

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Н. О. Лащевська
С. О. Сокольський
А. С. Вєчерковська

НЕЙРОННІ МЕРЕЖІ ДЛЯ ОБРОБЛЕННЯ СИГНАЛІВ ЛАБОРАТОРНІ РОБОТИ

Навчальний посібник

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра за освітніми
програмами «Інтелектуальні технології радіоелектронної техніки»,
«Інформаційна та комунікаційна радіоінженерія», «Радіотехнічні
комп'ютеризовані системи» спеціальності «Електроніка, електронні
комунікації, приладобудування та радіотехніка»*

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

Рецензент	Зінченко О.В., д.т.н., доцент, завідувач кафедри штучного інтелекту ДУІКТ
Відповідальний редактор	Мовчанюк А.В., к.т.н., доцент, завідувач кафедри прикладної радіоелектроніки, РТФ

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 2 від 06.11.2025 р.)*

*за поданням Вченої ради Радіотехнічного факультету
(протокол № 05/2025 від 26.05.2025 р.)*

НЕЙРОННІ МЕРЕЖІ ДЛЯ ОБРОБЛЕННЯ СИГНАЛІВ ЛАБОРАТОРНІ РОБОТИ

Нейронні мережі для оброблення сигналів. Лабораторні роботи [Електронний ресурс] : навч. посіб. для студ. спеціальності G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка» / Н.О. Лащевська, С.О. Сокольський, А.С. Вечерковська; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 115 с

Методичний посібник включає теоретичні відомості, практичні завдання та приклади розв'язання задач для успішного виконання лабораторних робіт. У посібнику подано базову інформацію з відповідних тем та довідкові матеріали, які сприяють ефективному вивченню й глибшому засвоєнню навчального матеріалу. Основною метою лабораторних робіт є формування у здобувачів вищої освіти ґрунтовних практичних навичок у проєктуванні, навчанні та тестуванні штучних нейронних мереж для розв'язання прикладних задач у різних галузях радіотехніки та телекомунікацій.

УДК 004.03(226)

Реєстр. № НП 25/26-064 Обсяг 5,1 авт. арк

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК №5354 від 25.05.2017 р.

© Н.О. Лащевська, С.О. Сокольський, А.С. Вечерковська 2025
© КПІ ім. Ігоря Сікорського, 2025

Зміст

ВСТУП.....	4
МЕТА ТА ОСНОВНІ ЗАВДАННЯ ЛАБОРАТОРНИХ РОБІТ	5
ПРАВИЛА ТЕХНІКИ БЕЗПЕКИ ПІД ЧАС ВИКОНАННЯ	
ЛАБОРАТОРНИХ РОБІТ	7
Лабораторна робота №1: ДОСЛІДЖЕННЯ ПРОСТОГО НЕЙРОНА ТА ЙОГО	
АКТИВАЦІЙНИХ ФУНКЦІЙ.....	9
Лабораторна робота №2: МОДЕЛЮВАННЯ ПРОСТИХ НЕЙРОННИХ	
МЕРЕЖ З ВИКОРИСТАННЯМ MATLAB DEEP LEARNING TOOLBOX	26
Лабораторна робота №3: ПОБУДОВА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ	
РОЗПІЗНАВАННЯ ЛІТЕР У MATLAB.....	43
Лабораторна робота №4: ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ	
КЛАСИФІКАЦІЇ, ПРОГНАЗУВАННЯ ТА АНАЛІЗУ ДАНИХ У MATLAB	54
Лабораторна робота №5: ОПРАЦЮВАННЯ СТИСНУТИХ ЗОБРАЖЕНЬ ЗА	
ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ У MATLAB.....	63
Лабораторна робота №6: РОЗПІЗНАВАННЯ АУДІОФАЙЛІВ ЗА	
ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ У MATLAB.....	76
Лабораторна робота №7: ОСНОВИ НЕЙРОННИХ МЕРЕЖ У MATLAB.....	87
Лабораторна робота №8: ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ У	
MATLAB ДЛЯ КЛАСИФІКАЦІЇ, ПРОГНОЗУВАННЯ ТА ВИЯВЛЕННЯ	
АНОМАЛІЙ.....	94
Лабораторна робота №9: РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ ЗА	
ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ У MATLAB.....	104
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ	113
ДОДАТОК А. СТРУКТУРА ВИКОНАННЯ ЗВІТУ	114
ДОДАТОК Б. ТИТУЛЬНА СТОРІНКА ЗВІТУ	115

ВСТУП

Методичний посібник «*Нейронні мережі для оброблення сигналів. Лабораторні роботи*» розроблений для студентів спеціальності G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка», які навчаються за освітніми програмами «Інтелектуальні технології радіoeлектронної техніки», «Інформаційна та комунікаційна радіоінженерія» та «Радіотехнічні комп'ютеризовані системи». Посібник спрямований на формування у студентів практичних навичок використання штучних нейронних мереж для вирішення прикладних задач у галузях радіотехніки, телекомунікацій, оброблення сигналів та інформаційних технологій.

У цьому посібнику подано:

- детальні програми виконання лабораторних робіт, які передбачають поетапне виконання завдань із поясненнями;
- короткі теоретичні пояснення для кожної теми, яка розглядається в лабораторній роботі;
- перелік теоретичних матеріалів та практичні приклади для успішного виконання завдань;
- контрольні запитання для самоперевірки.

Окрему увагу приділено застосуванню нейронних мереж у різних сферах оброблення сигналів, включаючи розпізнавання образів, фільтрацію шумів, класифікацію та прогнозування.

Лабораторні роботи виконуються у програмному середовищі *Matlab*, яке є сучасним і зручним інструментом для моделювання нейромережових структур, оброблення цифрових сигналів, аналізу даних та візуалізації результатів.

Офіційну студентську версію *Matlab* можна безкоштовно отримати на офіційному сайті: <https://www.mathworks.com>.

МЕТА ТА ОСНОВНІ ЗАВДАННЯ ЛАБОРАТОРНИХ РОБІТ

Метою лабораторних робіт з навчальної дисципліни «*Нейронні мережі для оброблення сигналів*» є поглиблення теоретичних знань і формування у студентів практичних навичок проєктування, навчання, тестування та використання штучних нейронних мереж. Студенти навчатимуться самостійно обирати відповідну архітектуру мережі, налаштовувати її параметри, навчати на основі готових або самостійно сформованих наборів навчальних вибірок, а також застосовувати отримані моделі для розв’язання прикладних задач у сфері телекомунікацій та радіотехніки.

Кожна лабораторна робота спрямована на досягнення конкретних навчальних результатів та включає наступні етапи:

- формулювання мети та постановка задачі;
- вивчення відповідної структури або архітектури нейронної мережі;
- реалізація алгоритму навчання або прогнозування у програмному середовищі *MATLAB* із використанням *Neural Network Toolbox* та *Deep Learning Toolbox*;
- візуалізація та інтерпретація результатів роботи мережі у вигляді графіків, спектрограм, матриць тощо;
- аналіз ефективності побудованої моделі (класифікаційна точність, помилки прогнозу, виявлення аномалій тощо);
- узагальнення результатів і формулювання висновків щодо виконаної роботи.

Кожна лабораторна робота містить короткий теоретичний блок, який дозволяє студенту швидко повторити ключові поняття, методи і підходи, необхідні для виконання практичного завдання. Для більш ефективного засвоєння матеріалу студенту рекомендовано самостійно опрацювати відповідні теми за підручниками та навчальними посібниками, що наведені у розділі «*Список рекомендованої літератури*».

По завершенню виконання всіх лабораторних робіт студенти зможуть:

- аналізувати поставлену задачу та визначати доцільність її реалізації за допомогою штучної нейронної мережі;
- вибирати оптимальний тип та параметри нейронної мережі для конкретних задач по обробленню сигналів;
- формувати навчальні вибірки або використовувати готові набори даних для тренування моделей;
- розробляти, навчати та тестувати моделі нейронних мереж в програмному пакеті MATLAB *Neural Network Toolbox* та *Deep Learning Toolbox*;
- оцінювати ефективність розроблених моделей та коригувати їх параметри для покращення результатів.

Отриманні навички вкрай важливі під час підготовки до реальної інженерної діяльності у галузі штучного інтелекту, цифрового оброблення сигналів, автоматизації та телекомунікацій.

ПРАВИЛА ТЕХНІКИ БЕЗПЕКИ ПІД ЧАС ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Загальні положення:

- 1) До виконання лабораторних робіт допускаються лише ті студенти, які ознайомилися з інструкцією з техніки безпеки та погоджуються дотримуватись правил поведінки у комп'ютерному класі.
- 2) Робота за комп'ютерами дозволяється виключно під наглядом викладача, лаборанта, інженера або іншої уповноваженої особи.
- 3) Вхід до комп'ютерного класу сторонніх осіб можливий лише з дозволу викладача.
- 4) У перервах між заняттями аудиторія обов'язково повинна провітрюватися, при цьому студенти тимчасово залишають приміщення.

Дії перед початком роботи:

- 1) Провести зовнішній огляд робочого місця: переконатися у відсутності пошкоджень обладнання та кабелів.
- 2) Перевірити правильність підключення пристроїв до електромережі
- 3) Увімкнути комп'ютер і переконатися у його стабільній роботі.

Забороняється під час занять:

- 1) Знаходитись у комп'ютерному класі у верхньому одязі біля робочого місця.
- 2) Споживати їжу чи напої в комп'ютерному класі, що може призвести до пошкодження обладнання.
- 3) Знаходитись позаду чи з боків працюючих моніторів інших користувачів.
- 4) Підключати або вимикати дроти, проводити маніпуляції із роз'ємами чи дротами без дозволу викладача чи іншої уповноваженої особи.
- 5) Самостійно переміщувати обладнання або відкривати корпус комп'ютера.
- 6) Самовільно усувати несправності або втручатись у налаштування системи.

- 7) Вмикати чи вимикати техніку без дозволу викладача.
- 8) Закривати вентиляційні отвори комп'ютерів та моніторів.
- 9) Використовувати комп'ютери для несанкціонованих цілей: ігор, перегляду сторонніх сайтів або запуску сторонніх програм.

Обов'язки студентів під час роботи:

- 1) Дотримуватись дисципліни та порядку у класі.
- 2) Виконувати всі інструкції викладача чи технічного персоналу.
- 3) Працювати згідно з встановленим графіком.
- 4) По завершенню роботи — зберегти дані, коректно вимкнути комп'ютер та залишити робоче місце охайним.

Правила ергономіки під час роботи:

- 1) Відстань від очей до екрана монітора повинна становити 70–80 см.
- 2) Спина повинна залишатися прямою.
- 3) Плечі мають бути розслабленими та опущеними.
- 4) Ноги повинні рівно стояти на підлозі, не бути схрещені.
- 5) Руки повинні лежати на одній лінії: зап'ястя, кисті та передпліччя знаходяться в одній площині.

Дії у разі виникнення аварійних ситуацій:

- 1) Якщо під час роботи виникають збої у роботі програмного забезпечення або обладнання, потрібно негайно звернутись до викладача, лаборанта чи іншої уповноваженої особи.
- 2) У разі виявлення запаху гару, диму або незвичних звуків — негайно припинити роботу, вимкнути обладнання та повідомити відповідальну особу.

Дотримання цих правил є обов'язковим і гарантує безпеку всіх учасників навчального процесу.

Лабораторна робота №1

ДОСЛІДЖЕННЯ ПРОСТОГО НЕЙРОНА ТА ЙОГО АКТИВАЦІЙНИХ ФУНКЦІЙ

Мета роботи

Ознайомитися з принципами функціонування штучного нейрона, дослідити особливості різних функцій активації, їх вплив на результат роботи нейрона та побачити взаємозв'язок між вхідними сигналами, вагами та виходом.

Теоретичні відомості

1. Структура елементарного нейрона

Основним елементом штучної нейронної мережі є **нейрон**. Його найпростіша структура передбачає наявність скалярного вхідного сигналу p , що множиться на скалярний ваговий коефіцієнт w . Результируючий зважений вхід $w \cdot p$ є аргументом функції активації нейрона f , яка формує вихідний сигнал a .

Математично роботу такого нейрона можна описати як: $a = f(w \cdot p)$. Структура нейрона з єдиним скалярним входом показана на рисунку 1.1 а.

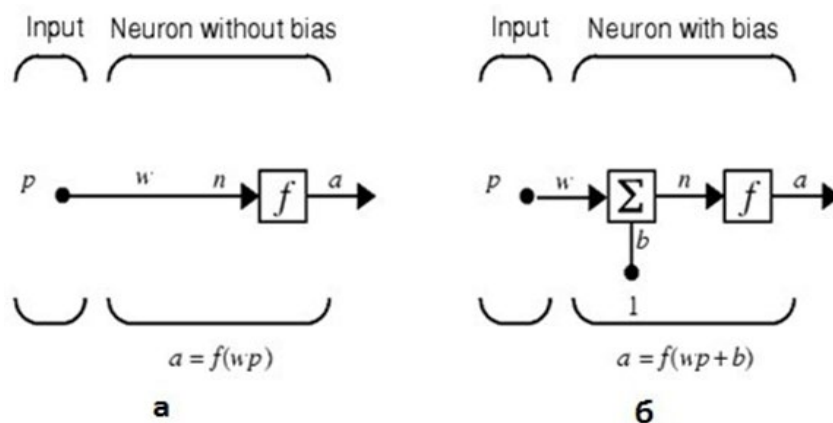


Рисунок 1.1 — Простий нейрон

Для більшої гнучкості до моделі нейрону додається скалярне **зміщення** (*bias*) b (рис. 1 б). Зміщення підсумовується з виваженим входом $w \cdot p$ і призводить до зрушення аргументу функції f на величину b . Дію зміщення можна звести до схеми зважування, якщо уявити, що нейрон має другий вхідний сигнал із значенням, рівним 1. Вхід n функції активації нейрона як і раніше залишається

скалярним і рівним сумі зваженого входу і зміщення b . Ця сума $(w \cdot p + b \cdot 1)$ є аргументом функції активації f , а виходом функції активації є сигнал a . Константи w і b є скалярними параметрами нейрона. Основний принцип роботи нейронної мережі полягає в налаштуванні параметрів нейрона з тим, щоб функціонування мережі відповідало деякій бажаній поведінці. Регулюючи ваги або параметри зміщення, можна «навчити» мережу виконувати конкретну роботу; можливо також, що мережа сама буде коригувати свої параметри, щоб досягти необхідного результату.

Рівняння нейрона зі зміщенням має вигляд:

$$a = f(w \cdot p + b \cdot 1)$$

Як уже зазначалося, зміщення b – налаштовуваний скалярний параметр нейрона, який не є входом. В цьому випадку b – вага, а константа 1, яка управляє зміщенням, розглядається як вхід і може бути врахована у вигляді лінійної комбінації векторів входу:

$$n = [w \quad b] \begin{bmatrix} p \\ 1 \end{bmatrix} = w \cdot p + b \cdot 1$$

Зміщення дозволяє змістити область активації нейрона, що покращує здатність мережі моделювати складні функціональні залежності.

Активация перетворює внутрішню суму сигналів на вихід, придатний для подальшої обробки. У звичайному режимі мережі активними стають лише частина нейронів — саме вони визначають підсумкову відповідь моделі на конкретний вхід, тоді як інші можуть лишатися «мовчазними» для даного запиту.

2. Функції активації

Функція активації (передавальна функція) f — це правило, за яким нейрон «вирішує», як реагувати на комбінацію зваженої суми входів. Вона вводить у модель потрібну нелінійність, без якої поведінка мережі зводилася б до лінійного перетворення, що недостатньо для відображення складних закономірностей у даних. Саме завдяки активаціям нейромережі здатні моделювати складні залежності й виявляти приховані патерни.

Нижче розглянуті найбільш поширені функції активації з аргументом n і виходом a .

- Одинична функція активації з жорстким обмеженням **hardlim**

Ця функція показана на рис. 1.2 і описується співвідношенням:

$$a = \text{hardlim}(n) = 1(n).$$

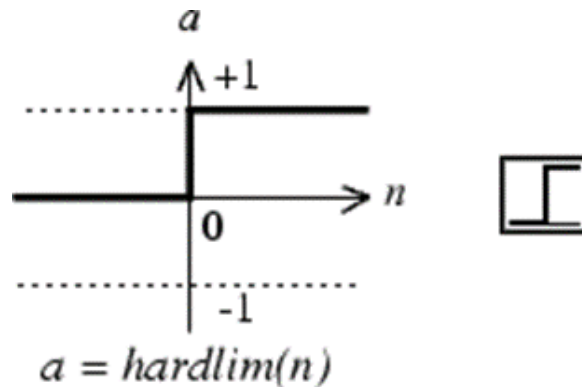


Рисунок 1.2 — Функція активації hardlim

Вона дорівнює 0, якщо $n < 0$, і дорівнює 1, якщо $n \geq 0$:

$$a = \begin{cases} 0, & \text{якщо } n < 0 \\ 1, & \text{якщо } n \geq 0 \end{cases}$$

Функція активації hardlim найчастіше використовується в логічних нейронних моделях з двома класами, коли потрібно прийняти чітке «так/ні»-рішення (наприклад, вмикати опалення, якщо температура нижча за поріг).

На рисунку 1.2 в правому верхньому куті в квадраті зображено символ, який характеризує функцію активації. Ці зображення вказують на структурних схемах нейронних мереж.

До складу пакета *Neural Network Toolbox* (NNT) входить М-функція *hardlim*, що реалізує функцію активації з жорсткими обмеженнями.

Переваги: інтуїтивна та проста у впровадженні; зручна для навчальних прикладів і логічних моделей.

Недоліки: непридатна для складних задач із багатьма класами або неперервними виходами; не забезпечує градієнтів для навчання. Якщо кілька нейронів одночасно видають «1», мережі складно однозначно визначити клас при багатокласовій постановці.

- Лінійна функція активації **purelin**

Ця функція активація (рис. 1.3) повертає пропорційний до входу результат і найчастіше описується співвідношенням:

$$a = \text{purelin}(n) = n$$

Або

$$a = w \cdot n + b$$

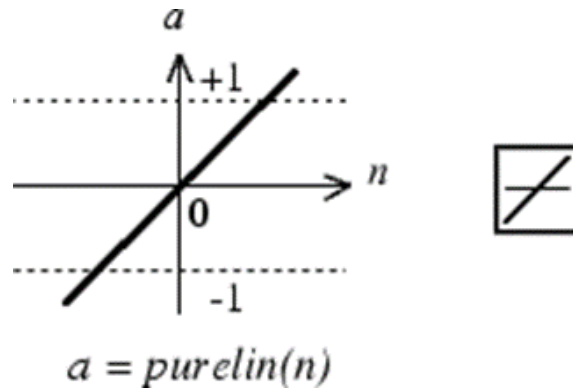


Рисунок 1.3 — Лінійна функція активації purelin

Таким чином, вихід функції purelin — це зважена сума характеристик без додаткових викривлень, що зручно під час застосування для регресійних задач (наприклад, прогноз зміни ціни з історичних даних) або як вихідний шар. Вихід такої функції не обмежений зверху/знизу.

Переваги: гранична простота; корисна в демонстраційних прикладах та на вихідному шарі регресійних моделей.

Недоліки: не створює нелінійності, тому не здатна описувати складні залежності у внутрішніх шарах.

- Функція ReLU

ReLU (Rectified Linear Unit) є однією з найуживаніших активацій у глибоких мережах і описується таким співвідношенням:

$$a = \max(0, n)$$

ReLU «обнуляє» від'ємні значення входу й пропускає додатні без змін. Приклад графіка функції зображено на рисунку 1.4.

Функція кусково-лінійна й допускає обчислення похідної, а отже — навчання методом зворотного поширення помилки (backpropagation). ReLU

широко використовують у комп'ютерному зорі: аналізуючи інтенсивності пікселів, мережа вчиться виділяти контури, текстури й інші ознаки.

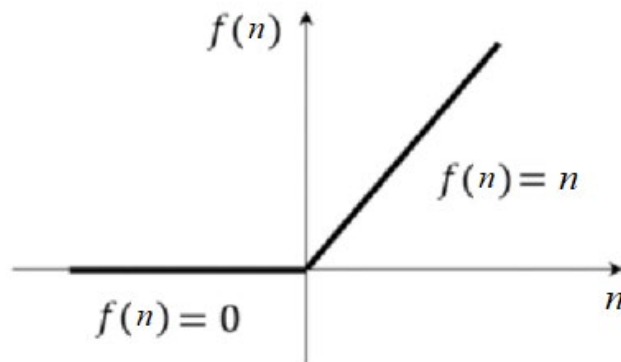


Рисунок 1.4 — Функція активації ReLU

Переваги: обчислювальна ефективність; відсутність насичення для $n > 0$; утримує нейрони деактивованими для $n < 0$, завдяки цьому пришвидшує навчання.

Недоліки: для $n < 0$ градієнт нульовий — можливе «вмирання» нейронів (вони перестають оновлюватися, а модель відповідно ефективно навчатись).

- Функція Leaky ReLU

Щоб послабити проблему «мертвих» нейронів у ReLU, застосовують Leaky ReLU (рис. 1.5). Вона описується наступним співвідношенням:

$$a = \max(\alpha n, n), \quad \alpha \approx 0,01$$

На від'ємних значеннях вхід не занулюється повністю, а пропускається з малою «пропускною» похилою. Це зберігає градієнт і дозволяє коректно навчати ваги навіть для $n < 0$.

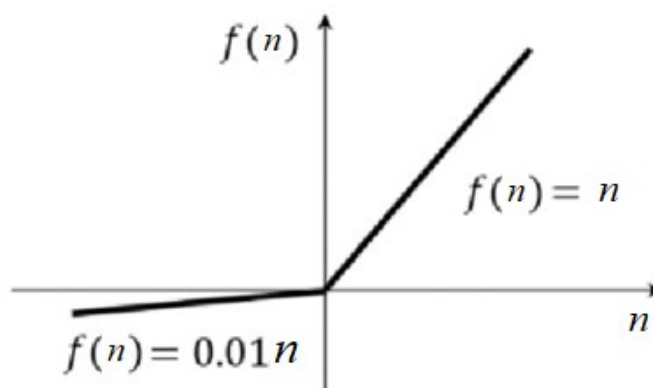


Рисунок 1.5 — Функція активації Leaky ReLU

- Сигмоїдальна логістична функція активації **logsig**

Ця функція активації належить до класу *сигмоїдальних* функцій та перетворює довільне дійсне значення (від $-\infty$ до $+\infty$) вхідного аргументу на вихідне значення в інтервалі від 0 до 1 (рис. 1.6). Для опису даної функції використовують наступне співвідношення:

$$a = \sigma(n) = \frac{1}{1 + e^{-n}}$$

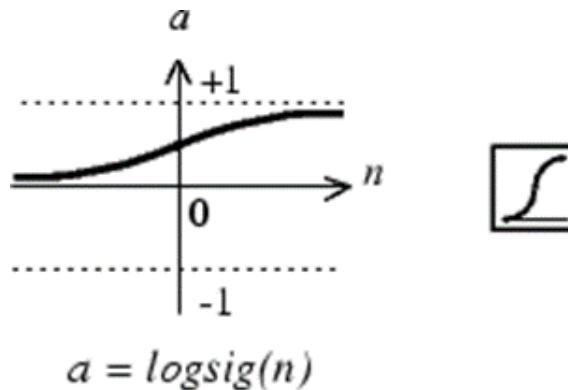


Рисунок 1.6 — Сигмоїдальна функція активації **logsig**

Функція **logsig** диференційована та часто використовується там, де вихід зручно трактувати як ймовірність (переважно у бінарній класифікації). У центральній області крива найкрутіша, тож невеликі зміни входу дають помітні зміни виходу; на далеких «хвостах» настає насичення.

Переваги: вводить нелінійність і надає ймовірнісну інтерпретацію виходу.

Недоліки: значення не центровані відносно нуля; можливі труднощі з навчанням через затухання градієнтів у зонах насичення.

В пакеті Neural Network Toolbox така сигмоїдальна функція представлена М-функцією *logsig*. Завдяки властивості до диференціювання ця функція часто використовується в мережах із навчанням на основі методу зворотного поширення помилки.

- Функція активації гіперболічний тангенс **tanh**

Гіперболічний тангенс — «нуль-центрована» альтернатива сигмоїді з діапазоном $[-1, 1]$ і описується наступним рівнянням:

$$a = \tanh(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}}$$

Завдяки центрованості середнє значення активацій у прихованих шарах близьке до нуля, що часто полегшує оптимізацію. Градієнти, як правило, більші за відповідні в сигмоїди в околі нуля, однак ефект насичення на «хвостах» також присутній.

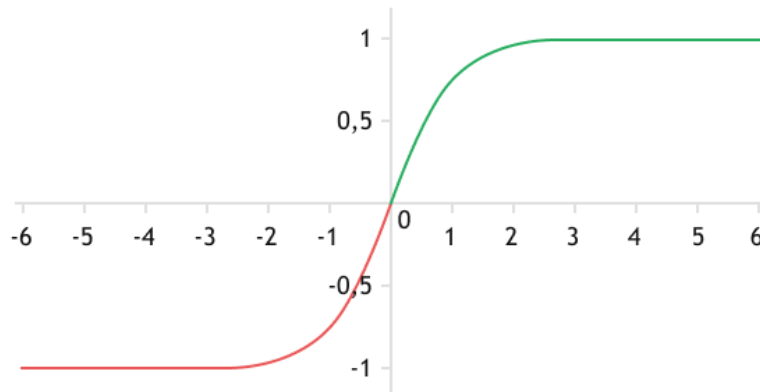


Рисунок 1.7 — Гіперболічний тангенс $\tanh$

Переваги: нульовий центр; здатність адекватно відтворювати як позитивні, так і негативні виходи.

Недоліки: схильність до затухання градієнтів при великих $|x|$.

- Функція активації Softmax

Softmax — нелінійна активаційна функція, яка перетворює вектор логітів (набір чисел) z на розподіл ймовірностей по класах (рис. 1.8). Вона описується наступним виразом:

$$a(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

де z_i — елемент вектору вхідних значень z ;

e^{z_i} — значення експоненти від z_i ;

$\sum_{j=1}^n e^{z_j}$ — значення суми всіх елементів вектору z .

Ця функція застосовується на вихідному шарі для багатокласової класифікації, коли об'єкт належить саме до одного з можливих класів

(взаємовиключні категорії) — наприклад, визначення типу об'єкта на зображенні чи теми документа.

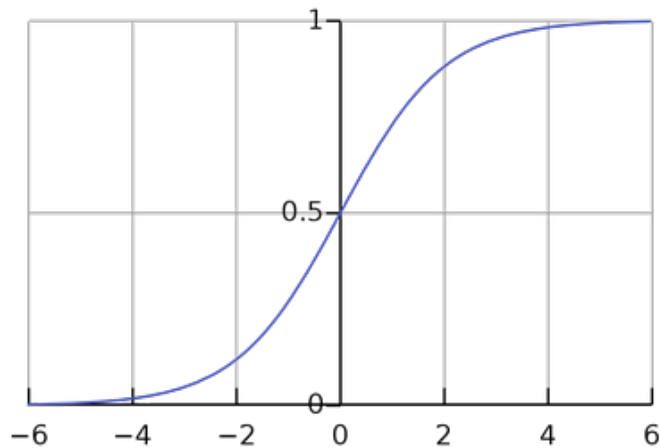


Рисунок 1.8 — Функція активації Softmax

Переваги: зручна інтерпретація у вигляді ймовірностей; дає чіткий нормований розподіл по класах.

Недоліки: не підходить для мультилейбл-класифікації (коли одночасно можливі кілька міток); при великій кількості класів окремі ймовірності можуть ставати дуже малими.

У пакет Neural Network Toolbox у MATLAB включені і інші функції активації. Використовуючи розширення інструментарію пакету або мову програмування, користувач може створювати й свої власні унікальні функції активації.

Для того, щоб правильно обрати функцію активації, необхідно визначити, який тип задачі нам потрібно розв'язати:

Східчаста: нині майже не використовується у внутрішніх шарах; припустима на виході для суто бінарних рішень.

- ✓ **Лінійна:** вихідний шар у регресії, де результат — довільне дійсне число.
- ✓ **ReLU:** стандарт для прихованих шарів завдяки швидкому навчанню.
- ✓ **Leaky ReLU:** альтернатива ReLU за наявності «мертвих» нейронів.
- ✓ **Сигмоїда:** вихідний шар для бінарної класифікації, коли потрібні ймовірності.
- ✓ **Tanh:** заміна сигмоїди, якщо потрібна нуль-центрована активація.

✓ **Softmax**: вихідний шар для багатокласових задач із взаємовиключними класами.

3. Нейрон з векторним входом

У реальних задачах нейрон має кілька входів. У таких випадках вхід нейрону описують у вигляді вектору $\mathbf{p}$ з R елементами $p_1, p_2, \dots, p_R$ (рис. 1.9). Тут кожен елемент входу множиться на вагові коефіцієнти $w_{11}, w_{12}, \dots, w_{1R}$ відповідно, далі зважені значення передаються на суматор. Їх сума дорівнює скалярному добутку вектора-рядка $\mathbf{W}$ на вектор-стовпець входу $\mathbf{p}$.

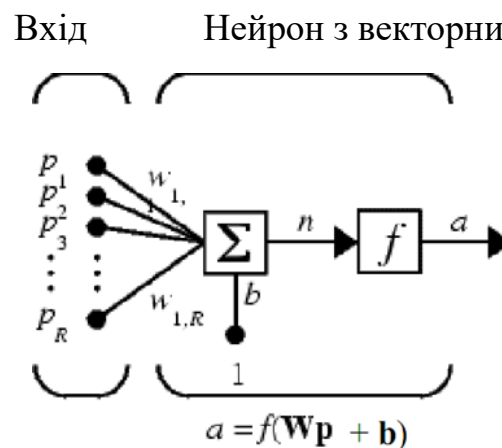


Рисунок 1.9 — Функціональна схема нейрона

Нейрон має зміщення b , яке додається до зваженої суми входів. Остаточна сума n визначається за наступною формулою:

$$n = w_{11}p_1 + w_{12}p_2 + \dots + w_{1R}p_R + b$$

Вона служить аргументом функції активації f . В нотації мови MATLAB цей вислів записується так:

$$n = \mathbf{W} * \mathbf{p} + b;$$

$$a = f(n);$$

Структура нейрона, яка показана на рис. 5, містить багато зайвих деталей. Тому при розгляді мереж, що складаються з великої кількості нейронів, для компактного зображення використовують укрупненні структурні схеми нейронів (рис. 1.10).

Кожен раз, коли використовується скорочена назва мережі, розмірність матриць вказується під іменами векторно-матричних змінних. Ця система позначень пояснює будову мережі і пов'язану з нею матричну математику.

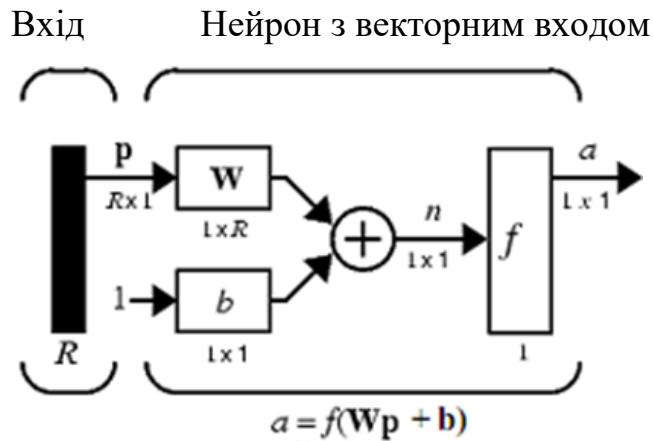


Рисунок 1.10 — Структурна схема нейрона

На укрупненій структурній схемі для позначення типу функції активації застосовуються спеціальні графічні символи, деякі з них наведені на рис. 1.11, де а – ступінчаста, б – лінійна, в – логістична функції.

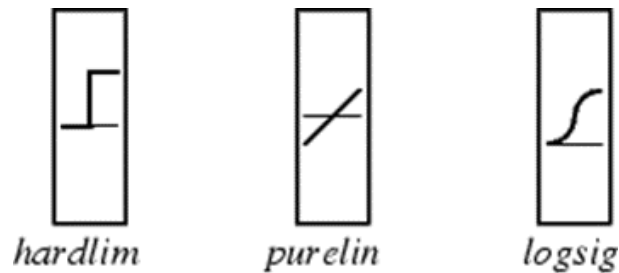


Рисунок 1.11 — Позначення функцій активації

Хід роботи

Завдання 1: Дослідити функцію активацію **'hardlim'**

1.1. Використовуючи MATLAB, вивести на екран інформацію про функцію hardlim.

```
name = hardlim('name');
inrange = hardlim('active');
outrange = hardlim('output');
```

1.2. Побудувати графік функції **'hardlim'** та її похідної **'dhardlim'**:

```
n = -5:0.1:5;
a = hardlim(n);
da = dhardlim(n);
plot(n, a, 'r'); hold on;
plot(n, da, 'c'); grid on;
legend('hardlim', 'dhardlim');
```

title('Функція hardlim та її похідна');

Програмний скрипт реалізації завдання записати у окремий М-файл під назвою **`hardlim.m`**.

Щоб запустити цей скрипт у основному файлі MATLAB з кодом виконання лабораторної роботи (рис. 1.12), виконайте в команду:

hardlim

Завдання 2: Дослідити симетричну функцію активації з жорсткими обмеженнями **hardlims** та її похідну **dhardlms**, що обумовлені наступними співвідношеннями:

$$\begin{aligned} \text{hardlims}(n) &= \begin{cases} -1, & \text{якщо } n < 0 \\ 1, & \text{якщо } n \geq 0 \end{cases} \\ \text{dhardlms}(n) &= \begin{cases} 0, & \text{якщо } n < 0 \\ 0, & \text{якщо } n \geq 0 \end{cases} \end{aligned}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати їхні графіки і розрахувати вектори виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в окремий файл під назвою **`hardlimsfile.m`**.

Завдання 3: Дослідити функцію **purelin** та її похідну **dpurelin**, що обумовлені такими співвідношеннями:

$$\text{purelin} = N;$$

$$\text{dpurelin} = 1;$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям **`purelinfile.m`**

Завдання 4: Дослідити додатну лінійну функцію активації **poslin** та її похідну **dposlin**, що обумовлені такими співвідношеннями:

$$\text{poslin}(n) = \begin{cases} 0, & \text{якщо } n < 0 \\ n, & \text{якщо } n \geq 0 \end{cases}$$

$$dposlin(n) = \begin{cases} 0, & \text{якщо } n < 0 \\ 1, & \text{якщо } n \geq 0 \end{cases}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям *`poslinfile.m`*

Завдання 5: Дослідити лінійну функцію активації з обмеженнями **satlin** та її похідну **dsatlin**, що обумовлені такими співвідношеннями:

$$satlin(n) = \begin{cases} 0, & \text{якщо } n < 0 \\ n, & \text{якщо } 0 \leq n \leq 1 \\ 1, & \text{якщо } n > 1 \end{cases}$$

$$dsatlin(n) = \begin{cases} 0, & \text{якщо } n < 0 \\ 1, & \text{якщо } 0 \leq n \leq 1 \\ 0, & \text{якщо } n > 1 \end{cases}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям *`satlinfile.m`*

Завдання 6: Дослідити симетричну лінійну функцію активації **satlins** та її похідну **dsatlins**, що обумовлені такими співвідношеннями:

$$satlins(n) = \begin{cases} -1, & \text{якщо } n < -1 \\ n, & \text{якщо } -1 \leq n \leq 1 \\ 1, & \text{якщо } n > 1 \end{cases}$$

$$dsatlins(n) = \begin{cases} 0, & \text{якщо } n < -1 \\ 1, & \text{якщо } -1 \leq n \leq 1 \\ 0, & \text{якщо } n > 1 \end{cases}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям *`satlinsfile.m`*

Завдання 7: Дослідити радіальну базисну функцію активації **radbas** та її похідну **dradbas**, що обумовлені такими співвідношеннями:

$$\begin{aligned} \text{radbas} &= e^{-2n}; \\ \text{dradbas} &= -2ne^{-2n}; \end{aligned}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям **'radbasfile.m'**

Завдання 8: Дослідити трикутну функцію активації **tribas** та її похідну **dtribas**, що обумовлені такими співвідношеннями:

$$\begin{aligned} \text{tribas}(n) &= \begin{cases} 0, & \text{якщо } n < -1 \\ 1 - \text{abs}(n), & \text{якщо } -1 \leq n \leq 1 \\ 0, & \text{якщо } n > 1 \end{cases} \\ \text{dtribas}(n) &= \begin{cases} 0, & \text{якщо } n < -1 \\ 1, & \text{якщо } -1 \leq n \leq 0 \\ -1, & \text{якщо } 0 < n < 1 \\ 0, & \text{якщо } n > 1 \end{cases} \end{aligned}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям **'tribasfile.m'**

Завдання 9: Дослідити логістичну функцію активації **logsig** та її похідну **dlogsig**, що обумовлені такими співвідношеннями:

$$\begin{aligned} \text{logsig} &= \frac{1}{(1 + e^{-n})}; \\ \text{dlogsig} &= \frac{e^{-n}}{(1 + e^{-n})^2}; \end{aligned}$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу *hardlimfile*. Новий скрипт записати в файл під ім'ям **'logsigfile.m'**

Завдання 10: Дослідити гіперболічну тангенціальну функцію активації **tansig** та її похідну **dtansig**, що обумовлені такими співвідношеннями:

$$tansig = \frac{2}{(1 + e^{-2n})} - 1;$$

$$dtansig = 1 - tansig^2 n;$$

Використовуючи MATLAB, вивести на екран інформацію про ці функції, побудувати графіки функції та її похідної, реалізувати розрахунки векторів виходу, скориставшись скриптом з М-файлу **hardlimfile**. Новий скрипт записати в файл під ім'ям **`tansigfile.m`**

Завдання 11: Для конкуруючої функції активації **compet**, яка використовується для формування ймовірних і самоорганізованих нейронних мереж, виконати наступні дії:

11.1. Вивести на екран інформацію про цю функцію за допомогою наступного скрипта:

```
name = compet('name'); % Назва функції
inrange = compet('active'); % Діапазон вхідних значень
outrange = compet('output'); % Діапазон вихідних значень

fprintf('Name: %s\n', name);
fprintf('Inrange: %s\n', inrange);
fprintf('Outrange: %s\n\n', outrange);
```

11.2. Побудувати стовпчикові діаграми для векторів входу та виходу, використовуючи шар з чотирьох нейронів:

```
N = [0; 1; - 0.5; 0.5];
A = compet (N);
subplot (2, 1, 1); % висновок у 1-е підвікно
bar (N); % – побудова стовпчикової діаграми входів;
ylabel ( 'N'); % – мітка осі ординат;
title('Вхідний вектор N');

subplot (2, 1, 2);
bar (A);
```

```
ylabel ('A'); % – мітка у 2-му подвікні  
title('Вихідний вектор A після compet');
```

11.3. Розглянуту послідовність команд оформити у вигляді скрипта в М-файл з ім'ям *'competfile.m'*.

Завдання 12: Виконати дії із 11-го завдання для конкуруючої функції активації з м'яким максимумом *softmax*, записавши при цьому новий скрипт в М-файл з ім'ям *'softmaxfile.m'*.

Завдання 13: Реалізація логічного *NOT*

Для простого нейрона з одним подвійним входом P і функції активації *hardlim* підібрати ваговий коефіцієнт W і зміщення b таким чином, щоб забезпечити інвертування вхідного сигналу, тобто заміну нуля одиницею, а одиницю нулем.

Завдання 14: Реалізація логічного *AND* та *OR*

Для нейрона з двома подвійними входами p_1 і p_2 , а також функцією активації *hardlim* підібрати вагові коефіцієнти W_{11} , W_{12} та зміщення b таким чином, щоб нейрон виконував функції логічного додавання і логічного множення.

Завдання 15: Класифікація бінарних входів

Для нейрона з двома подвійними входами p_1 і p_2 , а також функцією активації *hardlim* підібрати вагові коефіцієнти W_{11} , W_{12} та зміщення b таким чином, щоб класифікувати вхідні подвійні набори на два класи – нульовий і перший:

- а) $\{00, 01\}$ – нульовий клас, $\{10, 11\}$ – перший клас;
- б) $\{11\}$ – нульовий клас, $\{00, 01, 10\}$ – перший клас;
- в) $\{00, 11\}$ – нульовий клас, $\{01, 10\}$ – перший клас;
- г) $\{00, 11\}$ – перший клас, $\{01, 10\}$ – нульовий клас.

Завдання 16: Класифікація бінарних входів

Для нейрона з двома безперервними входами p_1 і p_2 та функцією активації **hardlim** побудувати графік поділяючої лінії, яка визначається рівнянням:

$$W_{11}p_1 + W_{12}p_2 + b = 0$$

Вважаємо, що вагові коефіцієнти W_{11} , W_{12} та зміщення b задані. Переконайтеся, що набори входів p_1 і p_2 по різні боки від поділяючої лінії належать різним класам і що не всяку множину наборів значень входів можна розділити на два класи, використовуючи нейрон розглянутого типу.

Завдання 17: За результатами виконання завдань, оформіть звіт використовуючи структуру показану у Додатку А.

Візьміть до уваги, що у цій лабораторній роботі *основний скрипт М-файлу з послідовним викликом функції реалізації завдань*, бажано оформлювати приблизно, як показано на *рисунку 1.12*.

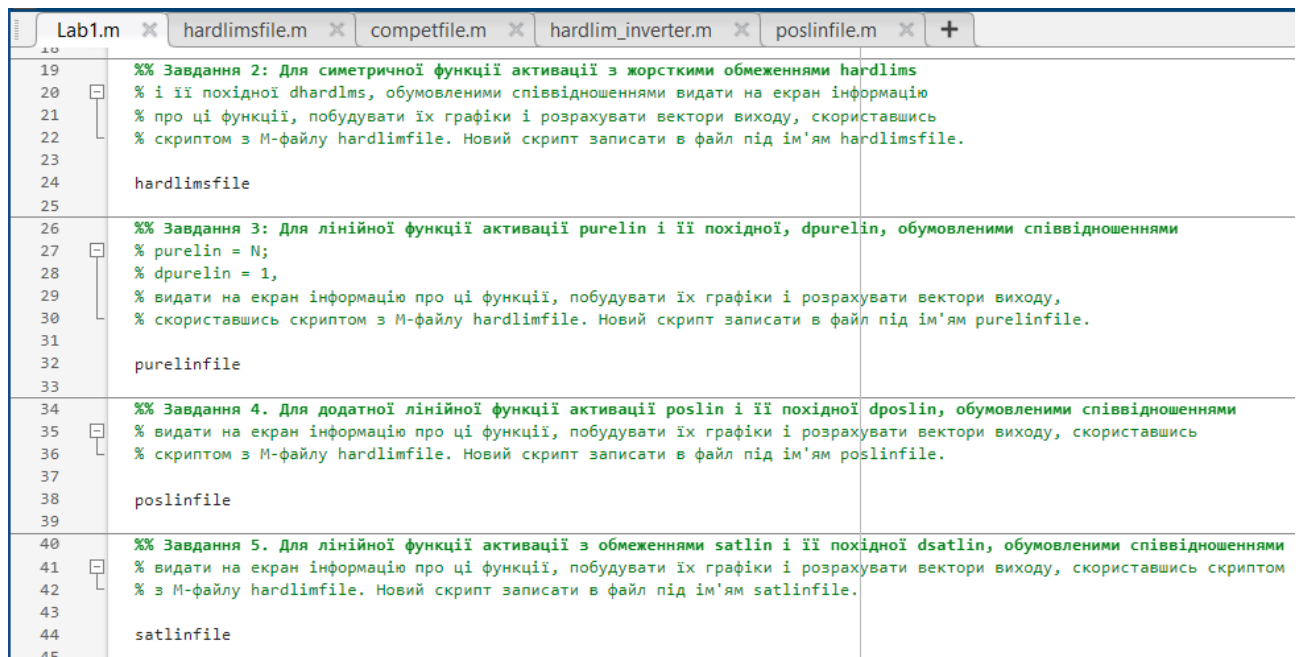


Рисунок 1.12 — Приклад оформлення основного скрипту

Представте виконану та оформлену роботу викладачеві для перевірки та захисту.

Контрольні питання

- 1) Які основні характеристики штучного нейрона?
- 2) Як впливає зміщення b на роботу нейрона?

- 3) Які типи функцій активації існують та в яких випадках вони використовуються?
- 4) Які властивості має функція `logsig`, і чому вона часто використовується у зворотному поширенні помилки?
- 5) Чим відрізняються симетричні та несиметричні функції активації?
- 6) Як інтерпретується робота нейрона з двійковими входами у задачах логічної класифікації?
- 7) Яке значення має похідна функції активації для процесу навчання нейронної мережі?
- 8) У чому полягає функціональне призначення функцій `compot` і `softmax`?
- 9) Як можна за допомогою одного нейрона реалізувати логічні операції (AND, OR, NOT)?
- 10) Що означає поняття “розділяюча лінія” в контексті класифікації вхідних даних?
- 11) Які обмеження має одношаровий нейрон з функцією `hardlim` при класифікації наборів даних?
- 12) Як впливає зміна вагових коефіцієнтів w на результат функціонування нейрона?

Лабораторна робота №2

МОДЕЛЮВАННЯ ПРОСТИХ НЕЙРОННИХ МЕРЕЖ З ВИКОРИСТАННЯМ MATLAB DEEP LEARNING TOOLBOX

Мета роботи

Ознайомлення з базовими принципами створення, навчання та аналізу штучних нейронних мереж у середовищі MATLAB із використанням Deep Learning Toolbox, навчитися використовувати інструменти toolbox для моделювання найпростіших типів нейронних мереж — персептрона та багатошарових мереж прямого поширення.

Теоретичні відомості

Нейронні мережі — це клас обчислювальних моделей, які імітують функціонування людського мозку для вирішення складних задач. Основною ідеєю є здатність системи самонавчатися на основі вхідних даних і адаптуватися до нової інформації без необхідності явного програмування всіх правил. Такі мережі широко застосовуються у задачах класифікації, регресії, обробки зображень і сигналів, розпізнавання образів, прогнозування часових рядів, виявлення аномалій, а також в інших напрямках машинного навчання та штучного інтелекту.

У середовищі MATLAB розробка нейронних мереж здійснюється за допомогою спеціалізованого середовища — **Deep Learning Toolbox**, який забезпечує функціональність для створення, налаштування, навчання та візуалізації штучних нейронних мереж (рис. 2.1). Toolbox підтримує два підходи:

- 1) *Програмний* — опис мережі та навчання програмним кодом (визначення шарів, графа мережі, параметрів навчання, запуск тренування).
- 2) *Інтерактивний* — через графічні засоби (зокрема Deep Network Designer) для побудови блок-схеми, вибору шарів, активацій, розмірностей, а також для запуску навчання та моніторингу процесу.

Deep Network Designer надає можливість:

- створювати й змінювати архітектуру як послідовність/граф шарів, обирати тип активаційної функції, налаштовувати кількість нейронів тощо;
- імпортувати дані після побудови архітектури (зображення, таблиці ознак, послідовності), налаштувати параметри навчання: обирати розмір міні-пакета, кількість епох, метод оптимізації, швидкість навчання;
- переглядати графіки навчання (зміна похибки/точності у часі);
- експортувати навчений варіант мережі в робочий простір або згенерувати MATLAB-код для відтворення експерименту.

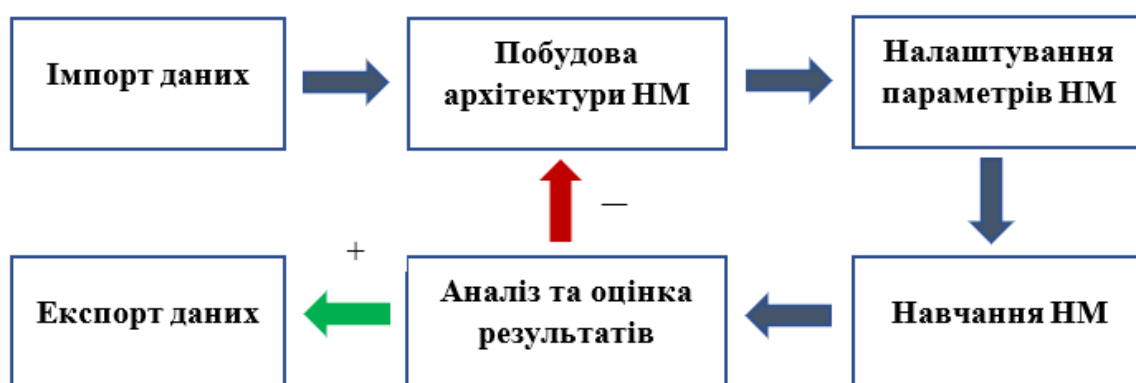


Рисунок 2.1 — Блок-схема робочого процесу Deep Network Designer

Розглянемо основні типи нейронних мереж, які вивчаються у цій лабораторній роботі:

1. **Перцептрон** — найпростіший тип штучної нейронної мережі, який складається лише з вхідного та вихідного шару. Він здатен реалізовувати лише лінійно роздільні функції, такі як логічна функція «І» або «АБО». Для нелінійно роздільних функцій, таких як «XOR», необхідна архітектура з прихованими шарами (багатошарова).

2. **Багатошарова нейронна мережа (Multilayer Perceptron, MLP)** — містить один або кілька прихованих шарів із нелінійними активаціями функціями (рис.2.2). Це дозволяє вирішувати більш складні задачі, включаючи узагальнену апроксимацію неперервних функцій різної складності на компактних множинах (за наявності достатньої кількості нейронів), що робить MLP універсальним інструментом класифікації й регресії.

3. *Мережі для регресії* — архітектури нейронних мереж, які використовуються для прогнозування неперервних значень. Вони мають відповідну структуру вихідного шару (найчастіше один нейрон без обмежувальної активації і regressionLayer) та відповідні функції втрат (наприклад, середньоквадратична помилка).

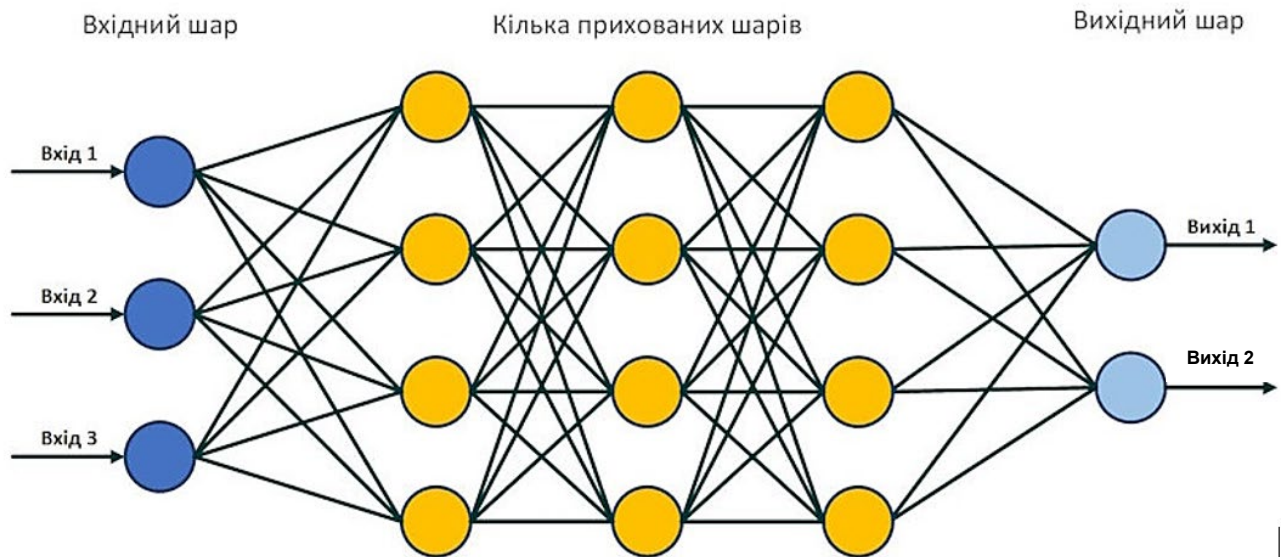


Рисунок 2.2 — Узагальнена структура багатошарової нейронної мережі (MLP)

Архітектурні елементи та типові шари.

У MATLAB архітектура мережі визначається як послідовність або граф шарів. Основними елементами є:

- ✓ **Вхідний шар**, що задає форму даних: розмірність ознак та розмір зображення.
- ✓ **Повнозв'язні шари** (*fullyConnected*) реалізують афінне перетворення $Wp+b$. Кількість нейронів визначає розмір проміжного представлення.
- ✓ **Активаційні шари** вмикають нелінійність. Найчастіше на практиці використовують ReLU, tanh або сигмоїду (для специфічних випадків).
- ✓ **Нормалізація/регуляризація** — batchNormalization та dropout шари.
- ✓ Комбінацією **вихідних шарів** для багатокласової класифікації є softmax + класифікаційний шар (рис. 2.3); для бінарної класифікації — аналогічно, з відповідною інтерпретацією виходу; для регресії ж це поєднання лінійного вихіду та regressionLayer.

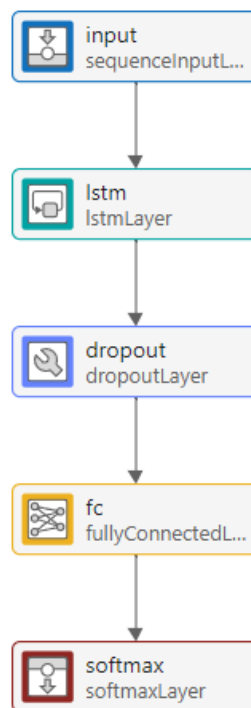


Рисунок 2.3 — Приклади конфігурацій НМ: класифікація з шаром softmax
Підготовка даних.

Якість навчання значною мірою залежить від коректної підготовки даних, що включає наступні етапи:

1) Навчальні, валідаційні та тестові дані **розбивають на підвибірки** зазвичай у співвідношенні 70/15/15 або 80/10/10. Валідація використовується для контролю перенавчання й підбору гіперпараметрів, тест — для фінальної оцінки.

2) **Масштабування ознак** — стандартизація (нульове середнє, одинична дисперсія) або нормування діапазону; для зображень — приведення діапазону інтенсивностей і розміру.

3) **Балансування класів** — рівномірна вибірка класів у міні-пакетах, аугментація менш представлених класів та за потреби зважування втрат.

4) **Аугментація** для зображень та послідовностей — повороти, зсуви, віддзеркалення, масштабування з метою підвищення узагальнювальної здатності без додаткового збору даних.

5) Встановлення генератора випадкових чисел для **фіксації випадковостей** та відтворюваності таких сценаріїв (однакове розбиття та ініціалізація у повторних запусках).

Навчання та оптимізація.

Навчання нейронної мережі зводиться до мінімізації функції втрат L на навчальній вибірці. Вектор параметрів (W, b) оновлюється ітераційно методами оптимізації. Найчастіше в ЛР2 використовується *Adam*, який змінює ваги і зміщення в нейронах для зменшення похибки між реальним та передбаченим значенням. Процес навчання триває кілька ітерацій (epoch), після чого модель здатна прогнозувати або класифікувати нові приклади.

Середовище MATLAB дозволяє ретельно та поглиблено налаштовувати основні гіперпараметри навчання:

- **Кількість epoch** — кількість повних проходів по сценарію навчання. Якщо їх буде замало, то мережа буде у стані недонавчання, а якщо забагато — перенавчання (зростає розрив між тренувальною та валідаційною кривими).
- **Розмір міні-пакета** (Mini-Batch Size) впливають на стабільність градієнта та ефективність обчислень. Великі пакети згладжують градієнт і пришвидшують обчислення, але можуть погіршити узагальнення; малі — додають «шуму» оновленням і інколи покращують результат.
- **Переупорядкування** (Shuffle) — перетасування даних щоепохи для зменшення кореляції між послідовними міні-пакетами.
- **План зміни швидкості навчання** (LR scheduler) — покрокове зменшення (piecewise) та експоненційне спадання кроків оновлення.
- **Обмеження градієнта** (gradient clipping) запобігає «вибуху» градієнтів.
- **Валідація під час навчання** — періодична оцінка на валідаційному наборі, відстеження найкращого стану моделі, рання зупинка (early stopping) за відсутності покращень (рис.2.4).

Типовими проблемами під час навчання є:

- **Недонавчання** — коли і тренувальна, і валідаційна помилки надто високі.

Для рішення цієї проблеми потрібно збільшити кількість нейронів/шарів, подовжити час та підвищити швидкість навчання (але дуже обережно), а також поліпшити підготовку ознак.

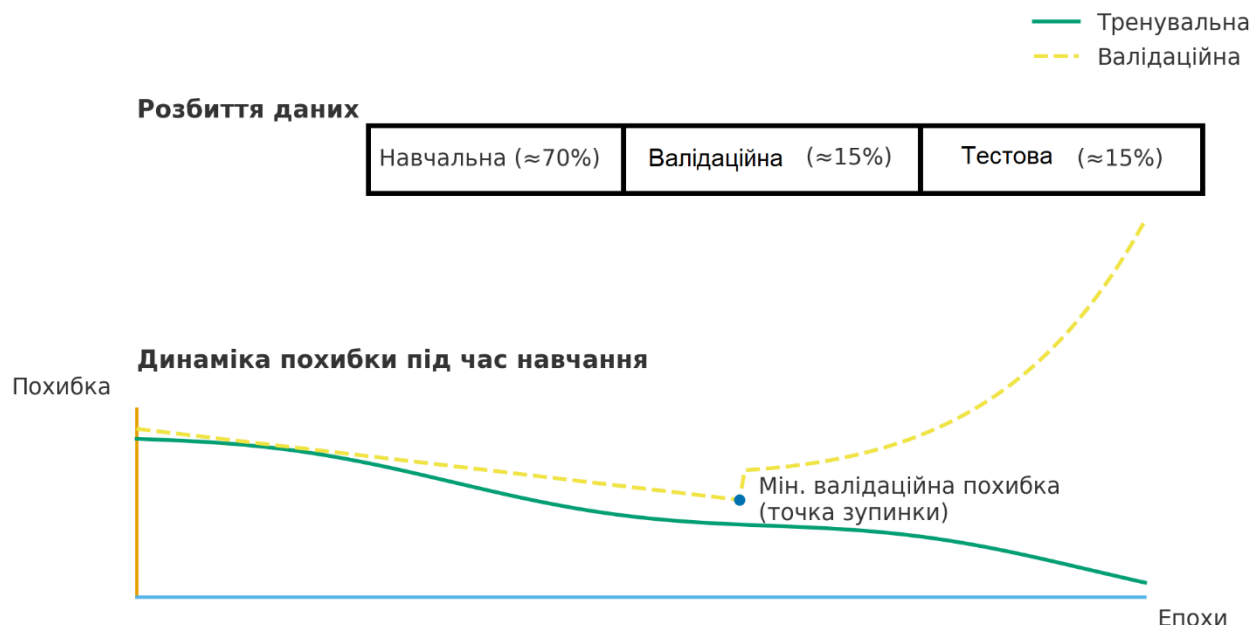


Рисунок 2.4 — Схема розбиття вибірки та контроль перенавчання за валідаційною похибкою

- **Перенавчання** — коли тренувальна помилка низька, а валідаційна — зростає. Щоб вирішити це питання, потрібно додати *L2-регуляризацію* (weight decay) — штраф за велику норму ваг, що зменшує чутливість до шуму, або *dropout* — випадкове «вимикання» частини нейронів у прихованих шарах під час навчання, це знизить ко-адаптацію ознак. Також можна ввести ранню зупинку або зменшити складність моделі.

- Для усунення **нестабільності градієнтів**, тобто стрибків втрат, ефективним кроком буде додавання batch normalization та увімкнення clipping.

- **Дисбаланс класів** — висока точність при поганих показниках для меншого класу. Рішенням цього моменту може бути правильний підбір порогу, зважування втрат, аугментація та балансування міні-пакетів.

Для **оцінювання результатів** класифікації окрім «точності», аналізують матрицю невідповідностей (за якими класами помилки), визначають *precision/recall/F1*, а також ROC-криві для бінарних задач.

Для аналізу регресій разом з MSE/MAE корисно дивитися R^2 і графіки «передбачення vs істина» для виявлення нелінійних спотворень або залежності помилки від діапазону значень.

Хід роботи

Завдання 1: Ознайомлення з графічним інтерфейсом *Deep Network Designer* для роботи з нейронними мережами.

1.1. Запустіть графічний інтерфейс MATLAB Deep Network Designer за допомогою команди:

```
deepNetworkDesigner
```

1.2. Ознайомтесь із елементами інтерфейсу та основними функціональними блоками, такими як:

- Створення нейронної мережі.
- Вибір та налаштування шарів мережі.
- Вибір алгоритмів навчання.
- Тестування та збереження мережі.

1.3. Завантажте готову прикладну модель згідно з номером варіанту (табл. 2.1) і перегляньте її архітектуру. Ознайомтесь із параметрами навчання для цієї моделі, а також функціональністю перегляду результатів.

1.4. Зробіть дві модифікації архітектури згідно ваших параметрів за таблицею 2.1:

- а) додайте (*парний* варіант) / видаліть (*непарний* варіант) 1 прихований *fullyConnectedLayer* з розміром $H1$ або $H2$ (якщо є) = № варіанту ;
- б) замініть функцію активації.

1.5. Порівняйте отримані дані. Висновки запишіть у звіт.

Завдання 2: Побудова та навчання персептрона для реалізації логічної функції "І" (AND).

2.1. Створіть одношарову нейронну мережу з одним виходом для реалізації логічної функції AND із двома бінарними входами. Для цього використовуйте інтерфейс Deep Network Designer або наступний код:

```
layers = [  
    featureInputLayer(2)  
    fullyConnectedLayer(1)  
    sigmoidLayer
```

```

regressionLayer];

options = trainingOptions('adam', ...
    'MaxEpochs', 100, ...
    'Plots', 'training-progress');

X = [0 0; 0 1; 1 0; 1 1]; % Входи
Y = [0; 0; 0; 1]; % Виходи логічної функції "І"

net = trainNetwork(X, Y, layers, options);

```

2.2. Після створення та навчання мережі перевірте її роботу, використовуючи тестові дані.

2.3. Додайте на вхід мережі шум за наступним виразом:

$$X_{\varepsilon} = X + \varepsilon \cdot \text{№ варіанту}$$

де ε обирається з таблиці 2.1.

2.4. Порівняйте результати роботи мережі за основними критеріями.

Висновки запишіть у звіт.

Завдання 3: Моделювання логічної функції "**АБО**" (OR).

3.1. Створіть нейронну мережу для моделювання логічної функції "АБО" (OR). Використовуйте архітектуру мережі з одним шаром та одним нейроном.

3.2. Навчіть мережу, використовуючи ті ж підходи, що і в завданні 2.

3.3. Перевірте роботу мережі на наборі вхідних даних:

```

X = [0 0; 0 1; 1 0; 1 1]; % Входи
Y = [0; 1; 1; 1]; % Виходи для "АБО"

```

3.4. Додайте вхідний шум та проаналізуйте отримані результати.

Завдання 4: Моделювання логічної функції "**ВИКЛЮЧНЕ АБО**" (XOR).

4.1 Побудуйте нейронну мережу для реалізації логічної функції "виключне АБО". Для цього потрібно використовувати багатошарову нейронну мережу (MLP) з нелінійними активаційними функціями та принаймні один

прихований шар, оскільки XOR є лінійно нероздільною функцією за своєю природою.

4.2 Архітектура мережі може бути наступною:

```
layers = [  
    featureInputLayer(2)  
    fullyConnectedLayer(2)  
    reluLayer  
    fullyConnectedLayer(1)  
    sigmoidLayer  
    regressionLayer];  
  
options = trainingOptions('adam', ...  
    'MaxEpochs', 500, ...  
    'Plots', 'training-progress');
```

```
X = [0 0; 0 1; 1 0; 1 1];  
Y = [0; 1; 1; 0];
```

```
net = trainNetwork(X, Y, layers, options);
```

4.3 Після навчання протестуйте мережу, переконавшись у коректності її відповіді.

4.4 За таблицею 2.1 замініть функцію активації та додайте другий прихований шар $H2 = \mathcal{N}_2$ варіанту.

4.5 Заново навчіть мережу та проаналізуйте якість її роботи. Звірте епохи до збіжності та середню помилку. Висновки записати у звіт.

Завдання 5: *Апроксимація математичної функції* за допомогою нейронної мережі.

5.1. Використовуючи *Deep Network Designer* або командний інтерфейс, створіть нейронну мережу для апроксимації математичної функції:

$$y = \alpha x_1^2 + \beta x_2 + \gamma$$

де коефіцієнти α , β , γ обчислюються за номером варіанту v , d_1 — остання цифра v , $d_2 = \lfloor v/10 \rfloor$:

$$\alpha = 1 + 0,01 \cdot d_1; \quad \beta = 1 + 0,01 \cdot v; \quad \gamma = 1 + 0,1 \cdot d_2;$$

5.2. Визначте вхідні дані

5.3. Налаштуйте мережу з наступною архітектурою:

- два вхідних параметри (x_1, x_2);
- один прихований шар із $10+H1$ нейронами (табл. 2.1);
- якщо варіант v **непарний** — додати 2-й прихований шар із $10+2 \cdot H2$ нейронами;
- вихідний шар: один лінійний нейрон; функція втрат — MSE;
- функції активації за таблицею 2.1.

5.4. Проведіть навчання моделі і перевірте її здатність апроксимувати нові значення

Завдання 6: За результатами виконання завдань, *оформіть звіт*, що обов’язково повинен містити елементи, вказані у *Додатку А*.

Представте виконану та оформлену роботу викладачеві для перевірки та захисту.

Таблиця 2.1 — Параметри індивідуального завдання

№ варіанту	Модель НМ	Н1 (нейронів у 1-му прихованому шарі)	Функція активації	Коефіцієнт шуму, ϵ
1, 11, 21, 31, 41	squeezenet	2	ReLU	0.01
2, 12, 22, 32, 42	googlenet	3	tanh	0.02
3, 13, 23, 33, 43	inceptionv3	4	leakyReLU(0.01)	0.03
4, 14, 24, 34, 44	mobilenetv2	5	ReLU	0.04
5, 15, 25, 35, 45	resnet18	6	tanh	0.05
6, 16, 26, 36, 46	xception	2	leakyReLU(0.02)	0.01
7, 17, 27, 37, 47	shufflenet	3	ReLU	0.02
8, 18, 28, 38, 48	xception	4	tanh	0.03
9, 19, 29, 39, 49	alexnet	5	leakyReLU(0.03)	0.04
10, 20, 30, 40, 50	vgg19	6	ReLU	0.05

Приклади виконання основних завдань

Приклад 1: Створення та навчання нейронної мережі за допомогою *Deep Network Designer*.

Крок 1: Запуск інтерфейсу *Deep Network Designer*.

Виконайте команду ***deepNetworkDesigner***, це запустить графічний інтерфейс, у якому ви зможете створювати, модифікувати та навчати нейронні мережі

Крок 2: Створення нової простої мережі

Для створення нової мережі, у головному вікні *Deep Network Designer* натисніть ***New*** → ***Create New Network***. У новому вікні оберіть тип мережі: наприклад, для створення *feedforward* мережі, можна вибрати ***Feedforward Network***.

Крок 3: Додавання шарів

У графічному інтерфейсі можна додавати та видаляти шари, налаштовуючи архітектуру мережі:

1) **Додамо вхідний шар**, для цього обираємо ***Input Layer***. Вводимо потрібні дані, наприклад, *featureInputLayer(2)* для двох входів.

2) **Додамо приховані шари.**

Для обробки даних обираємо шар ***Fully Connected Layer*** — наприклад, *fullyConnectedLayer(10)*, щоб додати 10 нейронів.

Для активації додайте ***Activation Layer***, такий як *reluLayer* або *sigmoidLayer*.

3) **Додамо вихідний шар *Output Layer*** для одного виходу *fullyConnectedLayer(1)* та активаційний шар для регресії (*regressionLayer*) чи класифікації (*classificationLayer*) в залежності від задачі .

Крок 4: Налаштування параметрів мережі

Клацніть на кожен шар і налаштуйте його параметри. Наприклад, для прихованих шарів можна змінити кількість нейронів, тип активаційної функції тощо. У графічному інтерфейсі ви можете бачити схему побудови нейронної

мережі та вносити до неї зміни, додаючи або видаляючи шари, переміщуючи їх місцями

Крок 5: Налаштування параметрів навчання

Налаштуйте параметри навчання у вкладці **Training Options**:

- a) оберіть алгоритм навчання (наприклад, *adam*);
- b) встановіть кількість *epoch*. Добір проводиться експериментально, орієнтуйтеся на точку, де валідаційна похибка перестає зменшуватись.
- c) встановіть розмір міні-пакетів. Він має узгоджуватись з доступною пам'яттю ПК. Для табличних даних часто беруть 32–256.
- d) встановіть початкову швидкість навчання. Типовими значення для алгоритму Adam є $1e-3 \dots 1e-4$. За потреби швидкість можна динамічно зменшити під час навчання.
- e) активувати ранню зупинку — зупиняти навчання, якщо валідаційна метрика не покращується певну кількість перевірок поспіль. Зберігаємо найкращу за валідацією мережу.
- f) Оберіть спосіб виводу графіку навчання.

Крок 6: Навчання мережі

Завантажте вхідні дані через вкладку **Data** або вручну через робоче середовище MATLAB. Натисніть кнопку **Train**, щоб запустити процес навчання мережі. Під час навчання ви побачите графіки процесу навчання.

Крок 7: Візуалізація та збереження мережі

Після завершення навчання ви можете зберегти вашу мережу та переглянути її архітектуру. Для візуалізації можна використовувати команду: `plot(net)`

Це побудує графік, що показує шари нейронної мережі та їх зв'язки.

Якщо ви створюєте мережу для класифікації або регресії, це буде виглядати приблизно так:

Input Layer → Fully Connected Layer → ReLU Layer → Fully Connected Layer → Softmax/Regression Layer.

- Input Layer (розмір входу: 28x28 зображення)

- *Convolutional Layer (32 фільтри розміру 3x3)*
- *ReLU Layer*
- *Max Pooling Layer (розмір вікна 2x2)*
- *Fully Connected Layer (10 нейронів для 10 класів)*
- *Softmax Layer (для класифікації)*
- *Classification Layer*

У результаті у вас буде чітка візуалізація шарів, їх зв'язків і налаштувань в архітектурі нейронної мережі, яку можна налаштовувати в реальному часі за допомогою Deep Network Designer.

Приклад 2: Створення і навчання нейронної мережі в командному режимі для логічної функції "І".

% Вхідні дані для логічної функції "І"

X = [0 0; 0 1; 1 0; 1 1]; % Входи

Y = [0; 0; 0; 1]; % Виходи для логічної функції "І"

% Архітектура мережі з двома входами та одним виходом

layers = [

featureInputLayer(2) % Вхідний шар для двох параметрів

fullyConnectedLayer(1) % Повнозв'язний шар з одним нейроном

sigmoidLayer % Сигмоїдна активаційна функція

regressionLayer]; % Шар для регресії

% Параметри навчання

options = trainingOptions('adam', ...

'MaxEpochs', 100, ...

'Plots', 'training-progress'); % Графічне відображення процесу навчання

% Навчання мережі

net = trainNetwork(X, Y, layers, options);

% Перевірка роботи мережі

% Тестові входи

X_test = [0 0; 0 1; 1 0; 1 1];

% Передбачені виходи

```
YPred = predict(net, X);
```

% Відображення результатів

```
disp('Реальні виходи:');
```

```
disp(Y);
```

```
disp('Прогнозовані виходи:');
```

```
disp(round(Y_pred)); % Округлюємо для отримання 0 або 1
```

>> Реальні виходи:

0

0

0

1

>> Прогнозовані виходи:

0

0

0

1

Приклад 3: Використання попередньо навченої моделі (наприклад, *SqueezeNet*)

Крок 1: Завантаження попередньо навченої моделі

```
net = squeezenet; % Завантаження моделі
```

Крок 2: Завантаження зображення та його попередня обробка

```
img = imread('peppers.png'); % Завантажити зображення 'peppers.png', що знаходиться у  
тій самій папці, що й проект
```

Важливо правильно вказати шлях до зображення, якщо воно знаходиться у
іншій папці, аніж проект.

```
imgResized = imresize(img, [227 227]); % Зміна розміру для сумісності з SqueezeNet
```

Крок 3: Класифікація зображення

```
label = classify(net, imgResized); % Класифікація зображення
```

```
imshow(img);
```

```
title(string(label)); % Виведення результату на екран
```

Приклад 4: Апроксимація математичної функції

Метою цього прикладу є побудова нейронної мережі для апроксимації
функції $y = x_1^2 + x_2$.

% Вхідні дані

```

X = [1 0.5; -2 0; 0.5 1; 1 1]; % Входи
Y = [1.25; 4; 1.5; 2];          % Виходи (цілі)

% Створення мережі
layers = [
    featureInputLayer(2)      % Вхідний шар (2 входи)
    fullyConnectedLayer(10)   % Прихований шар із 10 нейронами
    reluLayer                  % Активаційна функція ReLU
    fullyConnectedLayer(1)    % Вихідний шар із 1 нейроном
    regressionLayer];         % Шар для регресії (оскільки задача не класифікаційна)

% Налаштування параметрів навчання
options = trainingOptions('adam', ...
    'MaxEpochs', 500, ...
    'Plots', 'training-progress');

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

% Перевірка роботи нейронної мережі
% Прогноз на нових даних
X_test = [1 0.5; -2 0; 0.5 1; 1 1]; % Тестові вхідні дані

% Прогнозовані виходи
Y_pred = predict(net, X_test);

% Відображення результатів
disp('Реальні виходи:');
disp(Y);
disp('Прогнозовані виходи:');
disp(Y_pred);

>> Реальні виходи:                                >> Прогнозовані виходи:
    1.2500                                           1.2654
    4.0000                                           3.9975

```

1.5000

1.5160

2.0000

1.9762

Якщо отримані прогнозовані результати будуть сильно відрізнятись, можна:

- збільшити *кількість епох* (наприклад, до 1000);
- змінити *кількість нейронів* у прихованому шарі (наприклад, 20);
- використати *інші активаційні функції*.

Приклад 5: Побудова та навчання простої згорткової нейронної мережі для класифікації зображень

% Завантаження набору даних зображень (наприклад, CIFAR-10)

[XTrain, YTrain, XTest, YTest] = loadCIFARData();

% Створення простої мережі для класифікації

layers = [

 imageInputLayer([32 32 3]) % Вхідний шар для зображень 32x32x3

 convolution2dLayer(3, 8, 'Padding', 'same') % Конволюційний шар

 batchNormalizationLayer % Шар нормалізації

 reluLayer % ReLU активація

 maxPooling2dLayer(2, 'Stride', 2) % Шар pooling

 fullyConnectedLayer(10) % Вихідний шар на 10 класів

 softmaxLayer % Softmax для класифікації

 classificationLayer]; % Шар для класифікації

% Налаштування параметрів навчання

options = trainingOptions('adam', ...

 'MaxEpochs', 20, ...

 'InitialLearnRate', 1e-4, ...

 'Plots', 'training-progress');

% Навчання мережі

net = trainNetwork(XTrain, YTrain, layers, options);

% Оцінка точності мережі

```
YPred = classify(net, XTest);  
accuracy = sum(YPred == YTest) / numel(YTest);  
disp(['Точність: ', num2str(accuracy)]);
```

Контрольні питання

- 1) Що таке нейронна мережа та з яких елементів вона складається?
- 2) Які активаційні функції використовуються в глибокому навчанні?
- 3) Що таке процедура навчання нейронної мережі?
- 4) Яка різниця між лінійно роздільними та нероздільними множинами?
- 5) Які основні етапи створення та навчання нейронної мережі у MATLAB?
- 6) Для чого використовується `featureInputLayer` в архітектурі мережі?
- 7) У яких випадках використовується `regressionLayer`, а у яких — `classificationLayer`?
- 8) Як вибір кількості шарів і нейронів у них впливає на здатність моделі до навчання?
- 9) Чому для реалізації функції XOR потрібна багатошарова мережа?
- 10) Яку роль відіграє функція активації `reluLayer` у багатошаровій мережі?
- 11) Що таке оптимізатор у процесі навчання та які найпоширеніші оптимізатори підтримуються в MATLAB?
- 12) Як впливає кількість епох на якість навчання моделі?
- 13) Які переваги надає інтерфейс `Deep Network Designer` порівняно з командною реалізацією?
- 14) У яких випадках доцільно використовувати попередньо навчені моделі (наприклад, `SqueezeNet`)?
- 15) Як можна оцінити ефективність навчання побудованої моделі у MATLAB?

Лабораторна робота №3

ПОБУДОВА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ЛІТЕР У MATLAB

Мета роботи

Ознайомитися з основами побудови нейронних мереж для задачі розпізнавання символів (літер), навчитися працювати з наборами даних, налаштовувати архітектуру мережі, аналізувати ефективність навчання, тестувати точність моделі та візуалізувати результати класифікації використовуючи MATLAB Deep Learning Toolbox.

Теоретичні відомості

Розпізнавання зображень, зокрема літер, є класичним прикладом задачі багатокласової класифікації. Такі задачі ефективно розв'язуються за допомогою *згорткових нейронних мереж* (*Convolutional Neural Networks, CNN*). CNN – це спеціалізований тип глибоких нейронних мереж, розроблений для обробки даних що мають сіткову структуру, наприклад зображення, де інформація організована у вигляді матриці пікселів. На відміну від звичайних повнозв'язних мереж, які намагаються одразу працювати з усім зображенням як з одним довгим вектором ознак, CNN працює з частинами зображення — аналізує локальні ділянки (рецептивні поля). Тобто кожен фільтр «дивиться» лише на невеликий фрагмент зображення, наприклад 3×3 або 5×5 пікселів, і навчається виділяти характерні елементарні ознаки: краї, кути, штрихи, перепади яскравості. Далі, на наступних шарах, ці прості елементи комбінуються у складніші структури (наприклад, дуга літери «С», вертикальний штрих «І», діагональ «Х» тощо). У підсумку мережа поступово будує все більш абстрактні описи символу (наприклад, контури й форми літер). Така ієрархія ознак добре відповідає природі рукописних і друкованих символів.

Основні елементи згорткових нейронних мереж:

- **Вхідний шар** (*Input Layer*) приймає зображення фіксованого розміру, наприклад 28×28 пікселів у відтінках сірого (один канал). Кожен піксель має

значення яскравості, тому кожне зображення інтерпретується як матриця інтенсивностей. Такий формат потрібно витримувати для всіх прикладів, щоб вони були узгоджені за розмірністю.

- **Згортковий шар** (*Convolutional Layer*) використовує набір фільтрів (ядра згортки), які проходять по всьому зображенню та виявляють ключові ознаки, такі як краї, текстури чи контури. В результаті формується карта ознак (*feature map*), яка показує, де саме на зображенні виявлено відповідний шаблон.

Кількість параметрів одного згорткового шару є помірною. Для прикладу, якщо використовується 8 фільтрів 3×3 для одноканального (чорно-білого) зображення 28×28 , то загальна кількість ваг дорівнює $3 \cdot 3 \cdot 1 \cdot 8 = 72$ плюс 8 параметрів зміщення, тобто всього **80** параметрів. Це значно менше, ніж у випадку повнозв'язного шару, який би зв'язував кожен піксель з кожним нейроном.

Розмір вихідної карти ознак після згортки залежить від розміру ядра, кроку та заповнення. Якщо позначити висоту та ширину вхідного зображення як H та W відповідно, розмір ядра як K , крок (stride) як S , а «рамку» (padding) як P , тоді просторовий розмір виходу (висота $H_{\text{вих}}$ та ширина $W_{\text{вих}}$) обчислюється за наступними виразами:

$$H_{\text{вих}} = \left(\frac{H + 2P - K}{S} \right) + 1,$$
$$W_{\text{вих}} = \left(\frac{W + 2P - K}{S} \right) + 1$$

У MATLAB режим *'same'* зберігає розмір карти — додається заповнення по краях, а у режимі *'valid'* (без заповнення) вихід стає меншим.

- **Шар нормалізації** (*Batch Normalization Layer*) нормалізує значення активацій усередині мініпакета. Практично це вирівнює масштаби проміжних сигналів, що робить навчання стабільнішим і зазвичай прискорює збіжність. Нормалізація також дозволяє використовувати вищу швидкість навчання без «розвалу» градієнтів.

- **Активаційний шар** (*Activation Layer*).

Після згортки і нормалізації зазвичай застосовується нелінійність. Типовий вибір — **ReLU** (*Rectified Linear Unit*), яка пропускає лише додатні значення, а від’ємні значення зануляються. Це підсилює здатність мережі до відокремлення класів і допомагає уникати проблеми зникнення градієнта.

В окремих випадках застосовують варіанти ReLU, наприклад leaky ReLU, яка залишає маленький ненульовий градієнт і для від’ємних значень, щоб нейрон не «вимикався» повністю.

- **Шар підвибірки** (*Pooling Layer*) зменшує розмірність вхідних даних і кількість параметрів на наступних етапах. Найчастіше використовується **MaxPooling** розміром 2×2 зі stride 2. Ця функція обирає лише максимальне значення з кожного підвікна, що забезпечує ефективне стискання даних (менше обчислень у наступних шарах), фільтрацію «шуму» та часткова інваріантність до невеликих зсувів і деформацій (якщо літера зміщена на 1 піксель, важливі елементи все одно будуть «захоплені» максимальною вибіркою).

- **Повнозв’язний шар** (*Fully Connected Layer*) на завершальних етапах зв’язує всі нейрони з виділеними ознаками (штрихи, дуги, перехрестя, контури символів) між собою. Його завдання — на основі цих ознак прийняти остаточне рішення, до якого класу належить зображення. Повнозв’язний шар поводить себе як «класичний класифікатор», який бачить сукупність усіх витягнутих ознак одночасно.

- **Шар softmax** перетворює вихідні значення останнього повнозв’язного шару на набір ймовірностей: для кожного класу (кожної літери) обчислюється значення в діапазоні від 0 до 1, і сума всіх цих значень дорівнює 1. Таким чином ми отримуємо ймовірнісну інтерпретацію відповіді.

- **Шар класифікації** (*Classification Layer*) обирає остаточний клас (літеру) із максимальною ймовірністю.

У контексті розпізнавання літер маємо 26 класів — по одному на кожну велику букву англійського алфавіту (A...Z). Для коректної інтерпретації важливо, щоб мітки були узгоджені з вихідними нейронами (правильний порядок класів).

Окремо варто відзначити поняття рецептивного поля. На нижніх рівнях один згортковий фільтр «бачить» маленьку ділянку (наприклад, 3×3 пікселі), тобто може виявити локальну риску чи краєчок контуру. У міру того, як ми накладаємо кілька згорток і пулінгів поспіль, ефективна область огляду стає ширшою. На вищих шарах нейрон фактично аналізує вже значну частину або й всю букву цілком. Це і є причина, чому CNN добре справляється з літерами: воно спочатку вчиться бачити окремі штрихи, а потім — їх поєднання.

Процес навчання згорткових нейронних мереж для розпізнавання літер зазвичай включає наступні етапи:

1. Підготовка даних.

Усі зображення повинні бути приведені до однакового розміру (наприклад, $28 \times 28 \times 1$, де «1» означає один канал яскравості). Важливо також нормувати інтенсивність, наприклад привести значення пікселів у проміжок $[0;1]$. Також зображення повинні бути згруповані за мітками класу (літерами). Мітки зручно зберігати як значення категорій, сумісні з класифікаційним шаром MATLAB.

Додатково на цьому етапі часто застосовують аугментацію даних — невеликі повороти (наприклад, $\pm 5 \dots 10^\circ$), зсуви на 1–2 пікселі, зміна масштабу. Ці ускладнення роблять мережу більш стійкою. Для рукописних ж літер це критично, бо один і той самий символ може бути написаний з різним нахилом, різною товщиною штриха, у трохи іншому положенні відносно центру.

2. Розподіл на тренувальний, валідаційний і тестовий набори.

Тренувальний набір використовується безпосередньо для оновлення ваг мережі.

Валідаційний набір потрібен для контролю перенавчання: ми не навчаємося на цих даних, а лише перевіряємо, чи покращується якість. Це дозволяє коректно обрати момент зупинки навчання та налаштувати гіперпараметри (швидкість навчання, кількість епох).

Тестовий ж набір застосовується після завершення навчання і не повинен впливати на його перебіг. На тесті оцінюється реальна здатність мережі класифікувати «нові», раніше не бачені приклади.

Окремо звертають увагу на баланс класів. Якщо деякі літери зустрічаються значно рідше, модель може гірше їх розпізнавати. Це проявляється тим, що певні символи плутаються між собою (наприклад, «I» і «l», або «O» і «Q»). У таких випадках або збільшують кількість прикладів рідкісних класів, або застосовують спеціальну аугментацію саме до цих класів.

3. Налаштування архітектури мережі.

На цьому етапі визначається конфігурація шарів:

- кількість згорткових шарів і кількість фільтрів у кожному;
- розмір ядра згортки (3×3 , 5×5 тощо);
- місця вставки шарів нормалізації;
- де саме виконувати пулінг і який саме (max pooling, average pooling);
- розміри повнозв'язних шарів у «хвості» перед softmax.

Зазвичай для зображень 28×28 достатньо компактної мережі: один-два згорткових блоки (згортка $\rightarrow$ нормалізація $\rightarrow$ ReLU $\rightarrow$ пулінг), після яких іде повнозв'язний шар і класифікація. Надто глибока архітектура в такій задачі часто не дає суттєвого виграшу, але ускладнює навчання і збільшує ризик перенавчання.

4. Навчання моделі .

Навчання мережі — це процес підбору ваг (коефіцієнтів фільтрів, зсувів, коефіцієнтів повнозв'язних шарів), який виконується ітераційно за допомогою алгоритму зворотного поширення помилки (*backpropagation*).

Спочатку вибирається *початкова швидкість навчання* (*Initial Learn Rate*) та *оптимізатор* — алгоритм, який оновлює ваги на основі похідних функції втрат. В MATLAB типово використовують стохастичний градієнт із моментом (sgdm) або адаптивні методи на кшталт *Adam* (адаптивний метод моментів). Далі визначається кількість *epoch* (MaxEpochs).

Якість навчання відстежується через функцію втрат (*loss function*). Вона показує, наскільки сильно відрізняється передбачене значення від правильного.

Вибір функції втрат та метрик залежить від типу задачі. Метриками для **багатокласової класифікації** (крос-ентропія) в основному є точність, матриця невідповідностей (confusion matrix), повнота, точність (precision), F1-міра.

У задачах **бінарної класифікації** (бінарна крос-ентропія) додатково виділяють ROC-криві та AUC (Area Under Curve — площа під ROC кривою) для підбору порогу.

Для оцінювання **регресії** використовують середньоквадратичну помилку (MSE), середню абсолютну помилку (MAE), коефіцієнт детермінації R^2 , а також графіки кореляції між прогнозом і «правдою» для виявлення нелінійних спотворень або залежності помилки від діапазону значень.

У MATLAB для класифікаційних виходів зазвичай використовують фінальну комбінацію softmax + класифікаційний шар (ймовірнісна інтерпретація), для регресії — регресійний шар з MSE.

5. Тестування

Після завершення навчання модель перевіряється на тестовому наборі зображень, які не використовувалися ні для оновлення ваг, ні для вибору гіперпараметрів. На цьому етапі оцінюється фінальна точність розпізнавання.

Додатково корисно переглянути окремі приклади класифікації: показати саме зображення літери, передбачений клас, справжній клас і значення впевненості (ймовірність softmax). Також варто розглянути випадки, де мережа впевнено помиляється. Саме такі приклади найкраще виявляють систематичні слабкі місця, наприклад, неякісне центрування символу, занадто слабкий контраст або надто схожі форми між різними класами.

6. Аналіз результатів

Після завершення тестів, модель порівнює передбачені мітки з реальними та виконується узагальнений і візуальний аналіз для окремих прикладів. Насамперед обчислюються базові метрики (точність, а за потреби — precision/recall/F1 для кожного класу) і будується матриця невідповідностей, що дає змогу побачити, між якими класами трапляються систематичні помилки.

Для задач розпізнавання літер доцільно окремо переглянути приклади з найбільшою впевненістю моделі, але з неправильним результатом — це допомагає виявити типові «плутанини» (наприклад, $I \leftrightarrow l$, $O \leftrightarrow Q \leftrightarrow C$). Рекомендовано сформулювати підбірку з кількох десятків випадків: правильно класифіковані складні приклади та хибні спрацьовування з коротким коментарем (яка ознака могла ввести модель в оману).

Для підсумкової валідації демонструються графіки процесу навчання (зміна втрати та, за наявності, точності на train/val наборах), а також приклади класифікації окремих зображень із зазначенням передбаченої мітки, реального класу та оцінки впевненості. За потреби виконується підбір порога прийняття рішення (для бінарних або багатокласових із «інший» класом), аналізується вплив такого підбору на метрики й робиться короткий висновок щодо узагальнювальної здатності моделі (наявність/відсутність перенавчання, стабільність на тесті).

Регуляризація та зміцнення узагальнення.

- **L2-регуляризація** (weight decay) — до функції втрат додається штраф на велику норму ваг, що обмежує складність моделі та зменшує чутливість до шуму у даних. Практично це дозволяє уникати «надмірно різких» кордонів рішень та покращує стабільність на валідації.

- **Dropout** дає можливість випадково «вимикати» частину нейронів у прихованих шарах під час навчання. Такий прийом знижує ко-адаптацію ознак, змушуючи модель покладатися на більш стійкі патерни, і часто помітно зменшує перенавчання. Інтенсивність dropout добирається обережно: надто високі значення можуть уповільнити збіжність.

- **Batch Normalization** нормалізує проміжні активації усередині мережі, стабілізуючи їх розподіли. Це пришвидшує навчання, дозволяє використовувати більші швидкості навчання та зазвичай підвищує підсумкову якість. У згорткових нейронних мережах BN ставлять після згортки і перед нелінійністю.

- **Аугментація даних** — штучне розширення навчальної вибірки за рахунок допустимих перетворень (для зображень — повороти, зсуви, масштаб,

віддзеркалення, варіації яскравості). Аугментація підвищує різноманіття прикладів, роблячи модель стійкішою до реальних варіацій вхідних даних. Важливо зберігати семантику класу й не вносити перетворень, що змінюють мітку.

- **Крос-валідація** оцінює стабільності моделі на різних розбиттях даних. Корисна вона тоді, коли вибірка невелика або результати нестійкі до конкретного спліту. Дозволяє отримати більш надійні оцінки якості та вибрати налаштування, що краще узагальнюються.

Хід роботи

Завдання: Побудувати просту нейронну мережу для розпізнавання літер. Використати готові дані для тренування та тестування. Проаналізувати результати роботи нейронної мережі. Оформіть звіт відповідно до вимог та захистить лабораторну роботу.

1) *Створіть або завантажте готовий набір даних зображень літер англійського алфавіту.*

У якості готового набору даних зображень літер можна використати *Handwritten English Letters Dataset*. Для зручності використовуйте зображення у форматі 28x28 пікселів у градаціях сірого.

2) *Створіть функцію для оброблення та завантаження даних у MATLAB*

% Завантаження даних

```
[XTrain, YTrain, XTest, YTest] = loadLetterDataset();
```

Приклад структури функції *loadLetterDataset* для завантаження даних:

```
function [XTrain, YTrain, XTest, YTest] = loadLetterDataset()
```

```
% Визначаємо шлях до файлу CSV та директорії з зображеннями.
```

```
% Завантажуємо таблицю CSV.
```

```
% Отримуємо список файлів та міток.
```

```
% Перевіряємо можливі порожні або некоректні значення, наприклад, видаляємо ці рядки.
```

```
% Завантажуємо зображення, якщо необхідно, змінюємо розмір до 28x28.
```

```
% Розділяємо дані на навчальний та тестовий набори.
```

end

Також для роботи з зображеннями можна використовувати `'imageDatastore'`.

3) Побудуйте згорткову нейронну мережу, адаптовану до задачі класифікації літер.

Кількість фільтрів та розмір ядра в першому згортковому шарі задайте згідно індивідуального завдання (табл. 3.1).

```
layers = [  
    imageInputLayer([28 28 1])           % Вхідний шар для зображень 28x28  
    convolution2dLayer(3, 8, 'Padding', 'same') % Перший згортковий шар  
    batchNormalizationLayer              % Нормалізація партії  
    reluLayer                            % Функція активації ReLU  
    maxPooling2dLayer(2, 'Stride', 2)    % Шар підвибірки  
    fullyConnectedLayer(26)              % Повнозв'язний шар (26 класів для 26 літер)  
    softmaxLayer                         % Шар softmax  
    classificationLayer                  % Шар класифікації  
];
```

Якщо у вас непарний варіант, додайте до архітектури мережі шар `dropoutLayer(0.20)` після функції активації ReLU.

4) Налаштуємо індивідуальні параметри навчання за таблицею 3.1

```
options = trainingOptions('sgdm', ...  
    'MaxEpochs', 20, ...           % Кількість епох  
    'InitialLearnRate', 0.01, ...   % Початкова швидкість навчання  
    'Verbose', false, ...  
    'Plots', 'training-progress');   % Візуалізація процесу навчання
```

5) Проведіть навчання мережі на тренувальних даних

```
net = trainNetwork(XTrain, YTrain, layers, options);
```

6) Оцініть точність мережі на тестовому наборі даних

```
YPred = classify(net, XTest);  
accuracy = sum(YPred == YTest) / numel(YTest);
```

```
fprintf('Точність: %.2f%%\n', accuracy*100);
```

7) *Оцініть точність та візуалізуйте результати для окремих зображень*

```
idx = randi([1, numel(YTest)], 1, 5); % Випадкові індекси
for i = 1:length(idx)
    figure;
    imshow(XTest(:,:,idx(i)));
    title(['Прогноз: ', char(YPred(idx(i))), ', Реальний: ', char(YTest(idx(i)))]);
end
```

8) *Зробіть висновки по роботі*

Проведіть аналіз результатів роботи нейронної мережі та опишіть, які покращення можна внести в архітектуру або процес навчання для підвищення точності розпізнавання мінімум до 85%.

9) *Повторіть кроки 3 — 8 для оновленої нейронної мережі.*

10) Використовуючи Додаток А, *оформіть звіт* за виконаною роботою.

Зауважте, що представлені графічні **результати** роботи нейронної мережі мають містити графіки процесу навчання, точність та приклади класифікації.

Під час **аналізу** результатів, наведіть можливі помилки, шляхи їх виправлення та ідеї для покращення роботи нейронної мережі.

Таблиця 3.1. Варіанти індивідуальних параметрів нейронної мережі

<i>№ варіанту</i>	<i>К-сть фільтрів (Conv 1)</i>	<i>Розмір ядра</i>	<i>Кількість epoch</i>	<i>Початкова швидкість навчання</i>
1, 11, 21, 31, 41	8	3×3	20	0.010
2, 12, 22, 32, 42	12	3×3	25	0.005
3, 13, 23, 33, 43	16	3×3	30	0.001
4, 14, 24, 34, 44	24	5×5	35	0.010
5, 15, 25, 35, 45	32	5×5	20	0.005
6, 16, 26, 36, 46	8	3×3	25	0.001
7, 17, 27, 37, 47	12	3×3	30	0.010
8, 18, 28, 38, 48	16	3×3	35	0.005
9, 19, 29, 39, 49	24	5×5	20	0.001
10, 20, 30, 40, 50	32	5×5	25	0.010

Контрольні питання

1. Що таке згорткова нейронна мережа (CNN) і де вона найчастіше застосовується?
2. Яке призначення має шар *imageInputLayer* у побудові мережі?
3. Які функції виконує *convolution2dLayer* у згортковій нейронній мережі?
4. Для чого потрібен шар *batchNormalizationLayer*?
5. У чому полягає роль *reluLayer* в архітектурі нейронної мережі?
6. Яке призначення шару *maxPooling2dLayer*?
7. Що робить *fullyConnectedLayer* у мережі для класифікації літер?
8. Яку функцію виконує *softmaxLayer*?
9. Для чого використовується *classificationLayer*?
10. Яким чином розраховується точність нейронної мережі?
11. Як відбувається попередня обробка зображень у MATLAB перед подачею до мережі?
12. Які типи похибок можуть виникати під час розпізнавання літер?
13. Як можна покращити точність моделі? Назвіть хоча б два способи.
14. Чим відрізняється тестова вибірка від тренувальної?
15. Що таке епоха в процесі навчання нейронної мережі?

Лабораторна робота №4

ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ КЛАСИФІКАЦІЇ, ПРОГНАЗУВАННЯ ТА АНАЛІЗУ ДАНИХ У MATLAB

Мета роботи

Ознайомитися з можливостями використання різних типів нейронних мереж у середовищі MATLAB для вирішення прикладних задач класифікації даних, прогнозування часових рядів, кластеризації та виявлення аномалій. Навчитися змінювати архітектуру мереж, обирати параметри та оцінювати результати роботи моделей на реальних і синтетичних наборах даних.

Теоретичні відомості

У сучасних задачах аналізу даних широко застосовуються різні архітектури нейронних мереж, кожна з яких має свою спеціалізацію. У цій роботі розглядаються чотири популярні типи мереж: *MLP*, *LSTM*, *SOM* та *автокодувальники*. Нижче подано стислу інтуїцію, ключові конструкції та практичні рекомендації, що допоможуть свідомо обирати архітектуру під конкретну постановку задачі.

1. MLP – Багатошаровий персептрон (Multilayer Perceptron)

MLP є класичною формою штучної нейронної мережі, яка складається з одного або більше прихованих шарів з нелінійними активаційними функціями. Їх обирають для обробки статичних (непослідовних) даних: *класифікація* (softmax на виході) та *регресія* (лінійний вихід). MLP добре працює з табличними ознаками, агрегованими дескрипторами та ознаками, отриманими іншими моделями. Ця мережа особливо корисна у випадках, де результат залежить від попередніх станів.

Вхідний шар отримує ознаки (*features*), що можуть бути нормовані та стандартизовані для пришвидшення навчання, особливо при tanh.

Приховані шари виконують обчислення з використанням зважених сум та функцій активації (*ReLU, sigmoid, tanh* тощо). Достатньо одного прихованого шару із достатньою шириною, аби MLP апроксимував неперервні функції (теорема універсальної апроксимації), однак на практиці помірна глибина (1–3 шари) часто дає кращу узагальнюваність.

Вихідний шар відображає результат класифікації або регресії.

2. LSTM – Довга короткострокова пам'ять (Long Short-Term Memory)

LSTM — це особливий тип *рекурентної нейронної мережі* (RNN), який здатен зберігати історичну інформацію на довгих часових відстанях. Це досягається за допомогою спеціальної структури LSTM-блоку, який містить елементи пам'яті та керуючі ворота (*input, forget, output gates*).

LSTM використовується для задач:

- *прогнозування часових рядів;*
- *аналізу послідовностей;*
- *розпізнавання сигналів, що змінюються з часом.*

Ця мережа особливо корисна у випадках, де поточний результат залежить від попередніх станів.

У LSTM є три основні **робочі режими**. «*Sequence-to-one*» — коли робиться один прогноз на всю історію (класифікація послідовності, регресія по вікну). «*Sequence-to-sequence*» робить прогноз на кожен крок (наприклад, «на крок уперед»). *Багатокроковий прогноз* — ітеративний (*feeding back*) або прямий багатокроковий вихід.

3. SOM – Самоорганізуюча карта Кохонена (Self-Organizing Map)

SOM є ненаглядною нейронною мережею, яка виконує кластеризацію та візуалізацію даних у нижчому вимірі. SOM проєктує вхідні дані на 2D-карту, де схожі зразки розташовуються близько один до одного.

Особливості SOM мереж:

- Всі нейрони SOM мають *вектор ваг* такої ж розмірності, як і вхідні дані.
- SOM *не має функції активації* та не виконує навчання з учителем.

- Використовується для кластеризації, аналізу структури даних і візуалізації високовимірних просторів.
- Топологія карти прямокутна/гексагональна з розміром орієнтовно $\approx 5\sqrt{N}$ нейронів (для N точок) — стартова евристика.

SOM зазвичай обирають, коли потрібно «побачити» структуру даних без міток, знайти природні кластери/перехідні зони, а також коли є інтерес до топологічних зв'язків між кластерами, не лише до розбиття.

4. Автокодувальники (АЕ)

Автокодувальники — це безнаглядні нейронні мережі, які навчаються кодувати вхідні дані у компактне представлення (*latent space* — латентний простір) та відновлювати їх назад. Така мережа складається з двох частин:

- *Encoder* — зменшує розмірність вхідного вектора.
- *Decoder* — реконструює вхід на основі коду.

Застосовувати автокодувальники потрібно, коли необхідно зменшити розмірність даних із мінімальною втратою інформації, шукати відхилення без міток, виявлення аномалій через аналіз помилки реконструкції та попередньої обробки даних перед класифікацією.

Якщо модель добре відновлює нормальні приклади, але має високу помилку на аномальних, це ознака потенційних відхилень або помилок у даних.

Хід роботи

Завдання 1: Класифікація даних за допомогою MLP.

1.1) Завантаження та підготовка даних *Iris*

Використовуємо набір *fisheriris* для класифікації 3 видів ірисів:

```
load fisheriris;
```

```
inputs = meas'; % Вхідні дані (розміри квітів)
```

```
targets = full(ind2vec(grp2idx(species'))); % Цільові мітки у векторній формі
```

1.2) Створення та навчання багатопередового перцептрона (MLP)

Побудуємо мережу з двома прихованими шарами. Кількість нейронів кожного шару MLP обираємо за таблицею 4.1:

```
net = patternnet([10, 5]); % Два шари з 10 і 5 нейронами
net = train(net, inputs, targets); % Навчання мережі
```

1.3) Прогнозування та оцінка результатів класифікації

Побудуємо матрицю плутанини для візуалізації точності:

```
predictions = net(inputs);
figure;
plotconfusion(targets, predictions); % Матриця плутанини
```

У **результаті** буде побудовано графік матриці плутанини з показником точності класифікації (рис. 4.1) .

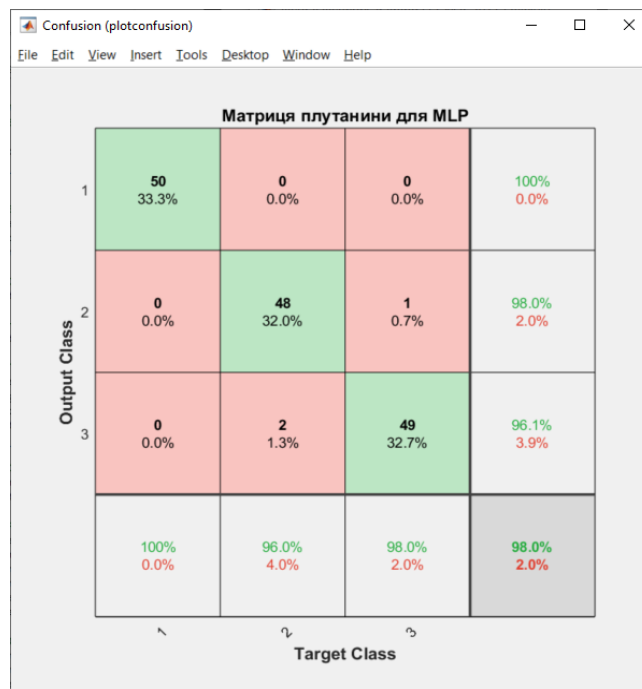


Рисунок 4.1 — Графік матриці плутанини

Завдання 2: Прогнозування часових рядів за допомогою LSTM.

2.1) Створимо штучний часовий ряд у вигляді синусоїдального сигналу з випадково генерованим шумом:

```
timeSteps = 1:100;
data = sin(timeSteps * 0.1) + randn(size(timeSteps)) * 0.1;
```

2.2) Розділяємо дані на навчальні та тестові:

```
trainRatio = 0.8;
numTrain = round(trainRatio * numel(data));
XTrain = data(1:numTrain);
YTrain = data(2:numTrain+1);
```

```
XTest = data(numTrain+1:end-1);
```

```
YTest = data(numTrain+2:end);
```

2.3) *Форматуємо дані для LSTM ($1 \times 1 \times N$):*

```
XTrain = num2cell(XTrain);
```

```
YTrain = num2cell(YTrain);
```

```
XTest = num2cell(XTest);
```

```
YTest = num2cell(YTest);
```

2.4) *Створюємо архітектуру LSTM мережі. Кількість нейронів у LSTM задаємо за таблицею 4.1.*

```
layers = [  
    sequenceInputLayer(1)  
    lstmLayer(50, 'OutputMode', 'sequence') % 50 нейронів у LSTM  
    fullyConnectedLayer(1)  
    regressionLayer  
];
```

2.5) *Налаштовуємо параметри навчання мережі за таблицею 4.1:*

```
options = trainingOptions('adam', ...  
    'MaxEpochs', 100, ...  
    'MiniBatchSize', 10, ...  
    'Shuffle', 'every-epoch', ...  
    'Verbose', false, ...  
    'Plots', 'training-progress');
```

```
% Навчання мережі LSTM
```

```
net = trainNetwork(XTrain, YTrain, layers, options);
```

```
% Прогнозування та оцінка
```

```
YPred = predict(net, XTest);
```

2.6) *Візуалізуємо результати прогнозу:*

```
figure;  
plot(cell2mat(YTest), 'b', 'LineWidth', 2);  
hold on;  
plot(cell2mat(YPred), 'r--', 'LineWidth', 2);  
legend('Реальні дані', 'Прогноз LSTM');  
title('Прогноз часових рядів за допомогою LSTM');
```

grid on;

На рисунку 4.2 зображено приблизний результат прогнозу часових рядів нейронною мережею LSTM.

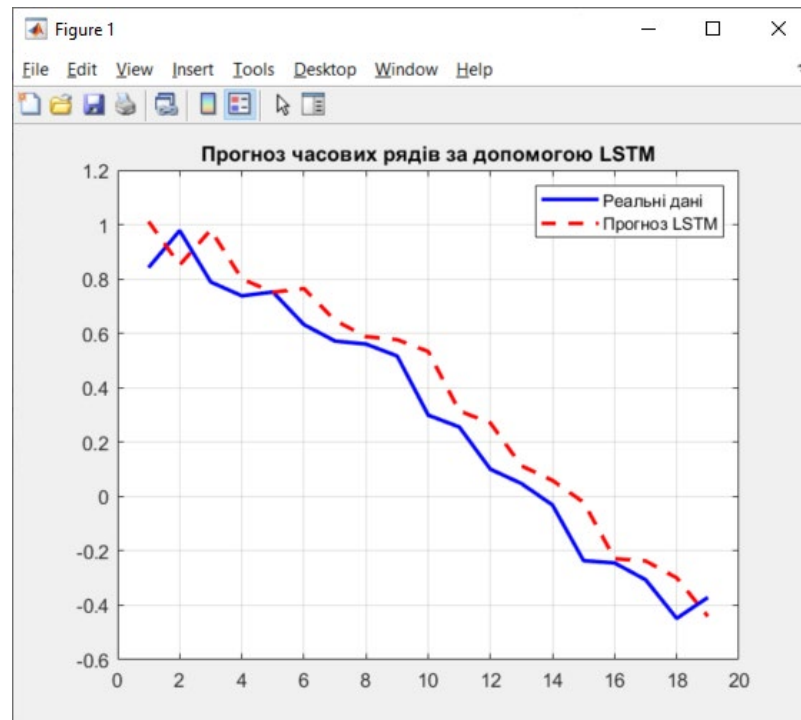


Рисунок 4.2 — Графік реальних та прогнозованих значень часового ряду

Завдання 3: Кластеризація даних за допомогою SOM.

3.1) Генеруємо випадкові 2D-дані:

```
data = rand(2, 100); % 100 точок у 2D-просторі
```

3.2) Створюємо SOM (розмір карти обираємо з табл. 4.1):

```
net = selforgmap([5 5]); % Карта розміром 5x5
```

3.3) Запускаємо навчання нейронної мережі

```
net = train(net, data);
```

3.4) Візуалізуємо кластери:

```
plotsompos(net, data);
```

```
title('Кластеризація даних за допомогою SOM');
```

У результаті виконання програми MATLAB побудує графік відображення кластерів та позицій нейронів на самонавчальній карті (рис 4.3).

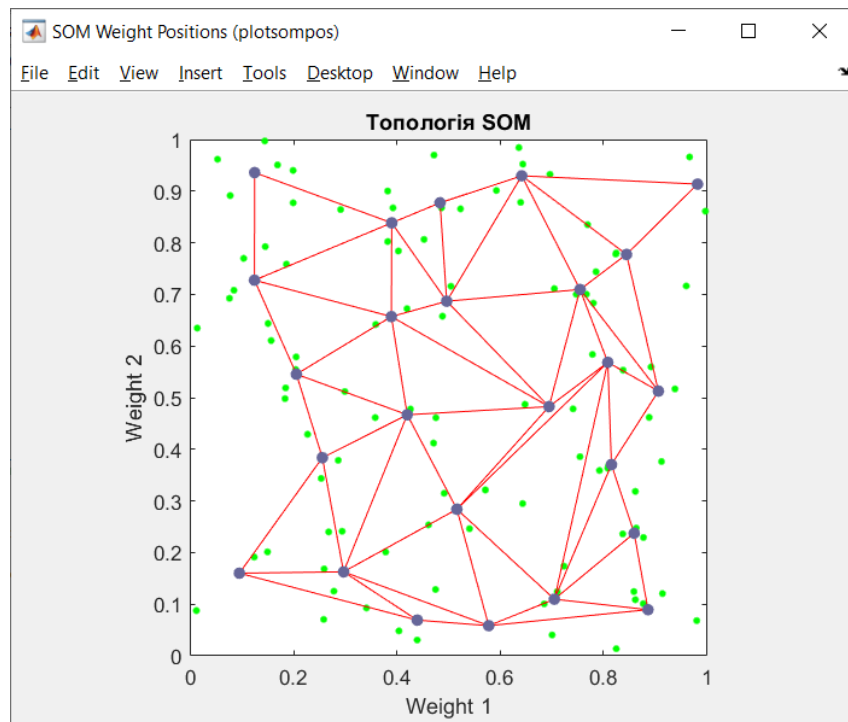


Рисунок 4.3 — Графік відображення кластерів та позицій нейронів

Завдання 4: Виявлення аномалій у наборі даних із шумом за допомогою автокодувальника.

4.1) Генеруємо дані із аномаліями:

```
data = [randn(1, 100), 10*randn(1, 5)]; % Дані з 5 аномаліями
```

4.2) Створюємо автокодувальник (АК).

Кількість нейронів у прихованому шарі АК та епох обираємо відповідно до варіанту (табл. 4.1):

```
autoenc = trainAutoencoder(data', 3, 'MaxEpochs', 50); % Створення та навчання
```

4.3) Отримуємо реконструйовані дані:

```
encodedData = encode(autoenc, data');
```

```
reconstructedData = decode(autoenc, encodedData);
```

4.4) Обчислимо помилки та задамо поріг виявлення аномалій:

```
errors = abs(reconstructedData' - data);
```

```
anomalies = find(errors > 2); % Поріг аномалії
```

4.5) Візуалізуємо отримані результати:

```
figure;
```

```
plot(data, 'b'); hold on;
```

```
plot(anomalies, data(anomalies), 'ro'); % Аномальні значення
```

```
legend('Дані', 'Аномалії');
title('Виявлення аномалій');
```

На рисунку 4.4 зображено приклад графіку оброблених даних із виділеними аномальними значеннями.

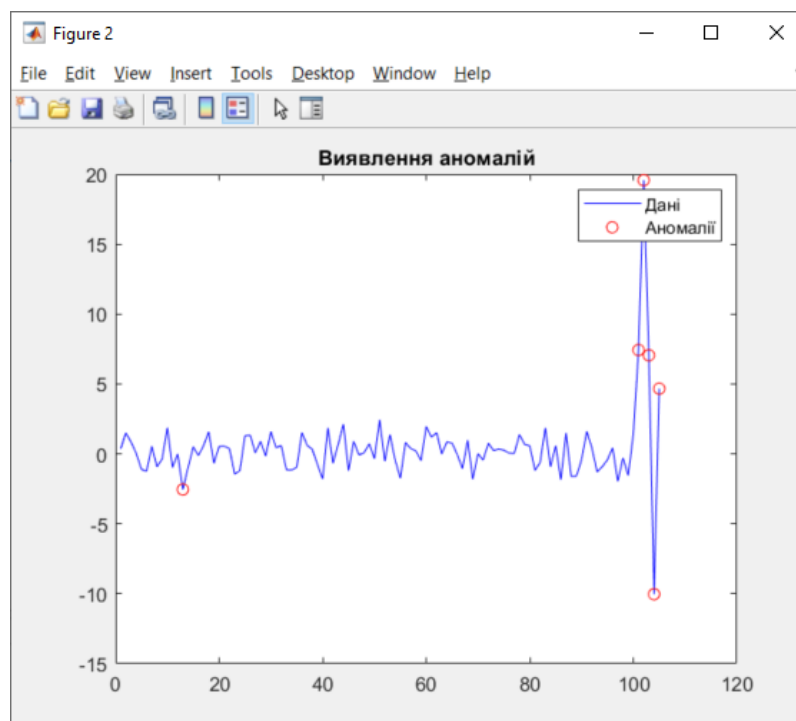


Рисунок 4.4 — Графік даних із виділеними аномальними значеннями

Завдання 5: *Оформіть звіт за виконаною роботою, відповідно до вимог у додатку А.*

Таблиця 4.1. Варіанти індивідуальних параметрів для нейронних мереж

№ варіанту	Розмір				Кількість epoch	Розмір міні- пакету
	MLP	LSTM	SOM	AK		
1, 11, 21, 31, 41	[10, 5]	50	[4 4]	3 нейрони	100	16
2, 12, 22, 32, 42	[12, 6]	30	[5 5]	5 нейрони	120	32
3, 13, 23, 33, 43	[8, 4]	80	[6 6]	8 нейрони	80	48
4, 14, 24, 34, 44	[16, 8]	100	[8 6]	4 нейрони	150	64
5, 15, 25, 35, 45	[10, 5]	50	[4 4]	6 нейрони	100	16
6, 16, 26, 36, 46	[12, 6]	30	[5 5]	3 нейрони	120	32
7, 17, 27, 37, 47	[8, 4]	80	[6 6]	5 нейрони	80	48
8, 18, 28, 38, 48	[16, 8]	100	[8 6]	8 нейрони	150	64
9, 19, 29, 39, 49	[10, 5]	50	[4 4]	4 нейрони	100	16
10, 20, 30, 40, 50	[12, 6]	30	[5 5]	6 нейрони	120	32

Контрольні питання

1. Що таке багатошаровий персептрон (MLP) і для яких задач його доцільно використовувати?
2. Які параметри можна змінити у багатошаровому персептроні для підвищення точності?
3. Яку роль виконують приховані шари в нейронній мережі MLP?
4. Чим рекурентна нейронна мережа LSTM відрізняється від звичайної feedforward-мережі?
5. Як вибір вибору нейронів у LSTM впливає на якість прогнозу?
6. Як побудована архітектура LSTM у MATLAB?
7. Що таке самонавчальна карта (SOM) і в чому її відмінність від мереж із вчителем?
8. Як здійснюється процес кластеризації за допомогою SOM?
9. Які переваги мають SOM при роботі з високовимірними даними?
10. Що таке автокодувальник і яка його структура?
11. Як автокодувальники допомагають у виявленні та усуненні аномалій у даних?
12. Чим відрізняється кодування (encode) від декодування (decode) в автокодувальниках?
13. Яким чином обирається поріг для виявлення аномалій на основі помилки реконструкції?
14. Як змінюється поведінка автокодувальника при зміні кількості нейронів у прихованому шарі?
15. Які є загальні етапи створення, навчання та оцінювання моделі нейронної мережі у MATLAB?
16. Які переваги MATLAB для роботи з нейронними мережами?

Лабораторна робота №5

ОПРАЦЮВАННЯ СТИСНУТИХ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ У MATLAB

Мета роботи

Ознайомитися з принципами опрацювання стиснутих зображень у задачах розпізнавання та класифікації за допомогою згорткових нейронних мереж у MATLAB. Закріпити практичні навички роботи з набором даних CIFAR-10, а також оцінити точність класифікації на прикладі JPEG зображень з різним ступенем стиснення.

Теоретичні відомості

У сучасних системах комп'ютерного зору широко використовуються *згорткові нейронні мережі* (ЗНМ) для класифікації зображень, розпізнавання об'єктів та інших задач аналізу візуальної інформації. Одним із важливих аспектів реального застосування таких систем є робота зі стиснутими зображеннями, оскільки більшість зображень, що надходять у системи (наприклад, з інтернету або камер спостереження), зберігаються у форматах JPEG, PNG тощо.

Стиснення зображень.

Стиснення — це процес зменшення обсягу цифрових даних, які описують зображення, з метою економії місця на носіях або прискоренні передачі даних у мережах. Стиснення буває:

- **Без втрат** (*lossless*) — оригінал зображення можна повністю відновити (наприклад, PNG, TIFF).
- **З втратами** (*lossy*) — частина інформації втрачається безповоротно, часто у високих просторових частотах (наприклад, JPEG).

З погляду нейронних мереж головне не тільки зменшення розміру файлу, а *зміна статистики зображень*: інтенсивності, текстури, локальні контури і

навіть розподіли кольору стають іншими. Це породжує *доменне зміщення* між тренувальними і тестовими даними.

JPEG-стиснення.

JPEG — один із найпоширеніших форматів стискання зображень з втратами. Цей алгоритм включає в себе:

- 1) Перетворення кольорового простору з *RGB* в *YCbCr* із подальшим субдискретизуванням хроми (часто 4:2:0), де кольорові канали мають нижчу роздільність, ніж яскравість..
- 2) Розбиття зображення на *блоки 8×8 пікселів*.
- 3) Застосування *дискретного косинусного перетворення* (ДКП) до кожного блоку.
- 4) *Квантизація* коефіцієнтів ДКП згідно з матрицею квантування. Саме на цьому етапі відбувається основна втрата інформації.
- 5) *Ентропійне кодування* (зазвичай методом Хаффмана).

Ступінь стиснення в JPEG визначається параметром **якість** (*Quality*), який зазвичай задається у відсотках (від 1 до 100). Чим менше значення якості — тим вищий рівень стиснення і, відповідно, більші втрати інформації. Це безпосередньо впливає на точність та ефективність розпізнавання моделей глибокого навчання, оскільки відбувається розмиття країв об'єктів, спотворення або втрата ключових ознак, а також створення завад у вигляді артефактів, які можуть бути інтерпретовані моделлю як нові, неіснуючі ознак.

Нейронна мережа, яка навчена лише на оригінальних (нестиснутих) зображеннях, може сильно втрачати в точності при роботі зі стиснутими зображеннями. Саме тому важливо аналізувати, як змінюється точність класифікації в залежності від ступеня стиснення, і в разі потреби — проводити перенавчання моделі з урахуванням стиснених даних.

У даній лабораторній роботі ми дослідимо, як ступінь стиснення JPEG впливає на якість класифікації зображень та навчимо модель згорткової нейронної мережі класифікувати зображення з різним рівнем стиснення.

Набір даних CIFAR-10

CIFAR-10 (*Canadian Institute for Advanced Research*) — це широко використовуваний еталонний набір даних для задач класифікації зображень. Він містить 60 000 кольорових зображень розміром 32×32 пікселі, які розділені на 10 класів по 6000 зображень у кожному класі.

Класи зображень CIFAR-10:

- *Airplane* (літак)
- *Automobile* (автомобіль)
- *Bird* (птаха)
- *Cat* (кіт)
- *Deer* (олень)
- *Dog* (собака)
- *Frog* (жаба)
- *Horse* (кінь)
- *Ship* (корабель)
- *Truck* (вантажівка)

Набір даних CIFAR-10 ідеально підходить для швидкої перевірки ефективності різних архітектур нейронних мереж, тестування поведінки моделей при зміні гіперпараметрів та аналізу стійкості до спотворень, таких як стиснення, шум, масштабування тощо.

Оскільки набір має просту структуру, збалансовані класи та малий розмір, це дозволяє проводити експерименти навіть на комп'ютерах із середнім рівнем обчислювальної потужності.

Завантаження CIFAR-10 у MATLAB:

Спочаток 1: Використання *cifar10Dataset* (після завантаження з Інтернету)

1. Спершу завантажте архів CIFAR-10, виконавши команду *cifar10Dataset*.

```
cifar10Data = 'cifar-10-batches-mat';  
if ~exist(cifar10Data, 'dir')  
    disp('Завантаження CIFAR-10 (потрібне інтернет-з'єднання)...');  
    cifar10Dataset;  
end
```

2. Після завантаження даних створіть *ImageDatastore* для роботи з CIFAR-10 у форматі MATLAB.

```
% Створення ImageDatastore для CIFAR-10
imds = imageDatastore(cifar10Data, ...
    'IncludeSubfolders', true, ...
    'LabelSource', 'foldernames');
```

Спосіб 2: Використання *matlab.io.datastore.DsFileSet*

Якщо у вас є архів *cifar-10-matlab.tar.gz*, його можна завантажити вручну з офіційного сайту CIFAR-10 і розпакувати в папці робочої директорії MATLAB. Далі можна створити *ImageDatastore* аналогічно, як описано вище.

Спосіб 3: Використання *deepLearningImportExample* (вбудована функція в *Deep Learning Toolbox*)

Якщо у вас встановлений *Deep Learning Toolbox*, ви також можете спробувати скористатися наступною командою:

```
cifar10Data = deepLearningImportExample;
imds = imageDatastore(cifar10Data, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
```

Після цього набір *imds* буде містити дані CIFAR-10, які можна використовувати для навчання та тестування нейронної мережі.

Хід роботи

Завдання: Створити, навчити та протестувати згорткову нейронну мережу для класифікації зображень у форматі JPEG з різним рівнем стиснення: 10%, 30%, 50%, 70% та 90%. Провести аналіз впливу рівня стиснення на точність класифікації.

1) Завантаження та розпакування набору даних CIFAR-10

Скачуємо архів *cifar-10-matlab.tar.gz* та розпаковуємо його у папку з проектом, наприклад, за допомогою функції *untar* у MATLAB:

```
if ~exist(cifar10DataFolder, 'dir')
    fprintf('🔄 Розпаковка архіву CIFAR-10...\n');
    untar(cifar10Archive, cifar10DataFolder);
```

```

fprintf('✅ CIFAR-10 розпаковано!\n');
else
    fprintf('📁 CIFAR-10 вже було розпаковано.\n');
end

```

У цьому коді змінна *cifar10Archive* показує шлях, куди було завантажено архів CIFAR-10, а *cifar10DataFolder* визначає місце, куди ви хочете розпакувати набір даних (бажано у робочу директорію проекту), наприклад:

```

cifar10Archive = 'C:\Users\Downloads\cifar-10-matlab.tar.gz'; % Шлях до архіву
cifar10DataFolder = 'C:\Users\Program Files \MATLAB\Lab5\cifar-10-batches-mat'; % місце
для розпакування

```

У результаті у папці *cifar-10-batches-mat* з'являться файли MATLAB (.mat), які містять зображення та мітки.

2) Створення функції *helperCIFAR10Data* для завантаження CIFAR-10 у MATLAB

Створюємо окремий файл з функцією *helperCIFAR10Data*, що читає всі .mat файли з набору CIFAR-10 та завантажує навчальні та тестові дані у MATLAB. У якості вхідного параметру (*cifar10DataFolder*), функція приймає шлях до розпакованого CIFAR-10. На виході ми маємо отримати наступні змінні:

- *trainingImages* — 4D масив (32x32x3x50000) навчальних зображень;
- *trainingLabels* — категоріальні мітки (50 000 міток);
- *testImages* — 4D масив (32x32x3x10000) тестових зображень;
- *testLabels* — категоріальні мітки (10 000 міток).

Можливий приклад реалізації функції *helperCIFAR10Data*:

```

function [trainingImages, trainingLabels, testImages, testLabels] =
helperCIFAR10Data(cifar10DataFolder)

% Імена файлів навчальних batch-ів
batchFiles = ["data_batch_1.mat", "data_batch_2.mat", ...
    "data_batch_3.mat", "data_batch_4.mat", "data_batch_5.mat"];

% Ініціалізація масивів
trainingImages = zeros(32, 32, 3, 50000, 'uint8'); % 50 000 навчальних зображень

```

```

trainingLabels = strings(50000, 1);

% Завантаження навчальних даних
index = 1;
for i = 1:length(batchFiles)
    batchPath = fullfile(cifar10DataFolder, batchFiles(i));
    batchData = load(batchPath);

    % CIFAR-10 зберігає зображення у вигляді [3072xN] (де 3072 = 32*32*3)
    images = reshape(batchData.data', 32, 32, 3, []);
    labels = batchData.labels + 1; % MATLAB використовує 1-based індексацію

    % Збереження у загальний масив
    numImages = size(images, 4);
    trainingImages(:, :, :, index:index+numImages-1) = images;
    trainingLabels(index:index+numImages-1) = categorical(labels);

    index = index + numImages;
end

% Завантаження тестових даних
testFile = fullfile(cifar10DataFolder, 'test_batch.mat');
testData = load(testFile);
testImages = reshape(testData.data', 32, 32, 3, []);
testLabels = categorical(testData.labels + 1);

fprintf('✅ CIFAR-10 успішно завантажено!\n');
end

```

3) Перевірка завантажених даних

Після завантаження даних, перевіримо їх відобразивши кілька випадкових зображень з CIFAR-10 (рис. 5.1)

```

% Візуалізація 10 рандомних зображень
figure;

```

```

perm = randperm(50000, 10);
for i = 1:10
    subplot(2,5,i);
    imshow(trainingImages(:,:,perm(i)));
    title(char(trainingLabels(perm(i))));
end

```

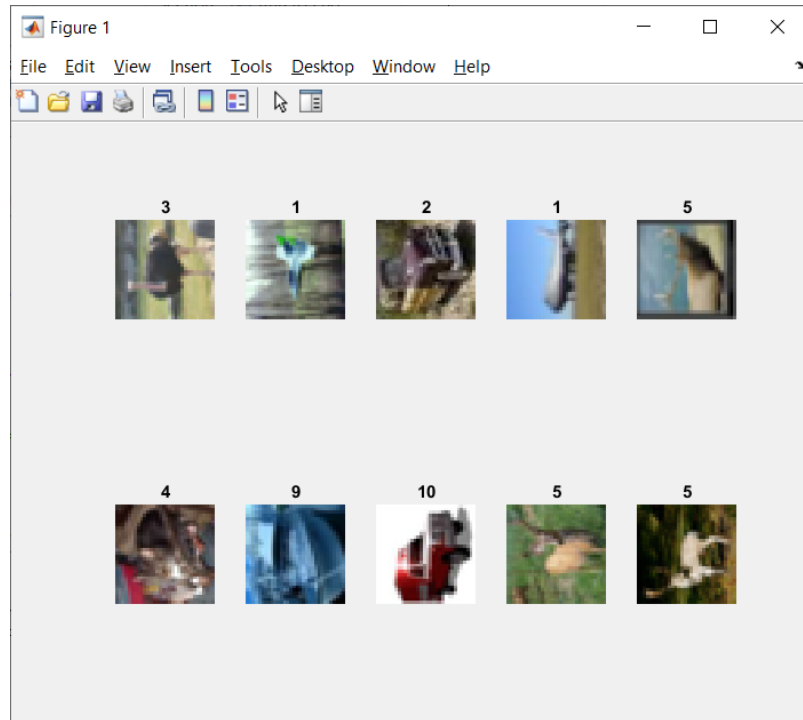


Рисунок 5.1 — Випадкові зображення з набору CIFAR-10

4) Стиснення зображень із різним рівнем якості

Копії кожного зображення у форматі JPEG слід зберегти в окремі папки з різними рівнями стиснення: 10%, 30%, 50%, 70% та 90%.

```

compressionLevels = [10, 30, 50, 70, 90]; % Рівні стиснення JPEG
% Ім'я папка для збереження стиснутих зображень
compressedFolder = 'compressed_images';
% Переконаємося, що папка існує
if ~exist(compressedFolder, 'dir')
    mkdir(compressedFolder);
end

% Створюємо підпапки для кожного рівня стиснення і класу
uniqueLabels = unique(trainingLabels); % Отримуємо список класів

```

```

% Створюємо підпапки для кожного рівня стиснення
for i = 1:numel(compressionLevels)
    for j = 1:numel(uniqueLabels)
        levelFolder = fullfile(compressedFolder, ['JPEG_' num2str(compressionLevels(i))],
char(uniqueLabels(j)));
        if ~exist(levelFolder, 'dir')
            mkdir(levelFolder);
        end
    end
end

% Стискання та збереження зображень у JPEG з різними рівнями якості
fprintf('🔄 Починаємо стиснення зображень...\n');
parfor i = 1:size(trainingImages, 4)
    img = trainingImages(:,:,i);
    label = char(trainingLabels(i)); % Отримуємо мітку класу

    for j = 1:numel(compressionLevels)

        % Шлях для збереження (JPEG_X/КЛАС/image_X.jpg)
        filePath = fullfile(compressedFolder, ['JPEG_' num2str(compressionLevels(j))], label, ...
            ['image_' num2str(i) '.jpg']);

        % Збереження у JPEG з різною якістю
        imwrite(img, filePath, 'JPEG', 'Quality', compressionLevels(j));
    end
end

fprintf('✅ Усі зображення стиснуті та збережені!\n');

У результаті у директорії compressed_images створено 5 підпапок (рис. 5.2), куди було збережено стиснуті зображення у 5 рівнях якості (10%, 30%, 50%, 70%, 90%).

```

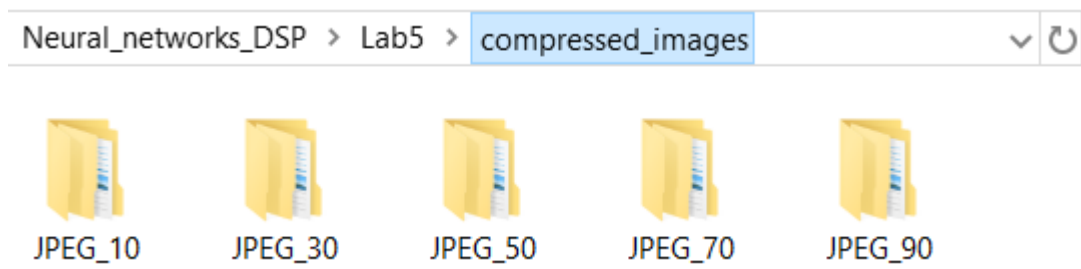


Рисунок 5.2 — Каталог директорії compressed_images

5) Завантаження стиснутих зображень у MATLAB

Тепер завантажмо стиснуті зображення як окремі набори *ImageDatastore*:

```

imds10 = imageDatastore(fullfile(compressedFolder, 'JPEG_10'), 'IncludeSubfolders',
true, 'LabelSource', 'foldernames');
imds30 = imageDatastore(fullfile(compressedFolder, 'JPEG_30'), 'IncludeSubfolders',
true, 'LabelSource', 'foldernames');
imds50 = imageDatastore(fullfile(compressedFolder, 'JPEG_50'), 'IncludeSubfolders',
true, 'LabelSource', 'foldernames');
imds70 = imageDatastore(fullfile(compressedFolder, 'JPEG_70'), 'IncludeSubfolders',
true, 'LabelSource', 'foldernames');
imds90 = imageDatastore(fullfile(compressedFolder, 'JPEG_90'), 'IncludeSubfolders',
true, 'LabelSource', 'foldernames');
```

Для перевірки результатів, аналогічно до *пункту 3*, виведіть на екран 10 випадкових стиснутих зображень, наприклад, з якістю 30%.

6) Налаштування нейронної мережі та опцій навчання

Створимо згорткову нейронну мережу для класифікації зображень із різним рівнем стиснення. Кількість фільтрів та шарів залежить від обчислювальних ресурсів та складності задачі. Обираємо індивідуальні параметри для першого та другого згорткових шарів за таблицею 5.1.

Приблизний вигляд архітектури може бути наступний:

```

layers = [
    imageInputLayer([32 32 3])

    convolution2dLayer(3, 16, 'Padding', 'same')
    batchNormalizationLayer
```

```

reluLayer
maxPooling2dLayer(2, 'Stride', 2)

convolution2dLayer(3, 32, 'Padding', 'same')
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2, 'Stride', 2)

fullyConnectedLayer(10)
softmaxLayer
classificationLayer
];

```

Таблиця 5.1. Індивідуальні параметри для побудови нейронної мережі

№ варіанту	Згортковий шар: <i>фільтри</i> × <i>ядро</i>		Шар dropout (після 2-го ReLU)
	Conv1	Conv2	
1, 11, 21, 31, 41	16 × 3×3	32 × 3×3	0.00
2, 12, 22, 32, 42	16 × 3×3	32 × 3×3	0.20
3, 13, 23, 33, 43	24 × 3×3	48 × 3×3	0.00
4, 14, 24, 34, 44	24 × 5×5	48 × 3×3	0.20
5, 15, 25, 35, 45	32 × 5×5	64 × 3×3	0.20
6, 16, 26, 36, 46	16 × 3×3	32 × 3×3	0.00
7, 17, 27, 37, 47	16 × 3×3	32 × 3×3	0.20
8, 18, 28, 38, 48	24 × 3×3	48 × 3×3	0.00
9, 19, 29, 39, 49	24 × 5×5	48 × 3×3	0.20
10, 20, 30, 40, 50	32 × 5×5	64 × 3×3	0.20

Налаштуємо параметри навчання за таблицею 5.2, наприклад:

```

options = trainingOptions('adam', ...
    'InitialLearnRate', 0.001, ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 64, ...
    'Shuffle', 'every-epoch', ...
    'Verbose', false, ...
    'Plots', 'training-progress');

```

Таблиця 5.2. Індивідуальні параметри для навчання нейронної мережі

<i>№ варіанту</i>	<i>Швидкість навчання</i>	<i>Кількість epoch</i>	<i>Розмір міні-партії</i>
1, 11, 21, 31, 41	1e-3	10	16
2, 12, 22, 32, 42	5e-4	12	32
3, 13, 23, 33, 43	1e-3	15	48
4, 14, 24, 34, 44	1e-3	18	64
5, 15, 25, 35, 45	5e-4	10	16
6, 16, 26, 36, 46	1e-3	12	32
7, 17, 27, 37, 47	1e-3	15	48
8, 18, 28, 38, 48	5e-4	18	64
9, 19, 29, 39, 49	1e-3	10	16
10, 20, 30, 40, 50	1e-3	12	32

7) Навчання та оцінка точності моделі

Навчимо нейронну мережу для кожного рівня стиснення та оцінимо точність на валідаційному наборі.

% Навчання моделі для різних рівнів стиснення

accuracyResults = zeros(numel(compressionLevels), 1);

for k = 1:numel(compressionLevels)

% Вибір набору даних згідно з рівнем стиснення

switch compressionLevels(k)

case 10, imdsTrain = imds10;

case 30, imdsTrain = imds30;

case 50, imdsTrain = imds50;

case 70, imdsTrain = imds70;

case 90, imdsTrain = imds90;

end

% Розподіл на навчальний і тестовий набори

[imdsTrainSplit, imdsValidationSplit] = splitEachLabel(imdsTrain, 0.8, 'randomized');

% Навчання нейронної мережі

trainedNet = trainNetwork(imdsTrainSplit, layers, options);

```

% Оцінка точності на валідаційному наборі
YPred = classify(trainedNet, imdsValidationSplit);
YValidation = imdsValidationSplit.Labels;
accuracy = sum(YPred == YValidation) / numel(YValidation);
accuracyResults(k) = accuracy;

disp(['Точність для JPEG ' num2str(compressionLevels(k)) '%: ' num2str(accuracy *
100) '%']);
end

```

8) Аналіз впливу рівня стиснення на точність

Порівняємо точність моделі при різних рівнях стиснення зображень.

```

% Графічне представлення точності
figure;
bar(compressionLevels, accuracyResults * 100);
xlabel('Рівень стиснення JPEG (%)');
ylabel('Точність (%)');
title('Залежність точності класифікації від рівня стиснення');

```

9) Тестування моделі на новому зображенні

Після завершення навчання необхідно протестувати мережу на нових стиснутих зображеннях, щоб перевірити точність класифікації.

Завантажуємо нове зображення, якщо воно знаходиться у директорії проекту, то у якості аргумента функції ***imread***, потрібно передати тільки ім'я зображення (наприклад, *new_image.jpg*), якщо ж зображення знаходиться в іншому місті — вказати повний шлях до зображення.

```

newImage = imread('new_image.jpg');
newImageCompressed = imresize(newImage, [32 32]);
imwrite(newImageCompressed, 'temp.jpg', 'JPEG', 'Quality', 30); % Стиснення до 30%
compressedTestImage = imread('temp.jpg');
compressedTestImage = imresize(compressedTestImage, [32 32]);

% Класифікація
predictedLabel = classify(trainedNet, compressedTestImage);

```

```
imshow(compressedTestImage);  
title(['Передбачений клас: ', char(predictedLabel)]);
```

10) Згідно з додатком А, *оформіть звіт* за виконаною лабораторною.

Контрольні питання

1. Що таке стиснення зображень, і як воно впливає на точність класифікації?
2. Які основні формати стиснення зображень ви використовували в лабораторній роботі?
3. Як змінюється точність класифікації при різних рівнях стиснення зображень?
4. Які переваги дає використання згорткових нейронних мереж (CNN) для класифікації стиснутих зображень?
5. Як працює функція `imageDatastore` для роботи з великими наборами зображень у MATLAB?
6. Як впливає рівень стиснення JPEG на якість класифікації зображень?
7. Який рівень стиснення є оптимальним для забезпечення високої точності розпізнавання?
8. Які основні етапи навчання нейронної мережі у MATLAB?
9. Як оцінити точність моделі класифікації зображень?
10. Як за допомогою MATLAB можна тестувати модель на нових стиснутих зображеннях?
11. Як впливає вибір архітектури нейронної мережі на точність класифікації стиснутих зображень?
12. Які переваги використання MATLAB для обробки стиснутих зображень та їх класифікації?

Лабораторна робота №6

РОЗПІЗНАВАННЯ АУДІОФАЙЛІВ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ У MATLAB

Мета роботи

Ознайомитися з використанням нейронних мереж для класифікації та розпізнавання звуків з аудіофайлів, а також навчитися застосовувати інструменти MATLAB для попередньої обробки аудіо, екстракції ознак і побудови класифікаційних моделей.

Теоретичні відомості

Розпізнавання звуків і аудіофайлів є важливою задачею в області обробки сигналів, що вимагає використання різних методів машинного навчання, таких як нейронні мережі. Найбільш ефективними є згорткові нейронні мережі (CNN) і мережі з довгою короткостроковою пам'яттю (LSTM), оскільки вони можуть обробляти часо- та просторові залежності сигналів.

Аудіофайли представлені як часові сигнали, які містять змінні амплітуди звуку в часі. Однак для більш зручного аналізу аудіо часто піддаються попередній обробці, щоб виділити важливі характеристики. Цей процес включає в себе кілька етапів:

Попередня обробка аудіофайлів

а) Нормалізація даних

Нормалізація аудіосигналу дозволяє привести його амплітуду до стабільного діапазону, наприклад, від -1 до 1. Це допомагає уникнути варіацій, які можуть вплинути на точність класифікації та спричинити зміщення сигналу.

б) Фільтрація

Для поліпшення якості сигналу або для видалення небажаних частотних компонентів застосовуються різні види фільтрів: *високочастотні* (ФВЧ), *низькочастотні* (ФНЧ) або *смугові фільтри*. Вони допомагають покращити характеристики аудіо та підсилити ті частоти, що є важливими для подальшого аналізу.

с) Зменшення шуму

Існує кілька методів зменшення шуму в аудіосигналах.

- **Спектральне віднімання** дозволяє видалити шум, вирахувавши його спектр і віднявши його від зашумленого сигналу. Цей метод є ефективним для стаціонарних шумів, хоча може впливати на якість відновленого сигналу.

- **Вінеровська фільтрація** використовує статистичні властивості сигналу і шуму для мінімізації похибки між вихідним і очищеним сигналом. Це дозволяє досягти кращих результатів у зменшенні шуму, особливо для нестабільних шумів.

- **Адаптивна фільтрація** дозволяє фільтрувати шум в реальному часі, оцінюючи зміни в характеристиках сигналу та адаптуючи фільтр під поточні умови.

д) Видалення тиші

Видалення періодів тиші з аудіофайлів може зменшити витрати обчислювальних ресурсів, зосереджуючи аналіз на важливих частинах сигналу, де є активний звуковий контент.

е) Попереднє виділення частот

Попереднє підсилення високочастотних компонентів звуку сприяє покращенню співвідношення сигнал/шум, що дозволяє виділити важливі характеристики звуку. Високочастотні компоненти зазвичай мають краще співвідношення сигнал/шум, що робить їх більш ефективними для подальшого аналізу.

ф) Сегментація

Аудіосигнал ділиться на менші сегменти або кадри певної тривалості (зазвичай 15–30 мс), щоб аналізувати зміни в сигналі на коротких часових інтервалах. Це дозволяє краще зафіксувати зміни в сигналі та його варіації.

г) Особливості отримання спектру

Щоб отримати спектр аудіосигналу, що змінюється в часі, сигнал розбивається на фрейми, після чого до кожного сегмента застосовується віконна функція. Віконні функції зменшують спектральний витік, покращують роздільну

здатність та мінімізують різкі переходи між сусідніми частинами сигналу. Вибір віконної функції (наприклад, *Ханна* чи *Хеммінга*) залежить від вимог до точності в часовій або частотній області.

h) ***Швидке перетворення Фур'є (ШПФ)***

Для кожного сегмента аудіосигналу застосовується перетворення Фур'є для аналізу частотного вмісту. Вибір довжини ШПФ має вплив на спектральну роздільну здатність і обчислювальні витрати, де оптимальною довжиною є ступінь двійки, що дозволяє мінімізувати спектральний витік та досягти хороших результатів.

Виділення звукових ознак.

Виділення ознак з аудіосигналу (Audio Feature Extractor) — це процес вилучення значущої та релевантної інформації з необроблених аудіосигналів для того, щоби представити аудіо у такому вигляді, який можна буде обробити за допомогою алгоритмів машинного навчання. Ці ознаки є вхідними даними для нейронних мереж, що дозволяє їм навчатися на аудіоданих і робити прогнози на основі нових, раніше невідомих звуків.

У підході глибокого навчання, обробка ознак відбувається автоматично, що дозволяє уникнути ручного вилучення ознак. Для представлення аудіосигналів у вигляді, що підходить для нейронних мереж, зазвичай використовуються спектрограми, мел-спектрограми та мел-частотні кепстральні коефіцієнти (MFCC). Розглянемо кожен з цих методів, їхні переваги та недоліки.

- ***Спектрограма;***

Спектрограма є візуальним відображенням спектру частот звукового сигналу, що змінюється з часом (рис. 6.1). Вона отримується шляхом застосування швидкого перетворення Фур'є (ШПФ) до коротких часових сегментів сигналу. Спектрограма дозволяє детально побачити, як змінюється частотний склад звуку в часі, що є корисним для задач, які потребують часово-частотної інформації, таких як визначення висоти, тембру або ритму звуків.

Основний *недолік* спектрограм — це їхня висока розмірність, що потребує великої кількості пам'яті та обчислювальних ресурсів. Крім того, для їхнього

коректного використання потрібно ретельно налаштовувати співвідношення час-частота, що може бути складним завданням.

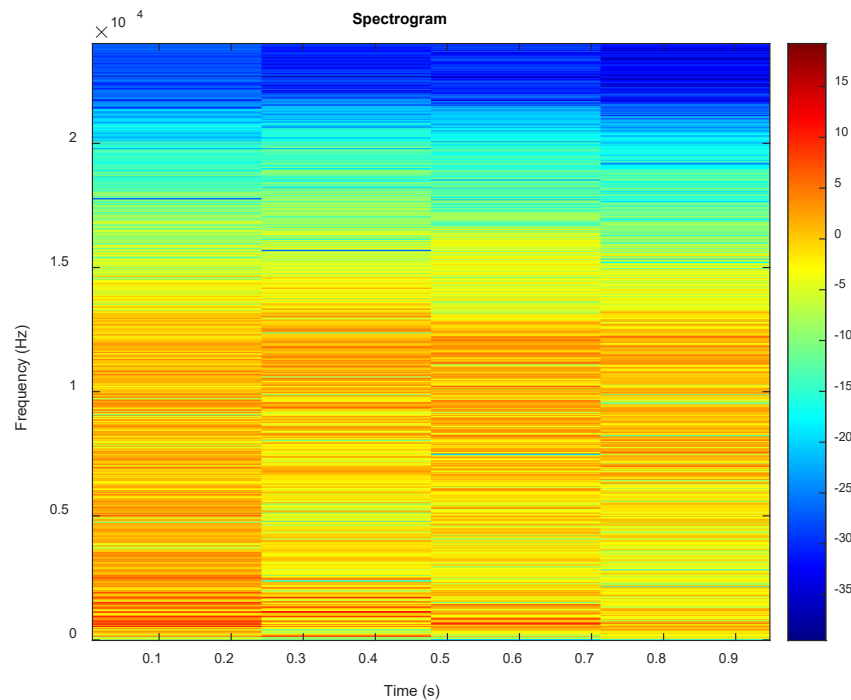


Рисунок 6.1 — Спектрограма аудіосигналу

- **Мел-спектрограма;**

Мел-спектрограма (рис 6.2) є модифікацією спектрограми, де частоти (f) перетворені в шкалу мела (m) за такою формулою:

$$m = 2595 \cdot \lg \left(1 + \frac{f}{700} \right)$$

Шкала мела краще відповідає тому, як людське вухо сприймає звук. Вона дає більшу точність для частот, які є важливими для сприйняття, та знижує значення частот, які малозначущі.

Мел-спектрограма зберігає більше спектральної інформації, ніж MFCC, і зазвичай використовується в задачах, де важливі дрібнозернисті частотні деталі, наприклад, у розпізнаванні мовних сигналів або музичних інструментів.

Однак, одним з недоліків є вища розмірність у порівнянні з MFCC, що може ускладнити обробку, особливо для великих наборів даних.

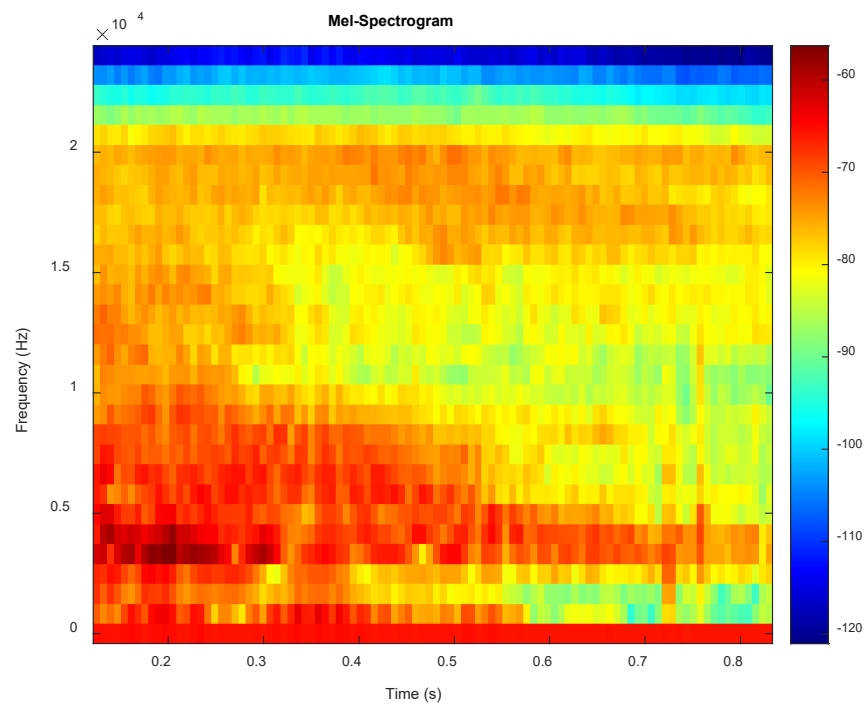


Рисунок 6.2 — Мел-спектрограма аудіосигналу

- **Мел-частотні кепстральні коефіцієнти (MFCC);**

MFCC — це коефіцієнти, що отримуються за допомогою косинусного перетворення логарифму спектра частотного сигналу (рис 6.3). Вони широко використовуються в задачах розпізнавання мови та класифікації аудіо. MFCC є стійкими до шуму, зберігають важливу спектральну інформацію, водночас зменшуючи розмірність простору ознак.

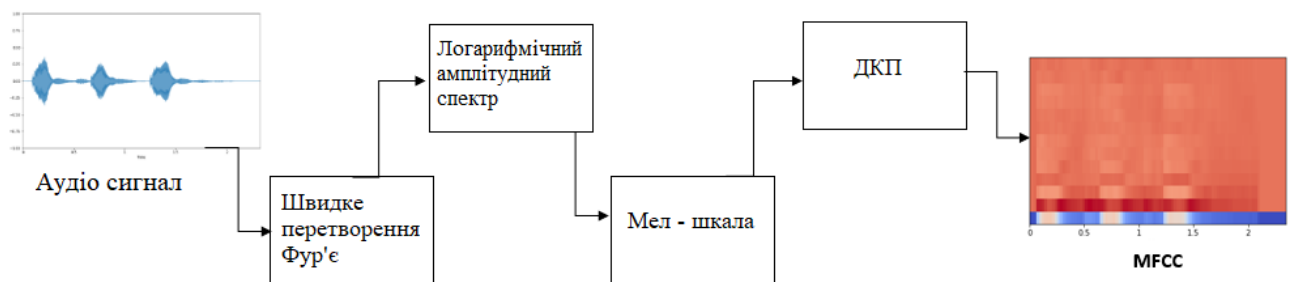


Рисунок 6.3 — Кроки для виділення MFCC з аудіосигналу

Найбільша *перевага* MFCC — це їх стійкість до варіацій джерела звуку та шуму. Вони добре підходять для задач, де потрібно зберегти основні характеристики звуку, при цьому зменшуючи розмірність ознак. Однак, вони можуть втрачати дрібні спектральні деталі, що є важливим для задач, де точна частотна інформація є критичною.

Для обчислення MFCC необхідно застосувати до логарифмічно масштабованих виходів мел-фільтрів дискретне косинусне перетворення (ДКП), що дозволяє виділити найбільш значущі спектральні компоненти.

Формула для обчислення MFCC:

$$X(k) = \sum_{n=1}^{N-1} x(n) \cos\left(\frac{\pi n}{C} \left(k - \frac{1}{2}\right)\right), \quad k = 0, 1 \dots N - 1$$

де $X(k)$ — вихідна послідовність коефіцієнтів ДКП;

$x(n)$ — вхідний аудіосигнал завдовжки N .

Оптимальна кількість коефіцієнтів MFCC може варіюватися в залежності від конкретної задачі. Зазвичай це визначається через експерименти, проводячи перевірку на валідаційному наборі даних.

Хід роботи

Завдання 1: Провести попередню обробку аудіофайлів: нормалізацію, обмеження частотного діапазону та видалення шуму.

1) Завантаження аудіофайлу

Спершу необхідно завантажити аудіофайл у форматі *WAV* або *MP3*. Для цього використовуємо функцію ***audioread***, яка дозволяє зчитувати аудіофайли з диску та завантажувати їх у робоче середовище MATLAB. У якості вхідного аргументу необхідно вказати шлях до розташування файлу, наприклад:

```
[audioData, fs] = audioread('C:\Users\Program Files\MATLAB\Lab6\audio1.wav');
```

У цьому коді ***audioData*** — це сам аудіосигнал, а ***fs*** — частота дискретизації (кількість зчитуваних зразків на секунду).

2) Візуалізація аудіосигналу

Після завантаження аудіофайлу можна візуалізувати його, щоб побачити зміну амплітуди сигналу в часі.

```
t = (0:length(audioData)-1) / fs;  
figure;  
plot(t, audioData);  
xlabel('Час (с)');
```

```
ylabel('Амплітуда');  
title('Аудіосигнал');
```

3) Обробка сигналу

У якості прикладу, зробимо нормалізацію аудіосигналу до діапазону від -1 до 1, щоб уникнути спотворень через варіації амплітуди.

```
audioData = audioData / max(abs(audioData));
```

Завдання 2: Виконати екстракцію ознак з аудіофайлів

1) Обчислення спектрограми

Спектрограма дозволяє візуалізувати частотний склад звукового сигналу, що змінюється з часом. Для її побудови застосуємо функцію *spectrogram*.

```
win = hamming(512); % Вікно Хеммінга  
[S, F, T] = spectrogram(audioData, win, 256, 512, fs);
```

```
% Візуалізація спектрограми  
figure;  
imagesc(T, F, 20*log10(abs(S)));  
axis xy;  
xlabel('Час (с)');  
ylabel('Частота (Гц)');  
title('Спектрограма аудіосигналу');  
colorbar;
```

2) Обчислення Мел-частотних кепстральних коефіцієнтів

MFCC дозволяють отримати узагальнене представлення звуку, що стійке до шуму. Для їх розрахунку застосуємо функцію *mfcc*.

```
coeffs = mfcc(audioData, fs);  
imagesc(coeffs');  
axis xy;  
xlabel('Час (с)');  
ylabel('Коефіцієнти MFCC');  
title('MFCC');
```

Завдання 3: Побудувати та навчити нейронну мережу класифікувати аудіо файли.

1) Завантажити набір аудіофайлів у MATLAB

У якості набору даних будемо використовувати «*Google Speech Commands Dataset*» — це набір аудіо даних з розмовними словами, призначений для тренування та оцінки систем розпізнавання ключових слів.

Для завантаження файлів напишемо невеличкий скріпт на кшталт такого:

```
dataFolder = 'C:\Users\Program Files\MATLAB\AudioDataStore\google_speech'; %  
місце розташування набору даних
```

```
ads = audioDatastore(dataFolder, ...
```

```
    'IncludeSubfolders', true, ... % Завантажуємо всі підпапки
```

```
    'LabelSource', 'foldernames'); % Використовуємо назви підпапок як мітки класів
```

```
% Розподіл на 80% тренувальних та 20% тестових даних
```

```
[trainAds, testAds] = splitEachLabel(ads, 0.8, 'randomized');
```

```
disp([' Навчальних файлів: ', num2str(numel(trainAds.Files))]);
```

```
disp([' Тестових файлів: ', num2str(numel(testAds.Files))]);
```

```
% Екстракція MFCC для навчального та тестового наборів
```

```
trainFeatures = extractFeatures(trainAds);
```

```
testFeatures = extractFeatures(testAds);
```

```
% Перетворюємо у формат 4D (height × width × channels × samples)
```

```
trainFeatures = cat(4, trainFeatures{:});
```

```
testFeatures = cat(4, testFeatures{:});
```

2) Створення архітектури нейронної мережі

Для побудови моделі може використовуватись нейронна мережа на базі *CNN* або *LSTM* — рекурентної нейронної мережі, яка має здатність зберігати інформацію про довготривалі залежності в даних. У цьому прикладі ми застосуємо просту CNN для класифікації аудіосигналів.

Мережа буде складатись з кількох шарів: вхідного шару, згорткових шарів, шару активації, шару підвибірки, повнозв'язного шару, шару softmax і класифікаційного шару. Індивідуальні параметри обрати з таблиці 5.1.

```
layers = [  
    imageInputLayer([size(trainFeatures,1) size(trainFeatures,2) 1]) % Вхідний шар  
    (розмір MFCC)  
  
    convolution2dLayer(3, 16, 'Padding', 'same') % Згортковий шар  
    batchNormalizationLayer  
    reluLayer  
    maxPooling2dLayer(2, 'Stride', 2)  
  
    convolution2dLayer(3, 32, 'Padding', 'same') % Другий згортковий шар  
    batchNormalizationLayer  
    reluLayer  
    maxPooling2dLayer(2, 'Stride', 2)  
  
    fullyConnectedLayer(numel(unique(trainLabels))) % Визначаємо кількість класів  
    автоматично  
    softmaxLayer  
    classificationLayer  
];
```

3) *Налаштування параметрів навчання* (табл. 5.2):

```
options = trainingOptions('adam', ...  
    'InitialLearnRate', 0.001, ...  
    'MaxEpochs', 10, ...  
    'MiniBatchSize', 64, ...  
    'Shuffle', 'every-epoch', ...  
    'Verbose', false, ...  
    'Plots', 'training-progress');
```

Тепер мережу можна навчити мережу на екстрагованих ознаках аудіо та відповідних етикетках (мітках).

```
trainedNet = trainNetwork(trainFeatures, trainLabels, layers, options);
```

Завдання 4: Оцінити точність роботи нейронної мережі на валідаційному наборі даних.

```
% Класифікація тестових даних
YPred = classify(trainedNet, testFeatures);

% Обчислення точності
accuracy = sum(YPred == testLabels) / numel(testLabels);
disp(['🎯 Точність моделі: ', num2str(accuracy * 100), '%']);
```

Завдання 5: Провести тестування моделі на нових аудіофайлах.

1) Завантаження нового аудіофайлу

Для тестування мережі використовуємо новий аудіофайл, який мережа не бачила в процесі навчання.

```
[newAudio, fs] = audioread('new_audio_file.wav');
newCoeffs = mfcc(newAudio, fs);
```

Якщо необхідно, то перетворіть коефіцієнти вхідного файлу у потрібний формат, наприклад, таким чином:

```
newCoeffs = reshape(newCoeffs, [size(newCoeffs,1), size(newCoeffs,2), 1, 1]);
```

2) Класифікуємо новий аудіофайл

```
newPred = classify(trainedNet, newCoeffs);
disp(['🔊 Передбачений клас: ', char(newPred)]);
```

Завдання 6: Відповідно до вимог у додатку А, оформіть звіт з висновками по роботі.

Контрольні питання

- 1) Що таке аудіосигнал і які основні параметри його характеризують?
- 2) Для чого виконується нормалізація аудіосигналу перед подачею на вхід нейронної мережі?
- 3) Які методи фільтрації аудіосигналів ви знаєте та в яких випадках вони застосовуються?
- 4) Що таке спектрограма і як вона обчислюється?

- 5) Поясніть сутність мел-частотних кепстральних коефіцієнтів (MFCC) та їх переваги.
- 6) Які типи ознак найчастіше використовуються для задач розпізнавання звуку?
- 7) Чим відрізняється мел-спектрограма від звичайної спектрограми?
- 8) Яку архітектуру нейронної мережі доцільно використовувати для класифікації аудіосигналів і чому?
- 9) Які основні шари згорткової нейронної мережі використовуються у задачах обробки аудіо?
- 10) Як у MATLAB можна візуалізувати частотну структуру звукового сигналу?
- 11) Які етапи включає процес навчання нейронної мережі у MATLAB?
- 12) Як визначити точність класифікації аудіофайлів нейронною мережею?
- 13) Яким чином відбувається тестування навченої моделі на нових аудіофайлах?
- 14) Які виклики можуть виникати при розпізнаванні аудіофайлів у реальних умовах (шум, тиша, перешкоди)?
- 15) Які особливості має попередня обробка звукових сигналів у задачах машинного навчання?

Лабораторна робота №7

ОСНОВИ НЕЙРОННИХ МЕРЕЖ У MATLAB

Мета роботи

Ознайомитися з принципами побудови штучних нейронних мереж у середовищі MATLAB, навчитися створювати одношарові нейронні мережі, налаштовувати їх архітектуру, здійснювати процес навчання, а також аналізувати точність апроксимації математичних функцій залежно від кількості нейронів у прихованому шарі.

Теоретичні відомості

Штучні нейронні мережі (ШНМ) — це параметричні математичні моделі, які натхнені біологічними нейронними мережами. Вони складаються з кількох нейронів, організованих у шари: вхідний шар, один або більше прихованих шарів та вихідний шар. Кожен нейрон у мережі отримує вхідні дані, застосовує лінійне перетворення та нелінійність для їхнього оброблення і передає результат далі. Поєднання таких перетворень дозволяє вирішувати різноманітні завдання, такі як класифікація, апроксимація функцій, розпізнавання образів, прогнозування та інші, з якими класичні алгоритми часто не справляються.

У практиці MATLAB це реалізується двома підходами. **Високорівневий** підхід має вже готові обгортки (*patternnet*, *feedforwardnet*) і функцію *trainNetwork* з описом шарів. А **низькорівневий** використовує кастомні цикли: *dlnetwork* + *автоматичне диференціювання* (*custom training loop*) для повного контролю.

Основні типи нейронних мереж:

Одношарова нейронна мережа складається лише з одного прихованого шару, що має обмежену здатність вирішувати задачі з лінійною або нелінійною апроксимацією. Вона застосовується до простих задач, де вхідні дані мають лінійні або прості нелінійні взаємозв'язки.

Багатшарова ШНМ (MLP) — це нейронна мережа з кількома прихованими шарами і здатна моделювати складні нелінійності, добре працює з табличними й агрегованими ознаками.

Вона є більш потужною на відміну від одношарової, тому і здатна розв'язувати задачі, що включають розпізнавання образів, мовлення, складні моделі прогнозування тощо. Багатошарові мережі можуть навчатися на великих наборах даних і оптимізувати кілька нелінійних функцій одночасно.

На практиці для статичних даних (без часової, просторової структур) MLP — базовий вибір, а для послідовностей та зображень зазвичай беруть спеціалізовані архітектури (RNN/CNN), але базові принципи навчання ті самі.

Основними елементами штучних нейронних мереж є:

1) Нейрон з вхідними даними, які множаться на ваги, до яких додається зміщення. Після цього результат проходить через активаційну функцію.

2) Активаційна функція є нелінійною функцією, яка визначає вихід нейрона на основі його входу. Без цієї функції нейронна мережа була б лінійною, що обмежило б її здатність вирішувати складні задачі.

З минулих робіт ви вже знаєте про функції активації. А для виконання цієї роботи достатньо знати, що *ReLU*-подібні функції пришвидшують збіжність глибоких моделей, *tanh* підходить для центрованих даних, а *сигмоїда* — для ймовірнісних виходів у бінарних задачах.

3) Функція втрат (*Loss function*) — це міра того, наскільки передбачений результат мережі відрізняється від реального. Вона використовується для оцінки ефективності моделі. Для задач класифікації часто використовують крос-ентропію, а для задач регресії — середньоквадратичну помилку (MSE).

4) Алгоритм навчання визначає, як мережа оновлює свої параметри (ваги та зміщення) для мінімізації функції втрат. З *найпоширеніших* алгоритмів виділяють наступні:

- **Гرادієнтний спуск** мінімізує функцію втрат, обчислюючи її градієнт по відношенню до ваг і застосовує зміну ваг у напрямку зниження цього градієнта.

- **Зворотне поширення помилки** — це алгоритм, який обчислює помилку на кожному рівні мережі та передає її назад до вихідного шару для оновлення параметрів.

Процес навчання нейронної мережі.

Процес навчання нейронної мережі можна умовно поділити на кілька етапів, кожен з яких потрібно ретельно контролювати:

1) Ініціалізація: визначаються коректні початкові значення ваг і зміщень, що зменшують «залипання» в поганих режимах. Для ReLU доцільна He-ініціалізація (у MATLAB — властивість `WeightsInitializer` для шарів).

2) Введення даних: тензори подаються на вхід мережі та мають стабільну форму: для зображень — $H \times W \times C \times N$, для ознак — $F \times N$. Тип даних — здебільшого `single`.

3) Прямий прохід: вхідні дані обробляються через нейрони, і обчислюється вихід мережі шарами зверху вниз. Корисно контролювати розміри через `analyzeNetwork` або перевірку графа шарів.

4) Обчислення функції втрат: Порівнюється передбачений результат з реальним, і розраховується функція втрат. Також одразу визначаєте метрики, що відповідатимуть бізнес-цілі (точність, F1, RMSE).

5) Зворотне поширення помилки та оновлення: обчислюється, як кожен параметр (вага, зміщення) впливає на результат і відбувається оновлення параметрів.

6) Вибір оптимізатора та швидкості навчання: надто великий крок дає нестабільність, надто малий — повільну збіжність.

7) Повторення та валідація: процес повторюється на всіх даних кілька разів (epoch), поки мережа не досягне бажаного рівня точності.

Дані та передобробка у MATLAB.

1) Кодування міток. Для класифікації зручно використовувати *categorical* (єдиний алфавіт класів). Для старих API — one-hot матриці (напр., *ind2vec*).

2) Масштабування ознак. Z-score або приведення до [0;1] пришвидшує навчання і робить ваги інтерпретованішими.

3) Робота з датасетами. *imageDatastore*, *audioDatastore*, *fileDatastore* дозволяють потоково читати дані та застосовувати *transform/augmented·Datastore* для обробки «на льоту».

4) **Баланс класів.** При дисбалансі — стратифікований спліт, таргетна аугментація рідкісних класів, класові ваги (кастомний шар втрат або вибірка з вагами).

Практичні поради щодо відтворюваності та продуктивності.

- **Фіксація випадковості:** `rng(seed,'twister')` для повторюваних результатів.
- **GPU-прискорення:** `trainingOptions('ExecutionEnvironment','auto')` або явний `gpuArray` (за наявності CUDA-сумісного GPU).
- **Типи даних:** зберігайте тензори як *single*; непотрібне *double* — зайва пам'ять і час.
- **Перевірка графа:** `analyzeNetwork` перед стартом навчання допоможе знайти помилки у розмірах.
- **Чекпони:** `CheckpointPath` для довгих тренувань.

Хід роботи

Завдання 1: Створення та навчання одношарової нейронної мережі для апроксимації функції **$y=\sin(x)$** на інтервалі $x \in [0, 2\pi]$.

1. Згенерувати навчальні дані.

Створимо набір даних для навчання нейронної мережі. Крок дискретизації взяти з таблиці 7.1.

```
x = 0:0.1:2*pi; % Інтервал [0, 2π]
```

```
y = sin(x);
```

2. Створення архітектури нейронної мережі

Створимо одношарову нейронну мережу з **10 + № варіанту** нейронів у прихованому шарі за допомогою функції ***feedforwardnet***:

```
net = feedforwardnet(10);
```

3. Навчіть створену мережу на підготовлених даних:

```
net = train(net, x, y);
```

4. Тестування та візуалізація результатів.

Виконайте прогнозування на навчальному наборі та порівняйте отримані результати з реальними значеннями:

% Прогнозування результатів

y_pred = net(x);

% Візуалізація результатів

figure;

plot(x, y, 'b', x, y_pred, 'r--');

legend('Справжнє значення', 'Прогнозоване значення');

title('Апроксимація функції $\sin(x)$ ');

xlabel('x');

ylabel('y');

grid on;

5. Дослідити вплив кількості нейронів у прихованому шарі
6. Повторити завдання 1 для мереж з 5, 10 та 20 нейронами.
7. Побудувати графіки результатів для кожної конфігурації.

Таблиця 7.1 — Індивідуальні параметри

<i>Остання цифра № варіанту</i>	<i>Крок Δx</i>	<i>Алгоритм навчання $net.trainFcn$</i>	<i>Коментар</i>
0 або 5	0.05	trainlm	Швидкий на малих мережах
1 або 6	0.05	trainscg	Стабільний, економний по пам'яті
2 або 7	0.1	trainbr	Байєсівська регуляризація (валідування можна вимкнути)
3 або 8	0.1	trainrp	Resilient backprop, без LR
4 або 9	0.15	trainlm	

Завдання 2: Створити та навчити нейронну мережу для апроксимації функції $y=\cos(x)$ на інтервалі $x \in [0, 2\pi]$. Повторіть всі пункти завдання 1.

Завдання 3: Створити та навчити нейронну мережу для апроксимації квадратичної функції $y=x^2$ на інтервалі $x \in [-2, 2]$. Повторіть всі пункти завдання 1.

Завдання 4: Створити та навчити нейронну мережу для апроксимації експоненціальної функції $y=e^x$ на інтервалі $x \in [0, 2]$. Повторіть всі пункти завдання 1.

Завдання 5: Створити та навчити нейронну мережу для апроксимації логарифмічної функції $y=\ln(x)$ на інтервалі $x \in [1,10]$. Повторіть всі пункти завдання 1.

Завдання 6: Створити та навчити нейронну мережу для апроксимації функції Гауса $y=e^{-x^2}$ на інтервалі $x \in [-2,2]$. Повторіть всі пункти завдання 1.

Завдання 7: Використовуючи матеріал у додатку А, напишіть висновки та оформіть звіт за лабораторною роботою.

Контрольні питання

- 1) Що таке нейронна мережа та які її основні складові?
 - Які типи нейронних мереж існують?
 - Що таке вхідний шар, прихований шар і вихідний шар?
 - Яка роль активаційних функцій в нейронній мережі?
- 2) Як змінюється точність апроксимації при збільшенні кількості нейронів у прихованому шарі?
- 3) Як збільшення кількості нейронів впливає на обчислювальні ресурси?
- 4) Які функції активації можна використовувати для нейронів?
 - Які особливості у функцій *sigmoid*, *tanh*, *ReLU*?
 - Як вибір функції активації впливає на процес навчання?
- 5) Як працює алгоритм навчання нейронної мережі у MATLAB?
- 6) Який метод оптимізації використовує MATLAB для навчання?
- 7) Як працює зворотне поширення помилки?
- 8) Що таке функція втрат і як вона використовується?
- 9) Чому важливо аналізувати результати апроксимації за допомогою графіків?
- 10) Які метрики можуть бути корисні для аналізу точності?
- 11) Як змінюється точність апроксимації при використанні різних математичних функцій ($\sin$, $\cos$, x^2 тощо)?

- 12) Як складність функції впливає на потреби в кількості нейронів?
- 13) Які є переваги та обмеження одношарових нейронних мереж для апроксимації складних функцій?
- 14) Чому багатошарові нейронні мережі можуть бути кращими для деяких задач?
- 15) Як можна покращити точність апроксимації за допомогою оптимізації параметрів? Які параметри навчання потрібно змінювати?
- 16) Як правильний вибір оптимізатора впливає на навчання мережі?
- 17) Як можна збільшити ефективність нейронної мережі для задачі апроксимації в MATLAB? Які додаткові методи навчання можна застосувати?
- 18) Як вибір структури мережі впливає на швидкість навчання та точність?
- 19) Які помилки можуть виникати під час навчання нейронної мережі, і як їх можна уникнути?
- 20) Що таке перенавчання і як його запобігти?

Лабораторна робота №8

ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ У MATLAB ДЛЯ КЛАСИФІКАЦІЇ, ПРОГНОЗУВАННЯ ТА ВИЯВЛЕННЯ АНОМАЛІЙ

Мета роботи

Вивчення методів побудови і навчання нейронних мереж для розв'язання практичних задач аналізу даних, зокрема у таких сферах як класифікація, прогнозування, кластеризація та виявлення аномалій, з подальшим оцінюванням ефективності кожного підходу.

Теоретичні відомості

Штучні нейронні мережі (ШНМ) — це універсальні апроксиматори, які навчаються не лише «правильним відповідям», а й поданню даних (представленню ознак). В основі нейронних мереж лежить ідея моделювання функцій людського мозку через взаємодію простих обчислювальних одиниць — нейронів. Нейрони організовані в шари, які можуть бути з'єднані різними способами, формуючи різноманітні архітектури для розв'язання конкретних задач.

На відміну від класичних методів, де необхідно ретельно інженерити ознаки вручну, ШНМ уміють витягувати корисні структури самотійно, що особливо критично для:

- **Класифікації** (визначення приналежності даних до певних класів)
- **Прогнозування** (передбачення майбутніх значень на основі історичних даних)
- **Кластеризації** (розподіл даних на групи за схожими ознаками без використання міток класів)
- **Зменшення розмірності** (спрощення представлення даних)
- **Виявлення аномалій** (ідентифікація незвичайних або нетипових значень)

Для вирішення цих завдань в MATLAB застосовують різні архітектури нейронних мереж, що закладають різні припущення про структуру даних — це є *індукційне упередження* моделі.

1. Багатошаровий персептрон (MLP).

MLP (*Multi-Layer Perceptron*) — це тип нейронної мережі прямого поширення (*fully connected*), яка ефективна для статичних векторів ознак (табличні дані, агреговані дескриптори). Вона складається з декількох шарів: вхідного шару, одного або декількох прихованих шарів і вихідного шару. Кожен шар містить нейрони з нелінійними активаційними функціями, що дозволяє мережі апроксимувати складні функції.

Основними перевагами багатошарових персептронів є :

- **Гнучкість** — ефективний для широкого кола задач, включаючи класифікацію та регресію.

Варто зауважити, що для часових рядів MLP працює, якщо контекст уже «закодований» у вибраних ознаках (вікна, лаги, статистики).

- **Можливість апроксимації нелінійних функцій.**
- **Простота і швидке навчання.**

У якості методу навчання найчастіше використовується алгоритм зворотного поширення помилки (*backpropagation*) з градієнтним спуском.

2. Рекурентні нейронні мережі LSTM

LSTM (*Long Short-Term Memory*) — це різновид рекурентних нейронних мереж (RNN), які спеціалізуються на аналізі послідовних часових рядів, подій, аудіо та тексту. Головна особливість LSTM — здатність пам'ятати інформацію протягом тривалих проміжків часу завдяки спеціальним механізмам "воріт" (вхідних, вихідних та воріт забування).

Основні перевагами LSTM є:

- **Ефективність у задачі прогнозування часових рядів та аналізу послідовностей** (наприклад, мова, текст, аудіо).
- **Здатність зберігати довгострокові залежності** у даних, що робить їх більш точними для задач з великою кількістю інформації про минулі події.

- **Широкі використання для задач передбачення (forecasting)** та розпізнавання мови.

3. Самоорганізовані карти (SOM)

Самоорганізовані карти (Self-Organizing Maps) — нейронні мережі без вчителя, що використовуються для кластеризації та візуалізації даних. SOM класифікують вхідні дані шляхом відображення їх на двовимірну карту, зберігаючи топологічну структуру даних. У результаті подібні дані групуються.

З основних переваг SOM виділяють наступні:

- **Навчання без вчителя:** використовують конкурентне навчання, де нейрони "змагаються" за представлення вхідних даних (розуміння «ландшафту» даних, підозрілі кластери — кандидати на аномалії).

- **Візуалізація та кластеризація високовимірних даних.**

- **Ефективні для виявлення структури в неструктурованих даних.**

4. Автокодувальники (Autoencoders)

Автокодувальники (AE) — нейронні мережі, які використовують для зменшення розмірності та вилучення ключових ознак із даних. Вони складаються з *кодувальника (encoder)*, що стискає вхідні дані до компактного прихованого представлення, та *декодувальника (decoder)*, який відновлює дані з цього компактного представлення.

Якщо модель добре реконструює «нормальні» приклади, а на відхиленнях помилка різко зростає, ця помилка працює як сигнал аномалії. Також AE дають компактні латентні ознаки для подальшої класифікації/кластеризації.

Основними перевагами автокодувальників є:

- **Зменшення розмірності даних** зі збереженням найважливішої інформації.
- **Ефективне виявлення аномалій.**
- **Генерація нових даних та шумозаглушення.**

Виявлення аномалій.

Виявлення аномалій за допомогою нейронних мереж здійснюється через прогнозування наступних значень на основі попередніх. Якщо фактичне

значення сильно відрізняється від прогнозу, воно може бути класифіковане як аномальне.

Виділяють декілька підходів для виявлення аномалій:

1) **Із використання автокодувальника.** У MATLAB використовують базовий сценарій *trainAutoencoder* або будують мережу через *layerGraph* для більшої гнучкості. Аномалійний бал при цьому еквівалентний помилці реконструкції.

2) **Прогнозний підхід із застосуванням LSTM або MLP.** Він дає свій прогноз стосовного наступного значення. Аномалійний бал в цьому випадку — помилка передбачення.

3) **SOM-підхід** будується на алгоритмі аналізу карти (щільність «хітів», великі міжнейронні відстані на U-matrix) для пошуку нестандартних регіонів.

Для аномалій головна складність — **порог**. Тому на практиці важливо правильно обрати статистичну процедуру (квантиль або медіана + $k \cdot \text{MAD}$) та перевіряти стабільність на валідаційних наборах.

Для класифікації ж корисно перевірити калібрування ймовірностей (навіть проста температурна корекція може покращити інтерпретованість впевненості). У звіті важливі не лише цифри метрик, а й аналіз помилок: приклади «важких» класів/епізодів, здогадки про причини і пропозиції щодо покращення.

Хід роботи

Завдання 1: Класифікація даних за допомогою MLP

1) *Завантажити набір даних Iris.*

```
load fisheriris;  
inputs = meas';  
targets = full(ind2vec(grp2idx(species)'));
```

2) *Створити багат шарову перцептронну мережу (MLP) з двома прихованими шарами.* Індивідуальні параметри для налаштування мережі визначаємо по таблиці 8.1.

```
net = patternnet([10, 5]);  
net.trainParam.epochs = 100;
```

```
net.trainFcn = trainFcn; % 'trainlm' / 'trainscg' / 'trainrp' / 'trainbr'
```

- 3) Розділити дані у співвідношенні — 70% навчання, 15% валідація та 15% тестових.

```
net.divideParam.trainRatio = 0.7;
```

```
net.divideParam.valRatio = 0.15;
```

```
net.divideParam.testRatio = 0.15;
```

- 4) Провести навчання мережі.

```
[net, tr] = train(net, inputs, targets);
```

- 5) Оцінити якість моделі.

```
testX = inputs(:, tr.testInd);
```

```
testT = targets(:, tr.testInd);
```

```
testY = net(testX);
```

- 6) Візуалізувати результати класифікації.

```
figure;
```

```
plotconfusion(testT, testY);
```

```
title('Матриця плутанини для MLP');
```

- 7) Змінити кількість нейронів у прихованих шарах і перевірити вплив на точність.

Таблиця 8.1 — Індивідуальні параметри для MLP на Iris

<i>Остання цифра № варіанту</i>	<i>Приховані шари</i>	<i>net.trainFcn</i>
0 або 5	[8,4]	trainlm
1 або 6	[10,5]	trainscg
2 або 7	[12,6]	trainbr
3 або 8	[16,8]	trainrp
4 або 9	[12,8]	trainlm

Завдання 2: Прогнозування часових рядів за допомогою LSTM.

- 1) Згенерувати синтетичний часовий ряд. Величину шум σ у синусоїді беремо з таблиці 8.2.

```
% Генерація даних
```

```
x = 0:0.01:10;
```

```
data = sin(x) + sigma*randn(size(x));
```

```
X = data(1:end-1);
```

```
Y = data(2:end);
```

- 2) *Створити LSTM мережу та задати її параметри згідно з таблицею 8.2.*

```
% Створення LSTM мережі
```

```
layers = [sequenceInputLayer(1), lstmLayer(50), fullyConnectedLayer(1),  
regressionLayer];
```

```
options = trainingOptions('adam', 'MaxEpochs', 100, 'Verbose', 0);
```

- 3) *Провести навчання мережі.*

```
% Навчання
```

```
net = trainNetwork(num2cell(X), num2cell(Y), layers, options);
```

- 4) *Оцінити прогнозу здатність.*

```
% Прогнозування
```

```
YPred = predict(net, num2cell(X));
```

```
figure; plot(Y, 'b'); hold on; plot(cell2mat(YPred), 'r--'); legend('Реальні',  
'Прогнозовані');
```

- 5) *Змінити кількість нейронів у шарі LSTM та перевірити результат.*

Таблиця 8.2 — Індивідуальні параметри для LSTM при функції синус

<i>Остання цифра № варіанту</i>	<i>Шум σ у синусі</i>	<i>Кількість нейронів</i>	<i>Кількість epoch</i>
0 або 5	0,05	40	80
1 або 6	0,10	50	100
2 або 7	0,15	60	120
3 або 8	0,10	80	100
4 або 9	0,05	30	80

Завдання 3: Кластеризація даних за допомогою SOM.

- 1) *Згенерувати двовимірні випадкові дані.*

- 2) *Створити карту SOM.* Індивідуальні параметри обираємо за таблицею 8.3.

- 3) *Провести навчання мережі.*

- 4) *Виконати кластеризацію та візуалізувати результати.*

```
% Генерація даних
```

```
data = rand(2, 100);
```

```
% Створення SOM
```

```
net = selforgmap([5 5], [], [], topo, dist);
```

```
net.trainParam.epochs = 100;
```

```

net = train(net, data);
% Кластеризація
outputs = net(data);
classes = vec2ind(outputs);
figure; plotsompos(net, data);

```

5) *Змінити розмір SOM та перевірити вплив на кластеризацію.*

Таблиця 8.3 — Індивідуальні параметри для SOM

Остання цифра № варіанту	Розмір карти	Топологія	Відстань	Епохи
0 або 5	4×4	hextop	linkdist	200
1 або 6	5×5	gridtop	linkdist	200
2 або 7	6×6	hextop	dist	250
3 або 8	8×8	hextop	mandist	300
4 або 9	6×6	gridtop	linkdist	300

Завдання 4: Автокодувальник для зменшення розмірності.

1) *Згенерувати багатовимірні дані.*

```

% Генерація 10-вимірних випадкових даних
data = rand(100, 10)';

```

2) *Створити автокодувальник.*

3) *Виконати навчання.*

```

autoenc = trainAutoencoder(data, codeDim, ...
    'MaxEpochs',100,'Denoising',denoise, ... % видалення шуму
    'SparsityRegularization',sReg, ...        % регуляризація розрідженості
    'SparsityProportion',sProp, ...          % коефіцієнт розрідженості
    'L2WeightRegularization',l2);            % L2-регуляризація
% Отримання зменшених ознак
encodedData = encode(autoenc, data);

```

4) *Візуалізувати в 3D зменшені дані (рис. 8.1)*

```

figure;
scatter3(encodedData(1,:), encodedData(2,:), encodedData(3,:));
title('Зменшення розмірності за допомогою автокодувальника');
xlabel('Компонента 1');
ylabel('Компонента 2');

```

zlabel('Компонента 3');

grid on;

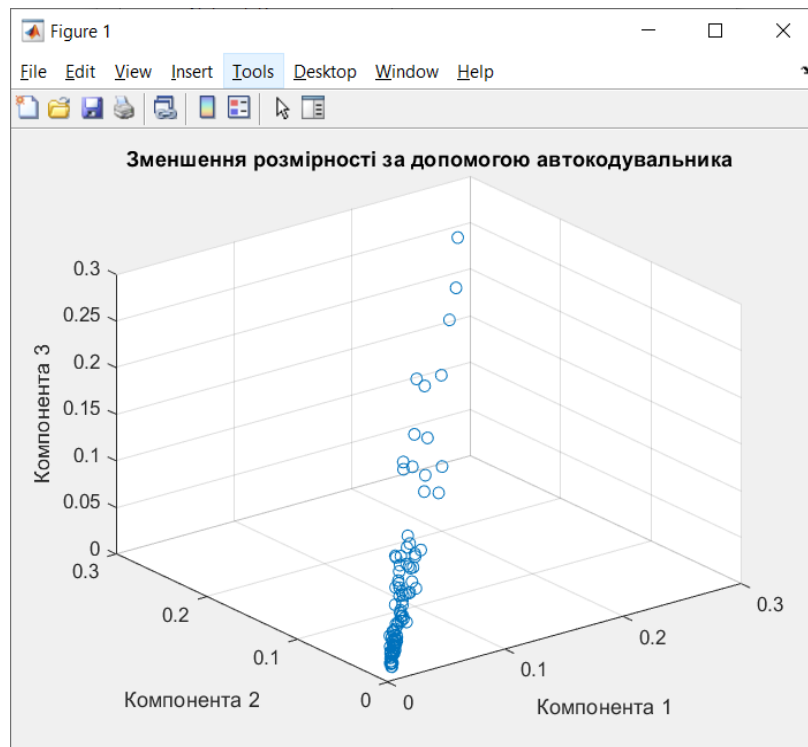


Рисунок 8.1 — Приклад 3D візуалізації зменшення розмірності даних

Таблиця 8.4 — Індивідуальні параметри для Автокодувальника

Остання цифра № варіанту	Розмір коду	Видалення шуму	Регуляриз. розріджен.	Коефіцієнт розріджен.	L2-рег.	Кількість epoch
0 або 5	2	false	0,00	—	0,004	60
1 або 6	3	true	0,10	0,05	0,002	70
2 або 7	5	false	0,15	0,1	0,001	80
3 або 8	3	true	0,20	0,1	0,002	90
4 або 9	2	false	0,10	0,05	0,004	80

Завдання 5: Виявлення аномалій за допомогою MLP.

1) *Згенерувати набір з аномаліями для вікна лагів $L > 1$ (табл.8.5).*

```
data = [randn(1, 100), 10*randn(1, 5)];
```

```
X = buffer(data(1:end-1), L, L-1, 'nodelay'); % L×N
```

```
Y = data(L+1:end); % 1×N
```

2) *Створити та навчити мережу. Кількість прихованих шарів так само обираємо за таблицею 8.5.*

% Створення MLP

- ```
net = feedforwardnet(H);
net = train(net, X, Y);
```
- 3) *Виконати прогнозування та обчислення помилок.*
- ```
pred = net(X);
errors = abs(pred - Y);
```
- 4) *Визначити аномальні значення (поріг обираємо з таблиці 8.5) .*
- ```
threshold = 2; % поріг
anomalies = find(errors > threshold);
```
- 5) *Візуалізувати результати (рис. 8.2)*
- ```
figure;
plot(data, 'b'); hold on;
plot(anomalies, data(anomalies), 'ro', 'MarkerSize', 8);
title('Виявлення аномалій');
legend('Дані', 'Аномалії');
```

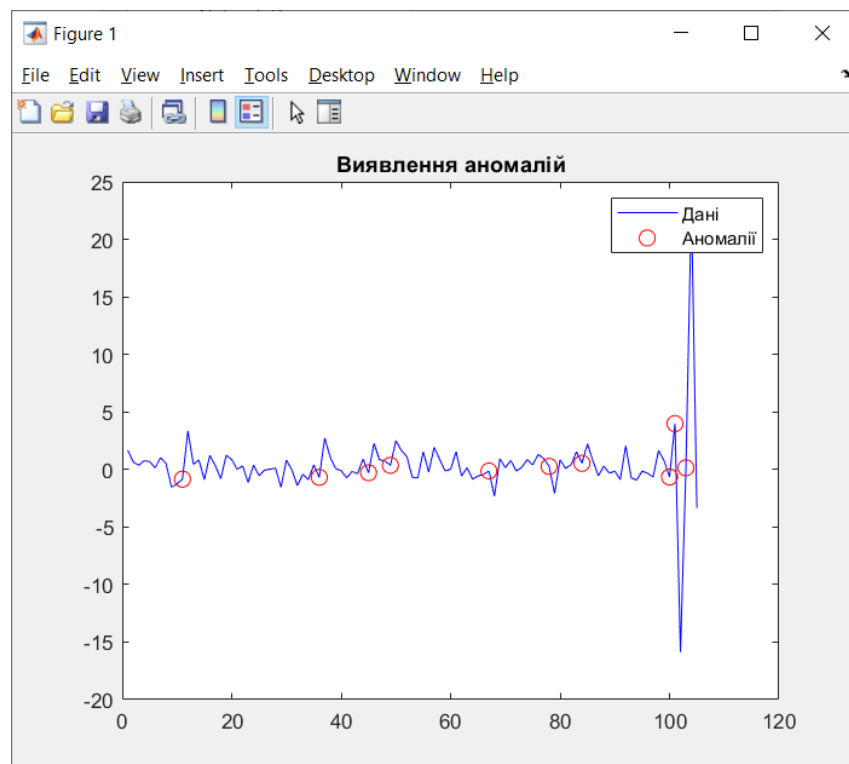


Рисунок 8.2 — Приклад візуалізації графіку з виявленими аномаліями

- 6) *Змінити параметри мережі та поріг для виявлення аномалій (взяти пару значень для порівняння).*
- 7) *Повторити кроки та візуалізувати результати.*

Таблиця 8.5 — Індивідуальні параметри для Автокодувальника

Остання цифра № варіанту	Вікно лагів, L	Кількість прихованих нейронів, H	Поріг аномалій
0 або 5	1	10	фіксовано 2
1 або 6	2	12	$\text{median}(\text{err}) + 3 * \text{mad}(\text{err}, 1)$
2 або 7	3	15	$\text{quantile}(\text{err}, 0.975)$
3 або 8	5	10	$\text{mean}(\text{err}) + 2.5 * \text{std}(\text{err})$
4 або 9	7	12	$\text{quantile}(\text{err}, 0.99)$

Завдання 6: Відповідно до структури у додатку А, оформіть звіт за виконаною роботою.

Контрольні питання

1. Що таке штучна нейронна мережа і які основні задачі вона вирішує?
2. Які типи нейронних мереж найчастіше використовують для аналізу даних?
3. Чим відрізняються навчання з учителем і навчання без учителя?
4. Що таке багатошаровий персептрон і які його особливості?
5. Які завдання найкраще вирішуються за допомогою MLP?
6. Як кількість нейронів і шарів впливає на точність роботи MLP?
7. Чому мережі типу LSTM ефективні для прогнозування часових рядів?
8. Які основні компоненти містить нейрон типу LSTM?
9. Як впливає кількість нейронів у шарі LSTM на точність прогнозування?
10. Що таке самоорганізована карта (SOM) і як вона працює?
11. Чим кластеризація за допомогою SOM відрізняється від інших методів кластеризації?
12. Як вибір розміру SOM впливає на результати кластеризації?
13. Що таке автокодувальник і які його основні компоненти?
14. Які завдання вирішуються за допомогою автокодувальників?
15. Як зміна кількості нейронів у прихованому шарі автокодувальника впливає на якість зменшення розмірності?
16. Які переваги використання нейронних мереж для виявлення аномалій порівняно з класичними статистичними методами?

Лабораторна робота №9

РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ У MATLAB

Мета роботи

Ознайомитися з основами методів розпізнавання об'єктів на зображеннях з використанням глибоких згорткових нейронних мереж (ЗНМ). Навчитися проводити попередню обробку зображень, будувати та налаштовувати ЗНМ-архітектури в середовищі MATLAB, тренувати моделі на стандартних наборах даних, оцінювати якість отриманих результатів та аналізувати вплив параметрів моделі на точність класифікації зображень.

Теоретичні відомості

Глибокі нейронні мережі є потужним інструментом для аналізу та розпізнавання образів, оскільки здатні виявляти складні та приховані взаємозв'язки у візуальних даних. Серед різноманітних типів нейронних мереж найбільш ефективними для класифікації зображень є згорткові нейронні мережі (Convolutional Neural Networks, CNN).

Архітектура згорткових нейронних мереж.

1. Згорткові шари (Convolutional Layers)

Згорткові шари застосовують набір фільтрів до вхідного зображення з метою виділення ознак. Кожен фільтр сканує зображення та створює карти ознак (feature maps). Наприклад, фільтри можуть визначати границі, текстури, форми або більш складні патерни.

Формула згортки:

$$(F * I)(x, y) = \sum_i \sum_j F(i, j) \cdot I(x - i, y - j)$$

де F — матриця фільтра, I — вхідне зображення.

2. Шари субдискреетизації (Pooling Layers)

Шари пулінгу зменшують розмірність вхідних карт ознак, вибираючи найбільш значущі елементи. Найпоширенішим є MaxPooling, який обирає

максимальне значення з локальної області. Це зменшує кількість параметрів і дозволяє мережі навчатися швидше та уникати перенавчання (overfitting).

3. Шари нормалізації (Batch Normalization Layers)

Шари нормалізації стабілізують процес навчання, нормалізуючи вихідні дані після згорткових шарів. Це значно прискорює процес навчання, зменшує чутливість моделі до вибору параметрів та допомагає запобігти перенавчанню.

4. Активаційні функції (Activation Functions)

Активаційні функції додають нелінійність до нейронних мереж, дозволяючи моделювати складніші взаємозв'язки:

- **ReLU (Rectified Linear Unit):**

$$f(x) = \max(0, x)$$

Швидко навчається і допомагає уникнути проблеми зникнення градієнту.

- **Softmax:**

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Використовується в останньому шарі мережі для перетворення виходів на ймовірності класів.

5. Повнозв'язні шари (Fully Connected Layers)

Це фінальні шари CNN, які здійснюють класифікацію на основі ознак, витягнутих згортковими шарами. Вони об'єднують усі витягнуті ознаки у вектор, який використовується для визначення приналежності зображення до певного класу.

Попередня обробка зображень

Перед подачею зображень у нейронну мережу здійснюється їх попередня обробка, яка включає:

- **Зміна розміру (Resizing):** приведення зображень до єдиного розміру.
- **Нормалізація:** приведення пікселів зображення до єдиного діапазону значень, наприклад $[0, 1]$.

- **Аугментація (Data Augmentation):** штучне розширення набору даних за допомогою поворотів, масштабування, зсувів, дзеркальних відображень для збільшення стійкості моделі.

Навчання CNN та оптимізація моделі

Процес навчання CNN включає мінімізацію функції втрат (наприклад, крос-ентропії) за допомогою оптимізаційних алгоритмів. Найчастіше використовують такі алгоритми:

- **SGD (Stochastic Gradient Descent):** поступово оновлює ваги у напрямку протилежному градієнту функції втрат.
- **Adam (Adaptive Moment Estimation):** адаптивний алгоритм оптимізації, що поєднує переваги SGD та методів адаптивного навчання, забезпечує швидке і стабільне навчання.

Параметри навчання CNN

При налаштуванні CNN важливі такі параметри:

- **Кількість епох (Epochs):**

Одна епоха відповідає одному проходженню всіх навчальних даних через мережу. Збільшення епох підвищує точність, але може призводити до перенавчання.

- **Розмір мініпартії (Batch Size):** кількість зображень, що обробляються одночасно перед оновленням параметрів мережі. Менший розмір може давати точніші результати, але повільніше навчання.

- **Початкова швидкість навчання (Learning Rate)** визначає, наскільки сильно змінюються ваги при кожному оновленні. Дуже велике значення може призвести до розходження моделі, а занадто мале — до повільного навчання.

Оцінка точності та продуктивності моделі

Після навчання модель оцінюється за допомогою різних метрик:

- **Точність (Accuracy)** — відношення правильно класифікованих зображень до загальної кількості зображень.

- **Матриця плутанини** (*Confusion Matrix*) — таблиця, яка показує кількість правильних і неправильних класифікацій, дозволяє визначити, які класи модель плутає між собою.

- **F1-score, Precision, Recall**: додаткові метрики, які враховують баланс між точністю та повнотою моделі, особливо важливі у випадках дисбалансу класів.

Типові задачі CNN в обробці зображень

CNN використовуються для вирішення різних задач:

- **Класифікація зображень** (визначення категорії зображення);
- **Об'єктна детекція** (визначення місця розташування і типу об'єкта на зображенні);
- **Сегментація зображень** (поділ зображення на семантичні області).

У лабораторній роботі основна увага зосереджена саме на задачі класифікації.

Хід роботи

Завдання 1: Підготовка та попередня обробка даних.

1) *Завантажити набір даних CIFAR-10 або підготувати власний набір зображень.*

У якості прикладу для виконання роботи використовується стандартний набір даних CIFAR-10, що містить 60000 кольорових зображень розміром 32×32 , які розподілені на 10 класів.

```
% Завантаження CIFAR-10
```

```
data = load('cifar10Data.mat');
```

```
imds = data.imds;
```

2) *Розподіл на тренувальні та тестові набори*

Розподіліть набір даних на навчальну (80%) та валідаційну (20%) вибірки для оцінки точності моделі.

```
[imdsTrain, imdsValidation] = splitEachLabel(imds, 0.8, 'randomized');
```

```
% Візуалізація даних
```

```
figure;
```

```
perm = randperm(10000, 20);
```

```

for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end

```

3) *Виконати попередню обробку, включаючи зміну розміру та нормалізацію.*

Для забезпечення стабільного навчання нейронної мережі перетворимо зображення у діапазон від 0 до 1.

```

% Функція для нормалізації
augmentedTrain = augmentedImageDatastore([32 32], imdsTrain,
'ColorPreprocessing', 'gray2rgb');
augmentedValidation = augmentedImageDatastore([32 32], imdsValidation,
'ColorPreprocessing', 'gray2rgb');

```

Завдання 2: Побудова та налаштування згорткової нейронної мережі.

- 1) *Створити модель CNN з кількома згортковими шарами.*
- 2) *Налаштувати параметри згорткових та повнозв'язних шарів.*

Для прикладу створимо згорткову нейронну мережу з двома згортковими блоками, які містять згорткові, нормалізаційні шари, функції активації ReLU та шари MaxPooling.

```

layers = [
    imageInputLayer([32 32 3])
    convolution2dLayer(3, 32, 'Padding', 'same') % 1-й згортковий шар
    batchNormalizationLayer % Шар нормалізації
    reluLayer % ReLU активація
    maxPooling2dLayer(2, 'Stride', 2) % Шар MaxPooling
    convolution2dLayer(3, 64, 'Padding', 'same') % 2-й згортковий шар
    batchNormalizationLayer % Шар нормалізації
    reluLayer % ReLU активація
    maxPooling2dLayer(2, 'Stride', 2) % Шар MaxPooling
    fullyConnectedLayer(10) % Повнозв'язний шар для 10 класів
    softmaxLayer % Шар softmax для класифікації
    classificationLayer % Класифікаційний шар
];

```

Зміна кількості фільтрів впливає на глибину представлення ознак, що може покращити точність класифікації, але збільшує витрати на обчислення.

Завдання 3: Навчання нейронної мережі.

1) Вибір параметрів навчання

Налаштуйте параметри для навчання CNN:

- ***InitialLearnRate***: початкова швидкість навчання.
- ***MaxEpochs***: кількість повних циклів навчання на всіх даних.
- ***MiniBatchSize***: кількість зображень, які одночасно обробляються мережею.

```
options = trainingOptions('adam', ...  
    'InitialLearnRate', 0.001, ...  
    'MaxEpochs', 15, ...  
    'MiniBatchSize', 64, ...  
    'Shuffle', 'every-epoch', ...  
    'ValidationData', augmentedValidation, ...  
    'ValidationFrequency', 30, ...  
    'Verbose', false, ...  
    'Plots', 'training-progress');
```

2) Виконати навчання нейронної мережі, використовуючи задані параметри та навчальний набір

```
trainedNet = trainNetwork(augmentedTrain, layers, options);
```

Під час навчання уважно стежте за графіком процесу навчання для визначення моменту перенавчання або стабілізації точності.

Завдання 4: Оцінка точності моделі.

1) Виконати класифікацію тестових зображень із валідаційного набору.

2) Обчислити точність моделі

```
YPred = classify(trainedNet, augmentedValidation);  
YValidation = imdsValidation.Labels;  
accuracy = sum(YPred == YValidation) / numel(YValidation);  
disp(['Точність моделі: ' num2str(accuracy * 100) '%']);
```

3) Побудувати матрицю плутанини.

```
figure; plotconfusion(YValidation, YPred);
```

Матриця плутанини дозволяє детально аналізувати, які класи модель плутає найбільше та відповідно корегувати архітектуру або навчання мережі.

Завдання 5: Тестування моделі на нових зображеннях.

Перевірте, як модель розпізнає нові зображення, що не входили до навчального або тестового набору.

1) Завантаження та підготовка нового зображення

```
newImage = imread('path_to_new_image.jpg');
```

```
newImage = imresize(newImage, [32 32]);
```

2) Виконати класифікацію та відобразити передбачений клас.

```
label = classify(trainedNet, newImage);
```

```
imshow(newImage);
```

```
title(['Передбачений клас: ', char(label)]);
```

Завдання 6: Повторити процес створення, налаштування та навчання моделі згорткової нейронної мережі, змінивши певні параметри моделі згідно з варіантом завдання, використовуючи дані з **таблиці 9.1**. Протестувати модель на нових зображеннях, оцінити точність і проаналізувати вплив змін параметрів на продуктивність.

1. ***Розмір вхідного зображення:*** У деяких варіантах використовується 32x32 пікселі (для CIFAR-10), у деяких - 64x64, що потребує перетворення зображень перед подачею на вхід нейронної мережі.

2. Змінюється ***кількість фільтрів у згорткових шарах*** для тестування ефективності з різними рівнями абстракції.

3. ***Кількість епох*** варіюється від 10 до 20. У звіті потрібно відмітити, як збільшення кількості епох впливає на точність моделі.

4. ***Швидкість навчання*** змінюється для аналізу стабільності і швидкості збіжності моделі. Нижчі значення можуть забезпечити стабільніше навчання, але довший час збіжності.

5. ***Розмір мініпартії*** впливає на час навчання і ефективність моделі, дозволяючи контролювати обсяг даних, оброблених у кожній ітерації.

Таблиця 9.1 — Варіанти завдань

№ варіанту	Розмір вхідного зображення	Кількість фільтрів у 1-му шарі	Кількість фільтрів у 2-му шарі	Кількість епох	Початкова швидкість навчання	Розмір мініпакета
1	32x32	16	32	10	0.001	64
2	32x32	32	64	15	0.001	32
3	32x32	16	32	20	0.0005	64
4	64x64	16	32	10	0.0001	32
5	64x64	32	64	15	0.0001	64
6	32x32	32	128	20	0.001	32
7	32x32	8	16	15	0.0005	128
8	64x64	16	64	10	0.001	64
9	64x64	16	128	20	0.0001	32
10	32x32	8	32	10	0.0005	64
11	32x32	32	64	10	0.001	32
12	64x64	8	16	15	0.0001	128
13	32x32	16	64	20	0.001	64
14	64x64	32	64	10	0.0005	32
15	32x32	16	128	15	0.0001	64
16	32x32	16	64	10	0.001	128
17	64x64	8	32	15	0.0005	64
18	32x32	8	16	20	0.0001	32
19	64x64	32	128	10	0.001	64
20	32x32	8	64	15	0.0005	32
21	64x64	16	128	20	0.0001	128
22	32x32	32	64	10	0.0005	64
23	64x64	8	32	15	0.001	32
24	32x32	8	128	20	0.0001	64
25	64x64	16	32	10	0.0005	128
26	32x32	8	64	15	0.0005	32
27	64x64	32	64	20	0.0001	64
28	32x32	16	128	10	0.001	128
29	64x64	32	128	15	0.0005	64
30	32x32	8	16	20	0.0001	32

Завдання 7: Використовуючи додаток А, оформіть звіт за виконаною лабораторною роботою.

Контрольні питання

1. Що таке згорткові нейронні мережі (CNN)? Які основні компоненти вони мають?
2. Чому важливо проводити попередню обробку зображень перед навчанням нейронної мережі?
3. Як впливає зміна розміру вхідного зображення на ефективність CNN?
4. Яку роль виконують згорткові (*Convolutional*) шари в CNN?
5. Для чого застосовують шари нормалізації (*Batch Normalization*)?
6. Яке призначення шару субдискретизації (*MaxPooling*)?
7. Які основні параметри потрібно встановити перед навчанням CNN?
8. Як впливає початкова швидкість навчання на процес тренування нейронної мережі?
9. Що таке перенавчання і як його можна уникнути?
10. Як оцінити точність нейронної мережі для класифікації зображень?
11. Що показує матриця плутанини (*Confusion Matrix*), і для чого її застосовують?
12. Як кількість фільтрів у згорткових шарах впливає на точність класифікації?
13. Як розмір мініпартії (*MiniBatchSize*) впливає на час навчання та стабільність моделі?
14. Яким чином кількість епох впливає на результати навчання мережі?
15. Які помилки можуть виникати під час класифікації зображень, і як їх можна виправити?
16. Чим відрізняється шар *Softmax* від інших активаційних функцій?
17. Як змінюється продуктивність CNN під час роботи з великими наборами даних?

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

- 1) 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.
- 2) Основи штучних нейронних мереж : навчальний посібник / В. П. Сидоренко, Л. М. Коваленко. – Київ : КНУ ім. Т. Шевченка, 2021. – 256 с.
- 3) Петров, О. В. Нейронні мережі: теорія і практика / О. В. Петров, І. М. Василенко. – Харків : ХНУРЕ, 2020. – 312 с.
- 4) Іванов, Д. С. Штучні нейронні мережі: моделювання та застосування / Д. С. Іванов. – Львів : Львівська політехніка, 2022. – 280 с.
- 5) Семенов, П. В. Нейронні мережі та їх застосування / П. В. Семенов. – Київ : Техніка, 2023. – 230 с.
- 6) Ткаченко, Ю. М. Методи та архітектури нейронних мереж / Ю. М. Ткаченко. – Дніпро : ДНУ, 2021. – 250 с.
- 7) Гавриленко, В. О. Асоціативні нейронні мережі / В. О. Гавриленко. – Запоріжжя : ЗНТУ, 2020. – 215 с.
- 8) Козлов, П. М. Двонаправлені нейронні мережі: теорія та практика / П. М. Козлов. – Вінниця : ВНТУ, 2022. – 198 с.
- 9) Лисенко, О. Б. Адаптивне навчання штучних нейронних мереж / О. Б. Лисенко. – Харків : ХАІ, 2021. – 270 с.
- 10) Добровська, Л. М. Теорія та практика нейронних мереж : навч. посіб. / Л. М. Добровська, І. А. Добровська. – Київ : НТУУ «КПІ», Вид-во «Політехніка», 2015. – 396 с. – Бібліогр. : с. 385–387. – ISBN 978-966-622-691-7.

ДОДАТОК А. СТРУКТУРА ВИКОНАННЯ ЗВІТУ

- *Титульний аркуш* (див. Додаток Б);
- *Мета роботи*;
- *Хід роботи*:
 - *Номер та умова завдання*;
 - *Опис послідовності виконання*;
 - *Відповідний фрагмент програмного коду з основними поясненнями*;

Наприклад:

```
% Визначення архітектури нейронної мережі
layers = [
    featureInputLayer(2)      % Вхідний шар з 2 входами
    fullyConnectedLayer(1)    % Один нейрон у прихованому шарі
    sigmoidLayer              % Функція активації (сигмоїда)
    regressionLayer];         % Регресійний шар (для логічної функції)

% Опції навчання
options = trainingOptions('adam', ...
    'MaxEpochs', 5000, ...    % Кількість епох (збільшив до 10000)
    'Plots', 'training-progress'); % Для відображення процесу навчання
```

- *Графіки отриманих результатів або скріншот даних з командного вікна для підтвердження результатів*;

Приклад підпису до графіка:

Рисунок 1 — Графік процесу навчання нейронної мережі

- *Проміжний аналіз отриманих даних*;

Приклад аналізу: «Отримана точність моделі складає 85%. Аналізуючи матрицю плутанини, можна побачити, що модель часто плутає класи «кіт» та «собака». Ймовірна причина — візуальна подібність цих класів. Для покращення результатів необхідно збільшити кількість епох навчання, змінити архітектуру CNN або застосувати інші методи обробки зображень».

Після виконання всіх завдань надати:

- *Повний лістинг коду з М-файлу*;
- *Загальні висновки по роботі*.

ДОДАТОК Б. ТИТУЛЬНА СТОРІНКА ЗВІТУ



МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
РАДІОТЕХНІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА ПРИКЛАДНОЇ РАДІОЕЛЕКТРОНІКИ



ЗВІТ

з лабораторної роботи № XX

«Назва роботи»

з дисципліни

“Нейронні мережі для оброблення сигналів”

Варіант № XX

Перевірив:

Виконав студент групи PE-??:

(ПІБ викладача)

(ПІБ студента)

(Дата та підпис викладача)

(Підпис студента)

Київ — 2025