Факультет комп'ютерних наук та кібернетики
Київського національного університету
імені Тараса Шевченка

Я.В. Терещенко
Б.В. Бобиль
А.О. Свистунов

Основи глибокого навчання

Конспект лекцій

Київ 2025

УДК 004.8

Рецензенти
д.ф.-м.н., професор Марченко О.О. (професор кафедри математичної інформатики факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка)
к.ф-м.н., доцент Криволап А.В. (доцент кафедри теорії та технології програмування Київського національного університету імені Тараса Шевченка)

Рекомендовано до друку вченою радою факультету комп'ютерних наук та кібернетики (протокол № _14_ від "_20_" __травня____ 2025 року)

Ухвалено науково-методичною комісією факультету комп'ютерних наук та кібернетики (протокол № _9_ від "_19_" __травня____ 2025 року)

доктор філософії, асистент Терещенко Ярослав Васильович
асистент Бобиль Богдан Володимирович
асистент Свистунов Антон Олександрович

Терещенко Я.В. , Бобиль Б.В., Свистунов А.О.
Основи глибокого навчання : консп. лек. / Я.В. Терещенко, Б.В. Бобиль, А.О. Свистунов – 2025. – 85 с.

Призначений для студентів, які розпочинають вивчення глибокого навчання — однієї з провідних технологій у сфері штучного інтелекту. Посібник охоплює базові теоретичні концепції та практичні аспекти побудови, навчання й оптимізації глибинних нейронних мереж. Розглядаються фундаментальні поняття: структура нейронних мереж, задача бінарної класифікації, алгоритм градієнтного спуску, побудова обчислювальних графів, активаційні функції, методи ініціалізації ваг, сучасні алгоритми оптимізації та техніки регуляризації.

Матеріал викладено у доступній формі з прикладами, формулами та поясненнями, що сприяє кращому засвоєнню. Курс покликаний сформувати в студентів цілісне уявлення про глибоке навчання як наукову дисципліну та інструмент для вирішення прикладних задач у сфері комп'ютерного зору, обробки мовлення, біоінформатики, робототехніки тощо.

Для студентів ОС «Магістр» факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, які навчаються за освітньо-науковою програмою «Математичні методи штучного інтелекту» спеціальності «Комп'ютерні науки».

© Терещенко Я.В., Бобиль Б.В., Свистунов А.О.

ЗМІСТ

ПЕРЕДМОВА....5
1. БАЗОВІ ПОНЯТТЯ НЕЙРОННИХ МЕРЕЖ....8
1.1 Що таке нейронна мережа?....8
1.2 Структура штучного нейрона....8
Обчислення зваженої суми вхідних сигналів....9
Додавання зміщення (bias)....9
Застосування функції активації....9
1.3 Архітектура нейронної мережі....10
1.4 Типи нейронних мереж....13
1.4.1 Перцептрон (Perceptron)....13
1.4.2 Багатошаровий перцептрон (MLP)....15
1.4.3 Згорткова нейронна мережа (CNN)....15
1.4.4 Рекурентна нейронна мережа (RNN)....17
1.4.5 Трансформери....17
1.4.6 Генеративні змагальні мережі (Generative Adversarial Networks, GANs)....19
1.4.7 Дифузійні моделі (Diffusion Models)....21
2. ЗАДАЧА БІНАРНОЇ КЛАСИФІКАЦІЇ....25
2.1 Формалізація задачі....25
2.2 Підходи до моделювання....28
Логістична регресія (Logistic Regression)....28
Лінійний перцептрон....29
Багатошаровий перцептрон....29
Підходи з попереднім перетворенням ознак (Feature Engineering)....30
Нелінійні моделі (SVM, Random Forest, Boosting)....31
Глибокі нейронні мережі....31
2.3 Логіти....32
2.4 Функція втрат. Бінарна крос-ентропія....33
2.5 Метрики оцінювання....34
Матриця помилок (confusion matrix)....35
Accuracy (Точність класифікації)....35
Precision (Точність позитивного класу)....36
Recall (Повнота або чутливість)....36
F1-score....36
Specificity (Специфічність)....37
Balanced Accuracy....37
AUC-ROC (Area Under the Receiver Operating Characteristic Curve)....37
PR-крива (Precision-Recall curve)....38
3. ГРАДІЄНТНИЙ СПУСК ДЛЯ НЕЙРОННИХ МЕРЕЖ ТА ОБЧИСЛЮВАЛЬНИЙ ГРАФ....41
3.1 Класичний алгоритм....42
3.2 Алгоритм оберненого розповсюдження похибки....43

Прямий прохід(forward pass): 43
Зворотний прохід (backward pass) 45
Оновлення параметрів 47
3.3 Обчислювальний граф 48
Forward pass (прямий прохід) 49
Backward pass (зворотний прохід) 50
Реалізація в фреймворках 51
4. ФУНКЦІЇ АКТИВАЦІЇ 53
4.1 Sigmoid 53
4.2 Tanh (гіперболічний тангенс) 54
4.3 ReLU (Rectified Linear Unit) 55
4.4 Leaky ReLU 56
4.5 Parametric ReLU (PReLU) 57
4.6 ELU (Exponential Linear Unit) 57
4.7 GELU (Gaussian Error Linear Unit) 58
4.8 Swish 59
4.9 Softmax 60
5. МЕТОДИ ІНІЦІАЛІЗАЦІЇ 62
5.1 Затухаючи та вибухові градієнти 62
5.2 Ініціалізація Xavier 64
6. МЕТОДИ ОПТИМІЗАЦІЇ 68
6.1 Градієнтний спуск (Gradient Descent) 68
6.2 Покращені методи оптимізації 69
Momentum 69
Nesterov Accelerated Gradient (NAG) 69
Adagrad 69
RMSProp 70
Adam (Adaptive Moment Estimation) 70
AdamW (Weight Decoupled Regularization) 71
Порівняльна таблиця методів 71
7. РЕГУЛЯРИЗАЦІЯ 74
7.1 Призначення регуляризації 74
7.2 Методи регуляризації 75
L2-регуляризація (Ridge, weight decay) 75
L1-регуляризація (Lasso) 76
Dropout 77
DropConnect 78
Label Smoothing 78
ElasticNet 79
Додаткові методи: 80
ВИСНОВКИ 81
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 83

ПЕРЕДМОВА

Штучний інтелект (ШІ) і машинне навчання (ML) за останні десятиліття зазнали стрімкого розвитку, ставши рушійними силами інновацій у різних галузях науки, техніки та бізнесу. Однією з найбільш потужних і перспективних технологій у межах штучного інтелекту є глибоке навчання (Deep Learning) — підхід, що базується на багатошарових штучних нейронних мережах і дозволяє автоматично знаходити закономірності у великих обсягах даних.

Глибоке навчання імітує принципи роботи біологічних нейронних мереж, здійснюючи багаторівневу трансформацію інформації для витягання високорівневих ознак. Завдяки своїй здатності до самостійного навчання із даних, без потреби у складній ручній розробці ознак, методи глибокого навчання відкрили нові можливості у вирішенні завдань, що раніше вважалися надзвичайно складними або неможливими.

Сьогодні технології глибокого навчання застосовуються у різноманітних сферах:

- Комп’ютерний зір (розпізнавання об'єктів, сегментація сцен, автоматичне водіння);
- Обробка природної мови (машинний переклад, чат-боти, текстова аналітика);
- Біомедицина (діагностика захворювань, аналіз ДНК, пошук нових ліків);
- Фінанси (прогнозування ринкових трендів, виявлення шахрайства);
- Робототехніка та автономні системи;
- Креативні індустрії (генерація музики, зображень, відео).

З урахуванням важливості та широкого спектра застосувань глибокого навчання, формування міцної бази знань у цій області є необхідною умовою для підготовки фахівців нового покоління.

Цей конспект лекцій розроблений для забезпечення студентів необхідними теоретичними знаннями та практичними навичками, що дозволять їм не лише розуміти, але й самостійно

розробляти та оптимізувати глибинні моделі для вирішення реальних завдань.

Рукопис орієнтований на студентів старших курсів, практикуючих інженерів та дослідників, які прагнуть заглибитись у сферу штучного інтелекту, усіх, хто планує розвивати кар'єру у галузях Data Science, Machine Learning або Computer Vision.

Головна мета рукопису - забезпечення системного розуміння фундаментальних концепцій глибокого навчання: архітектури нейронних мереж, механізмів тренування, методів оптимізації та боротьби з перенавчанням. Навчити студентів будувати, тренувати та оцінювати прості та середньої складності нейронні мережі. Ознайомити з основними практичними інструментами роботи з глибинними моделями, такими як PyTorch або TensorFlow. Підготувати студентів до участі у прикладних та дослідницьких проектах зі створення інтелектуальних систем. Сформувати аналітичне мислення для оцінки якості моделей та пошуку шляхів їх покращення.

Надати базу знань для подальшого вивчення більш складних архітектур: згорткових мереж (CNN), рекурентних мереж (RNN), трансформерів (Transformer) та їх варіацій.

Після проходження курсу студент повинен:

Знати:

- Основні типи нейронних мереж: багатошарові перцептрони (MLP), основи згорткових нейронних мереж (CNN), рекурентних нейронних мереж (RNN).
- Структуру обчислювального графа, принципи роботи алгоритму зворотного поширення помилки (backpropagation).
- Види функцій активації (Sigmoid, ReLU, Leaky ReLU, Tanh та інші) та їх особливості.
- Основні методи ініціалізації ваг (Random, Xavier, He) та їх вплив на процес навчання.
- Методики оптимізації (SGD, Adam, RMSProp) та їх особливості.

- Прийоми стабілізації процесу навчання: батч-нормалізація, регуляризація (Dropout, L2-регуляризація, DropConnect, Label Smoothing, ElasticNet).

Вміти:

- Ставити завдання класифікації та регресії у формулювання нейронних мереж.
- Самостійно реалізувати прості багатошарові нейронні мережі з нуля.
- Використовувати стохастичний градієнтний спуск та його варіанти для оптимізації моделей.
- Застосовувати батч-нормалізацію, регуляризацію та інші методи для покращення якості навчання.
- Виконувати візуалізацію процесу навчання (криві loss, accuracy) та аналізувати причини проблем у навчанні моделей (overfitting, underfitting).

Таким чином, курс спрямований як на розвиток теоретичних знань (поглиблене розуміння механізмів роботи нейронних мереж), так і на формування практичних компетентностей, необхідних для розробки, тренування та аналізу глибинних моделей у реальних проєктах.

1. БАЗОВІ ПОНЯТТЯ НЕЙРОННИХ МЕРЕЖ

1.1 Що таке нейронна мережа?

Штучна нейронна мережа (ШНМ) — це математична структура, яка моделює принципи обробки інформації, подібні до роботи біологічного мозку. Вона складається з великої кількості простих обчислювальних елементів — нейронів, які об'єднані у шари та взаємодіють через зважені з'єднання.

Основне призначення нейронної мережі полягає у перетворенні вхідних даних (векторів ознак) у вихідні значення (наприклад, ймовірності класів, передбачені значення у регресії) через послідовність обчислювальних операцій.

Ідея запозичена з біології: в реальному мозку нейрони передають сигнали один одному через синапси, причому сила сигналу залежить від ваги цих з'єднань. Так само у штучних нейронних мережах інформація передається через вагові коефіцієнти.

Ключові характеристики нейронних мереж:

1. Навчаються знаходити приховані закономірності у даних.
2. Можуть апроксимувати складні нелінійні функції.
3. Підтримують адаптацію через зміну ваг на основі алгоритмів оптимізації.

Загальна схема роботи нейронної мережі:

- Приймає на вхід вектор даних x.
- Перетворює його через послідовність шарів, де кожен шар виконує обчислення.
- Генерує вихід y, який може бути, наприклад, прогнозом класу або числовим значенням.

1.2 Структура штучного нейрона

Кожен штучний нейрон є елементарним блоком обчислень і виконує три основні операції:

Обчислення зваженої суми вхідних сигналів

Нехай вхідні дані представлені як вектор $\mathbf{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}]$, а ваги як вектор $\mathbf{w} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_n}]$.

Тоді нейрон обчислює лінійну комбінацію:

$$\mathrm{z} = \sum_{i=1}^{n} w_i \cdot x_i = \boldsymbol{w^T x}$$

Додавання зміщення (bias)

До результату додається додатковий параметр зміщення bb, який дозволяє моделі зміщувати функцію активації ліворуч або праворуч:

$$\mathrm{z} = \boldsymbol{w^T x} + b$$

Зміщення підвищує гнучкість моделі і дозволяє їй краще узгоджуватися з даними.

Застосування функції активації

Щоб надати моделі **нелінійності**, результат z передається через **функцію активації** $\phi(\mathrm{z})$.

Типові функції активації:

- Sigmoid: $\phi(\mathrm{z}) = \frac{1}{1+e^{-z}}$
- Tanh: $\phi(\mathrm{z}) = \tanh(z)$
- ReLU (Rectified Linear Unit): $\phi(\mathrm{z}) = \max(0, z)$

Функція активації дозволяє нейронній мережі моделювати складні залежності та робить її здатною вирішувати задачі класифікації і регресії.

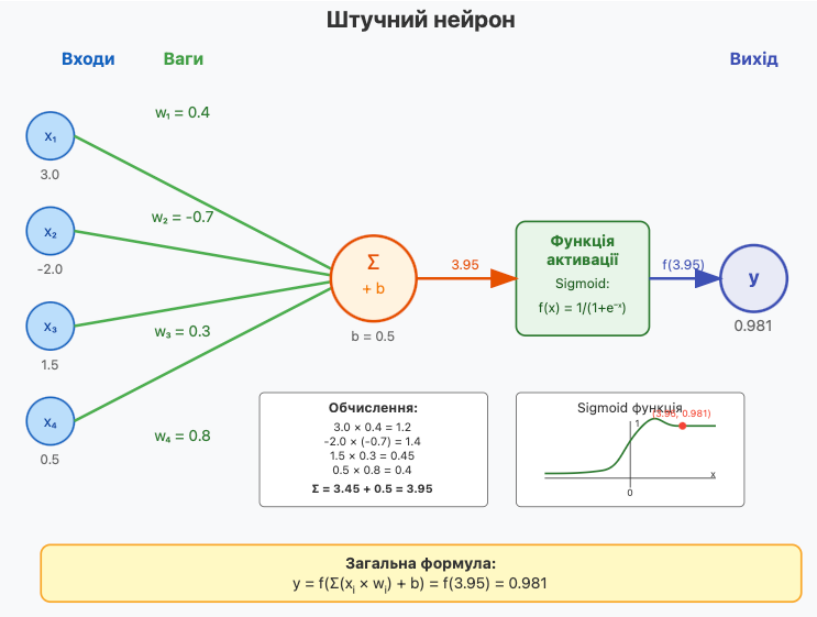

Рисунок 1. Структура штучного нейрона з функцією активації.

Аналогічно можна подати нейрон (Рисунок 1) формулою:

$$y = \phi(\sum_{i=1}^{n} w_i \cdot x_i + b)$$

1.3 Архітектура нейронної мережі

Штучна нейронна мережа (Рисунок 2) складається з трьох основних типів шарів:

- **Вхідний шар** (Input layer): приймає сирі дані (наприклад, пікселі зображення або ознаки об'єкта).
- **Приховані шари** (Hidden layers): виконують проміжні обчислення, витягаючи корисні ознаки.

- **Вихідний шар** (Output layer): генерує остаточний прогноз.

Число прихованих шарів і нейронів у них визначає архітектуру мережі і безпосередньо впливає на її потужність та здатність узагальнювати дані.

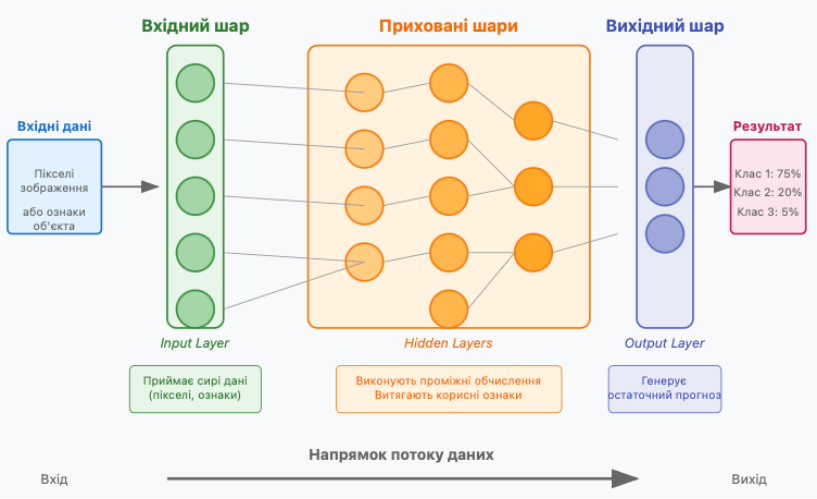

Рисунок 2. Приклад мережі прямого поширення.

Саме поняття “Deep learning” означає, що ми працюємо з нейронною мережою, яка має багато прихованих шарів (Рисунок3).

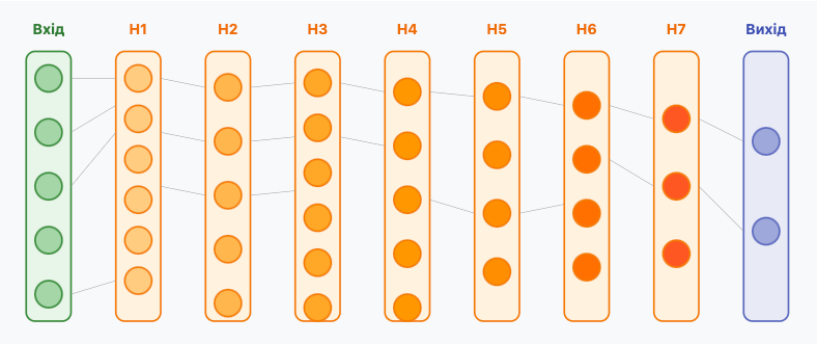

Рисунок 3. Нейронна мережа з вхідним, вихідним та 7-ма прихованими шарами.

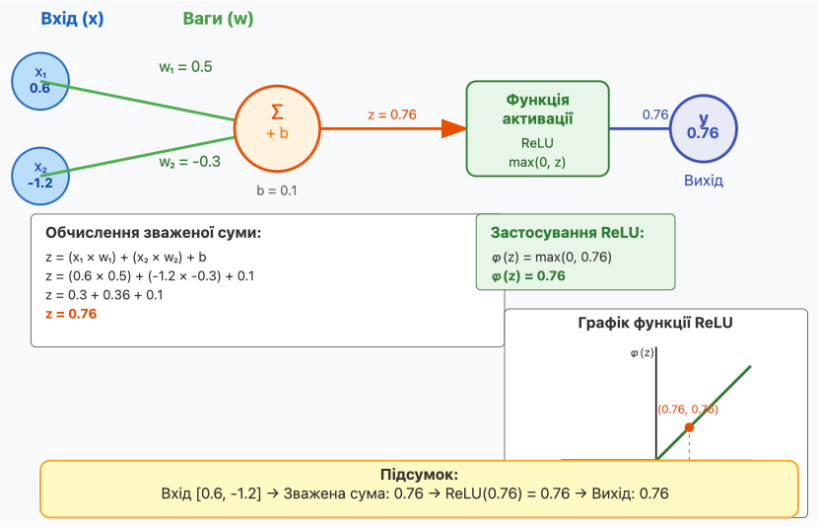

Рисунок 4. Найпростіша нейронна мережа з функцією активації.

Приклад обчислення нейрону з функцією активації:

- Вхід: $x = [0.6, -1.2]$
- Ваги: $x = [0.5, -0.3]$
- Зміщення: $b = 0.1$
- Функція активації: ReLU

Тоді:

1. Обчислюємо зважену суму:

$$z = (0.6 \times 0.5) + (-1.2 \times -0.3) + 0.1 = 0.3 + 0.36 + 0.1 = 0.76$$

2. Застосовуємо активацію ReLU:

$$\phi(z) = \max(0, 0.76) = 0.76$$

Висновок: Вихід нейрона дорівнює 0.76.

1.4 Типи нейронних мереж

Штучні нейронні мережі поділяються на різні типи залежно від їх архітектури та завдань, які вони вирішують. Розглянемо основні з них.

1.4.1 Перцептрон (Perceptron)

Перцептрон (Рисунок 5) — це найпростіша форма нейронної мережі, запропонована Френком Розенблаттом у 1958 році. Він складається з одного шару нейронів і призначений для вирішення задач лінійної класифікації.

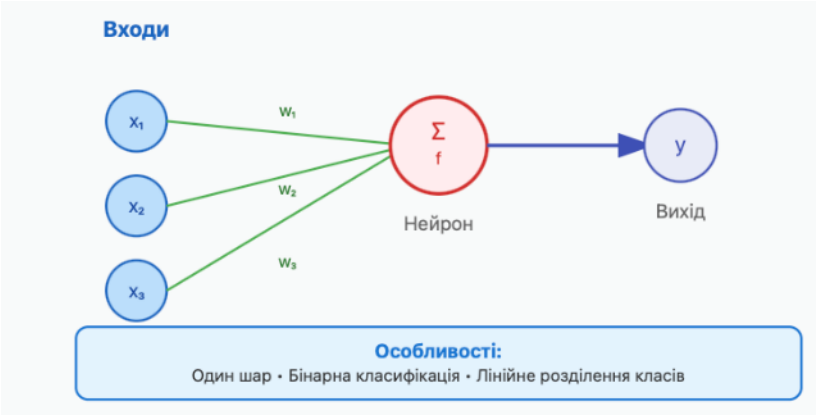

Рисунок 5. Схема. Роботи перцептрону Розенблата.

Математично вихід одного нейрона у перцептроні:

$$output = \phi(\sum_{i=1}^{n} w_i \cdot x_i \ + b)$$

де ϕ — функція активації (зазвичай поріг)

Обмеження: Перцептрон не може вирішувати задачі, що не є лінійно роздільними (наприклад, задачу XOR, Рисунок 6).

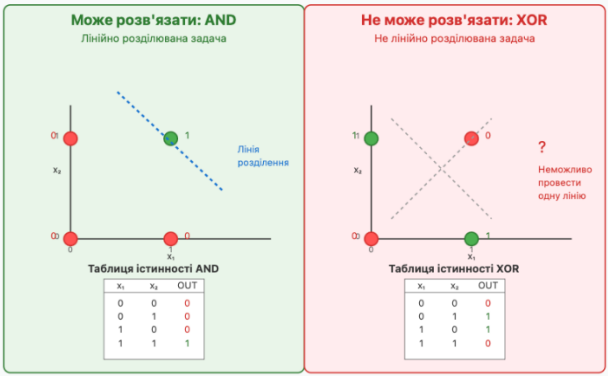

Рисунок 6. Приклади лінійно роздільної та лінійно нероздільної задач.

1.4.2 Багатошаровий перцептрон (MLP)

Багатошаровий перцептрон складається з:

- одного вхідного шару,
- одного або кількох прихованих шарів,
- одного вихідного шару.

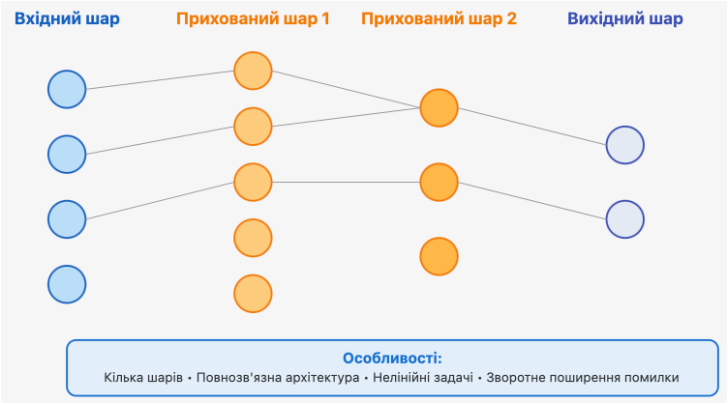

Рисунок 7. Схема. передачі даних між нейронами в мережі прямого поширення(feed-forward MLP).

На відміну від простого перцептрона, MLP (Рисунок 7) має **нелінійні функції активації** в прихованих шарах (наприклад, ReLU, tanh), що дозволяє йому вирішувати складні задачі з нелінійними залежностями.

Переваги:

- **Універсальність** (теорема про універсальну апроксимацію: MLP може апроксимувати будь-яку безперервну функцію за певних умов).
- Підходить для задач класифікації та регресії.

1.4.3 Згорткова нейронна мережа (CNN)

Згорткові нейронні мережі спеціалізуються на обробці даних із локальною структурою, зокрема зображень. Архітектура згоркової нейронної мережі запозичена з принципу роботи зорової системи людського ока. На перших рівнях ми

розпізнаємо низькорівневі ознаки: кола, границі, ліній. На глибших рівнях нейронної мережі ознаки описують семантичні звязки: виділення окремих обєктів, або їх частин (Рисунок 8).

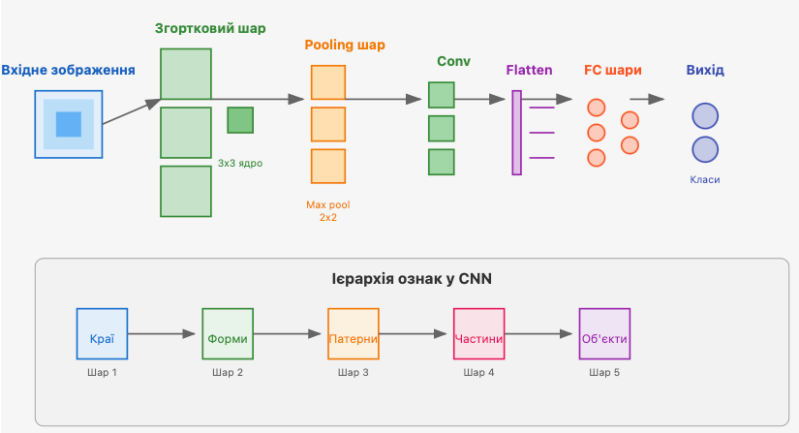

Рисунок 8. Базова архітектура згорткової нейронної мережі(CNN).

Ключові компоненти CNN:

- **Згорткові шари** (Convolutional layers): автоматично витягують локальні ознаки.
- **Шари субдискретизації** (Pooling layers): зменшують розмірність і виділяють важливі ознаки.
- **Повнозв'язані шари** (Fully Connected layers): виконують остаточну класифікацію.

Особливість: Замість обробки усіх вхідних ознак одразу, CNN "дивляться" на невеликі частини даних через фільтри, що значно зменшує кількість параметрів.

Основні області застосування: комп'ютерний зір, медична діагностика за знімками, розпізнавання об'єктів.

1.4.4 Рекурентна нейронна мережа (RNN)

Рекурентні нейронні мережі призначені для роботи з послідовностями даних, де важливий порядок елементів (наприклад, текст, аудіо, часові ряди).

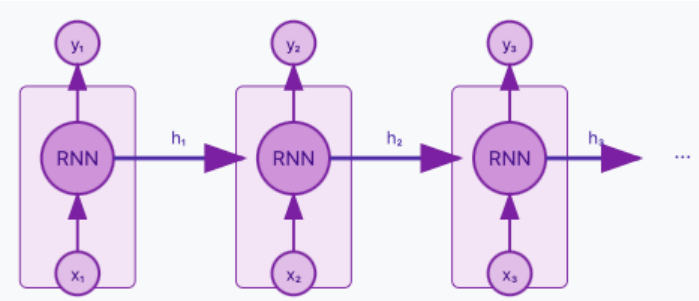

Рисунок 9. Базова архітектура рекурентної нейромережі(RNN).

Ключова особливості RNN (Рисунок 9) - вони мають **циклічні з'єднання**, що дозволяють моделі запам'ятовувати інформацію з попередніх кроків. Математично вихід у момент часу t:

$$h_t = \phi(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_{t-1} + b)$$

де h_t — стан пам'яті на кроці t.

Використання: переклад текстів, передбачення наступних символів, генерація музики.

1.4.5 Трансформери

Transformer — це архітектура нейронної мережі, спеціально розроблена для обробки **послідовностей даних** без використання рекурентних зв'язків (Рисунок 10).

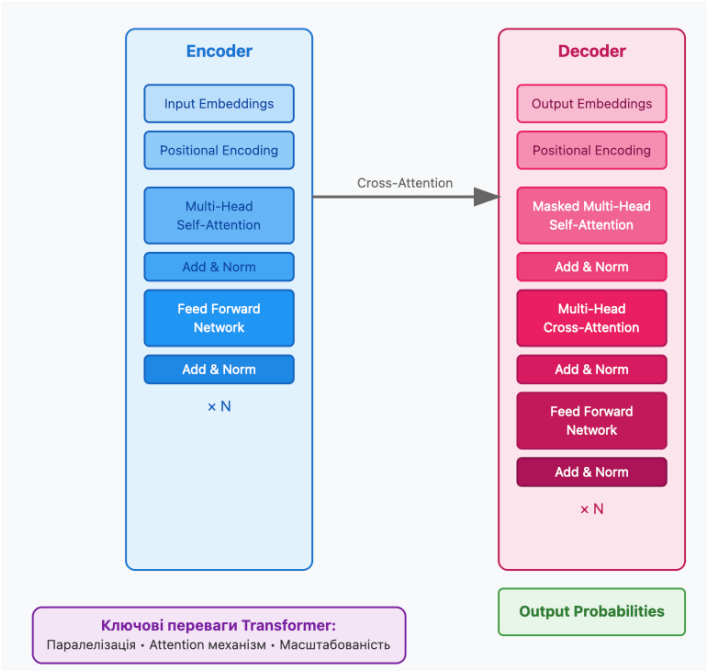

Рисунок 10. Базова архітектура трансформера.

Ключова особливість Transformer:

- Використання механізму **self-attention**: кожен елемент послідовності "бачить" всю іншу послідовність під час обробки.
- Паралельна обробка даних, що значно пришвидшує тренування порівняно з RNN.

Структура базового Transformer:

- **Encoder**: послідовність блоків, які перетворюють вхід у внутрішнє представлення.
- **Decoder**: генерує вихід на основі представлення від енкодера.

Основні компоненти:

1. **Self-Attention**: обчислює ваги важливості між елементами вхідної послідовності.
2. **Position Encoding**: додає інформацію про порядок елементів.
3. **Feed-Forward Networks**: застосовуються після кожної операції attention.
4. **Residual Connections та Layer Normalization** для покращення стабільності тренування.

Важливо: Transformer став основою для найпотужніших сучасних моделей, таких як BERT, GPT, T5.

Основні напрямки застосування: машинний переклад (наприклад, Google Translate), генерація текстів (ChatGPT, GPT-4), аналіз документів. На даний момент трансформери використовуються також для задач компютерного зору.

1.4.6 Генеративні змагальні мережі (Generative Adversarial Networks, GANs)

GAN — це архітектура, яка складається з двох нейронних мереж (Рисунок 11):

- **Генератора** (Generator): створює нові дані.
- **Дискримінатора** (Discriminator): намагається відрізнити справжні дані від згенерованих.

Рисунок 11. Процес навчання генеративних змагальних мереж(GAN).

Ці дві мережі навчаються у процесі змагання. Генератор прагне створювати настільки реалістичні зразки, щоб обдурити дискримінатор. Дискримінатор прагне точно розрізняти справжні та підроблені зразки.

Мета генератора:

$$G(z) \rightarrow \text{дані, подібні до реальних}$$

де z — вхідний випадковий шум.

Мета дискримінатора:

$$D(x) \approx 1, \qquad D(G(z)) \approx 0$$

де x - справжні дані, $G(z)$ — згенеровані дані.

Основні застосування GAN – це генерація реалістичних зображень (наприклад, облич, які не існують), покращення якості зображень (super-resolution), створення нових творів мистецтва, музики.

1.4.7 Дифузійні моделі (Diffusion Models)

Дифузійні моделі — це сучасний тип генеративних моделей, який базується на поступовому "зашумленні" даних і навчанню моделі відновлювати оригінальні дані з шуму (Рисунок 12).

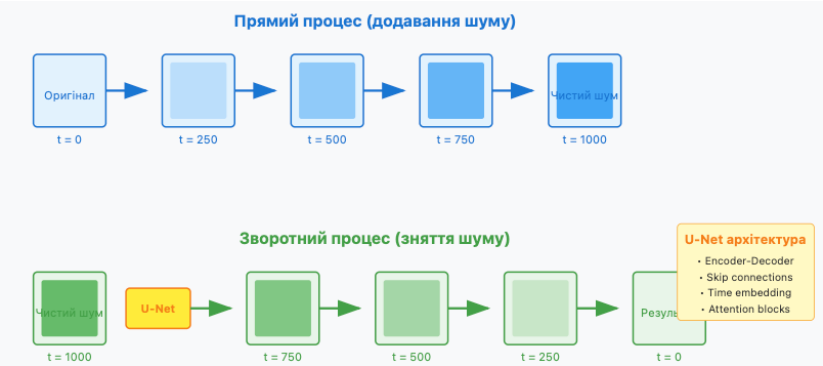

Рисунок 12. Приклад роботи дифузійної моделі.

Основна ідея:

- **Прямий процес:** поступово додається шум до даних (наприклад, зображення) за багато кроків.
- **Зворотний процес:** модель навчається поетапно відновлювати дані з шуму.

Таким чином, на етапі генерації ми стартуємо з випадкового шуму і за допомогою моделі крок за кроком "прибираємо шум", отримуючи нові дані. Ключова формула прямого процесу:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

де β_t — параметр, що контролює кількість доданого шуму на кожному кроці.

Приклади застосування Diffusion models:

- Створення високоякісних реалістичних зображень (наприклад, Stable Diffusion, DALL-E 2).

- Генерація 3D-об'єктів, музики.
- Редагування зображень (inpainting, style transfer).

Перевага над GAN:

1. Більш стабільне навчання.
2. Краща якість результатів без "mode collapse" (типова проблема GAN)

Архітектура	**Особливості**	**Основні області застосування**
MLP	Прості табличні дані, регресія, класифікація	Класифікація чисел, регресія
CNN	Згортки, локальні ознаки	Комп'ютерний зір
RNN	Послідовності, пам'ять стану	Текст, часові ряди, аудіо
Transformer	Self-Attention, паралельна обробка	Текст, зображення, звук
GAN	Змагання між моделями	Генерація реалістичних зразків
Diffusion	Поетапне відновлення з шуму	Генерація високоякісних даних

Таблиця 1. Порівняльна характеристика архітектур нейронних мереж.

Питання для самоконтролю

1. Що таке штучна нейронна мережа? У чому її подібність до біологічного мозку?
2. Які три основні операції виконує штучний нейрон?

3. Що таке функція активації? Назвіть приклади та поясніть їх відмінності.
4. Для чого використовується зміщення (bias) у нейроні?
5. Що таке обчислювальний граф у контексті нейронних мереж?
6. Чим багатошаровий перцептрон відрізняється від простого?
7. Як працюють згорткові нейронні мережі та де вони застосовуються?
8. Як RNN запам'ятовують інформацію з попередніх кроків?
9. Що таке self-attention у моделі Transformer і яку функцію він виконує?
10. У чому полягає принцип навчання GAN? Які ролі відіграють генератор і дискримінатор?
11. Як працює дифузійна модель? У чому полягає forward і reverse процес?
12. Поясніть переваги і недоліки Transformer у порівнянні з RNN.
13. У яких випадках доцільно використовувати Diffusion model, а не GAN?

Вправи

1. **Реалізуйте штучний нейрон з нуля (без фреймворків).** Напишіть Python-функцію, яка обчислює вихід нейрона:

```
import numpy as np

def simple_neuron(x, w, b, activation='relu'):
    z = np.dot(w, x) + b
    if activation == 'relu':
        return np.maximum(0, z)
    elif activation == 'sigmoid':
        return 1 / (1 + np.exp(-z))
    else:
        return z
```

```
# Приклад:
x = np.array([0.6, -1.2])
w = np.array([0.5, -0.3])
b = 0.1
print(simple_neuron(x, w, b))  # Output: 0.76 (ReLU)
```

2. Побудуйте свою першу MLP-модель у PyTorch

```
import torch
import torch.nn as nn

class SimpleMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(2, 4)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(4, 1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        return self.fc2(x)

model = SimpleMLP()
x = torch.tensor([[0.6, -1.2]])
output = model(x.float())
print(output)
```

2. ЗАДАЧА БІНАРНОЇ КЛАСИФІКАЦІЇ

Бінарна класифікація — це фундаментальна задача у машинному навчанні та глибокому навчанні, яка полягає в автоматичному віднесенні об'єкта до одного з **двох взаємовиключних класів**. Вона є основою для побудови багатьох систем прийняття рішень — від простих фільтрів спаму до складних систем медичної діагностики чи фінансової аналітики (Рисунок 13).

2.1 Формалізація задачі

Нехай:

- $x \in \mathbb{R}^n$— вхідний вектор ознак (features), що описує об'єкт або приклад. Кожен елемент вектора — це числове або категоріальне представлення певної характеристики.
- $y \in \{0,1\}$ — цільова змінна (label), яка вказує, до якого з двох класів належить приклад.

Завдання моделі полягає у побудові функції $f: \mathbb{R}^n \to [0,1]$, яка наближує ймовірність того, що приклад належить до класу 1, тобто:

$$f(x) \approx P(y = 1|x)$$

Якщо результат $f(x) \geq 0.5$, то модель передбачає клас 1, інакше — клас 0. Поріг 0.5 може змінюватися залежно від пріоритетів задачі (наприклад, у медичній діагностиці важливо уникати хибнонегативних результатів, тому поріг може бути зменшений).

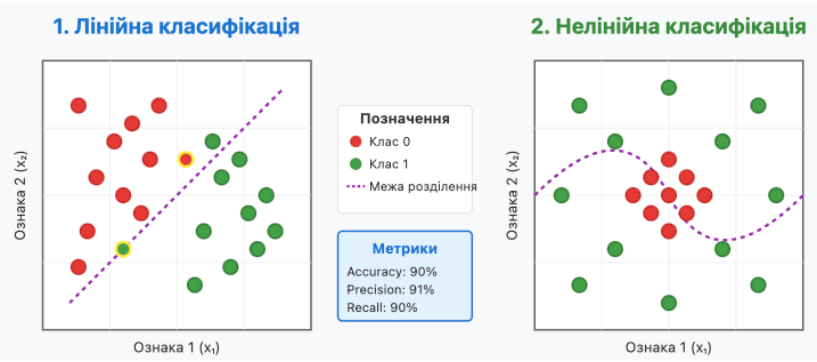

Рисунок 13. Типи задач класифікації відносно розміщення об’єктів у просторі ознак.

Галузь	**Приклад задачі**	**Клас 1**	**Клас 0**
Безпека	Виявлення шахрайських транзакцій	Транзакція є шахрайською	Нормальна транзакція
Медицина	Діагностика хвороб	Пацієнт має хворобу	Пацієнт здоровий
Комунікації	Фільтрація спаму	Повідомлення — спам	Повідомлення — не спам
Промисловість	Контроль якості	Деталь бракована	Деталь відповідає стандарту
Комп’ютерний зір	Виявлення об'єкта на зображенні	Об'єкт присутній	Об'єкт відсутній

Таблиця 2. Приклади задач бінарної класифікації.

Попри свою "простоту", задача бінарної класифікації є надзвичайно важливою у практиці, оскільки:

- Багато **складніших задач** (наприклад, багатокласова класифікація або сегментація зображень) можна звести до декількох незалежних задач бінарної класифікації (наприклад, one-vs-all).
- Вона дозволяє **тестувати нові архітектури**, функції активації, методи оптимізації у контрольованому середовищі.
- Є зручною для **аналізу помилок**, візуалізації граничних рішень і вивчення поведінки моделей.

Основні компоненти задачі бінарної класифікації

1. **Вектор ознак (feature vector)**. Містить числове представлення інформації про об'єкт. Приклади: пікселі зображення, частотність слів у тексті, результати медичних тестів.
2. **Модель**. Побудована на основі математичних обчислень (наприклад, логістична регресія, перцептрон, нейронна мережа), яка трансформує вхідні дані у ймовірність.
3. **Функція активації**. Найчастіше — **сигмоїда**:

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

яка перетворює вихід моделі у значення від 0 до 1.

4. **Функція втрат.** Для задач бінарної класифікації стандартною є бінарна крос-ентропія:

$$L = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})$$

5. **Метрика якості**. Вибирається залежно від задачі: accuracy, precision, recall, F1-score, AUC-ROC тощо.

яка перетворює вихід моделі у значення від 0 до 1.

Навчання моделі полягає у мінімізації функції втрат на навчальній вибірці:

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} L(f(x_i), y_i)$$

де:

- θ — параметри моделі,
- m — кількість прикладів у вибірці,
- $f(x_i)$ — передбачення моделі для прикладу ii,
- y_i — істинне значення класу.

2.2 Підходи до моделювання

У задачі бінарної класифікації існує декілька підходів до побудови моделей, які відрізняються як за принципами роботи, так і за складністю реалізації, тренуванням та областю застосування. Від найпростіших лінійних моделей до глибоких нейронних мереж — кожен підхід має свої переваги та обмеження.

Логістична регресія (Logistic Regression)

Модель передбачає ймовірність приналежності об'єкта до класу 1 як сигмоїду від лінійної комбінації ознак:

$$\mathrm{P}(y = 1|x) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Особливості:

- Є базовим і часто першим методом у задачах класифікації.

- Підходить для **лінійно відокремлюваних** задач.
- Добре інтерпретується (можна аналізувати ваги).

Недоліки:

- Не здатна моделювати **нелінійні залежності** між ознаками та мітками.

Лінійний перцептрон

Простий штучний нейрон, який обчислює:

$$y = step(w^T x + b)$$

Особливості:

- Є основою для багатьох складніших нейронних мереж.
- Працює лише з **лінійно роздільними даними**.
- Вчиться за допомогою **перцептронного алгоритму оновлення ваг**.

Через обмеження step-функції, перцептрон не може представити ймовірності та не диференційовний — що обмежує його у використанні в градієнтних методах оптимізації.

Багатошаровий перцептрон

Складається з декількох шарів нейронів з **нелінійними активаціями** (наприклад, ReLU, Tanh), що дозволяє моделювати складні функції:

$$y = \sigma(W_2 \phi(W_1 \mathrm{x} + b_1) + b_2$$

де ϕ — активаційна функція.

Особливості:

- Універсальний апроксиматор — може наближати довільні функції.
- Працює з **нелінійними залежностями**.
- Навчається через **зворотне поширення помилки (back-propagation)**.
- Може мати сотні, а іноді й тисячі параметрів.

Недоліки:

- Чутливість до гіперпараметрів.
- Схильність до перенавчання за відсутності регуляризації.
- Потребує великої кількості даних та обчислювальних ресурсів.

Підходи з попереднім перетворенням ознак (Feature Engineering)

У деяких випадках прості моделі (логістична регресія, лінійний класифікатор) можуть показувати хороші результати **за умови використання інформативних ознак**. Це підхід, коли основну увагу приділяють:

- побудові нових ознак (наприклад, добутки, логарифми, категоріальні фічі),
- нормалізації даних,
- зменшенню розмірності (наприклад, PCA).

У сучасному глибокому навчанні цей підхід все частіше замінюється **автоматичним витягуванням ознак**, яке здійснюють глибокі мережі.

Нелінійні моделі (SVM, Random Forest, Boosting)

Ці моделі не входять безпосередньо до глибокого навчання, проте заслуговують на згадку як **сильні базові бінарні класифікатори**.

- **Support Vector Machine (SVM)**: використовує гіперплощину з максимальною відстанню до найближчих точок обох класів.
- **Random Forest**: ансамбль дерев рішень, що дозволяє враховувати різні варіанти розбиття простору ознак.
- **Gradient Boosting**: послідовне навчання слабких моделей (дерев), що мінімізують похибку попередніх.

Глибокі нейронні мережі

Коли обсяг даних великий, а вхідні ознаки складні (наприклад, зображення, звук, текст), використовуються **глибокі моделі**, які складаються з багатьох шарів.

- **Convolutional Neural Networks (CNN)** — для зображень.
- **Recurrent Neural Networks (RNN)**, **GRU**, **LSTM** — для часових рядів та тексту.
- **Transformer-based моделі** — для обробки послідовностей (використовуються також у класифікації текстів, наприклад, BERT).

Переваги:

- Витягують релевантні ознаки автоматично.
- Надзвичайно гнучкі.
- Легко адаптуються до нових типів вхідних даних.

Недоліки:

- Вимогливі до ресурсів (GPU/TPU).

- Потребують великої кількості даних.
- Можуть бути складними для інтерпретації.

2.3 Логіти

У задачах бінарної класифікації модель має оцінити ймовірність того, що об'єкт належить до класу 1 (або 0). Для цього часто використовується логістична функція (сигмоїда), яка приймає на вхід логіт — лінійне поєднання ознак, і перетворює його у значення в інтервалі від 0 до 1.

Слово **логіт** походить від **log-odds** — логарифму відношення ймовірностей (коефіцієнтів шансів):

$$logit(p) = \log\left(\frac{p}{1-p}\right)$$

Це обернена функція до сигмоїди:

$$p = \sigma(z) \rightarrow z = \log\left(\frac{p}{1-p}\right)$$

Тобто, логіт — це **лінійна функція від ознак**, яка відображає, наскільки ймовірніший клас 1 порівняно з класом 0.

- $z = 0 \rightarrow \sigma(z) = 0.5$— модель "вагається".
- $z > 0 \rightarrow \sigma(z) > 0.5$— модель схиляється до класу 1.
- $z < 0 \rightarrow \sigma(z) < 0.5$ — модель схиляється до класу 0.

Чим більше абсолютне значення логіта, тим вища впевненість моделі.

Нехай маємо функцію ймовірності для класу 1**:**

$$P(y = 1|x) = p$$

Вважаємо, що шанси (odds) на користь класу 1:

$$\text{odds} = \frac{p}{1-p}$$

Логарифм від odds — це логіт:

$$\log\left(\frac{p}{1-p}\right) = \text{z} = \text{w}^{\text{T}}x + b$$

Зворотно отримаємо:

$$p = \frac{1}{1+e^{-z}} = \sigma(z)$$

Таким чином, ймовірність p може бути представлена через **сигмоїду від лінійної комбінації ознак**.

2.4 Функція втрат. Бінарна крос-ентропія

Функція втрат (англ. **loss function**) — це ключовий елемент у процесі навчання моделей машинного і глибокого навчання. Вона визначає, наскільки хорошим є передбачення моделі порівняно з фактичними мітками класу, і слугує орієнтиром для алгоритму оптимізації.

Мета функції втрат:

- Кількісно оцінити **відхилення** передбачення моделі від реального результату.
- Спрямовувати **навчання моделі**, мінімізуючи середню втрату на тренувальній вибірці.

Для задач бінарної класифікації найпоширенішою функцією втрат є бінарна крос-ентропія, яка також відома як логарифмічна функція втрат (log loss).

Нехай:

- $y \in \{0,1\}$ - справжня мітка класу.
- $\hat{y} \in (0,1)$ - передбачена ймовірність того, що приклад належить до класу 1.

Формула для одного прикладу:

$$L(y, \hat{y}) = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$$

Середня втрата для вибірки з NN прикладів:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

Якщо істинна мітка y=1, то втрата буде мінімальною, коли $\hat{y} \to 1$ (тобто модель "впевнена" у правильному класі). Якщо y=0, то втрата буде мінімальною, коли $\hat{y} \to 0$. Коли модель "помиляється з упевненістю" (наприклад, $\hat{y} \to 0.01$, а y=1), функція втрат різко зростає.

Крос-ентропія **штрафує впевнені помилки набагато сильніше**, ніж невпевнені.

2.5 Метрики оцінювання

Оцінювання якості моделі бінарної класифікації є критичним етапом у процесі розробки. Правильний вибір метрики дозволяє (Рисунок 14):

- адекватно порівнювати моделі між собою,
- виявляти проблеми перенавчання або недонавчання,
- приймати зважені рішення про запуск моделі в продакшн.

Матриця помилок (confusion matrix)

Для початку, розглянемо базову структуру **матриці передбачень**, яка складається з чотирьох основних величин:

Факт	**Позитивний (1)**	**Негативний (0)**
Позитивний (1)	TP (True Positive)	FN (False Negative)
Негативний (0)	FP (False Positive)	TN (True Negative)

Таблиця 3. Класифікація передбачень моделі бінарного класифікатора

- **TP (істинно позитивні)** — модель правильно передбачила клас 1.
- **TN (істинно негативні)** — модель правильно передбачила клас 0.
- **FP (хибнопозитивні)** — модель помилково передбачила клас 1.
- **FN (хибнонегативні)** — модель помилково передбачила клас 0.

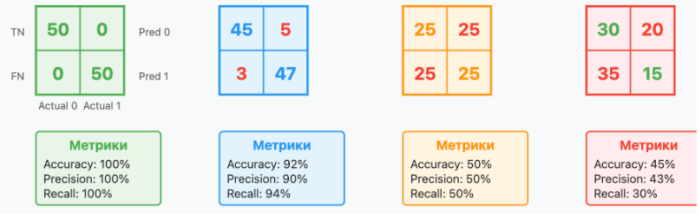

Рисунок 14. Приклад підрахунку метрик на основі матриці передбачень.

Accuracy (Точність класифікації)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Інтерпретація: Частка правильно передбачених прикладів серед усіх прикладів.

Добре працює, коли класи збалансовані, але неінформативна при **незбалансованих класах**. Наприклад, якщо 95% — клас 0, модель може передбачати завжди 0 і мати 95% точності, але бути марною.

Precision (Точність позитивного класу)

$$Precision = \frac{TP}{TP + FP}$$

Частка правильних позитивних передбачень серед усіх передбачених позитивних. Важлива, коли **вартість хибнопозитивних** помилок висока (наприклад, діагностика раку — не можна помилково сказати, що хвороба є).

Recall (Повнота або чутливість)

$$Recall = \frac{TP}{TP + FN}$$

Частка правильних позитивних передбачень серед усіх істинно позитивних прикладів. Важлива, коли **вартість пропущеного позитивного прикладу** є високою (наприклад, виявлення шахрайства, хвороб, небезпеки).

F1-score

$$Recall = 2\frac{Precision \cdot Recall}{Precision + Recall}$$

Гармонічне середнє між Precision та Recall. Добре показує **баланс** між точністю та повнотою.

Specificity (Специфічність)

$$Specificity = \frac{TN}{TN + FP}$$

Здатність моделі правильно відкинути негативні приклади. Часто використовується разом з Recall (Sensivity) у медичних задачах.

Balanced Accuracy

$$Balanced\ Accuracy = \frac{Recall + Specificity}{2}$$

Підходить для **незбалансованих вибірок** — враховує успішність передбачення кожного класу окремо.

AUC-ROC (Area Under the Receiver Operating Characteristic Curve)

ROC-крива — це графік, який зображує (Рисунок 15):

- **X-вісь**: False Positive Rate (FPR) $= \frac{FP}{FP+TN}$
- **Y-вісь**: True Positive Rate (TPR) $=$ Recall

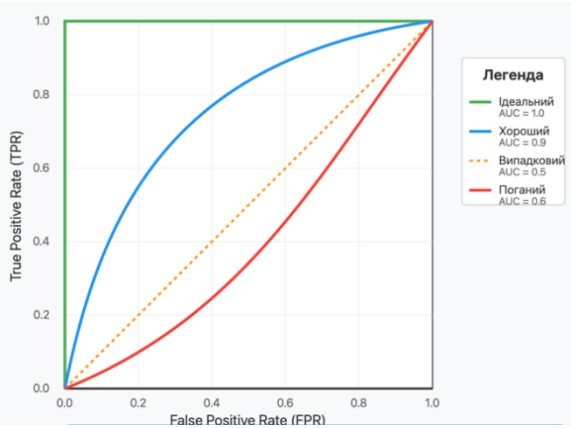

Рисунок 15. Приклад ROC-кривої.

AUC (Area Under Curve) — це площа під цією кривою.

- AUC=1.0 — ідеальна модель.
- AUC=0.5 — випадкове передбачення.
- AUC<0.5 — гірше випадковості (можливо, переплутані мітки).

Добре працює для **незбалансованих класів** і **ймовірнісних моделей**.

PR-крива (Precision-Recall curve)

Альтернатива ROC-криві для незбалансованих задач відображає (Рисунок 16):

- **X-вісь**: Recall
- **Y-вісь**: Precision

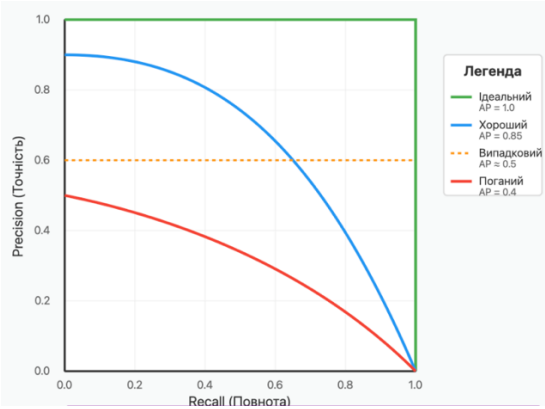

Рисунок 16. Приклад PR-кривої.

Якщо позитивний клас рідкісний — PR-крива дає більш інформативну оцінку.

Метрика	**Коли важлива**	**Можливі недоліки**
Accuracy	Збалансовані дані	Ігнорує структуру помилок
Precision	FP мають велику вартість (наприклад, банківські моделі)	Ігнорує FN
Recall	FN мають велику вартість (наприклад, медицина)	Ігнорує FP
F1-score	Компроміс між Precision і Recall	Складно інтерпретувати окремо
AUC-ROC	Враховує ймовірності передбачення	Може переоцінювати FP у задачах з рідким позитивним класом
PR-крива	Низький відсоток позитивного класу	Складна в інтерпретації порівняно з Accuracy

Таблиця 4. Порівняння метрик для задачі класифікації.

Жодна метрика не є універсальною. Завжди слід обирати метрику, яка відповідає бізнес-завданню або контексту застосування. Для незбалансованих задач точність (accuracy) може ввести в оману. Метрики типу Precision/Recall/F1/AUC важливі для об'єктивного аналізу класифікатора.

Питання для самоконтролю

1. Що таке гіперплощина в контексті лінійної класифікації?
2. Чому вихід лінійної моделі потрібно перетворювати сигмоїдою?
3. Як інтерпретувати вихід логістичної регресії?
4. Чому бінарна крос-ентропія є кращою функцією втрат для бінарної класифікації, ніж MSE?
5. Як змінюється функція втрат у залежності від відхилення передбачення від істинного класу?

Вправи

1. **Програмна вправа:** Реалізуйте функцію ***binary_cross_entropy(y_true, y_pred)*** самостійно без використання бібліотек.
2. **Графічна вправа:** Побудуйте вручну графік розділення двох класів за допомогою випадково згенерованих 2D-даних і лінійної моделі.
3. **Теоретична вправа:** Доведіть, що сигмоїда є гладкою, монотонною функцією і що її похідна визначається як:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

3. ГРАДІЄНТНИЙ СПУСК ДЛЯ НЕЙРОННИХ МЕРЕЖ ТА ОБЧИСЛЮВАЛЬНИЙ ГРАФ

Градієнтний спуск — це базовий метод оптимізації, що використовується для навчання нейронних мереж. Його мета — знайти такі значення параметрів (ваг) моделі, які мінімізують функцію втрат (loss function), тобто наблизити передбачення моделі до реальних значень.

Алгоритм працює за принципом руху у напрямку найшвидшого зменшення значення функції втрат — протилежному до градієнта цієї функції (Рисунок 17).

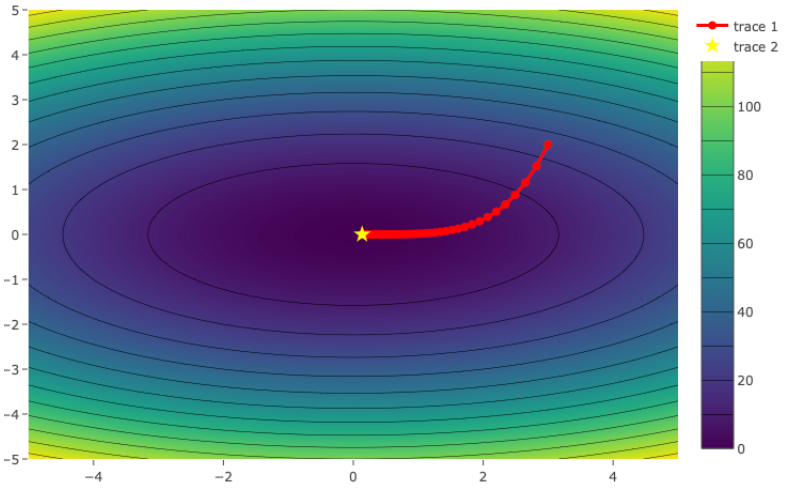

Рисунок 17. Приклад роботи градієнтного спуску.

3.1 Класичний алгоритм

Для одного параметра w оновлення виконується за формулою:

$$w = w - \eta \frac{\partial L}{\partial w}$$

де:

- η — коефіцієнт навчання (learning rate),
- $\frac{\partial L}{\partial w}$ — похідна (градієнт) функції втрат L відносно параметра w.

Цей процес повторюється ітеративно для всіх параметрів моделі до досягнення збіжності (minima loss).

Варіанти градієнтного спуску

1. **Повний градієнтний спуск (Batch Gradient Descent):**
   - Оновлення ваг виконується **після обчислення градієнтів по всій навчальній вибірці**.
   - Плюси: точне значення градієнта.
   - Мінуси: повільна робота на великих датасетах, потребує великої пам'яті.
2. **Стохастичний градієнтний спуск (SGD):**
   - Оновлення ваг після **кожного окремого зразка**:

$$w = w - \eta \frac{\partial L_i}{\partial w}$$

   - Плюси: швидке оновлення, дозволяє вирватись з локальних мінімумів.

- Мінуси: коливання, нестабільність, шум.

3. **Міні-батч градієнтний спуск:**
   - Компроміс між batch і SGD: оновлення ваг виконується після обробки **малої випадкової підмножини даних (батча)**.
   - Найбільш популярний варіант у практиці.
   - Звичайні розміри батчу: 16, 32, 64, 128 (залежно від задачі та об'єму пам'яті GPU/CPU).

3.2 Алгоритм оберненого розповсюдження похибки

Щоб оновити ваги, треба обчислити градієнти похідної втрат за кожним параметром (вагою). У складних багатошарових мережах пряме чисельне диференціювання — неефективне. **Backpropagation** - алгоритм, що дозволяє ефективно обчислювати градієнти для всіх параметрів нейронної мережі шляхом застосування **ланцюгового правила диференціювання** (chain rule).

Прямий прохід(forward pass):

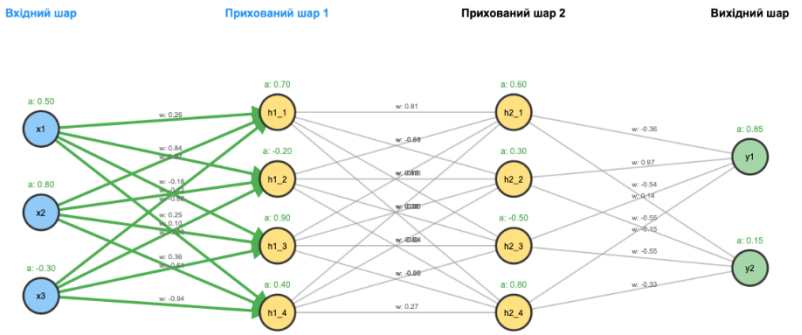

Рисунок 18. Обчислення значень на шарі 1.

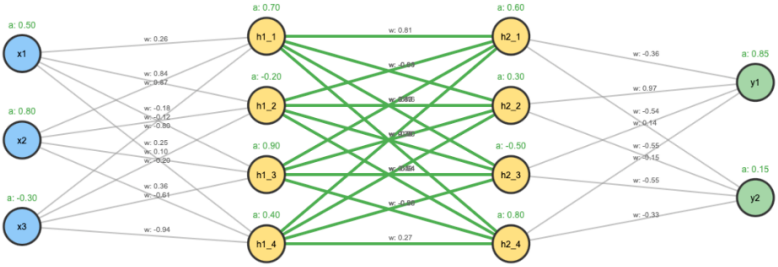

Рисунок 19. Обчислення значень на шарі 2.

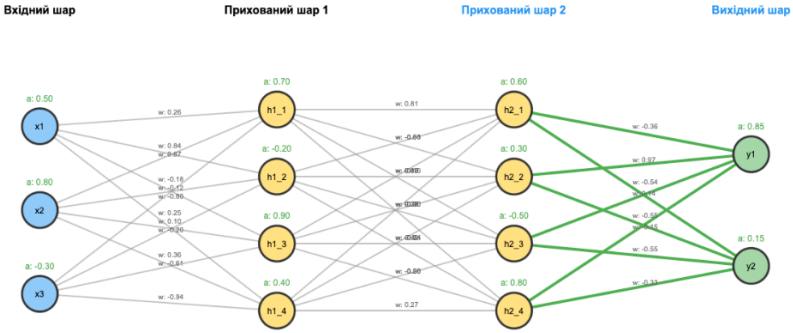

Рисунок 20. Обчислення значень на вихідному шарі.

Для кожного шару обчислюємо:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

де:

- $z^{(l)}$ — зважена сума входів у шар l,
- $a^{(l-1)}$ — активація шару l,
- $W^{(l)}$, $b^{(l)}$ — ваги і зсуви (bias) шару l,
- σ — функція активації (наприклад, ReLU або Sigmoid).

Обчислення витрат

$$L = L(y, \hat{y}) = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$$

де:

- y — справжнє значення,
- $\hat{y}$ — передбачене значення.

Зворотний прохід (backward pass)

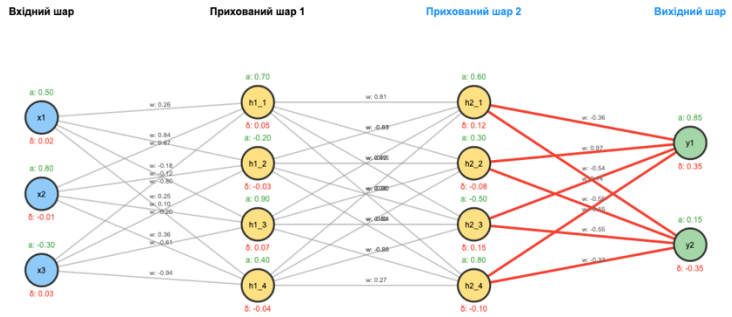

Рисунок 21. Обчислення градієнтів та помилок на вихідному шарі.

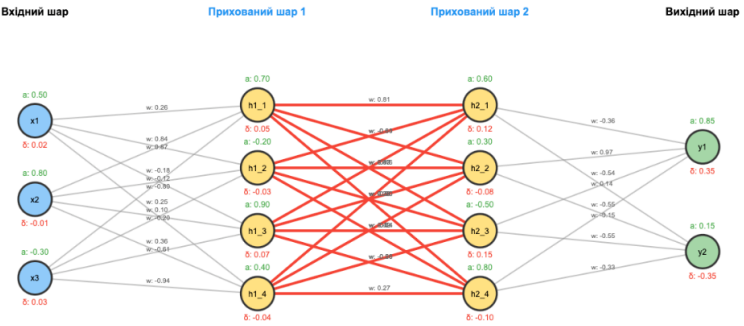

Рисунок 22. Обчислення градієнтів та помилок на шарі 2.

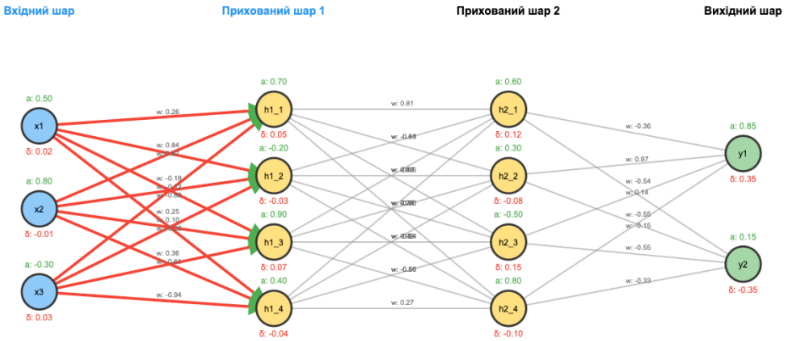

Рисунок 23. Обчислення градієнтів та помилок на шарі 1.

Розрахунок помилок та градієнтів для кожного шару:

1. Помилка на вихідному шарі:

$$\delta^{(L)} = \frac{\partial L}{\partial a^{(L)}} \cdot \sigma'(z^{(L)})$$

2. Помилка на прихованих шарах (рекурсивно):

$$\delta^{(l)} = (W^{(l+1)})^T \, \delta^{(l+1)} \cdot \sigma'(z^{(l)})$$

3. Градієнти для оновлення ваг і зсувів:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} \cdot (a^{(l-1)})^T$$

$$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$$

Оновлення параметрів

$$W^{(l)} \coloneqq W^{(l)} - \eta \cdot \frac{\partial L}{\partial W^{(l)}}$$
$$b^{(l)} \coloneqq b^{(l)} - \eta \cdot \frac{\partial L}{\partial b^{(l)}}$$

Проблеми, що виникають при використанні градієнтного спуску

1. **Місцеві мінімуми (local minima):**
   - Особливо актуальні для неглибоких мереж.
   - У складних моделей зазвичай виникає багато локальних мінімумів.
2. **Сідлові точки (saddle points):**
   - Точки, де градієнт дорівнює нулю, але вони не є мінімумами. Часто призводять до "застрягання" під час навчання.
3. **Вибух градієнтів (exploding gradients):**
   - Значення градієнтів стають дуже великими, що призводить до нестабільного оновлення ваг і переобучення або розвалу мережі.
   - Часто виникає в рекурентних мережах (RNN).
4. **В'ялі градієнти (vanishing gradients):**
   - Градієнти стають настільки малими, що оновлення ваг стає неефективним.
   - Проблема зникає частково завдяки ReLU та ініціалізації He.

Занадто малий $\eta \rightarrow$ повільне навчання, довго сходиться. Великий $\eta \rightarrow$ нестабільне навчання, можливі осциляції або вихід за межі оптимуму.

***Порада**: використовувати графік зміни функції втрат, щоб підібрати адекватне значення η. Також сучасні оптимізатори (наприклад, Adam) автоматично підлаштовують коефіцієнти. Використовувати **міні-батч** варіант у більшості практичних*

*задач. Поєднувати градієнтний спуск з **регуляризацією**, **батч-нормалізацією** та **оптимізаторами** для стабільнішого навчання. Проводити **візуалізацію** процесу навчання (криві loss/accuracy на train/test) для аналізу динаміки.*

3.3 Обчислювальний граф

Обчислювальний граф (computational graph) — це направлений ациклічний граф, у якому:

- **вузли (nodes)** представляють змінні, операції або константи;
- **ребра (edges)** означають залежності між цими елементами.

Обчислювальний граф дозволяє чітко відобразити порядок виконання обчислень у складних математичних виразах, таких як функції втрат нейронних мереж, і є основою автоматичного диференціювання, що критично важливе для навчання моделей.

Розглянемо простий приклад виразу:

$$L = (wx + b)^2$$

Обчислювальний граф для цього виразу виглядатиме так:

1. Вузли:
   - x: вхід, w: вага
   - b: зсув
   - wx: добуток
   - $wx + b$: сума
   - L: фінальний результат (квадрат)
2. Ребра:
   - від w та x до wx
   - від wx та b до $wx + b$
   - від $wx + b$ до L

Чітка структура дозволяє моделювати складні функції у вигляді набору простих операцій. На основі обчислювального графа можливо ефективно обчислювати градієнти для всіх параметрів шляхом зворотного проходу (алгоритм backpropagation). Граф можна аналізувати й оптимізувати (наприклад, об'єднання операцій, повторне використання результатів). Незалежні частини графа можна виконувати паралельно, що пришвидшує тренування моделей на GPU або TPU.

Forward pass (прямий прохід)

У прямому проході кожен вузол обчислює своє значення, використовуючи значення попередніх вузлів, аж до фінального результату.

У прикладі з $L = (wx + b)^2$:

- Обчислюємо wx

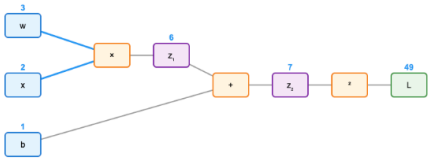

Рисунок 25. Обчислення значень wx.

- Потім $wx + b$

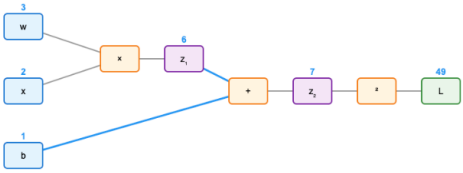

Рисунок 26. Обчислення значень $wx + b$.

- Потім підносимо до квадрату

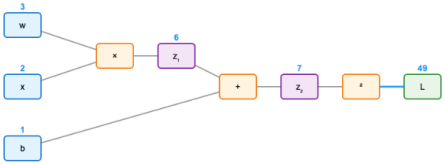

Рисунок 27. Обчислення значень $(wx + b)^2$

Backward pass (зворотний прохід)

Після прямого проходу обчислюється похідна (градієнт, Рисунок 28) функції втрат відносно кожного параметра моделі. Це реалізується за допомогою **ланцюгового правила** (chain rule).

У прикладі:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w}$$

де z $= wx + b$

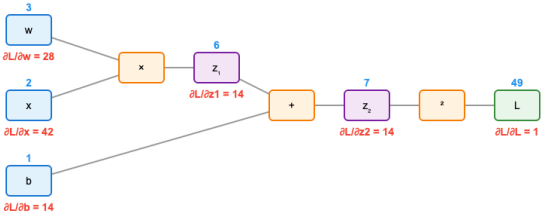

Рисунок 28. Обчислення градієнтів

Реалізація в фреймворках

Сучасні бібліотеки глибокого навчання, такі як **TensorFlow**, **PyTorch**, **JAX**, реалізують обчислювальні графи для автоматичного обчислення похідних:

- **TensorFlow (Static Graph)**: побудова графа виконується перед запуском, граф компілюється та оптимізується.
- **PyTorch (Dynamic Graph)**: граф формується динамічно під час виконання (eager execution), що дозволяє легше налагоджувати код.
- **JAX**: використовує XLA-компіляцію та дозволяє виконувати трансформації над графами (jit, grad, vmap).

Приклад коду з PyTorch:

```
import torch

# Параметри (вектори)
x = torch.tensor([3.0], requires_grad=False)  # вхідне значення
w = torch.tensor([2.0], requires_grad=True)   # ваги
b = torch.tensor([1.0], requires_grad=True)   # зміщення

# Forward pass
z = w * x + b
L = z ** 2  # функція втрат

# Backward pass (обчислення градієнтів)
L.backward()
```

В глибокому навчанні граф обчислення використовується для аналізу потоку градієнтів, виявлення проблем типу вибух/затухання градієнтів. Дає змогу будувати гнучкі

моделі з розгалуженими архітектурами та забезпечує відтворюваність та трасування обчислень, що важливо в наукових дослідженнях.

Питання для самоконтролю

1. У чому полягає різниця між Batch GD та Stochastic GD?
2. Чому в'ялі градієнти заважають навчанню? Як їх уникнути?
3. Як змінюється поведінка градієнтного спуску при зміні коефіцієнта навчання?
4. Чим пояснюється нестабільність SGD і як її зменшити?
5. Що таке обчислювальний граф і які його складові?
6. Як реалізується прямий і зворотний прохід?
7. У чому переваги динамічного графа над статичним?
8. Як обчислюється градієнт похідної за допомогою обчислювального графа?

Вправи

1. **Практична вправа:**
   - Реалізуйте mini-batch варіант SGD у Python.
   - Дослідіть вплив розміру батчу та значення $\eta\eta$ на швидкість і стабільність навчання.
2. **Дослідження:**
   - Проведіть серію експериментів із графіками loss та accuracy на synthetic dataset. Виведіть висновки.

4. ФУНКЦІЇ АКТИВАЦІЇ

Функції активації (activation functions) є ключовими компонентами штучних нейронних мереж. Їх основна роль — **вносити нелінійність** у модель. Без них нейронна мережа, незалежно від кількості шарів, поводилася б як проста лінійна модель, нездатна до апроксимації складних функцій.

Функції активації дозволяють моделі навчатися **нелінійним закономірностям** у даних, таким як розпізнавання образів, мовних конструкцій, складних залежностей тощо.

Вимоги до функції активації:

- **Нелінійність** — необхідна для ускладнення простору розв'язків.
- **Диференційованість** — важливо для застосування зворотного поширення похибки (backpropagation).
- **Обмежена область значень** — бажано, щоб значення не зростали до нескінченності.
- **Монотонність** — покращує стабільність градієнтів.
- **Обчислювальна ефективність** — важлива для великих моделей.

Основні функції активації

4.1 Sigmoid

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

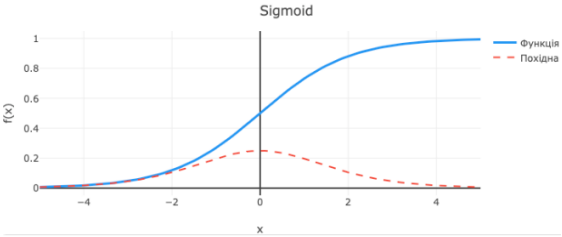

Рисунок 29. Графік сигмоїди

- Область значень: (0, 1)
- Використовується для ймовірнісного трактування результату (наприклад, у задачах бінарної класифікації).

Переваги:

- Зрозумілий інтерпретативний сенс.

Недоліки:

- В'ялі градієнти при великих або малих значеннях x (градієнт ≈ 0).
- Не нульове середнє значення → погана ініціалізація градієнтів у глибоких мережах.
- Повільна обчислювальна ефективність через e^{-x}

4.2 Tanh (гіперболічний тангенс)

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

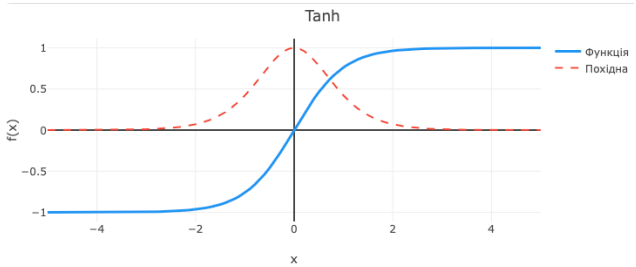

Рисунок 30. Графік tanh

- Область значень: (-1 , 1)

Переваги:

- Центрована відносно нуля → краще для симетричних оновлень ваг.
- Часто показує кращу збіжність, ніж Sigmoid.

Недоліки:

- Все ще має проблему в’ялих градієнтів (vanishing grads).

4.3 ReLU (Rectified Linear Unit)

$$\text{ReLU}\,(x) = \max\,(0, x)$$

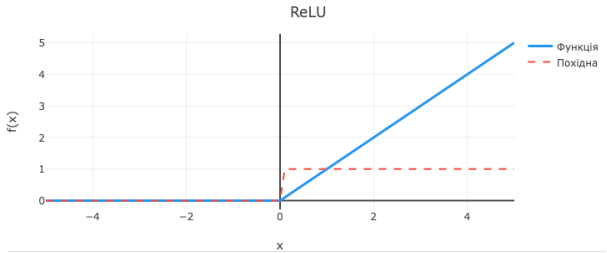

Рисунок 31. Графік ReLU

Найпопулярніша активація для глибоких нейронних мереж.

Переваги:

- Простота реалізації.
- Відсутність проблеми в'ялого градієнта для $x > 0$ Швидке обчислення.

Недоліки:

- Проблема "вмирання нейронів" (dead neurons) — якщо $x < 0$, то градієнт дорівнює 0, і ваги не оновлюються.

4.4 Leaky ReLU

$$\text{Leaky ReLU}\,(x) = \begin{cases} \text{x}, & \text{x} \geq 0 \\ \alpha\text{x}, & \text{x} < 0 \end{cases}$$

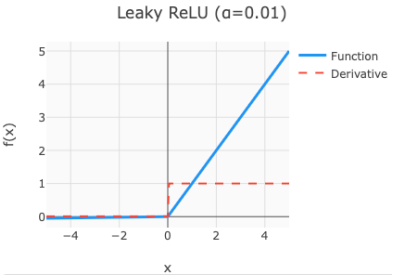

Рисунок 32. Графік Leaky ReLU

Значення α зазвичай невелике, наприклад 0.01.

Переваги:

- Дозволяє невеликий градієнт для $\text{x} < 0 \rightarrow$ менше "вмираючих" нейронів.

Недоліки:

- Значення α задається вручну і потребує тюнінгу.

4.5 Parametric ReLU (PReLU)

$$\text{PReLU}\,(x,\alpha) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

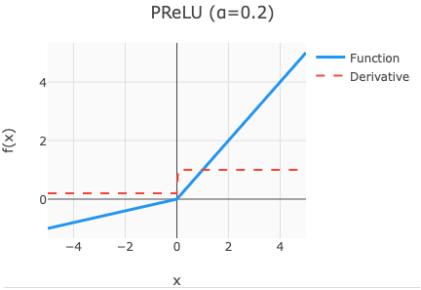

Рисунок 33. Графік PReLU

- На відміну від Leaky ReLU, параметр α навчається під час тренування.
- Показує покращення для деяких задач.

4.6 ELU (Exponential Linear Unit)

$$\text{ELU}\,(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^{x} - 1), & x < 0 \end{cases}$$

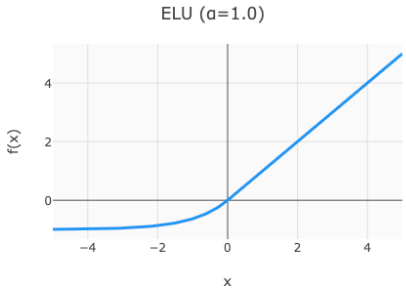

Рисунок 34. Графік ELU

Переваги:

- Менше "мертвих" нейронів.
- Центрує вихід навколо нуля.

Недоліки:

- Дорожче обчислювально через експоненту.

4.7 GELU (Gaussian Error Linear Unit)

$$GeLU\,(x) = x\Phi(x)$$

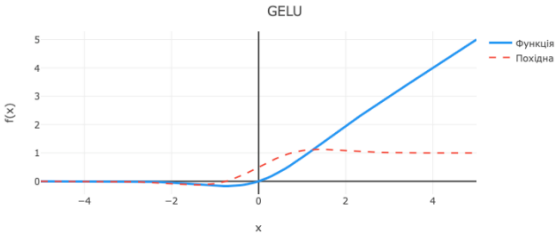

Рисунок 35. Графік GELU

де $\Phi(x)$ — кумулятивна функція нормального розподілу.

Використовується у **Transformers** (наприклад, BERT).

Переваги:

- Плавна активація, краще зберігає градієнти.

Недоліки:

- Обчислювально важка

4.8 Swish

$$\text{Swish}(x) = x\sigma(x) = \frac{x}{1 + e^{-x}}$$

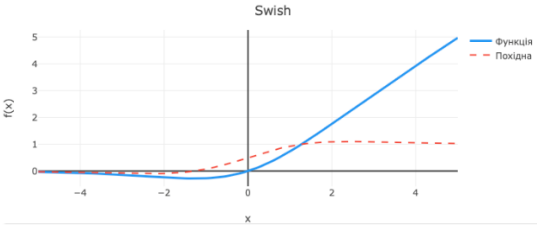

Рисунок 36. Графік Swish

Запропонована командою Google для використання в глибоких моделях (зокрема, у EfficientNet)

Переваги:

- Плавна, безшумна альтернатива ReLU.
- Утримує інформацію навіть при негативних x, на відміну від ReLU.
- Має **позитивну фазу градієнта** навіть при $x < 0$, що дозволяє уникати "мертвих" нейронів.

Недоліки:

- Трохи дорожча обчислювально через наявність $\sigma(x)$

Swish добре себе зарекомендувала в задачах з глибокими мережами, де важлива плавна зміна градієнтів. Її часто використовують у сучасних архітектурах у ролі заміни ReLU.

4.9 Softmax

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

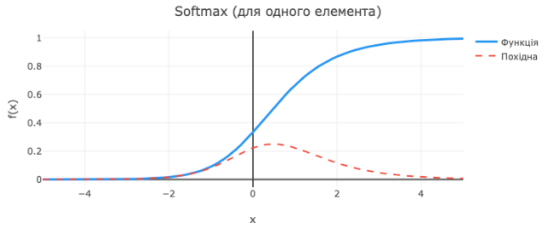

Рисунок 37. Графік Softmax

Використовується виключно у вихідному шарі багатокласових класифікаторів. Перетворює довільний вектор $x = [x_1, \dots x_n]$ у вектор ймовірностей, що сумуються до 1.

Переваги:

- Інтерпретовані результати — кожне значення вектору Softmax можна тлумачити як ймовірність належності до певного класу.

Недоліки:

- Вразлива до великих чисел (підвищена нестабільність без нормалізації або стабілізованих експонент).

Softmax зазвичай використовується **разом із функцією втрат CrossEntropyLoss**, яка поєднує логарифм Softmax із обчисленням ентропії для ефективнішої оптимізації.

Питання для самоконтролю

1. Чому важливо вносити нелінійність у нейронну мережу?
2. Які недоліки має функція Sigmoid?
3. У чому полягає проблема "мертвих" нейронів у ReLU?
4. Чим відрізняється Leaky ReLU від PReLU?
5. У яких випадках доцільно використовувати GELU?

Вправи

1. Побудуйте MLP для задачі бінарної класифікації. Проведіть експерименти зі зміною функцій активації (ReLU, Tanh, Sigmoid, Leaky ReLU) та порівняйте результати (accuracy, швидкість збіжності, стабільність навчання).
2. Поясніть, чому ReLU сприяє прискоренню навчання в порівнянні з Sigmoid.

5. МЕТОДИ ІНІЦІАЛІЗАЦІЇ

При навчанні глибоких нейронних мереж параметри (ваги) оновлюються за допомогою зворотного поширення помилки (backpropagation). Якщо початкові значення ваг обрані невдало:

- **Градієнти можуть затухати** — і модель майже не навчається.
- **Градієнти можуть вибухати** — і параметри стають дуже великими або NaN.
- **Мережа може швидко переобратися** до несиметричних рішень.

5.1 Затухаючи та вибухові градієнти

У глибоких нейронних мережах значення сигналу (активацій) та градієнтів проходять через **багато шарів**, кожен із яких трансформує їх за допомогою матриці ваг і функції активації.

Рисунок 38. Правильна ініціалізація

Якщо **дисперсія значень активації або градієнтів** не зберігається сталою в кожному шарі, тоді під час прямого проходження:

- сигнал або **затухає** (наближається до нуля, Рисунок 37)

Рисунок 39. Затухання градієнтів

- або **вибухає** (надмірно зростає),
  особливо при великій глибині моделі.

Рисунок 40. Вибух градієнтів

Нехай вхід до шару - випадкова величина x, вага w, активація - a. Тоді вихід:

$$y = wx$$

Припустимо:

$$x \sim \mathrm{N}(0, \sigma_x^2)$$

$$w \sim \mathrm{N}(0, \sigma_w^2)$$

Тоді:

$$Var(y) = Var(wx) = Var(w) \cdot Var(x) = \sigma_x^2 \sigma_w^2$$

Після L шарів:

$$Var\left(y^{(L)}\right) = (\sigma_w^2)^L \cdot Var(x)$$

- Якщо $\sigma_w^2 > 1 \rightarrow$ експоненційне **зростання дисперсії → вибух градієнтів**
- Якщо $\sigma_w^2 < 1 \rightarrow$ експоненційне **зменшення → затухання градієнтів**

5.2 Ініціалізація **Xavier**

Xavier (Glorot) ініціалізація підбирає дисперсію початкових ваг так, щоб і прямі активації, і зворотні градієнти не "вигорали" (не $\rightarrow 0$) і не "вибухали" ($\rightarrow \infty$) у міру проходження через шари.

Необхідні (достатні) умови для ініціалізації Xavier

1. Нульові сподівані

Інакше у шарах накопичувалося б зміщення, а ми аналізуємо лише збереження дисперсії.

$$E\left[w_{jk}\right] = 0, E[x_k] = 0, E\left[\delta z_j\right] = 0$$

2. Однакова (скінченна) дисперсія

$$Var\left(w_{jk}\right) = \sigma_w^2 = Var(x_k) = \sigma_x^2 = Var\left(\delta z_j\right) = \sigma_\delta^2$$

3. Незалежність

- w_{jk} - незалежні між собою
- x_k - незалежні між собою
- w_{jk} незалежні від x_k і від δz_j

Малі кореляції на практиці прийнятні, але повна залежність спотворить дисперсійні оцінки.

4. Однаковий розподіл ваг

5. Достатньо велике fan-in, fan-out

Формулювання задачі ініціалізації Xavier

Розглянемо один лінійний шар

$$z = Wx, \quad W_{jk} \sim N(0, \sigma_w^2), \quad x_k \sim D(0, \sigma_x^2)$$

де

- n_{in} – кількість входів (fan-in),
- n_{out} – кількість виходів (fan-out).

Припустимо незалежність w_{jk} та x_k. Тоді для кожного виходу

$$Var[z_j] = n_{in}\sigma_w^2\sigma_x^2$$

Щоб дисперсія не змінювалась від шару до шару, вимагаємо

$$Var[z_j] = \sigma_x^2 \rightarrow n_{in}\sigma_w^2 = 1 \rightarrow \sigma_w^2 = \frac{1}{n_{in}}$$

У зворотному проході для похибки на вході маємо

$$\delta_k = \sum_{j=1}^{n_{out}} w_{jk}\delta z_j\,, \qquad Var[\delta_k] = n_{out}\sigma_w^2\sigma_\delta^2$$

Вимагаючи $Var[\delta_k] = \sigma_\delta^2$, одержуємо

$$n_{out}\sigma_w^2 = 1 \rightarrow \sigma_w^2 = \frac{1}{n_{out}}$$

Для знаходження компромісу між умовами прямих активацій та зворотніх градієнтів береться середнє значення:

$$\sigma_w^2 = \frac{2}{n_{in} + n_{out}}$$

Розподіл ваг	**Параметри через fan-in/out**
Нормальний	$w_{jk} \sim N\left(0, \frac{2}{n_{in} + n_{out}}\right)$
Рівномірний	$w_{jk} \sim U[-a, a], a = \sqrt{\frac{6}{n_{in} + n_{out}}}$

Таблиця 5. Приклад різних розподілів

Збереження енергії сигнала гарантує, що средньоквадратична «енергія» вектора не змінюється ні у прямому, ні у зворотному проході – кожен наступний шар спостерігає схожу шкалу значень. Вивід базується лише на лінійності шару, він не залежить від конкретної нелінійної функції активації (але для ReLU дисперсія фактично падає вдвічі, тому для ReLU-мереж частіше беруть Не ініціалізацію $\sigma_w^2 = \frac{2}{n_{in}}$). Поки n_{in} достатньо велике, сума $w_{jk}x_k$ наближено нормально розподілена, й аналіз дисперсій коректно описує поведінку навіть для слабко корельованих вхідних ознак.

Деякі зауваження:

- **Незалежність** між вагами й активаціями – теоретична; на практиці кореляції з'являються, але наближення все ще корисне.
- **Нелінійності** зі значним насиченням (sigmoid) усе-таки втрачають масштаб; тому спільно з Xavier часто застосовують BatchNorm або змінюють нелінійність (tanh, ReLU).
- **Глибина мережі**. Для дуже глибоких моделей навіть точне збереження дисперсії не гарантує відсутність затухання інформації – тут допомагають ресідуал зв'язки та сучасні нормалізації.

Порівняння з іншими схемами

Ініціалізація	**Дисперсія**	**Коли використовувати**
Xavier (tanh/linear)	$2/(n_{in} + n_{out})$	сигмоїд/tanh, лінійні шари
He (Kaiming)	$2/n_{in}$	ReLU, Leaky-ReLU
Lecun	$1/n_{in}$	SELU / self-normalizing NNs

Таблиця 6. Порівняння методів ініціалізації

Xavier ініціалізація математично обґрунтовується вимогою **зберегти дисперсію як прямих активацій, так і градієнтів**. Розв'язуючи це рівняння для нормального та рівномірного розподілів, отримуємо просту, але дуже ефективну формулу, яка й зробила можливим глибоке навчання мереж із сигмоїдами. Таким чином, Xavier ініціалізація – це компроміс, що мінімізує загрозу затухання/вибуху градієнтів і робить навчання стабільним уже «з коробки».

6. МЕТОДИ ОПТИМІЗАЦІЇ

Процес навчання нейронної мережі — це задача **мінімізації функції втрат (loss function)**. Оптимізатори — це алгоритми, що коригують параметри (ваги й зсуви) мережі для зменшення значення функції втрат на навчальному наборі даних. Правильний вибір оптимізатора суттєво впливає на **швидкість збіжності**, **стабільність навчання**, **якість узагальнення** та **час навчання**.

6.1 Градієнтний спуск (Gradient Descent)

Стандартний алгоритм:

$$\theta = \theta - \eta \nabla_\theta L(\theta)$$

- θ — параметри моделі (ваги).
- η — коефіцієнт навчання (learning rate).
- $\nabla_\theta L(\theta)$— градієнт функції втрат відносно параметрів.

Типи:

- **Batch Gradient Descent**: використовує весь датасет для одного оновлення (висока точність, але повільний).
- **Stochastic Gradient Descent (SGD)**: оновлення після кожного прикладу (швидкий, але шумний).
- **Mini-batch Gradient Descent**: оновлення після кожного невеликого піднабору прикладів (компроміс між двома крайнощами, використовується найчастіше).

Проблеми:

- Застрягання в **локальних мінімумах**.
- **Вибухи або згасання градієнтів**, особливо у глибоких мережах.
- Повільне зближення в **анізотропних просторах**, коли напрямки мають дуже різні кривизни.

6.2 Покращені методи оптимізації

Momentum

Додає "інерцію" до оновлення параметрів, щоб уникати осциляцій та пришвидшувати збліження:

$$\nu_t = \gamma \nu_{t-1} + \eta \nabla_\theta L(\theta)$$

$$\theta = \theta - \nu_t$$

- $\gamma \in [0.5, 0.99]$ - коефіцієнт імпульсу.
- **Ідея**: згладжування градієнтів по напрямку руху.

Nesterov Accelerated Gradient (NAG)

Покращення Momentum: оцінює градієнт не в поточній точці, а в "поглянутій наперед" точці**:**

$$\nu_t = \gamma \nu_{t-1} + \eta \nabla_\theta L(\theta - \gamma \nu_{t-1})$$

$$\theta = \theta - \nu_t$$

Adagrad

Індивідуально масштабує коефіцієнт навчання для кожного параметра, ґрунтуючись на історії градієнтів:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + e}} \nabla_\theta L(\theta)$$

- G_t — сума квадратів попередніх градієнтів.

- **Перевага**: добре працює при **рідкісних** ознаках (наприклад, у NLP).
- **Недолік**: зменшення навчання з часом → зупинка навчання.

RMSProp

Розв’язує проблему Adagrad за допомогою експоненційного згладжування:

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho){g_t}^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{E[g^2]_t + e}} g_t$$

- $\rho \approx 0.9$
- Підходить для нелінійних, нестабільних функцій втрат, часто застосовується для RNN.

Adam (Adaptive Moment Estimation)

Об'єднує ідеї Momentum та RMSProp: обчислює адаптивне навчання і "імпульс" одночасно:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2){g_t}^2$$

$$\hat{m}_t = \frac{m_t}{1-{\beta_1}^t} \ , \qquad \hat{v}_t = \frac{v_t}{1-{\beta_2}^t}$$

$$\theta_{t+1} = \theta_t - \frac{\hat{m}_t}{\sqrt{\hat{v}_t + e}}$$

- Стандартні значення: $\beta_1 = 0.9,\ \beta_2 = 0.999, e = 10^{-8}$
- Найпопулярніший на практиці оптимізатор через стабільність і швидку збіжність.

AdamW (Weight Decoupled Regularization)

Варіант Adam із правильним застосуванням регуляризації L2:

- Відокремлює **weight decay** від градієнтів.
- Підтримується в PyTorch (torch.optim.AdamW), рекомендовано для **трансформерів**.

Порівняльна таблиця методів

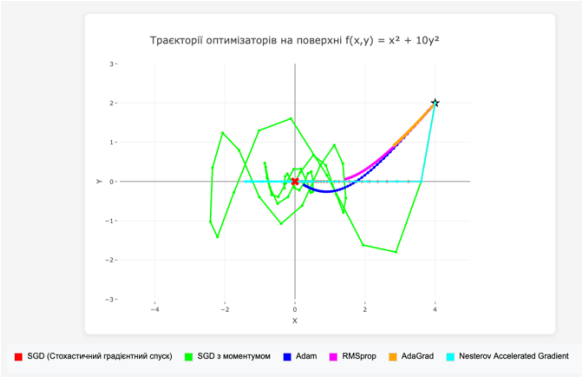

Рисунок 41. Приклад оптимізації функції різними методами.

Метод	Швидкість збіжності	Стабільність	Адаптивність	Переваги	Недоліки
SGD	Низька	Середня	Ні	Проста реалізація, контроль	Вимогливий до гіперпараметрів

				ована поведінка	
Momentum	Середня	Висока	Ні	Зменшує осциляції, пришвидшує навчання	Вимагає налаштування імпульсу
Adagrad	Висока на старті	Висока	Так	Підходить для рідких ознак	Навчання затухає з часом
RMSProp	Висока	Висока	Так	Добре працює на RNN	Не зберігає "імпульс"
Adam	Висока	Висока	Так	Найбільш універсальний оптимізатор	Може "застрягати" на плато
AdamW	Висока	Висока	Так	Краща регуляризація, стабільніша збіжність	Дещо складніший у налаштуванні

Таблиця 6. Порівняння методів оптимізації

- **Adam** — вибір за замовчуванням для більшості завдань.
- **SGD з Momentum** — рекомендований для великих моделей (ResNet, EfficientNet) на ImageNet.

- Для **мовних моделей** (NLP) або **трансформерів** краще використовувати **AdamW**.
- Значення **learning rate** сильно впливає на якість — використовуйте **learning rate scheduler** (наприклад, ReduceLROnPlateau, CosineAnnealing, OneCycle).
- Обов'язково моніторьте **втрати на валідації**, а не тільки на тренуванні.

Питання для самоконтролю

1. Що таке оптимізатор у контексті нейронної мережі?
2. Чим відрізняються Batch, Stochastic та Mini-batch градієнтний спуск?
3. Яку проблему вирішує метод Momentum у порівнянні зі стандартним SGD?
4. У чому полягає основна ідея методу Adagrad?
5. Чому RMSProp вважається вдосконаленням Adagrad?
6. Які дві моменти (похідні величини) використовуються в алгоритмі Adam?
7. Навіщо у формулі Adam використовується корекція упередженості $\hat{m}_t$, $\hat{v}_t$?
8. Яка ключова відмінність між Adam і AdamW?
9. У яких випадках варто надавати перевагу SGD замість Adam?
10. Як впливають занадто великі або занадто малі значення learning rate на процес навчання?

7. РЕГУЛЯРИЗАЦІЯ

7.1 Призначення регуляризації

Однією з головних проблем при тренуванні глибоких нейронних мереж є **перенавчання (overfitting)** — ситуація, коли модель демонструє високу точність на тренувальних даних, але погано узагальнює на нових, невідомих прикладах. Причиною є надто складна модель, яка запам'ятовує шум замість вивчення узагальнених закономірностей (Рисунок 41).

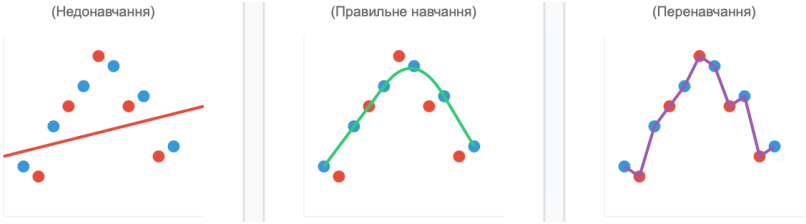

Рисунок 42. Приклад оптимізації функції різними методами.

Регуляризація — це набір методів, які спрямовані на обмеження складності моделі та покращення її здатності до узагальнення.

Рисунок 43. Оптимізація без регуляризації

7.2 Методи регуляризації

L2-регуляризація (Ridge, weight decay)

Рисунок 44. Оптимізація з L2 регуляризацією

Ідея: штрафування великих значень вагових коефіцієнтів, щоб зробити модель "гладкою" і менш схильною до перенавчання.

Математична формула:

Модифікована функція втрат:

$$L_{reg} = L_{orig} + \lambda \sum_{i} {w_i}^2$$

де:

- L_{orig} - початкова функція втрат (наприклад, крос-ентропія),
- w_i -вагові коефіцієнти моделі,
- λ - гіперпараметр регуляризації.

Інтерпретація: модель прагне зменшити як втрати, так і норму ваг.

L1-регуляризація (Lasso)

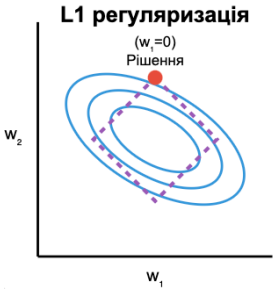

Рисунок 45. Оптимізація з L1 регуляризацією

Ідея: замість квадратів ваг використовується їх абсолютне значення.

$$L_{reg} = L_{orig} + \lambda \sum_i |w_i|$$

Особливість: L1-регуляризація сприяє зануленню деяких ваг → виникає **розрідженість (sparsity)** → ефективний метод для відбору ознак (feature selection).

Dropout

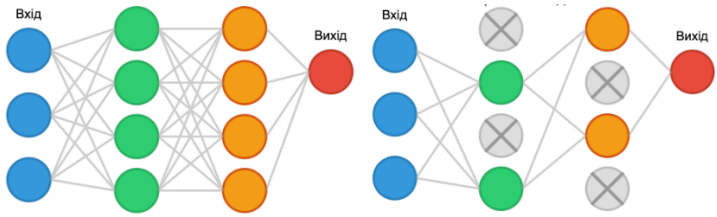

Рисунок 46. Приклад Dropout

Ідея: під час тренування випадково "відключати" (обнуляти) деякі нейрони з ймовірністю p. Це запобігає залежності між нейронами і покращує узагальнення.

Математично: під час тренування, на кожній ітерації для кожного нейрона генерується маска $m \sim Bernulli(p)$, і:

$$\tilde{h} = \mathrm{m} \cdot \mathrm{h}$$

де $\tilde{h}$ — вихід нейрона до **dropout**.

На етапі тестування значення не відкидаються, але масштабуються на p, щоб зберегти очікуване значення.

Переваги:

- Простий та ефективний спосіб боротьби з перенавчанням.
- Діє як ансамбль моделей без додаткових витрат.

DropConnect

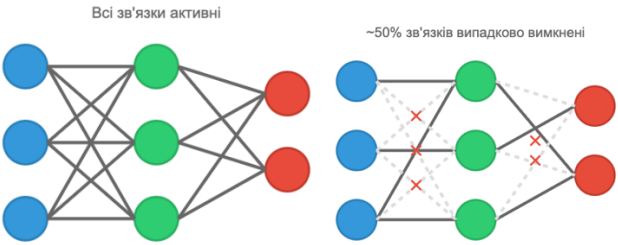

Рисунок 47. Приклад DropConnect

Аналог Dropout, але замість вимикання нейронів - вимикаються окремі ваги. Замість того, щоб зануляти активації нейронів, DropConnect випадково зануляє елементи матриці ваг:

$$\widetilde{W} = \mathrm{M} \cdot \mathrm{W}$$

де M — бінарна матриця з елементами з $Bernulli(p)$, що визначає, які з'єднання залишити, а які вимкнути.

Переваги:

- Дає додатковий рівень регуляризації.
- Може забезпечити кращі результати на деяких задачах, порівняно з Dropout.

Label Smoothing

Ідея: замість жорстких "one-hot" міток, використовуються **розгладжені мітки**, які моделюють невизначеність у даних.

Наприклад, замість мітки $\mathrm{y} = [0,0,1,0]$ використовують:

$$y_{smooth} = [0.05, 0.05, 0.85, 0.05]$$

Мета:

- Уникнути надмірної впевненості моделі.
- Покращити узагальнення.
- Зменшити переобучення.

ElasticNet

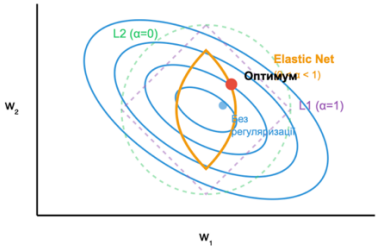

Рисунок 48. Приклад ElasticNet регуляризації

Комбінація L1 і L2 регуляризацій:

$$L_{reg} = L_{orig} + \lambda_1 \sum_i |w_i| + \lambda_2 \sum_i {w_i}^2$$

Переваги:

- Контролює як складність, так і кількість активних ознак.
- Часто використовується, коли потрібно як регуляризація, так і розрідженість.

Додаткові методи:

- **Early stopping** - зупинка тренування, коли валідаційна похибка перестає покращуватись.
- **Data augmentation** — розширення навчальної вибірки штучно зміненими прикладами.
- **Noise injection** — додавання шуму до входів або ваг під час тренування.

Регуляризація - ключовий елемент сучасного машинного навчання, без якого глибокі моделі не можуть бути ефективно застосовані до реальних задач. Правильний вибір методу регуляризації залежить від конкретної задачі, розміру вибірки, складності моделі та типу даних.

Питання для самоконтролю

1. У чому різниця між L1 і L2 регуляризацією?
2. Як працює dropout і чому він ефективний?
3. У чому переваги Label Smoothing?
4. Як впливає регуляризація на функцію втрат?
5. У яких випадках доцільно використовувати DropConnect замість Dropout?

Вправи

1. Реалізуйте модель MLP з L2-регуляризацією та протестуйте її на датасеті MNIST.
2. Порівняйте ефективність Dropout та DropConnect у простій задачі класифікації.
3. Побудуйте модель з Label Smoothing на задачі багатокласової класифікації та порівняйте її з класичною "one-hot" постановкою.

ВИСНОВКИ

Глибоке навчання — це сучасна парадигма машинного навчання, яка докорінно змінила уявлення про обробку даних, автоматизацію прийняття рішень і створення інтелектуальних систем. Вивчення цієї дисципліни вимагає розуміння як теоретичних основ, так і практичних аспектів побудови, навчання та аналізу нейронних мереж. У цьому методичному посібнику ми здійснили поступовий і системний рух від фундаментальних понять до ключових методів, що використовуються у створенні сучасних глибинних моделей.

Ми розпочали з базових принципів нейронних мереж: розглянули архітектуру штучного нейрона, схему його роботи та принципи побудови шарів. Далі, через задачу бінарної класифікації, ми ознайомились із лінійними моделями, функцією логіту та логістичною регресією. Це дало нам змогу закласти основу для розуміння того, як нейронна мережа приймає рішення.

Особливу увагу ми приділили функціям активації, які дозволяють мережі моделювати складні, нелінійні залежності. Було проаналізовано найпоширеніші активації, зокрема Sigmoid, Tanh, ReLU, Leaky ReLU, а також розкрито їхні переваги та недоліки. Далі ми детально зупинилися на методах ініціалізації ваг, адже саме цей етап суттєво впливає на збіжність та стабільність навчання: було пояснено мотивацію появи підходів Xavier та He, які є стандартом де-факто у сучасному глибокому навчанні.

Оптимізація — серце процесу навчання — була подана через розгляд градієнтного спуску та його покращених варіантів: Momentum, RMSProp, Adam. Кожен метод супроводжувався інтуїцією, формулами, прикладами коду, що дозволяє глибоко зрозуміти їхню логіку та сферу застосування.

Ще однією ключовою темою стала регуляризація. Ми розглянули класичні методи (L1, L2, Elastic Net), стохастичні техніки (Dropout, DropConnect), а також сучасні підходи до стабілізації навчання, як-от Label Smoothing. Було продемонстровано, як ці методи запобігають перенавчанню, зберігаючи при цьому здатність моделі узагальнювати.

Зміст посібника формує системне розуміння ключових компонентів глибокого навчання та готує читача до вивчення більш спеціалізованих тем — згорткових нейронних мереж (CNN), рекурентних моделей (RNN, LSTM), трансформерів, дифузійних моделей, а також застосування глибокого навчання у задачах комп'ютерного зору, обробки мови, генеративного моделювання та робототехніки.

Ми щиро сподіваємося, що цей посібник:

- дав змогу студентам сформувати міцну базу знань із глибокого навчання,
- надихнув на дослідження та створення власних моделей,
- сприяв усвідомленню місця штучного інтелекту в сучасному технологічному ландшафті.

Глибоке навчання — це не просто інструмент, це спосіб осмислення складності світу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. — MIT Press, 2016. — 800 p. — https://www.deeplearningbook.org/
2. Bishop C. M. Pattern Recognition and Machine Learning. — Springer, 2006. — 738 p.
3. Nielsen M. A. Neural Networks and Deep Learning. — Determination Press, 2015. — 224 p. — http://neuralnetworksanddeeplearning.com/
4. Chollet F. Deep Learning with Python. — 2nd ed. — Manning Publications, 2021. — 504 p.
5. Zhang A., Lipton Z. C., Li M., Smola A. J. Dive into Deep Learning. — 2021. — arXiv Open Source Book.
6. Rumelhart D. E., Hinton G. E., Williams R. J. Learning representations by back-propagating errors // Nature. — 1986. — Vol. 323. — P. 533–536.
7. Hochreiter S., Schmidhuber J. Long short-term memory // Neural Computation. — 1997. — Vol. 9(8). — P. 1735–1780.
8. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition // Proceedings of the IEEE. — 1998. — Vol. 86(11). — P. 2278–2324.
9. Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks // In: Proc. of AISTATS 2010. — P. 249–256.
10. Glorot X., Bordes A., Bengio Y. Deep Sparse Rectifier Neural Networks // In: Proc. of AISTATS 2011. — P. 315–323.
11. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: A Simple Way to Prevent

Neural Networks from Overfitting // Journal of Machine Learning Research. — 2014. — Vol. 15(1). — P. 1929–1958.
12. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proc. of ICML. — 2015. — P. 448–456.
13. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization // Proc. of ICLR. — 2015. — arXiv:1412.6980.
14. He K., Zhang X., Ren S., Sun J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification // Proc. of ICCV. — 2015. — P. 1026–1034.
15. Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y. Generative Adversarial Networks // Proc. of NIPS. — 2014. — arXiv:1406.2661.
16. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need // In: Advances in Neural Information Processing Systems (NeurIPS), 2017. — arXiv:1706.03762.
17. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proc. of CVPR. — 2016. — P. 770–778.

Навчальне видання

Терещенко Ярослав Васильович
Бобиль Богдан Володимирович
Свистунов Антон Олександрович

Основи глибокого навчання

Конспект лекцій