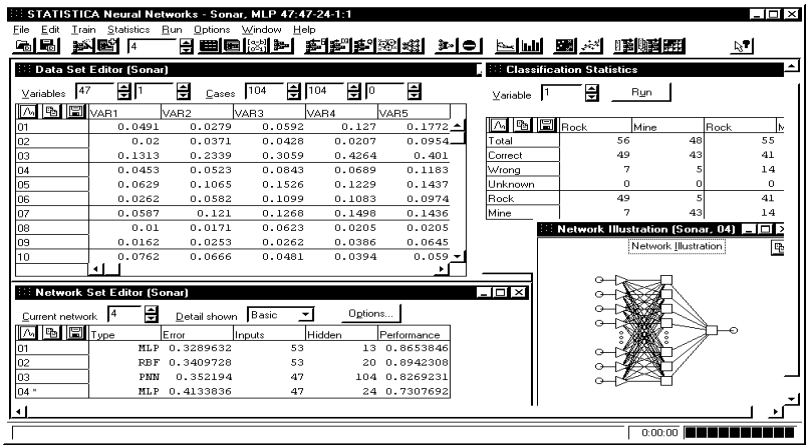


Рисунок 6.19 – Головне вікно пакета Statistica Neural Networks

Розглянемо призначення основних пунктів меню пакета.

Меню File (Файл) містить пункти: New – новий(ва) (Network – нейронна мережа, Intelligent Problem Solver – інтелектуальний вирішувач задач, Data Set – вибірка даних); Open – відкрити файл, Close – закрити файл, Save – зберегти в поточний файл, Save as – зберегти у файл з ім'ям; Network Set – набір нейромереж; Exit – вихід.

Меню Edit (Редагувати) містить пункти: Variables – змінні, ознаки; Cases – випадки, екземпляри; Undo – скасувати дію; Cut – вирізати; Copy – копіювати до буфера; Paste – вставити з буфера; Clear – очистити; Data Set – набір даних; Network Set – набір

нейромереж; Pre/post Processing – попередня / пост-обробка; Network – нейромережа.

Меню Train (Навчання) містить пункти: Multilayer Perceptrons – багатошарові персептрони; Radial-Basis Functions – радіально-базисні мережі; Linear – лінійні мережі; Kohonen – мережі Кохонена (SOM); Class labels – мітки класів; LVQ – мережі LVQ; Probabilistic – імовірнісні мережі; General Regression – узагальнено-регресійні мережі; Principal Components – аналіз головних компонент; Auxiliary – помічник: Best Network – добір кращої мережі, Stopping Conditions – критерії зупинення, Set Weights – установка ваг, Weigend Regularization – регуляція ваг, Feature Selection – відбір ознак.

Меню Statistics (Статистика) містить пункти: Training Graph – графік навчання; Case Errors – помилки для екземплярів; Classification – класифікація; ROC Curve – крива характеристики дії одержувача; Regression – регресія; Weight Distribution – розподіл ваг; Win Frequencies – частоти переможців; Sensivity – чутливість; Network Illustration – зображення мережі; Messages – повідомлення.

Меню Run (Запуск) містить пункти: Single Case – запуск для одного екземпляра; Data Set – запуск для набору даних; One-off – запуск для одного з прикладів, обраних інтерактивно; Activations – активація мережі; Time Series – часові ряди; Response Graph – графік відгуку; Response Surface – поверхня відгуку; Topological Map – топологічна карта; Cluster Diagram – діаграма кластерів; Code Generation – генерація коду.

Вибір вхідних змінних – це винятково важливий етап при побудові нейронної мережі. Перед тим, як безпосередньо починати працювати з пакетом Statistica Neural Networks, має сенс зробити попередній відбір змінних, використовуючи при цьому свої знання в предметній області та стандартні статистичні критерії. Потім, уже засобами пакета Statistica Neural Networks можна буде спробувати різні комбінації вхідних змінних.

У пакеті Statistica Neural Networks є можливість «ігнорувати» деякі змінні, так що отримана мережа не буде використовувати їх як входи. Можна по черзі

експериментувати з різними комбінаціями входів, будуючи кожного разу нові варіанти мереж.

Процес побудови нейромережі (після вибору вхідних змінних) складається з таких кроків.

1. Вибрати початкову конфігурацію мережі (наприклад, один проміжний шар з числом елементів у ньому, рівним півсумі кількості входів і кількості виходів – Наставник (Network Advisor) пакета Statistica Neural Networks запропонує таку конфігурацію за замовчуванням).

2. Провести ряд експериментів з різними конфігураціями, запам'ятовуючи при цьому кращу мережу (у змісті контрольної помилки). У пакеті Statistica Neural Networks передбачене автоматичне запам'ятовування кращої мережі під час експерименту. Для кожної конфігурації варто провести кілька експериментів, щоб не одержати помилковий результат через те, що метод навчання потрапив у локальний мінімум цільової функції.

3. Якщо в черговому експерименті спостерігається недонавчання (мережа не видає результат прийнятної якості), спробувати додати додаткові нейрони в проміжний шар (шари). Якщо це не допомагає, спробувати додати новий проміжний шар.

4. Якщо має місце перенавчання (контрольна помилка стала зростати), спробувати видалити кілька схованих елементів (а можливо і шарів).

Багаторазове повторення евристичних експериментів у кращому випадку є досить утомливим, і тому в пакет Statistica Neural Networks включений спеціальний метод автоматичного пошуку.

Автоматичний конструктор мережі – *Automatic Network Designer* проведе експерименти з різним числом схованих елементів, для кожної спробної архітектури мережі виконає кілька прогонів навчання, відбираючи при цьому найкращу мережу по показнику контрольної помилки з виправленням на розмір мережі. В Автоматичному конструкторі мережі реалізовані складні методи пошуку, у тому числі метод «імітації відпалу» (simulated annealing), за допомогою яких можна

перепробувати сотні різних мереж, виділяючи з них особливо перспективні, або швидко знаходити «грубе і просте» рішення.

У задачі класифікації мережа повинна віднести кожне спостереження до одного з декількох класів (чи, у більш загальному випадку, оцінити імовірність приналежності спостереження до кожного з класів). У пакеті Statistica Neural Networks для класифікації використовуються номінальна вихідна змінна – різні її значення відповідають різним класам.

У пакеті Statistica Neural Networks класифікацію можна здійснювати за допомогою мереж наступних типів: багатошарового персептрона, радіально-базисних функцій, мережі Кохонена, імовірнісної нейромережі та лінійної мережі. Єдина з мереж пакета Statistica Neural Networks, не призначена для задач класифікації, – це узагальнено-регресійна мережа.

Номінальні змінні представляються в пакеті Statistica Neural Networks в одному з двох видів (перший з них підходить тільки для змінних із двома значеннями): 1) бінарному (два стани) та 2) один-з-N. При бінарному поданні змінній відповідає один вузол мережі, при цьому значення 0,0 означає активний стан, а 1,0 – неактивний. При кодуванні 1-з-N на кожен стан виділяється один елемент, так що кожен конкретний стан представляється як 1,0 у відповідному елементі та 0,0 у всіх інші.

Номінальні вхідні змінні в пакеті Statistica Neural Networks можуть бути перетворені одним з цих методів як на етапі навчання мережі, так і при її роботі. Цільові вихідні значення для елементів, що відповідають номінальним змінним, також легко визначаються під час навчання. Більш великі зусилля вимагаються для того, щоб за результатами роботи мережі визначити вихідний клас.

Кожний з вихідних елементів буде містити числові значення в інтервалі від 0,0 до 1,0. Щоб упевнено визначити клас по наборі вихідних значень, мережа повинна вирішити, чи «досить близькі» вони до нуля або одиниці. Якщо такої близькості не спостерігається, клас вважається «невизначеним».

Крім того, у пакеті Statistica Neural Networks для інтерпретації вихідних значень використовуються довірчі рівні (пороги прийняття і відкидання). Ці граничні значення можна корегувати, щоб змусити мережу бути більш, чи, навпаки, менш

«рішучою» при оголошенні класу. Схеми тут несильно розрізняються для випадків бінарного та 1-з-N подань.

Бінарне подання. Якщо вихідне значення елемента перевищує поріг прийняття, то вибирається клас 1,0. Якщо вихідне значення лежить нижче порога відкидання, вибирається клас 0,0. Якщо вихідне значення лежить між порогами, клас вважається не визначеним.

Один-з-N. Визначений клас вибирається тільки в тому випадку, якщо значення відповідного вихідного елемента вище порога прийняття, а всіх інших вихідних елементів – нижче порога відкидання. Якщо ж дана умова не виконана, клас не визначається.

При кодуванні методом 1-з-N є одна особливість. На перший погляд здається, що «найбільш рішучою» буде мережа з порогами прийняття і відкидання, рівними 0,5. Це дійсно так для бінарного кодування, але вже не так для кодування 1-з-N. Можна зробити так, щоб поріг прийняття був нижче порога відкидання, і найбільш рішучою буде мережа, у якої поріг прийняття 0,0 , а поріг відкидання 1,0. При такому, на перший погляд, дивному настроюванню мережі елемент із найвищим рівнем активації буде визначати клас поза залежністю від того, що відбувається в інших елементах.

Схема дії *методу визначення класу* в пакеті Statistica Neural Networks полягає у виконанні таких кроків.

1. Вибирається елемент із найвищим вихідним сигналом. Якщо його вихідний сигнал вище або дорівнює порогові прийняття, а вихідні сигнали всіх інших елементів нижче порога відкидання, то як відповідь видати клас, обумовлений цим елементом.

2. При порозі прийняття 0,0 вихідний сигнал елемента, що виграв, завжди буде прийнятий, а при порозі відкидання 1,0 всі інші елементи неминуче будуть відкинуті, і тому метод зводиться до простого вибору елемента, що виграв. Якщо ж обоє граничних значення – прийняття та відкидання – установити на рівні 0,5, мережа цілком може залишитися в нерішучості (у випадках, коли в переможця результат нижче 0.5 чи в когось із тих, що програли – вище 0,5).

Усе сказане відноситься до механізму вибору класу для більшості типів мереж. На відміну від них, мережа Кохонена діє зовсім інакше.

У мережі Кохонена елементом топологічної карти (вихідного шару), що виграв, є той, у якого найвищий рівень активації (він вимірює відстань від вхідного приклада до точки, координати якої зберігаються в елементі мережі). Деякі чи навіть усі елементи топологічної карти можуть бути позначені іменами класів. Якщо ця відстань є достатньо малою, то даний випадок зараховується до відповідного класу (за умови, що зазначене ім'я класу). У пакеті Statistica Neural Networks значення порога прийняття – це найбільша відстань, на якій приймається позитивне рішення про класифікацію спостереження. Якщо ж вхідний випадок лежить від елемента, що виграв, на більш далекій відстані або якщо елемент, що виграв, не був позначений (або якщо його мітка не відповідає жодному зі значень вихідний номінальної змінної), то випадок залишається некласифікованим. Поріг відкидання в мережах Кохонена не використовується.

У наших розглядах ми припускали, що «позитивному» рішенню про класифікацію повинне відповідати значення, близьке до 1,0, а «негативному» – близьке до 0,0. Це дійсно так, якщо на виході використовуються логістичні функції активації. Крім того, це зручно, оскільки імовірність може приймати значення від 0,0 до 1,0. Однак, у деяких ситуаціях може виявитися більш зручним використовувати інший діапазон. Іноді застосовується зворотна упорядкованість, так що позитивне рішення відповідає малим вихідним значенням. Пакет Statistica Neural Networks підтримує кожної з цих варіантів роботи.

Спочатку як границі діапазону для кожної змінної використовуються значення мінімум/середнє і максимум/стандартне відхилення. Для логістичної вихідної функції активації гарними значеннями за замовчуванням є 0,0 і 1,0. Деякі автори радять використовувати як функцію активації гіперболічний тангенс, що приймає значення в інтервалі (–1,0; +1,0) . Таким прийомом можна поліпшити навчання, тому що ця функція (на відміну від логістичної) симетрична. У цьому випадку потрібно змінити значення мінімум/середнє і

максимум/стандартне відхилення, і програма Statistica Neural Networks автоматично буде правильно інтерпретувати класи.

Зворотна упорядкованість, як правило, застосовується в двох ситуаціях. Одна з них – це мережі Кохонена, у яких вихідне значення є міра далекості, і її мале значення відповідає більшій довірі. Друга ситуація виникає при використанні матриці втрат (яка може бути додана у імовірнісну мережу на етапі її побудови або вручну – до мереж інших типів). Якщо використовується матриця втрат, то виходи мережі означають очікувані утрати від вибору того чи іншого класу, і ціль полягає в тім, щоб вибрати клас з найменшими втратами. Упорядкованість можна звернути, оголосивши вихідний сигнал не рівнем довіри, а мірою помилки. У такому випадку поріг прийняття буде нижче порога відкидання.

При виборі порогів прийняття/відкидання й оцінці здібностей мережі до класифікації дуже допомагає інформація, що міститься у вікні Статистики класифікації – *Classification Statistics*. У ньому вказується, скільки спостережень було класифіковано правильно, скільки неправильно взагалі чи не класифіковано. Крім того, видається інформація про те, скільки спостережень кожного класу було віднесено до інших класів. Усі ці дані видаються окремо для навчальної, контрольної і тестової множин.

У задачах оцінювання (регресії) метою є оцінка значення числової вихідної змінної за значеннями вхідних змінних. Задачі регресії в пакеті Statistica Neural Networks можна вирішувати за допомогою: багатошарових персептронів, мереж радіально-базисних функцій, узагальнено-регресійних і лінійних мереж. При цьому вихідні дані повинні мати стандартний числовий (не номінальний) тип.

Особливу важливість для регресії мають масштабування (шкалування) вихідних значень і ефекти екстраполяції.

Нейронні мережі найбільше часто використовуваних архітектур видають вихідні значення в деякому визначеному діапазоні (наприклад, на відрізку [0,1] у випадку логістичної функції активації). Для задач класифікації це не створює

труднощів. Однак для задач регресії зовсім очевидно, що отут є проблема, і деякі її деталі виявляються дуже тонкими.

Для початку застосуємо метод *масштабування*, щоб вихід мережі мав «прийнятний» діапазон. Найпростішою з масштабуючих функцій пакета Statistica Neural Networks є мінімаксна функція: вона знаходить мінімальне і максимальне значення змінної по навчальній множині і виконує лінійне перетворення (із застосуванням коефіцієнта масштабу і зсуву), так щоб значення лежали в потрібному діапазоні (як правило, на відрізку
[0,0; 1,0]). Якщо ці дії застосовуються до числової вихідної змінної, тобто є гарантія, що всі навчальні значення після перетворення потраплять в область можливих вихідних значень мережі, тоді мережа може бути навчена. Крім того, ми знаємо, що виходи мережі повинні знаходитися у визначених межах. Цю обставина можна вважати достоїнством або недоліком – тут ми приходимо до питань екстраполяції.

Припустимо, наприклад, що ми використовуємо багатошаровий персептрон. Застосування мінімакса по описаній вище схемі є дуже обмеженим. По-перше, функція не буде екстраполюватися, як би близько ми не знаходилися до навчальних даних (у дійсності ж, якщо ми лише мало-мало вийшли за область навчальних даних, екстраполяція цілком виправдана). По-друге, оцінка по середньому також не буде виконуватися: замість цього буде братися мінімум або максимум дивлячись по тому, зростала чи убувала в цьому місці оцінювана крива.

Щоб уникнути цих недоліків у персептроні використовується ряд прийомів.

По-перше, логістичну функцію активації у вихідному шарі можна замінити на лінійну, котра не змінює рівня активації (помітимо, що функції активації змінюються тільки у вихідному шарі; у проміжних шарах як і раніше залишаються логістичні та гіперболічні функції активації). Лінійна функція активації не насичується, і тому здатна екстраполювати (при цьому логістичні функції попередніх рівнів усе-таки припускають насичення на більш високих рівнях). Лінійні функції активації у персептроні можуть викликати визначені обчислювальні труднощі в методі

зворотного поширення, тому при його використанні варто брати малі (менш 0,1) швидкості навчання. Описаний підхід придатний для цілей екстраполяції.

По-друге, можна змінити цільовий діапазон мінімаксної масштабуючої функції (наприклад, зробити його [0,25; 0,75]). У результаті навчальні спостереження будуть відображатися в рівні, що відповідає середній частини діапазону вихідних значень. Цікаво помітити, що якщо цей діапазон обраний маленьким, і обидві його границі знаходяться поблизу значення 0,5, то він буде відповідати середній ділянці сигмоїдної кривої, на якому вона «майже лінійна», – тоді ми будемо мати практично ту ж схему, що й у випадку лінійного вихідного шару. Така мережа зможе виконувати екстраполяцію у визначених межах, а потім буде насичуватися. Усе це можна добре собі представити так: екстраполяція припустима у визначених границях, а поза ними вона буде припинятися.

Якщо застосовується перший підхід і у вихідному шарі поміщені лінійні елементи, то може вийти так, що взагалі немає необхідності використовувати метод масштабування, оскільки елементи і без масштабування можуть видавати будь-який рівень вихідних сигналів. У пакеті Statistica Neural Networks мається можливість для більшої ефективності узагалі відключити всі масштабування. Однак, на практиці повне відмовлення від масштабування приводить до труднощів в методах навчання. Дійсно, у цьому випадку різні ваги мережі працюють у різних масштабах, і це ускладнює початкову ініціалізацію ваг і (частково) навчання. Тому не рекомендується відключати масштабування, за винятком тих випадків, коли діапазон вихідних значень дуже малий і розташований поблизу нуля. Це ж розуміння свідчить на користь масштабування і при передобробці у персептронах (при який, у принципі, ваги першого проміжного шару можна легко коректувати, домагаючись цим будь-якого потрібного масштабування).

Радіальні мережі по своїй природі нездатні до екстраполяції. Чим далі вхідний приклад розташований від точок, що відповідають радіальним елементам, тим менше стають рівні активації радіальних елементів і (зрештою) тим менше буде

вихідний сигнал мережі. Вхідний приклад, розташований далеко від центрів радіальних елементів, дасть нульовий вихідний сигнал. Прагнення мережі не екстраполювати дані можна вважати достоїнством (це залежить від предметної області і думки користувача), однак убування вихідного сигналу (на перший погляд) не є достоїнством. Якщо ми прагнемо уникати екстраполяції, то для вхідних точок, що відрізняються великим ступенем новизни, у якості виходу ми, як правило, хочемо мати усереднене значення.

Для радіальних мереж у задачах регресії цього можна досягти за допомогою масштабуючої функції середнє/стандартне відхилення. Навчальні дані масштабуються таким чином, щоб середнє вихідне значення дорівнювало 0,0, а всі інші значення були б промасштабовані на стандартне відхилення вихідних сигналів. При обробці вхідних точок, що лежать поза областями дії радіальних елементів, вихідний сигнал мережі буде приблизно дорівнює середньому значенню.

Якість роботи мережі в задачі регресії можна перевірити декількома способами.

По-перше, мережі можна повідомити вихідне значення, що відповідає будь-якому спостереженню (чи якомусь новому спостереженню, що необхідно перевірити). Якщо це спостереження містилося у вихідних даних, то видається значення різниці (нев'язання).

По-друге, можуть бути отримані підсумкові статистики. До них відносяться середнє значення і стандартне відхилення, обчислені для навчальних даних і для помилки прогнозу. У загальному випадку середнє значення помилки прогнозу буде дуже близьким до нуля (зрештою, нульове середнє для помилки прогнозу можна одержати, попросту оцінивши середнє значення навчальних даних і зовсім не звертаючись до значень вхідних змінних). Найбільш важливим показником є стандартне відхилення помилки прогнозу. Якщо воно не виявиться істотно менше стандартного відхилення навчальних даних, це буде означати, що мережа працює не краще, ніж проста оцінка по середньому. Далі, у пакеті Statistica Neural Networks користувачу видається відношення стандартного відхилення помилки прогнозу до стандартного відхилення навчальних даних. Якщо

воно істотно менше одиниці (наприклад, нижче 0,1), то це говорить про гарну якість регресії. Це регресійне відношення (точніше, величину одиниця мінус це відношення) іноді називають часткою поясненої дисперсії моделі.

По-третє, можна вивести зображення поверхні відгуку. Насправді, зрозуміло, ця поверхня являє собою N+1–мірний об'єкт, де N – число вхідних елементів, а вимір, що залишився, відповідає висоті точки на поверхні. Зрозуміло, що безпосередньо візуально представити таку поверхню при N більшому двох неможливо (а реально N завжди більше двох). Проте, у пакеті Statistica Neural Networks можна виводити зрізи поверхні відгуку по будь-яким двом вхідним змінним. При цьому значення всіх інших вхідних змінних фіксуються, і змінюються тільки дві обрані. Всім іншим змінним можна додати будь-яке значення за своїм розсудом (за замовчуванням система Statistica Neural Networks візьме для них середні значення). Значення двох досліджуваних змінних можна змінювати в довільному діапазоні (за замовчуванням – у діапазоні зміни навчальних даних).

У *задачах аналізу часових рядів* метою є прогноз майбутніх значень змінної, залежної від часу, на основі попередніх значень її та/або інших змінних. Як правило, прогнозована змінна є числовою, тому прогнозування часових рядів – це окремий випадок регресії. Однак таке обмеження не закладене в пакет Statistica Neural Networks, так що в ньому можна прогнозувати і часові ряди номінальних змінних.

Звичайно чергове значення часового ряду прогнозується по деякому числу його попередніх значень (прогноз на один крок уперед у часі). У пакеті Statistica Neural Networks можна виконувати прогноз на будь-яке число кроків. Після того, як обчислене чергове передбачуване значення, воно підставляється назад і з його допомогою (а також попередніх значень) виходить наступний прогноз – це називається проекцією часового ряду. У пакеті Statistica Neural Networks можна здійснювати проекцію часового ряду і при покроковому прогнозуванні. Зрозуміло, що надійність такої проекції тим менше, чим більше кроків уперед ми намагаємося пророчити. У випадках, коли потрібно зовсім

визначена дальність прогнозу, розумно буде спеціально навчити мережу саме на таку дальність.

У пакеті Statistica Neural Networks для рішення задач прогнозу часових рядів можна застосовувати мережі всіх типів (тип мережі повинний підходити, у залежності від задачі, для регресії чи класифікації). Мережа конфігурується для прогнозу часового ряду установкою параметрів Часове вікно – Steps та Обрій – Lookahead.

Параметр Часове вікно задає число попередніх значень, які варто подавати на вхід, а параметр Обрій указує, як далеко потрібно будувати прогноз. Кількість вхідних і вихідних змінних може бути довільною. Однак, найчастіше в якості вхідної й одночасно (з урахуванням обрію) вихідної виступає єдина змінна.

При конфігуруванні мережі для аналізу часових рядів змінюється метод передобробки даних (витягаються не окремі спостереження, а їхні блоки), але навчання і робота мережі відбуваються точно так само, як і в задачах інших типів.

6.3 Моделювання нейронних мереж засобами бібліотек мови Python

Мова *Python* є однією з найбільш популярних мов програмування в останнє десятиріччя. Завдяки безкоштовним засобам розробки та потужним бібліотекам Python знаходить широке використання при розробці прикладного програмного забезпечення та вирішенні наукових задач. Зокрема велика кількість потужних бібліотек Python для моделювання нейронних мереж та розпізнавання образів дозволяє вирішувати задачі штучного інтелекту та інтелектуального аналізу даних.

Бібліотека NeuroLab (https://pythonhosted.org/neurolab/) – це бібліотека для Python, що містить моделі та методи навчання нейромереж з інтерфейсом, подібним до Neural Network Toolbox пакету MATLAB.

Інсталяція бібліотеки здійснюється одним зі способів:

– використовуючи setuptools:

```
easy_install neurolab
```

– використовуючи pip:

```
pip install neurolab
```

– використовуючи python:

```
python setup.py install
```

Модуль net – модуль, що містить базові архітектури нейромереж.

Тип мережі	Функція створення	Кількість шарів	Підтримка функцій навчання	Функція помилки
Одношаровий персептрон	newp	1	train_delta	SSE
Багатошаровий персептрон (БНМ)	newff	≥ 1	train_gd, train_gdm, train_gda, train_gdx, train_rprop, train_bfgs, train_cg	SSE
Конкуруючий шар (SOM)	newc	1	train_wta, train_cwta*	SAE
LVQ	newlvq	2	train_lvq	MSE
Мережа Елмана	newelm	≥ 1	train_gdx	MSE
Мережа Хопфілда	newhop	1	–	–
Мережа Хеммінга	newhem	2	–	–

`neurolab.net.newp(minmax, cn, transf)`– створення одношарового персептрона. Параметри: minmax – список списків, які містять мінімальне та максимальне значення відповідної вхідної ознаки, cn – кількість вихідних нейронів, transf – функція активації. Повертає мережу net.

```
>>> # створення мережі з двома входами та 10 нейронами
>>> net = newp([[-1, 1], [-1, 1]], 10)
```

`neurolab.net.newff(minmax, size, transf=None)` – створення багатошарового персептрона (БНМ). Параметри: minmax – список списків, які містять мінімальне та максимальне значення відповідної вхідної ознаки, size – перелік, довжина якого відповідає кількості шарів, не враховуючи вхідний шар, елементи якого відповідають кількості нейронів відповідного шару, transf – функція активації. Повертає мережу net.

```
>>> # створення нейромережі з двома входами
>>> # діапазон для кожного входу: [-0.5, 0.5]
>>> # 3 нейрони у прихованому шарі, 1 вихідний нейрон
>>> # 2 шари, включаючи прихований та вихідний шари
>>> net = newff([[-0.5, 0.5], [-0.5, 0.5]], [3, 1])
```

```
>>> net.ci
2
>>> net.co
1
>>> len(net.layers)
2
```

`neurolab.net.newhop(target, transf=None, max_init=10, delta=0)` – створює рекурентну мережу Хопфілда. Параметри: target – масив цільових шаблонів, transf – функція активації, max_init – максимальна кількість рекурентних ітерацій, delta – мінімальна різниця між двома виходами для зупинення рекурентного циклу. Повертає нейромережу net.

```
>>> net = newhop([[-1, -1, -1], [1, -1, 1]])
>>> output = net.sim([[-1, 1, -1], [1, -1, 1]])
```

`neurolab.net.newhem(target, transf=None, max_iter=10, delta=0)` – створення рекурентної мережі Хеммінга з двома шарами. Параметри: target – масив з навчальними шаблонами, transf – функція активації вхідного шару, max_init – максимальна кількість рекурентних ітерацій, delta – мінімальна різниця між двома виходами для зупинення рекурентного циклу. Повертає мережу net.

```
>>> net = newhem([[-1, -1, -1], [1, -1, 1]])
>>> output = net.sim([[-1, 1, -1], [1, -1, 1]])
```

`neurolab.net.newelm(minmax, size, transf=None)` – створення рекурентної мережі Елмана. Параметри: minmax – перелік мінімальних та максимальних значень для входів, size – список довжиною, що дорівнює кількостів шарів, без урахування вхідного шару, елементами якого є кількості нейронів відповідного шару. Повертає мережу net.

```
>>> # один вхід, діапазон входу [-1, 1],
>>> # один вихідний нейрон,
>>> # один шар, включаючи вихідний шар
>>> net = newelm([[-1, 1]], [1], [trans.PureLin()])
>>> # встановлення ваг для усіх вхідних нейронів в 1
>>> net.layers[0].np['w'][:] = 1
>>> # встановлення порогів усіх вхідних нейронів у 0
>>> net.layers[0].np['b'][:] = 0
>>> net.sim([[1], [1] ,[1], [3]])
array([[ 1.],
       [ 2.],
       [ 3.],
       [ 6.]])
```

`neurolab.net.newc(minmax, cn)` – створення конкурентного шару (мережі Кохонена). Параметри: minmax – перелік мінімальних та

максимальних значень для входів, cn – кількість вихідних нейронів. Повертає мережу net.

```
>>> # створення мережі з двома входами та 10 нейронами
>>> net = newc([[-1, 1], [-1, 1]], 10)
```

`neurolab.net.newlvq(minmax, cn0, pc)` – створює мережу LVQ. Параметри: minmax – список списків, які містять мінімальне та максимальне значення відповідної вхідної ознаки, cn0 – кількість нейронів у вхідному шарі, pc – список процентів, сума яких має дорівнювати одиниці. Повертає мережу net.

```
>>> # створення мережі з двома входами,
>>> # двома шарами та 10 нейронами у кожному шарі
>>> net = newlvq([[-1, 1], [-1, 1]], 10, [0.6, 0.4])
```

`trans` – функції активації нейронів.

`deriv(x, y)` – похідна функції активації.

```
>>> import numpy as np
>>> f = TanSig()
>>> x = np.linspace(-5,5,100)
>>> y = f(x)
>>> df_on_dy = f.deriv(x, y) # обчислення похідної
>>> f.out_minmax  # діапазон виходу [min, max]
[-1, 1]
>>> f.inp_active  # діапазон входу [min, max]
[-2, 2]
```

`class neurolab.trans.Competitive` – функція активації змагального шару. Параметри: x – вхідний масив. Повертає y – масив бінарних елементів: 1, якщо це – найменший елемент в x, 0 – інакше.

```
>>> f = Competitive()
>>> f([-5, -0.1, 0, 0.1, 100])
array([ 1.,  0.,  0.,  0.,  0.])
>>> f([-5, -0.1, 0, -6, 100])
array([ 0.,  0.,  0.,  1.,  0.])
```

`class neurolab.trans.HardLim` – порогова функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = HardLim()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([ 0.,  0.,  0.,  1.,  1.])
```

`class neurolab.trans.HardLims` – симетрична порогова функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = HardLims()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([-1., -1., -1.,  1.,  1.])
```

`class neurolab.trans.LogSig` – функція активації логарифмічний сигмоїд. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = LogSig()
>>> x = np.array([-np.Inf, 0.0, np.Inf])
>>> f(x).tolist()
[0.0, 0.5, 1.0]
```

`class neurolab.trans.PureLin` – лінійна функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> import numpy as np
>>> f = PureLin()
>>> x = np.array([-100., 50., 10., 40.])
>>> f(x).tolist()
[-100.0, 50.0, 10.0, 40.0]
```

`class neurolab.trans.SatLin` – насичена лінійна функція активації. Параметри: x – вхідний масив. Повертає y – вихід.

```
>>> f = SatLin()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([ 0. ,  0. ,  0. ,  0.1,  1. ])
```

`class neurolab.trans.SatLinPrm(k=1, out_min=0, out_max=1)` – лінійна функція активації з параметричним виходом. Параметри: k – дійсне число, out_min, out_max – мінімальне та максимальне значення виходу, x – вхідний масив. Повертає y – вихід.

```
>>> f = SatLinPrm()
>>> x = np.array([-5, -0.1, 0, 0.1, 100])
>>> f(x)
array([ 0. ,  0. ,  0. ,  0.1,  1. ])
>>> f = SatLinPrm(1, -1, 1)
>>> f(x)
array([-1. , -0.1,  0. ,  0.1,  1. ])
```

`class neurolab.trans.SatLins` – функція активації симетрична насичена лінійна.

```
>>> f = SatLins()
>>> x = np.array([-5, -1, 0, 0.1, 100])
>>> f(x)
array([-1. , -1. ,  0. ,  0.1,  1. ])
```

`neurolab.trans.SoftMax` – функція активації Soft max.

```
>>> from numpy import floor
>>> f = SoftMax()
>>> floor(f([0, 1, 0.5, -0.5]) * 10)
array([ 1.,  4.,  2.,  1.])
```

`neurolab.trans.TanSig` – сигмоїдна функція активації гіперболічний тангенс.

```
>>> f = TanSig()
>>> f([-np.Inf, 0.0, np.Inf])
array([-1.,  0.,  1.])
```

`init` – функції ініціалізації шарів мережі.

`neurolab.init.InitRand(minmax, init_prop)` – ініціалізація заданих властивостей шару випадковими числами у заданих обмеженнях.

`neurolab.init.init_rand(layer, min=-0.5, max=0.5, init_prop='w')` –ініціалізація заданої властивості шару випадковими числами у заданих обмеженнях.

`neurolab.init.init_zeros(layer)` – встановлення усіх властивостей шару layer у нуль.

`neurolab.init.initnw(layer)` – ініціалізація ваг шару layer методом Нгуена-Уідроу.

`neurolab.init.initwb_lin(layer)` – ініціалізація лінійного простору ваг та порогів layer невипадковими значеннями.

`neurolab.init.initwb_reg(layer)` – ініціалізація ваг та порогів шару layer у діапазоні, заданому активаційною функцією.

`neurolab.init.midpoint(layer)` – встановлення вагових коефіцієнтів шару layer у центрі діапазонів входів.

`train` – методи навчання нейромереж. Типові параметри: input – значення вхідних ознак розпізнаваних екземплярів, target – цільові значення вихідних ознак для розпізнаваних екземплярів, epochs – максимально припустима кількість циклів навчання, show – період відображення поточних результатів навчання, goal – значення цільової функції, при досягненні якого навчання зупиняється, lr – крок навчання.

Навчання одношарового персептрона.

`neurolab.train.train_delta`() – дельта-правило.

Градієнтні методи навчання БНМ. Типові параметри градієнтних методів (у доповнення до розглянутих вище параметрів train): adapt – тип навчання, rr – коефіцієнт регуляризації навчання, mc – константа моменту навчання, lr_inc – коефіцієнт збільшення швидкості навчання, lr_dec – коефіцієнт зменшення швидкості навчання, max_perf_inc – максимально припустиме збільшення цільової функції, rate_dec – зменшення зміни ваг, rate_inc – приріст зміни ваг, rate_min – мінімальне значення градієнта цільової функції, rate_max – максимальне значення зміни вагових коефіцієнтів.

`neurolab.train.train_gd()` – градієнтний спуск зі зворотним поширенням помилки.

`neurolab.train.train_gdm()` – градієнтний спуск з моментом зі зворотним поширенням помилки.

`neurolab.train.train_gda()` – градієнтний спуск з адаптивним навчальним кроком.

`neurolab.train.train_gdx()` – градієнтний спуск з моментом зі зворотним поширенням помилки та адаптивним навчальним кроком.

`neurolab.train.train_rprop()` – еластичне зворотне поширення помилки.

`neurolab.train.train_bfgs()` – метод Бройдена-Флетчера–Гольдфарба-Шанно.

`neurolab.train.train_cg()` – метод спряжених градієнтів.

`neurolab.train.train_ncg()` – метод спряжених градієнтів Ньютона.

Методи навчання на основі правила "переможець отримує усе" (Winner Take All).

`neurolab.train.train_wta()` – метод "переможець отримує усе" для мережі SOM. Параметри: input – значення вхідних ознак екземплярів вибірки, epochs – максимально припустима кількість епох, show – період друку у процесі навчання, goal –значення цільової функції, при якому навчання зупиняється.

`neurolab.train.train_cwta()` – совісливий метод "переможець отримує усе" для мережі SOM. Параметри: input – значення вхідних ознак екземплярів вибірки, epochs – максимально припустима кількість епох, show – період друку у процесі навчання, goal –значення цільової функції, при якому навчання зупиняється.

Методи навчання мереж LVQ.

`neurolab.train.train_lvq()` – метод навчання LVQ1. Параметри: input – значення вхідних ознак екземплярів вибірки, epochs – максимально припустима кількість епох, show – період друку у процесі навчання, goal –значення цільової функції, при якому навчання зупиняється, lr – крок навчання, adapt – тип навчання.

`error` – функції помилки нейромережі.

`deriv(target, output)` – похідна функції помилки нейромережі.

```
>>> msef = MSE()
>>> x = np.array([[1.0, 0.0], [2.0, 0.0]])
>>> msef(x, 0)
1.25
>>> # calc derivative:
>>> msef.deriv(x[0], 0)
array([ 1.,  0.])
```

`class neurolab.error.CEE` – функція помилки кросентропія. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

`class neurolab.error.MAE` – функція помилки середнє модулів помилки. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

`class neurolab.error.MSE` – середньоквадратична функція помилки. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

```
>>> f = MSE()
>>> x = np.array([[1.0, 0.0], [2.0, 0.0]])
>>> f(x, 0)
1.25
>>> f = MSE()
>>> x = np.array([1.0, 0.0])
>>> # calc derivative:
>>> f.deriv(x, 0)
array([ 1.,  0.])
```

`class neurolab.error.SAE` – функція помилки суми модулів. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

`class neurolab.error.SSE` – функція сумарної квадратичної помилки. Параметри: target – масив цільових значень, output – масив розрахованих мережею вихідних значень. Повертає: v – значення помилки.

Бібліотека Keras (https://keras.io/) є потужним засобом для моделювання нейронних мереж мовою Python.

За замовчуванням Keras використовує бібліотеку Tensorflow для маніпуляції тензорами. Тому перед інсталяцією Keras потрібно встановити бібліотеку Tensorflow.

Інсталяція бібліотеки Keras здійснюється одним із двох способів:

– з PyPI (рекомендується):

```
sudo pip install keras
pip install keras
```

– з джерела GitHub:

```
git clone https://github.com/keras-team/keras.git
cd keras
sudo python setup.py install
```

Розглянемо найважливіші методи бібліотеки Keras.

Метод `compile(optimizer, loss=None, metrics=None, loss_weights=None, sample_weight_mode=None, weighted_metrics =None, target_tensors=None)` конфігурує модель для навчання. Як аргументи використовує: optimizer – рядок з назвою оптимізатора, loss – рядок з ім'ям функції втрат, metrics – список метрик, що обраховуються під час навчання та тестування моделі, loss_weights – необов'язковий перелік скалярних коефіцієнтів втрат, sample_weight_mode –режим визначення вагових коефіцієнтів, weighted_metrics – перелік метрик, що обраховуються та зважуються вагами вибірки або вагами класу під час навчання та тестування, target_tensors – визначення керованих параметрів навчання.

Метод `fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch= None, validation_steps=None, validation_freq=1, max_queue_size=10, workers=1, use_multiprocessing=False)` навчає модель для заданої кількості циклів навчання (епох – проходів вибірки). Як аргументи використовує: x – вхідні дані, y – цільові дані, batch_size – розмір пакету (кількість екземплярів на оновлення градієнта), epochs – кількість циклів навчання моделі на усій вибірці даних, verbose – режим відображення ходу навчання, callbacks – список викликів, validation_split – доля навчальних даних, що використовуються для перевірки моделі, validation_data – дані для обрахунку функції втрат та метрик наприкінці кожної епохи навчання, shuffle – регулювання перемішування навчальних даних перед кожною епохою навчання, class_weight – ваги класів при розрахунку функції втрат, sample_weight – ваги екземплярів, враховувані при визначенні функції втрат, initial_epoch – епоха, з якої починається навчання, steps_per_epoch – кількість пакетів (підвибірок) екземплярів, що обробляються на одній епосі, validation_steps – кількість пакетів (підвибірок) екземплярів для перевірки моделі перед зупиненням, validation_freq – кількість епох навчання, що виконуються до запуску перевірки, max_queue_size – максимальний розмір черги генератора, workers – максимальна кількість процесів, що можуть прискорюватися при використання потоків на основі процесів, use_multiprocessing – прапорець

використання потоків. Повертає об’єкт History, властивість якого History.history – це запис значень втрат навчання та метрик навчання на успішних епохах, а також значень втрат та метрик тестування моделі.

Метод `predict(x, batch_size=None, verbose=0, steps= None, callbacks=None, max_queue_size=10, workers=1, use_multiprocessing=False)` генерує передбачення (оцінку) виходу для вхідних екземплярів. Як аргументи використовує: x – вхідні дані, що розпізнаються, batch_size – розмір пакету на оновлення градієнта, verbose – прапорець пояснень, steps – кількість кроків (пакетів екземплярів) перед завершенням циклу оцінювання, callbacks – список зворотних викликів, max_queue_size – максимальний розмір черги генератора, workers – максимальна кількість процесів, use_multiprocessing – прапорець використання багатопроцесорних потоків. Повертає масив(и) Numpy, що містять передбачення (оцінки) виходу.

Розглянемо використання засобів бібліотеки на прикладах фрагментів тексту програм, пояснення до яких наведено у коментарях (джерело: https://www.datacamp.com/).

Базовий приклад.

```
# підключення бібліотек та імпорт їхніх об'єктів
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
# генерація випадкового масиву даних
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
# створення структури моделі нейромережі
>>> model = Sequential()
>>> model.add(Dense(32, activation='relu', input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
# компіляція
>>> model.compile(optimizer='rmsprop',
loss='binary_crossentropy', metrics=['accuracy'])
# навчання моделі
>>> model.fit(data,labels,epochs=10,batch_size=32)
# запуск емуляції моделі для набору даних
>>> predictions = model.predict(data)
```

Дані мають зберігатися як масиви NumPy або як списки масивів NumPy. Бажано первинну вибірку даних розбити на навчальну та тестові вибірки.

Завантаження даних з наборів Keras.

```
>>> from keras.datasets import boston_housing, mnist,
cifar10, imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) =
boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) =
cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) =
imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Завантаження даних з мережі Інтернет.

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Попередня обробка даних.

```
# Заповнення послідовності
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
# Кодування даних
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
# Формування навчальної та тествої вибірок даних
>>> from sklearn.model_selection import train_test_split
>>> X_train5, X_test5, y_train5, y_test5 =
train_test_split(X, y, test_size=0.33, random_state=42)
# Нормалізація даних
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Визначення архітектури моделі нейромережі:

```
# Послідовна модель
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

– багатошаровий персептрон:

```
# бінарна класифікація:
>>> from keras.layers import Dense
>>> model.add(Dense(12, input_dim=8, kernel_initializer=
'uniform', activation='relu'))
>>> model.add(Dense(8, kernel_initializer='uniform',
```

```
activation='relu'))
>>> model.add(Dense(1, kernel_initializer='uniform',
activation='sigmoid'))
# багатокласова класифікація:
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,
)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
# оцінювання:
>>> model.add(Dense(64, activation='relu',
input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

– згорткова нейромережа:

```
>>> from keras.layers import Activation, Conv2D,
MaxPooling2D, Flatten
>>> model2.add(Conv2D(32, (3,3), padding='same',
input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32, (3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64, (3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64, (3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

– рекурентна нейромережа:

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,
recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Огляд моделі.

```
# форма виходу моделі
>>> model.output_shape
# зведене подання моделі
```

```
>>> model.summary()
# конфігурація моделі
>>> model.get_config()
# перелік усіх тензорів ваг моделі
>>> model.get_weights()
```
Компіляція моделей:

– багатошаровий персептрон:

```
# Бінарна класифікація
>>> model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])
# Багатокласова класифікація
>>> model.compile(optimizer='rmsprop',
loss='categorical_crossentropy', metrics=['accuracy'])
# Оцінювання
>>> model.compile(optimizer='rmsprop', loss='mse',
metrics=['mae'])
```
– рекурентна нейромережа:

```
# Визначення параметрів моделі
>>> model3.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
```
Навчання моделі.

```
>>> model3.fit(x_train4, y_train4, batch_size=32,
epochs=15, verbose=1, validation_data=(x_test4,
y_test4))
```
Обчислення якості моделі.

```
>>> score = model3.evaluate(x_test, y_test,
batch_size=32)
```
Емуляція (запуск) моделі.

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```
Збереження та завантаження моделі.

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```
Оптимізація параметрів моделі.

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
optimizer=opt, metrics=['accuracy'])
```
Раннє зупинення навчання моделі.

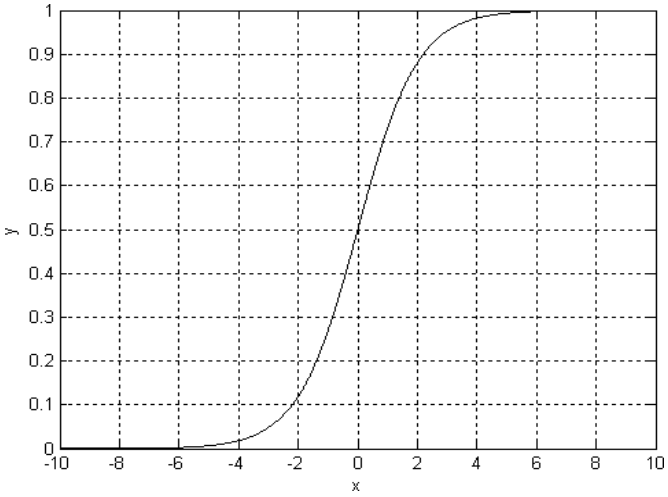
```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4, y_train4, batch_size=32,
epochs=15, validation_data=(x_test4, y_test4),
callbacks=[early_stopping_monitor])
```

6.4 Приклади виконання завдань

Приклад 1. Побудувати за допомогою пакету Matlab графік сигмоїдної функції активації.

```
% створюємо набір значень аргументу функції
% із певним кроком
x=-10:0.01:10;
% обчислюємо значення функції активації
% логістичний сигмоїд
y=logsig(x);
% будуємо графік залежності y(x)
% суцільною лінією чорного кольору
plot(x, y,'k-');
% підписуємо осі графіку
xlabel('x');
ylabel('y');
% додаємо на графік координатну сітку
grid(on);
```

У результаті виконання розробленої програми в окремій формі (графічному вікні) на екрані отримаємо такий графік.



Приклад 2. Моделювання і навчання радіально-базисних НМ.

```
%Задаємо значення ознак екземплярів навчальної
вибірки: %3 екземпляри (стовпці), 1 ознака (рядки).
x = [1 2 3];
%Задаємо значення параметра, що прогнозується,
%для 3 екземплярів навчальної вибірки.
y = [2.0 4.1 5.9];
%Створюємо і навчаємо радіально-базисну НМ
net = newrb(x, y);
%Обчислюємо по навченій мережі net значення параметра,
%що прогнозується, для екземплярів, що характеризуються
%набором значень ознак x та видаємо його на екран.
a = sim(net, x)
```

Приклад 3. Моделювання і навчання НМ Хопфілда.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%2 екземпляри (стовпці), 3 ознаки (рядки).
x = [-1    1;
     -1   -1;
      1    1];
net = newhop(x); % Створюємо і навчаємо НМ Хопфілда.
```

Приклад 4. Моделювання і навчання НМ LVQ.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%7 екземплярів (стовпці), 1 ознака (рядки).
x=[1 -2 2 0 4 -5 3];
%Задаємо номери класів для 7 екземплярів
%навчальної вибірки.
y=[1  2 1 2 1  2 1];
%Перетворюємо номери класів у внутрішній формат.
yc=ind2vec(y);
%Створюємо нейронну мережу net і визначаємо її
%топологію: діапазон зміни значень ознак визначається
%функцією minmax, кількість схованих нейронів
%(кластерів) – 4, апріорна імовірність віднесення
%екземплярів до одного класу – 0.6, до іншого – 0.4, у
%якості методу навчання НМ використовуємо метод LVQ1.
net=newlvq(minmax(x),4, [0.6 0.4], 'learnlv1');
%Задаємо максимально припустиму кількість циклів
%навчання (епох).
net. trainParam. epochs= 1000;
%Задаємо період відображення інформації про процес
%навчання на екрані в циклах (епохах) навчання.
net. trainParam. show= 100;
net. trainParam. lr=0.05; %Задаємо крок навчання.
%Навчаємо нейронну мережу net на основі навчальної
%вибірки, представленої набором значень ознак
```

```
    %екземплярів x і набором значень відповідних їм номерів
    %класів y.
    net=train(net, x, yc);
    %Обчислюємо по навченій мережі net номери класів для
    %екземплярів, що характеризуються набором значень
    %ознак x.
    a=sim(net, x);
    %Перетворюємо номери кластерів у зручний для сприйняття
    %формат і видаємо на екран.
    ac=vec2ind(a)
```

Приклад 5. Моделювання і навчання багатошарового персептрона.

```
    %Задаємо значення ознак екземплярів навчальної вибірки:
    %6 екземплярів (стовпці), 3 ознаки (рядки).
    x = [0.1 0.5 0.2 0.4 0.3 0.9;
         0.9 0.5 0.8 0.6 0.7 0.1;
         0.3 0.0 0.6 0.1 0.2 0.9];
    %Задаємо номери класів для 6 екземплярів навчальної
    %вибірки.
    y = [1   0   1   0   1   0];
    %Створюємо нейронну мережу net і визначаємо її
    %топологію: діапазон зміни значень ознак [0 1],
    %кількість ознак – 3, кількість вихідних змінних – 1, на
    %першому шарі – 2 нейрони, на другому шарі 1 – нейрон,
    %нейрони 1 і 2 шарів мають сигмоїдні функції активації
    %(logsig), для навчання мережі використовується метод
    % Левенберга-Марквардта (trainlm).
    net=newff(repmat([0 1], 3, 1),[2,1],{'logsig',
'logsig'}, 'trainlm');
    %Задаємо період відображення інформації про процес
    %навчання на екрані в циклах (епохах) навчання.
    net. trainparam. show=25;
    net. trainparam. lr= 0.01; %Задаємо крок навчання.
    %Задаємо максимально припустиму кількість циклів
    %навчання (епох).
    net. trainparam. epochs=500;
    %Задаємо максимально припустиме значення критерію
    %навчання (помилки навчання).
    net. trainparam. goal=0.01;
    %Визначаємо і запам'ятовуємо поточне значення лічильника
    %часу в змінній ct.
    ct=cputime;
    %Навчаємо нейронну мережу net на основі навчальної
    %вибірки, представленої набором значень ознак
    %екземплярів x і набором значень відповідних їм
    %номерів класів y.
    net=train(net, x, y);
    %Визначаємо поточне значення лічильника часу, віднімаємо
```

```
%від нього значення змінної ct – визначаємо час навчання
%НМ, що заносимо в змінну ct і видаємо на екран (ознака
%друку на екран – відсутність символу «;» наприкінці
%оператора).
ct=cputime-ct
%Обчислюємо по навченій мережі net номери класів для
%екземплярів, що характеризуються набором значень
%ознак x.
a=round(sim(net, x));
```

Приклад 6. Моделювання і навчання НМ SOM.

```
%Задаємо набір значень для однієї ознаки 400 екземплярів
%за допомогою генератора випадкових чисел.
x = rand(1,400);
%Створюємо нейронну мережу net і визначаємо її
%топологію: діапазон зміни значень ознак визначається
%функцією minmax, у першому шарі масив нейронів – 2x5.
net=newsom(minmax(x), [2 5]);
%Навчаємо нейронну мережу net (формуємо кластери) на
%основі навчальної вибірки, представленої набором
%значень ознак екземплярів x.
net=train(net, x);
%Обчислюємо по навченій мережі net номера кластерів,
%зіставлених нейронам вихідного шару НМ для екземплярів,
%що характеризуються набором значень ознак x.
y=sim(net, x);
%Перетворюємо номера кластерів у зручний для сприйняття
% формат і видаємо на екран.
yc=vec2ind(y)
```

Приклад 7. Моделювання і навчання мереж Елмана.

```
%Задаємо значення ознак екземплярів навчальної вибірки:
%8 екземплярів, 1 ознака.
x = [1 0 1 1 1 0 1 1]
%Задаємо значення цільової ознаки екземплярів.
y = [0 0 0 1 1 0 0 1]
%Створюємо мережу Елмана, що складається з 5 нейронів
%на першому шарі та 1 нейрона на другому шарі, які мають
%функції активації 'tansig' та 'logsig', відповідно.
net = newelm([0 1], [5 1], {'tansig', 'logsig'});
%Навчаємо мережу Елмана.
net = train(net, x, y);
%Обчислюємо по навченій мережі net номери класів для
%екземплярів, що характеризуються набором значень
%ознак x.
Y = sim(net, x);
```

Приклад 8. Використовуючи засоби пакету Matlab побудувати мережу Елмана для набору даних simpleseries_dataset.

```
% завантажуємо набір даних у вибірку <X, T>,
% де X – вхідні, а T – вихідні значення
[X, T] = simpleseries_dataset;
% створюємо мережу Елмана
% з затримками для шарів 1 і 2 та
% розміром схованого шару 10 нейронів
% задаємо як функцію навчання
% метод Левенберга-Марквардта
net = elmannet(1:2,10, 'trainlm');
% готуємо вхідні та цільові дані для моделювання
% та тренування мережі
[Xs, Xi, Ai, Ts] = preparets(net, X, T);
% запускаємо навчання мережі
net = train(net, Xs, Ts, Xi, Ai);
% емулюємо мережу
Y = net(Xs, Xi, Ai);
% визначаємо якість навчання мережі
perf = perform(net, Ts, Y)
```

Приклад 9. Написати функцію, що налагоджує ваги двошарової нейронної мережі, що містить $k1$ нейронів на першому шарі та $k2$ нейронів – на другому й навчається за вибіркою X (матриця незалежних змінних) і Y (вектор значень цільової функції). Точність апроксимації 0,01. Підрахувати кількість викликів функції помилки. У випадку, якщо не задані параметри $k1$, $k2$, X и Y, настроїти нейронну мережу, що містить 4 нейрони на першому шарі й 1 нейрон на другому шарі.

Розв'язок оформимо у вигляді двох файлів NNweights. m і ErrorPodbor. m.

Файл NNweights. m

```
function mean1 = NNweights(k1_, k2_, x1, y1)
%встановлюємо глобальні змінні
global Q k1 k2 net x y
%net - двошарова нейронна мережа
%k1 - кількість нейронів на 1-му шарі
%k2 - кількість нейронів на 2-му шарі
if (nargin~=4) % параметри не задані
k1=4; k2=1;
x=[елементимасивX]'; y=[елементимасивY];
%Тут замінити елементимасивX та елементимасивY
%конкретними числами
```

```
    end;
    if (nargin==4)        % усі параметри задані
        k1=k1_; k2=k2_; x=x1; y=y1;
    end;
    %Нормалізація X та Y
    x_min=min(x',[],1); x_max=max(x',[],1);
    for i=1:1: size(x,1)
       for j=1:1: size(x,2)
         x(i,j)=(x(i,j)-x_min(i))./abs(x_max(i)-x_min(i));
       end;
    end;
    y_max=max(y); y_min=min(y);
    y=(y-y_min)/abs(y_max-y_min);
    %Створення нейронної мережі прямого поширення, нейрони
    %якої використовують функції активації логістичний
    %сигмоїд, а для навчання застосовується метод
    %Левенберга-Марквардта (тут вказано просто як заглушка).
    net=newff(repmat([0   1],   size(x,1),1),   [k1,   k2],
{'logsig','logsig'},'trainlm');
    Q=0;
    %Запуск навчання нейромережі за допомогою
    %генетичного пошуку
    [W error] = ga(@ErrorPodbor, k1*(size(x,1)+1)+k2*(k1+1),
gaoptimset('FitnessLimit', 0.01));
    %Видаємо розраховані значення змінних на екран
    Q
    error
    net. IW{1,1}
    net. b{1,1}
    net. b{2,1}
    net. LW{2,1}
    mean1 = W'; % Присвоюємо функції значення
```

Файл ErrorPodbor. m

```
function mean1 = ErrorPodbor(w)
%встановлюємо глобальні змінні
global k1 k2 net x y Q
%net - двошарова нейронна мережа
%k1 - кількість нейронів на 1-му шарі
%k2 - кількість нейронів на 2-му шарі
%Q - номер поточної ітерації
%Встановлюємо значення ваг
net_IW=[];
for i=1:1: k1
    for j=1:1: size(x,1)
        net_IW(i, j)=w(j+size(x,1)*(i-1));
    end;
end;
```

```
net. IW{1,1}=net_IW; net. IW{1,1}; net_b11=[];
for i=1:1: k1
    net_b11(i)=w(i+size(x,1)*k1);
end;
net. b{1,1} = net_b11'; net_b21=[];
for i=1:1: k2
    net_b21(i)=w(i+size(x,1)*k1+k1);
end;
net. b{2,1} = net_b21; net_LW=[];
for i=1:1: k2
   for j=1:1: k1
    net_LW(i, j)=w(j+size(x,1)*(i-1)+size(x,1)*k1+k1+k2);
   end;
end;
net. LW{2,1} = net_LW;
%Емулюємо роботу мережі при встановлених вагах
a=sim(net, x);
%Обчислюємо помилку мережі
for i=1:1: size(y,1)
    E(i)=abs(y(i)-a(i))/(abs(y(i)));
end;
Q=Q+1;
A=[Q mean(E)];
mean1=mean(E); % результат функції - середня помилка
```

? 6.5 Контрольні питання

1. Які ви знаєте програмні засоби для моделювання НМ?
2. У якому модулі пакету Matlab містяться засоби для моделювання нейромереж?
3. Для чого призначений засіб nntool пакету Matlab?
4. Опишіть подання структури нейромережі у пакеті Matlab.
5. Функції для моделювання нейромереж у програмному та ручному режимах у пакеті Matlab.
6. Функції вирішення задач оптимізації на основі еволюційного пошуку бібліотеки Genetic Algorithm and Direct Search Toolbox пакету Matlab.
7. Засоби моделювання нейронних мереж у пакеті Statistica Neural Networks.

8. Процес побудови нейромережі пакеті Statistica Neural Networks.

9. Моделювання і навчання радіально-базисних НМ у пакеті Matlab.

10. Моделювання і навчання НМ Хопфілда у пакеті Matlab.

11. Моделювання і навчання НМ LVQ у пакеті Matlab.

12. Моделювання і навчання багатошарового персептрона у пакеті Matlab.

13. Моделювання і навчання НМ SOM у пакеті Matlab.

14. Моделювання і навчання мереж Елмана у пакеті Matlab.

15. Як впливає репрезентативність навчальної вибірки на результат побудови мережі?

16. Що таке нормалізація даних?

17. З якою метою проводиться масштабування даних?

18. Назвіть основні принципи створення навчальної вибірки.

19. Чи впливає репрезентативність навчальної вибірки на точність класифікації екземплярів тестової вибірки?

20. Чи впливає репрезентативність тестової вибірки на точність класифікації екземплярів тестової вибірки?

21. Чи впливає репрезентативність тестової вибірки на точність навчання мережі по навчальній вибірці?

22. Чи залежить якість навчання нейромереж від якості та обсягу навчальної вибірки?

23. Що таке генеральна сукупність, вибірка, екземпляр, ознака?

24. Вимоги до навчальних вибірок даних.

25. Що таке репрезентативна вибірка даних?

26. Чи повинна навчальна вибірка бути репрезентативною?

27. Чи повинна тестова вибірка бути репрезентативною?

28. Чи впливає обсяг навчальної вибірки на швидкість навчання нейромереж?

29. Модуль Neural Network Toolbox.

30. Пакет Statistica Neural Networks.

31. Моделі нейроелементів у пакеті Matlab.

32. Проаналізуйте внутрішню структуру функції ga пакету Matlab: основні змінні, параметри, методи та допоміжні функції, їх призначення та використання.

33. Опишіть основні засоби бібліотеки NeuroLab.

34. Як інсталювати бібліотеку NeuroLab?

35. Як здійснюється моделювання багатошарових нейромереж у бібліотеці NeuroLab?

36. Як здійснюється моделювання нейромереж Хопфілда у бібліотеці NeuroLab?

37. Як здійснюється моделювання нейромереж Кохонена у бібліотеці NeuroLab?

38. Як здійснюється моделювання нейромереж Хеммінга у бібліотеці NeuroLab?

39. Як здійснюється моделювання нейромереж Елмана у бібліотеці NeuroLab?

40. Опишіть основні засоби бібліотеки Keras.

41. Як інсталювати бібліотеку Keras?

42. Розглянемо найважливіші методи бібліотеки Keras.

43. Як конфігурувати модель для навчання у бібліотеці Keras?

44. Як здійснити навчання нейромережі у бібліотеці Keras?

45. Як отримати передбачення (оцінку) для розпізнаваного екземпляра нейромережею у бібліотеці Keras?

6.6 Практичні завдання

Завдання 1. Лабораторна робота

«Нейромережі прямого поширення»

*Мета роботи***:** Вивчити архітектури формального нейрона, багатошарового персептрона і радіально-базисної мережі, а також методи їхнього навчання; ознайомитися з програмними продуктами, що моделюють персептрони та радіально-базисні мережі.

Завдання до роботи.

1. Ознайомитися з рекомендованою літературою.

2. Вивчити архітектури формального нейрона, багатошарового персептрона та радіально-базисної мережі, а також методи їхнього навчання.

3. Використовуючи документацію програмних засобів (Python з бібліотекою Keras або Matlab, або Statistica Neural Networks), вивчити їх архітектуру і компоненти, призначені для моделювання і навчання нейромереж прямого поширення, введення і виведення даних, підготовки звітів (виведення і відображення результатів роботи).

4. Сформувати навчальну і тестову вибірки даних. Визначити за табл. 6.4 кількості шарів та нейронів у шарах для побудови багатошарових персептронів, а також функцію активації для завдання п. 6.

Таблиця 6.4 – Параметри нейромереж для варіантів

Номер варіанта	Багатошаровий персептрон		Назва функції активації
	Кількість шарів	Кількості нейронів у шарах	
1	2	3–1	логістична сигмоїдна
2	3	3–3–1	тангенційна сигмоїдна
3	4	3–3–3–1	логістична сигмоїдна
4	2	2–1	тангенційна сигмоїдна
5	3	2–2–1	логістична сигмоїдна
6	4	2–2–2–1	тангенційна сигмоїдна
7	2	5–1	логістична сигмоїдна
8	3	5–5–1	тангенційна сигмоїдна
9	4	5–5–5–1	логістична сигмоїдна
10	2	4–1	тангенційна сигмоїдна
11	3	4–4–1	логістична сигмоїдна
12	4	4–4–4–1	тангенційна сигмоїдна
13	2	6–1	логістична сигмоїдна
14	3	6–3–1	тангенційна сигмоїдна
15	4	6–4–3–1	логістична сигмоїдна
16	2	7–1	тангенційна сигмоїдна
17	3	7–7–1	логістична сигмоїдна
18	4	7–3–4–1	тангенційна сигмоїдна
19	2	8–1	логістична сигмоїдна
20	3	8–8–1	тангенційна сигмоїдна
21	4	8–7–6–1	логістична сигмоїдна
22	2	2–1	тангенційна сигмоїдна
23	3	2–3–1	логістична сигмоїдна
24	4	2–4–4–1	тангенційна сигмоїдна
25	2	3–1	логістична сигмоїдна
26	3	6–2–1	тангенційна сигмоїдна
27	4	7–6–3–1	логістична сигмоїдна
28	2	4–1	тангенційна сигмоїдна
29	3	9–2–1	логістична сигмоїдна
30	4	9–3–2–1	тангенційна сигмоїдна

5. Використовуючи програмний засіб для відповідних варіанту студента вхідних даних вирішити прикладну задачу на основі одношарового та багатошарового персептронів,

використовуючи різні методи навчання та різні функції активації. Вирішити ту саму задачу на основі радіально-базисної мережі.

6. Змінюючи значення кроку навчання, для одношарового та багатошарового персептронів з заданою функцією активації, що відповідає номеру варіанту студента (див. табл. 6.4) дослідити, як впливає величина кроку навчання на час навчання. Побудувати графіки залежності часу навчання персептронів від величини кроку навчання.

7. Зберегти у файлі на диску початкові параметри нейромоделей, результати їхнього навчання (матрицю ваг та структуру з зазначенням функцій активації і кількості нейроелементів у шарах) та роботи для навчальної та контрольної вибірок (помилку класифікації (оцінювання), значення на входах і виході, час навчання, час класифікації).

8. Результати виконання пп. 5–7 занести у таблицю, стовпці якої повинні мати назви: назва архітектури нейромережі, кількість шарів, кількість нейронів у шарах, функції активації нейронів у шарах, метод навчання, час навчання, час класифікації (оцінювання) навченої мережі для навчальної вибірки, помилка класифікації (оцінювання) для навчальної вибірки, час класифікації (оцінювання) навченої мережі для тестової вибірки, помилка класифікації (оцінювання) для тестової вибірки.

9. Проаналізувати отримані результати і зробити висновки про те, як впливає вид функції активації формального нейрона на час навчання і час класифікації (оцінювання), а також величину помилки навчання і класифікації (оцінювання) одношарового персептрона; як впливають вид функції активації і вид коригувального правила ваг (метод навчання) на час навчання і класифікації (оцінювання), а також величину помилки навчання та класифікації багатошарового персептрона; яка модель нейромереж прямого поширення краще підходить для вирішуваної задачі.

10. Порівняти одношаровий та багатошаровий персептрони, радіально-базисну мережу та багатошаровий персептрон за швидкістю навчання і точністю класифікації; принципами побудови архітектури.

11. Оформити звіт з роботи.

Зміст звіту.

1. Тема та мета роботи.

2. Завдання до роботи, що містить стислий опис вирішуваної практичної задачі (не більше 0,5 сторінки).

3. Короткі теоретичні відомості (не більше 2 сторінок), що містять стислий опис архітектур формального нейрона, багатошарового персептрона і радіально-базисної мережі та методів їхнього навчання.

4. Опис процесу виконання роботи (не більше 3 сторінок).

5. Тексти програм, розроблених (модифікованих) студентом.

6. Вхідні дані та результати роботи програм, узагальнюючі таблиці, графіки.

7. Висновки, що відображують результати виконання роботи та їх критичний аналіз.

Завдання 2. Лабораторна робота

«Нейромережі зі зворотними зв'язками»

Мета роботи – вивчити моделі і методи навчання нейромереж зі зворотними зв'язками, розглянути приклади їхнього практичного використання; порівняти їхні можливості з можливостями нейромереж прямого поширення; ознайомитися зі стандартними програмними засобами для моделювання мереж зі зворотними зв'язками.

Завдання до роботи.

1. Ознайомитися з рекомендованою літературою.

2. Вивчити архітектури нейромереж Хопфілда та Елмана, а також методи їхнього навчання.

3. Використовуючи документацію Python або Matlab, вивчити його архітектуру і компоненти, призначені для моделювання і навчання нейромереж зі зворотними зв'язками, введення і виведення даних, підготовки звітів (виведення і відображення результатів роботи).

4. Сформувати навчальну і тестову вибірки даних.

5. Використовуючи Python або Matlab для сформованих вибірок даних вирішити задачу побудови нейромереж Хопфілда та Елмана.

6. Зберегти у файлі на диску початкові параметри нейромоделей, результати їхнього навчання (матрицю ваг та структуру з зазначенням функцій активації і кількості

нейроелементів у шарах) та роботи для навчальної та контрольної вибірок (помилку класифікації (оцінювання), значення на входах і виході, час навчання, час класифікації).

7. Результати виконання пп. 5–6 занести у таблицю, стовпці якої повинні мати назви: назва архітектури нейромережі, кількість шарів, кількість нейронів у шарах, функції активації нейронів у шарах, метод навчання, час навчання, час класифікації навченої мережі для навчальної вибірки, помилка класифікації для навчальної вибірки, час класифікації навченої мережі для тестової вибірки, помилка класифікації для тестової вибірки.

8. Проаналізувати отримані результати і дати порівняльну характеристику мереж Хопфілда та Елмана. Дати порівняльну характеристику мереж зі зворотними зв'язками та мереж прямого поширення за швидкістю навчання і точністю класифікації; принципами побудови архітектури.

9. Оформити звіт з роботи.

Зміст звіту.

1. Тема та мета роботи.

2. Завдання до роботи, що містить стислий опис вирішуваної практичної задачі (не більше 0,5 сторінки).

3. Короткі теоретичні відомості (не більше 2 сторінок), що містять стислий опис архітектур та методів навчання нейромереж зі зворотними зв'язками.

4. Опис процесу виконання роботи (не більше 3 сторінок).

5. Тексти програм, розроблених (модифікованих) студентом.

6. Вхідні дані та результати роботи програм, узагальнюючі таблиці, графіки.

7. Висновки, що відображують результати виконання роботи та їх критичний аналіз.

Завдання 3. Лабораторна робота

«Нейромережі з латеральними зв'язками»

Мета роботи – вивчити моделі нейронних мереж з латеральними зв'язками та методи їхнього навчання; порівняти можливості нейромереж з латеральними зв'язками з можливостями нейромереж прямого поширення та можливостями мереж зі зворотними зв'язками; ознайомитися з

програмними засобами, що моделюють нейромережі з латеральними зв'язками.

Завдання до роботи.

1. Ознайомитися з рекомендованою літературою.

2. Вивчити архітектури нейромереж з латеральними зв'язками, а також методи їхнього навчання.

3. Використовуючи документацію Python або Matlab, або Statistica Neural Networks, вивчити їх архітектуру і компоненти, призначені для моделювання і навчання нейромереж з латеральними зв'язками, введення і виведення даних, підготовки звітів (виведення і відображення результатів роботи).

4. Сформувати навчальну і тестову вибірки даних.

5. Використовуючи Python або Matlab, або Statistica Neural Networks для сформованих вибірок навчити нейромережу Кохонена LVQ. При цьому окремо дослідити властивості різних методів навчання. Для тієї ж задачі сформувати кластери на основі SOM.

6. Зберегти у файлі на диску початкові параметри нейромоделей, результати їхнього навчання (матрицю ваг та структуру з зазначенням функцій активації і кількості нейроелементів у шарах) та роботи для навчальної та контрольної вибірок (помилку класифікації (оцінювання), значення на входах і виході, час навчання, час класифікації).

7. Результати виконання пп. 5–6 занести у таблицю, стовпці якої повинні мати назви: назва архітектури нейромережі, кількість шарів, кількість нейронів у шарах, функції активації нейронів у шарах, метод навчання, час навчання, час класифікації навченої мережі для навчальної вибірки, помилка класифікації для навчальної вибірки, час класифікації навченої мережі для тестової вибірки, помилка класифікації для тестової вибірки.

8. Проаналізувати отримані результати і дати порівняльну характеристику мереж з латеральними зв'язками SOFM та LVQ. Дати порівняльну характеристику мереж з латеральними зв'язками, мереж зі зворотними зв'язками та мереж прямого поширення за швидкістю навчання і точністю класифікації; принципами побудови архітектури.

9. Оформити звіт з роботи.

Зміст звіту.

1. Тема та мета роботи.

2. Завдання до роботи, що містить стислий опис вирішуваної практичної задачі (не більше 0,5 сторінки).

3. Короткі теоретичні відомості (не більше 2 сторінок), що містять стислий опис архітектур та методів навчання нейромереж з латеральними зв’язками.

4. Опис процесу виконання роботи (не більше 3 сторінок).

5. Тексти програм, розроблених (модифікованих) студентом.

6. Вхідні дані та результати роботи програм, узагальнюючі таблиці, графіки.

7. Висновки, що відображують результати виконання роботи та їх критичний аналіз.

ЛІТЕРАТУРА

Основна література

1. Інформаційні технології. Словник термінів. Частина 34. Штучний інтелект. Нейронні мережі (EN ISO/IEC 2382-34:1999, IDT) : ДСТУ ISO/IEC 2382-34-2003. – [Чинний від 2004-10-01]. – К. : Держспоживстандарт, 2005. – 20 с. – (Національний стандарт України).

2. Субботін, С. О. Нейронні мережі : навчальний посібник / С. О. Субботін, А. О. Олійник ; під заг. ред. проф. С. О. Субботіна. – Запоріжжя : ЗНТУ, 2014. – 132 с.

3. Руденко, О. Г. Штучні нейронні мережі / О. Г. Руденко, Є. В. Бодянський. – Харків : Компанія СМІТ, 2006. – 404 с.

4. Олійник, А. О. Інтелектуальний аналіз даних : навч. посіб. / А. О. Олійник, С. О. Субботін, О. О. Олійник. – Запоріжжя : ЗНТУ, 2011. – 271 с.

Додаткова література

5. Demuth, H. Neural Network Toolbox for use with Matlab: user's guide / H. Demuth, M. Beale. – Natick: Mathworks Inc, 1997. – 700 p.

6. Sumathi, S. Computational intelligence paradigms: theory and applications using Matlab / S. Sumathi, S. Paneerselvam. – Boca Raton: CRC Press, 2010. – 851 p.

7. Субботін, С. О. Неітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей : монографія / С. О. Субботін, А. О. Олійник, О. О. Олійник ; під заг. ред. С. О. Субботіна. – Запоріжжя : ЗНТУ, 2009. – 375 с.

8. Schmidhuber, J. Deep Learning in Neural Networks: An Overview // Neural Networks. – 2015. – Vol. 61. – P. 85–117.

9. Bengio, Y. Deep Learning / Y. Bengio, Y. LeCun, G. Hinton // Nature. – 2015. – Vol. 521. – P. 436–444.

10. Hochreiter, S. Long short-term memory / S. Hochreiter, J. Schmidhuber // Neural Computation. – 1997. – Vol. 9, No. 8. – P. 1735-1780.

11. Gers, F. Learning precise timing with LSTM recurrent networks / F. Gers, N. Schraudolph, J. Schmidhuber // Journal of Machine Learning Research. – 2002. – Vol. 3. – P. 115—143.

12. Generative Adversarial Networks [Electronic resource] / [I.

Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, Sh. Ozair, A. Courville, Aaron, J. Bengio]. – Access mode: https://arxiv.org/pdf/1406.2661.pdf