**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКИЙ КОЛЕГІУМ»**
**імені Т.Г. ШЕВЧЕНКА**

# Основи інтелектуального аналізу даних і машинного навчання

**МЕТОДИЧНІ ВКАЗІВКИ**
З ДИСЦИПЛІНИ «**Основи штучного інтелекту та інтелектуального аналізу даних**»

**Чернігів НУЧК 2025**

УДК 004.8 (072)

ORCID https://orcid.org/ 0000-0001-9290-7563
ORCID https://orcid.org/0000-0002-6178-6444

О-75 Основи інтелектуального аналізу даних і машинного навчання. Методичні вказівки з дисципліни “Основи штучного інтелекту та інтелектуального аналізу даних” / укладачі: Горошко Ю.В., Костюченко А.О., Чернігів: НУЧК, 2025, 81 с.

Методичні рекомендації складено для здобувачів освіти, які навчаються за освітньою програмою 122 Комп’ютерні науки першого (бакалаврського) рівня вищої освіти. Основною метою методичних вказівок є формування у студентів знань та практичних навичок з основних розділів дисципліни “Основи штучного інтелекту та інтелектуального аналізу даних”.

Укладачі:
Горошко Юрій Васильович, доктор педагогічних наук, професор кафедри інформатики і обчислювальної техніки Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка
Костюченко Андрій Олександрович, кандидат педагогічних наук, старший викладач кафедри інформатики і обчислювальної техніки Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка

Рецензенти:
Кандидат технічних наук наук, доцент, доцент кафедри інформаційних управляючих систем та технологій ДВНЗ “Ужгородський національний університет” Владислав Коцовський
Кандидат педагогічних наук, доцент, доцент кафедри інформатики і обчислювальної техніки Національного університету “Чернігівський колегіум” імені Т.Г.Шевченка Вінниченко Є.Ф.

Рекомендовано до друку вченою радою природничо-математичного факультету Національного університету «Чернігівський колегіум» імені Т.Г.Шевченка, протокол № 5 від 16 грудня 2024 р.

# Зміст

Інтелектуальний аналіз даних і машинне навчання – це взаємопов'язані галузі, які працюють з великими обсягами інформації, автоматизуючи процеси аналізу, виявлення закономірностей і прийняття рішень.

### 1. Інтелектуальний аналіз даних (Data Mining)

**Інтелектуальний аналіз даних** – це процес виявлення корисних, неочевидних закономірностей, трендів або інформації з великих наборів даних. Основна мета – перетворення даних у зрозумілу і корисну інформацію. Методи аналізу включають:

- Класифікацію (розподіл об'єктів по категоріях).
- Кластеризацію (групування схожих об'єктів).
- Виявлення аномалій (ідентифікація нестандартних поведінкових патернів).
- Асоціаційний аналіз (виявлення залежностей між змінними).
- Регресійний аналіз (пошук залежностей між змінними).

Приклади застосування:

- Аналіз клієнтської поведінки в бізнесі.
- Прогнозування фінансових ринків.
- Виявлення шахрайства у банківських операціях.

### 2. Машинне навчання (Machine Learning)

**Машинне навчання** – це підрозділ штучного інтелекту, що фокусується на створенні алгоритмів, здатних навчатися та покращувати свої показники без явного програмування. Модель "вчиться" на історичних даних і використовує отримані знання для прогнозів або прийняття рішень.

**Типи машинного навчання:**

*1.* ***Контрольоване навчання (Supervised Learning):***

- Навчання на розмічених даних (дані мають вхід і очікуваний вихід).
- Приклади: класифікація (пошук спаму в електронній пошті), регресія (прогнозування цін на нерухомість).

*2.* ***Неконтрольоване навчання (Unsupervised Learning):***

- Робота з нерозміченими даними.
- Приклади: кластеризація (групування клієнтів), зниження розмірності (спрощення складних наборів даних).

*3.* ***Навчання з підкріпленням (Reinforcement Learning):***

- Алгоритм вчиться шляхом проб і помилок, отримуючи винагороди за правильні дії.
- Приклад: створення агентів для ігор або управління роботами.

**Методи машинного навчання:**

- Лінійна/логістична регресія.
- Дерева рішень.
- Нейронні мережі та глибоке навчання (Deep Learning).
- Метод опорних векторів (SVM).
- Кластеризація (наприклад, метод K-середніх).

**Зв'язок між інтелектуальним аналізом даних і машинним навчанням**

**Інтелектуальний аналіз даних** є ширшою концепцією, яка охоплює різні методи, включаючи статистичний аналіз та машинне навчання.

**Машинне навчання** виступає інструментом для автоматизації процесів аналізу даних.

Ці галузі є ключовими в сучасному світі, оскільки їх застосування поширюється на медицину, фінанси, маркетинг, кібербезпеку та багато інших сфер.

## 3. Історія розвитку нейронних мереж

Історія розвитку **нейронних мереж** охоплює понад 70 років і характеризується етапами злетів і падінь. Вона пов'язана з дослідженнями в галузях біології, математики, комп'ютерних наук і штучного інтелекту. Ось ключові етапи виникнення та еволюції нейронних мереж:

### *1. Початок ідей (1940-ві роки)*

***• 1943 рік – перша модель нейрона:***

Воррен МакКаллок і Волтер Піттс запропонували першу математичну модель нейрона. У своїй роботі вони описали, як біологічний нейрон можна змоделювати за допомогою простих логічних операцій. Їхня модель стала основою для подальших досліджень штучних нейронів.

***• 1949 рік – правило Хебба:***

Дональд Гебб висунув гіпотезу про навчання біологічних нейронів: "Коли один нейрон активує інший, зв'язок між ними зміцнюється". Ця концепція стала ключовою ідеєю для механізму навчання нейронних мереж.

### *2. Ранні реалізації (1950–1960-ті роки)*

***• 1958 рік – персептрон Френка Розенблатта:***

Розенблатт створив персептрон — одну з перших реалізацій штучної нейронної мережі. Це була проста модель з одним шаром нейронів, здатна виконувати класифікацію. Однак, ця мережа мала обмеження: не могла розв'язувати задачі, що не є лінійно роздільними (наприклад, XOR-проблему).

***• 1960 рік – адаптивний резонансний перцептрон:***

Бернард Відроу і Марсель Хофф запропонували алгоритм навчання, відомий як **правило дельти**, який став основою для сучасного градієнтного спуску.

***3. "Зима" штучного інтелекту (1970–1980-ті роки)***

***• 1969 рік – критика від Мінського та Пейперта:***

У книзі "Персептрони" Марвін Мінський і Сеймур Пейперт показали обмеження одношарових мереж. Вони зазначили, що такі мережі не можуть розв'язувати задачі нелінійної класифікації, що спричинило втрату інтересу до цієї тематики.

***• Поява багатошарових мереж:***

У 1970-х роках дослідники розробили концепцію багатошарових нейронних мереж, але алгоритм навчання для таких мереж (зворотне поширення помилки) ще не був ефективно реалізований.

***4. Відродження нейронних мереж (1980-ті роки)***

***• 1986 рік – зворотне поширення помилки (Backpropagation):***

Девід Румельхарт, Джеффрі Хінтон і Рональд Вільямс запропонували алгоритм зворотного поширення помилки, що дозволило ефективно навчати багатошарові нейронні мережі. Це стало великим проривом у галузі.

***• 1989 рік – успіх конволюційних нейронних мереж:***

Ян ЛеКун застосував конволюційні нейронні мережі (CNN) для розпізнавання рукописних цифр. Його система використовувала згорткові операції і мала великий успіх у практичних задачах.

***5. Глибокі нейронні мережі (2000-ті – сьогодення)***

***• 2006 рік – епоха глибокого навчання:***

Джеффрі Хінтон і його колеги запропонували метод попереднього навчання глибоких нейронних мереж, що дозволило значно підвищити їх ефективність.

***• 2012 рік – прорив у комп'ютерному зорі:***

Алекс Крижевський, Ілля Суцкевер і Хінтон створили модель AlexNet, яка виграла змагання ImageNet. Її успіх довів, що глибокі мережі можуть обробляти великі набори даних із високою точністю.

• ***Сучасний розвиток (2015–2020-ті роки):***

Поява генеративних моделей, таких як **GAN** (Generative Adversarial Networks) і трансформери (наприклад, GPT). Удосконалення архітектур для обробки тексту, зображень і відео. Використання нейронних мереж у медичних дослідженнях, автономному транспорті, фінансах тощо.

**Ключові фактори успіху сучасних нейронних мереж:**

1. **Потужні апаратні засоби:** Розвиток графічних процесорів (GPU) та спеціалізованих чипів (TPU).
2. **Доступність великих обсягів даних:** Великі набори даних стали основою для навчання моделей.
3. **Оптимізація алгоритмів:** Розвиток нових методів оптимізації (наприклад, Adam, RMSProp).

Нині нейронні мережі є основою багатьох сучасних технологій, зокрема обробки природної мови (NLP), комп'ютерного зору, автономних систем і прогнозної аналітики.

## 4. Задачі, для яких використовуються нейронні мережі

**Нейронні мережі** використовуються для розв'язання широкого спектра задач, які потребують обробки великих обсягів даних, виявлення закономірностей і прийняття рішень. Основні сфери застосування включають:

**1. Обробка зображень і комп'ютерний зір**

• **Класифікація зображень:** Розпізнавання об'єктів, таких як тварини, транспортні засоби, обличчя.

*Приклад:* Системи безпеки, що ідентифікують обличчя, або програми для діагностики медичних знімків.

• **Сегментація зображень:** Визначення контурів і областей об'єктів на зображеннях.

*Приклад:* Виділення уражених тканин у медичних сканах.

• **Обробка відео:** Виявлення та відстеження об'єктів у відеопотоках.

*Приклад:* Системи для аналізу руху транспорту або безпілотного водіння.

**2. Обробка тексту та природна мова (NLP)**

- **Класифікація тексту:** Виявлення спаму в електронній пошті, категоризація новин, аналіз відгуків.
- **Машинний переклад:** Автоматичний переклад текстів (Google Translate).
- **Аналіз настроїв:** Визначення емоційного забарвлення тексту (позитивний, негативний, нейтральний).

*Приклад:* Аналіз відгуків клієнтів у соцмережах.

- **Генерація тексту:** Автоматичне створення статей, відповідей у чатах.

*Приклад:* GPT-моделі для генерації текстів.

- **Розпізнавання мови та синтез голосу:** Перетворення мови в текст і навпаки.

*Приклад:* Голосові асистенти (Siri, Alexa).

**3. Обробка аудіо**

- **Розпізнавання мовлення:** Перетворення звукового сигналу в текст.

*Приклад:* Голосові помічники, системи субтитрування.

- **Синтез мови:** Генерація природного голосу.

*Приклад:* Text-to-speech у навігаторах і навчальних програмах.

- **Аналіз звуку:** Розпізнавання музичних жанрів, діагностика технічного обладнання за звуками.

**4. Прогнозування та аналіз даних**

- **Прогнозування часу:** Передбачення змін у фінансових ринках, попиту на товари, погоди.
- **Виявлення аномалій:** Ідентифікація незвичайної поведінки систем.

*Приклад:* Виявлення шахрайства в банківських транзакціях.

- **Рекомендаційні системи:** Персоналізовані рекомендації продуктів, фільмів, музики.

*Приклад:* Netflix, Spotify, Amazon.

**5. Автономні системи**

- **Автономне водіння:** Виявлення об’єктів, розпізнавання дорожніх знаків і прийняття рішень у реальному часі.
- **Робототехніка:** Навчання роботів виконувати складні завдання.

*Приклад:* Маніпулятори на виробництві або роботи для домашнього використання.

**6. Генеративні моделі**

- **Генерація зображень:** Створення реалістичних зображень, графіки, 3D-моделей.

*Приклад:* Додатки для редагування фотографій або створення фотореалістичних облич.

- **Deepfake:** Створення відео або зображень із синтезованими елементами.
- **Генерація музики:** Композиція музичних треків на основі навчання на існуючих творах.

**7. Медична діагностика**

- **Аналіз медичних зображень:** Розпізнавання патологій у рентгенівських знімках, МРТ чи КТ.
- **Прогнозування ризиків:** Передбачення захворювань на основі медичних даних.

*Приклад:* Виявлення ризику серцевих захворювань.

- **Аналіз геномних даних:** Допомога в дослідженні ДНК.

**8. Ігри та моделювання**

- **Ігрові агенти:** Алгоритми, які можуть обігрувати людей у складних іграх (шахи, го, відеоігри).

*Приклад:* AlphaGo, OpenAI Five.

- **Моделювання процесів:** Сценарії реального світу для тренування автономних систем.

**9. Енергетика та екологія**

- **Оптимізація енергомереж:** Передбачення навантаження на енергосистеми.
- **Моніторинг навколишнього середовища:** Аналіз змін клімату або виявлення незаконного вирубування лісів.

**10. Фінансові технології**

- **Аналіз ризиків:** Оцінка кредитоспроможності клієнтів.
- **Торгівля на ринках:** Автоматизовані трейдингові системи.
- **Шахрайство:** Виявлення підозрілих транзакцій.

Нейронні мережі є універсальним інструментом, здатним адаптуватися до нових задач і сфер застосування, тому їх роль постійно зростає у всіх аспектах нашого життя.

## 5. Поняття штучного і біологічного нейронів

**Штучний нейрон** і **біологічний нейрон** є двома типами елементів, які мають спільну ідею: обробка та передача інформації. Однак вони значно відрізняються у своїй природі, структурі, функціональності та принципах роботи.

**1. Біологічний нейрон**

**Біологічний нейрон** – це клітина в нервовій системі живого організму, яка передає електрохімічні сигнали для обробки інформації та координації дій.

***Основна структура біологічного нейрона:***

1. **Сома (тіло клітини):** Основна частина нейрона, яка містить ядро і виконує метаболічні функції.
2. **Дендрити:** Гіллясті відростки, що отримують сигнали від інших нейронів.
3. **Аксон:** Довгий відросток, який передає сигнали іншим клітинам (нейронам, м'язам).
4. **Синапси:** Місця контакту між аксоном одного нейрона і дендритами іншого. Тут передача сигналів здійснюється за допомогою нейромедіаторів.

***Принцип роботи:***

- Нейрон отримує вхідні сигнали через дендрити.
- Якщо сукупний сигнал перевищує порогове значення, нейрон генерує **потенціал дії** (електричний імпульс).
- Потенціал дії передається вздовж аксона до синапсів, де викликає вивільнення нейромедіаторів, які впливають на наступний нейрон.

**2. Штучний нейрон**

**Штучний нейрон** – це математична модель, натхненна роботою біологічного нейрона, яка використовується в обчисленнях для створення штучних нейронних мереж.

***Основна структура штучного нейрона:***

1. **Входи (Input):** Сигнали (зазвичай числа), які надходять до нейрона.
2. **Ваги (Weights):** Числові коефіцієнти, що визначають важливість кожного входу.
3. **Зважена сума:** Нейрон обчислює суму всіх входів, помножених на відповідні ваги.

4. **Активуюча функція:** Математична функція, яка вирішує, чи повинен нейрон “активуватися” (передати сигнал далі).

*Приклади:* сигмоїдна функція, ReLU (Rectified Linear Unit), гіперболічний тангенс.

5. **Вихід (Output):** Результат обчислень, який передається іншим нейронам.

***Принцип роботи:***

- Нейрон отримує вхідні дані у вигляді чисел.
- Кожен вхід множиться на свою вагу, і результати додаються.
- Активуюча функція обчислює результат на основі цієї суми.
- Вихід передається на наступний шар мережі.

**Порівняння біологічного та штучного нейрона**

**Спільні риси:**

1. **Натхнення природою:** Штучний нейрон був створений як спрощена модель біологічного нейрона.
2. **Розпаралелення:** Як біологічні, так і штучні нейрони працюють у мережах, де одночасно активується багато елементів.
3. **Навчання:** Обидва типи здатні “вчитися” через зміну своїх параметрів (синаптична пластичність у біології, зміна ваг у штучному нейроні).

| Аспект | Біологічні нейрони | Штучні нейрони |
|---|---|---|
| **Природа** | Живі клітини в організмі | Математична модель у комп’ютері |
| **Складність** | Складна структура з тисячами дендритів і синапсів | Проста модель з обмеженою кількістю виходів |
| **Швидкість обробки** | Мілісекунди для передачі сигналу | Мікро- або наносекунди в обчисленнях |
| **Сигнал** | Електрохімічні імпульси | Числові значення |
| **Навчання** | Залежить від біологічних змін (пластичність синопсів) | Регулювання ваг через алгоритми навчання |
| **Зв'язок** | Тисячі входів і виходів (велика синоптична мережа) | Кількість входів і виходів обмежена моделлю |
| **Функції активації** | Складні нелінійні біологічні механізми | Просте математичне обчислення |

Біологічні нейрони – це основа роботи мозку, що дозволяє обробляти сенсорну інформацію і приймати рішення, тоді як штучні нейрони – це спрощені математичні аналоги, які використовуються для створення штучного інтелекту. Хоча вони значно відрізняються за природою, саме біологічні нейрони надихнули дослідників на розробку сучасних нейронних мереж.

## 6. Основні види нейронних мереж

Нейронні мережі поділяються на різні види залежно від архітектури, способу функціонування та застосувань. Основні види нейронних мереж такі:

### 6.1. Персептрон

**Персептрон** — це одна з перших математичних моделей штучної нейронної мережі, яка була створена для імітації роботи біологічних нейронів. Його запропонував **Френк Розенблатт** у 1958 році. Персептрон був здатний виконувати задачі класифікації, наприклад, розрізнення об'єктів за їх характеристиками.

**Структура персептрона**

Персептрон складається з трьох основних компонентів:

1. Вхідний шар:

   Містить нейрони (входи), які приймають вхідні дані.

   Кожен вхід має вагу $w_i$ , яка визначає, наскільки важливим є цей вхід.

2. Обчислювальний елемент:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

   Обчислює **зважену суму** вхідних сигналів:

   де $x_i$ — вхідні значення, $w_i$ — ваги, $b$ — зміщення (bias).

   Ця сума передається в **активаційну функцію**, яка визначає вихід нейрона.

**3. Вихідний шар:**

• Дає результат: 0 або 1 (для бінарної класифікації).

**Принцип роботи**

1. Вхідні дані $x_1, x_2, ..., x_n$ подаються на нейрон.
2. Для кожного входу $x_i$ його значення множиться на вагу $w_i$ .
3. Результати множення сумуються разом із зміщенням $b$ .

4. Зважена сума передається до активаційної функції, наприклад, **порогової функції**:

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

5. Отриманий результат $f(x)$ використовується як вихід класифікації.

**Алгоритм навчання персептрона**

Персептрон використовує простий алгоритм для навчання ваг:

1. **Ініціалізація ваг:** Початкові ваги $w_i$ та зміщення $b$ встановлюються випадковим чином або на нуль.
2. **Обчислення помилки:** Після передбачення помилка обчислюється як різниця між очікуваним результатом $y_{target}$ і фактичним $y_{pred}$ .
3. Оновлення ваг:

Якщо передбачення неправильне, ваги коригуються за формулою:

$w_i = w_i + \Delta w_i$,

де $\Delta w_i = \eta (y_{target} - y_{pred}) x_i$ ,

а $\eta$ — коефіцієнт навчання (learning rate).

4. **Повторення:** Процес повторюється для всіх навчальних прикладів до досягнення необхідної точності.

**Обмеження персептрона**

**1. Неможливість вирішувати нелінійно роздільні задачі:**

Наприклад, персептрон не може розв'язати задачу XOR (виняткове "або"), оскільки її класифікація потребує нелінійної межі.

**2. Одношаровість:**

Класичний персептрон є одношаровою моделлю, що обмежує його здатність вирішувати складні задачі.

**Переваги персептрона**

- Простота реалізації та навчання.
- Може ефективно вирішувати лінійно роздільні задачі.
- Став основою для розвитку більш складних моделей, таких як багатошарові персептрони (MLP).

**Історичне значення**

Персептрон став першим кроком у розвитку штучних нейронних мереж. Незважаючи на свої обмеження, він продемонстрував, що штучні нейрони можуть використовуватися для обчислень і розпізнавання патернів.

Подальші дослідження призвели до створення багатошарових моделей із використанням алгоритмів, таких як зворотне поширення помилки, що зробило нейронні мережі більш потужними.

### 6.2. Багатошаровий персептрон (Multilayer Perceptron, MLP)

**Багатошаровий персептрон (MLP)** — це один із найпоширеніших типів штучних нейронних мереж, який використовується для вирішення задач класифікації, регресії та прогнозування. Його головна відмінність від простого персептрона полягає в наявності одного або декількох прихованих шарів, що дозволяє моделювати нелінійні залежності.

**Структура багатошарового перцептрона**

**1. Вхідний шар (Input layer):**

- Містить нейрони, що отримують дані на вході.
- Кількість нейронів дорівнює кількості ознак у вхідних даних.

**2. Приховані шари (Hidden layers):**

- Один або більше шарів нейронів, які обробляють дані.
- Використовуються нелінійні функції активації (наприклад, ReLU, sigmoid, tanh), що дозволяє мережі навчати складні залежності між даними.

**3. Вихідний шар (Output layer):**

- Генерує результат. Кількість нейронів залежить від задачі:
- Один нейрон для регресії.
- Кількість нейронів відповідає числу класів у задачах класифікації.
- Для класифікації зазвичай використовується **softmax** або **sigmoid** як активаційна функція.

**Softmax** — це математична функція, яка перетворює вектор чисел (логітів) у вектор ймовірностей, де сума всіх значень дорівнює 1. Зазвичай використовується у вихідному шарі класифікаційних нейронних мереж для багатокласових задач.

**Формула Softmax**

Нехай $z = [z_1, z_2, ..., z_n]$ — вхідний вектор логітів (виходи попереднього шару мережі). Softmax обчислюється за формулою:

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

де:

$z_i$ — значення на вході для i -го класу,

чисельник дробу — експонента числа z_i ,

знаменник дробу — сума експонент усіх елементів вектора z .

**Властивості Softmax**

**1. Нормалізація:**

Значення кожного виходу функції $\sigma(z_i)$ лежить у межах від 0 до 1: $0 \le \sigma(z\_i) \le 1$.

Це забезпечує інтерпретацію результату як ймовірностей.

**2. Сума дорівнює 1:**

$$\sum_{i=1}^{n} \sigma\left(z_i\right) = 1$$

Сума всіх виходів завжди дорівнює 1:

**3. Чутливість до масштабу:**

Різниця між значеннями логітів впливає на вплив конкретного елемента у фінальній ймовірності.

**Роль у нейронних мережах**

**1. Класифікація:**

Softmax зазвичай використовується у багатокласових класифікаційних задачах. Він перетворює необроблені виходи (логіти) нейронної мережі у вектор ймовірностей для кожного класу.

**2. Функція втрат:**

У поєднанні з функцією крос-ентропії (cross-entropy loss) забезпечує стабільне навчання, що дозволяє моделі краще розрізняти класи.

**3. Вибір найімовірнішого класу:**

Після застосування Softmax найімовірніший клас визначається як той, який має найвищу ймовірність:

$$Class = arg\max_i \sigma\left(z_i\right)$$

**Приклад роботи Softmax**

Припустимо, є вектор логітів z = [2.0, 1.0, 0.1] . Обчислимо Softmax:

1. Обчислимо експоненти кожного елемента:

$$e^{z_1} = e^2, e^{z_2} = e^1 e^{z_3} = e^{0.1}$$

2. Знайдемо суму:

$$\sum \square e^2 + e^1 + e^{0.1}$$

3. Нормалізуємо кожен елемент:

Після виконання цих кроків отримаємо ймовірності для кожного класу.

**Переваги Softmax**

$$\sigma(z_1)=\frac{e^2}{\sum \square}, \sigma(z_2)=\frac{e^1}{\sum \square}, \sigma(z_3)=\frac{e^{0.1}}{\sum \square}$$

1. Інтерпретація результату як ймовірностей.
2. Забезпечує відносну оцінку впевненості моделі у кожному класі.
3. Добре працює у поєднанні з функцією крос-ентропії.

**Недоліки Softmax**

**1. Чутливість до великих значень:**

Може спричиняти чисельні нестабільності, якщо вхідні значення z мають великий діапазон. Це вирішується шляхом віднімання максимуму з z перед обчисленням:

$$\sigma(z_i)=\frac{e^{z_i-max(z)}}{\sum_{j=1}^{n} e^{z_i-max(z)}}$$

**2. Непропорційний розподіл ймовірностей:**

Якщо один з логітів значно більший за інші, Softmax робить його ймовірність домінуючою.

**Застосування**

- Багатокласова класифікація (NLP, комп'ютерний зір).
- Рекомендаційні системи.
- Задачі, що вимагають виходу у вигляді ймовірностей для кількох варіантів.

Softmax залишається основною функцією в багатокласових нейронних мережах завдяки своїй простоті та ефективності.

**Sigmoid (сигмоїдна функція)** — це нелінійна активаційна функція, яка часто використовується в нейронних мережах, особливо для задач, що потребують виходу у діапазоні від 0 до 1 (наприклад, бінарна класифікація). Функція визначається як:

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

**Функція визначається як:**

Графік S-подібний (sigmoid), симетричний відносно точки $x = 0$ .

Якщо x прямує до нескінченості, сигмоїд прямує до 1, якщо x прямує до мінус нескінченості, сигмоїд прямує до 0.

**Властивості Sigmoid**

**1. Нелінійність:**

Завдяки нелінійності Sigmoid дозволяє нейронним мережам моделювати складні залежності.

**2. Гладкість:**

Sigmoid є диференційовною функцією, що важливо для роботи алгоритмів зворотного поширення помилки (backpropagation).

**3. Монотонність:**

Функція зростає монотонно.

Похідна Sigmoid виражається через саму функцію:

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

Ця властивість спрощує обчислення під час навчання нейронних мереж.

**Роль у нейронних мережах**

**1. Активаційна функція:**

Sigmoid використовується в якості активаційної функції, яка переводить вихід нейрона у діапазон $(0, 1)$ . Це дозволяє інтерпретувати вихід як ймовірність.

**2. Бінарна класифікація:**

У задачах з двома класами Sigmoid застосовується у вихідному шарі разом із функцією втрат **бінарна крос-ентропія** (binary cross-entropy).

**3. Робота з імовірностями:**

Sigmoid підходить для задач, де потрібна ймовірність для одного вихідного класу.

**Недоліки Sigmoid**

**1. Ванішинг градієнт (затухання градієнта):**

- При великих або малих значеннях x , похідна функції стає дуже малою.

- Це ускладнює навчання глибоких нейронних мереж через повільне оновлення ваг.

**2. Нецентрований вихід:**

- Значення виходу лежать у діапазоні $(0, 1)$, а середнє значення $0.5$.
- Це може знижувати ефективність градієнтного спуску, бо викликає «зсув» градієнтів.

**3. Неефективність для багатокласових задач:**

- Для багатокласових задач частіше використовується **Softmax**, оскільки він враховує всі класи одночасно.

**Переваги Sigmoid**

- Проста реалізація.
- Інтерпретація вихідного значення як ймовірності.
- Добре працює у простих задачах, таких як логістична регресія.

**Альтернативи Sigmoid**

Через її обмеження в сучасних нейронних мережах часто використовуються альтернативи:

**1. ReLU (Rectified Linear Unit):**

Лінійна активаційна функція, що вирішує проблему затухаючого градієнта.

**2. Leaky ReLU:**

Модифікація ReLU, що дозволяє невеликі негативні значення.

**3. Tanh:**

Схожа на Sigmoid, але з виходом у діапазоні $(-1, 1)$. Центрована навколо нуля, що покращує збіжність.

**Принцип роботи MLP**

**1. Обчислення зваженої суми:**

Кожен нейрон обчислює зважену суму своїх входів:

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

де:

$w_{ji}$ — ваги з'єднань,

$x_i$ — вхідні значення,

$b_j$ — зміщення (bias).

2. Активація:

Результат $z_j$ передається через функцію активації, що додає нелінійність:

$a_j = f(z_j)$,

де f — функція активації.

**3. Пропуск даних через шари:**

Обчислення повторюється для кожного шару, передаючи вихід одного шару як вхід до наступного.

**4. Вихідні дані:**

Результати вихідного шару порівнюються з реальними значеннями для оцінки якості передбачення.

**Алгоритм навчання MLP**

Навчання багатошарового перцептрона виконується за допомогою алгоритму зворотного поширення помилки (backpropagation) та градієнтного спуску. Основні етапи:

**1. Прямий прохід (forward pass):**

Обчислення виходів усіх шарів до вихідного шару.

$$Loss = \frac{1}{n}\sum_{i=1}^{n}\left(y_i^{true} - y_i^{pred}\right)^2$$

**2. Обчислення помилки:**

•Використовується функція втрат (наприклад, MSE для регресії або крос-ентропія для класифікації):

**3. Зворотний прохід (backpropagation):**

Помилка поширюється назад через шари для обчислення градієнтів ваг.

**4. Оновлення ваг:**

Ваги коригуються за допомогою градієнтного спуску:

$$\omega_{ij} = \omega_{ij} - \eta\frac{\partial Loss}{\partial w_{ij}}$$

де $\eta$ — коефіцієнт навчання (learning rate).

**5. Повторення:**

Процес повторюється, доки функція втрат не стане досить малою або не досягне заданої кількості епох, тобто ітерацій перегляду всього датасету.

**Особливості багатошарового перцептрона**

**Переваги:**

1. **Здатність моделювати нелінійні залежності:** Завдяки використанню прихованих шарів і нелінійних функцій активації.
2. **Універсальність:** Підходить для різних задач: класифікації, регресії, прогнозування.
3. **Масштабованість:** Кількість шарів і нейронів можна змінювати залежно від складності задачі.

**Недоліки:**

1. **Складність навчання:** Велика кількість параметрів може призводити до перенавчання.
2. **Чутливість до гіперпараметрів:** Вимагає налаштування таких параметрів, як learning rate, кількість шарів, кількість нейронів.
3. **Обчислювальна складність:** Навчання великих моделей потребує значних обчислювальних ресурсів.

**Застосування MLP**

1. **Класифікація:** Розпізнавання образів, тексту, медичних діагнозів.
2. **Регресія:** Прогнозування цін, часових рядів.
3. **Прогнозування:** Аналіз трендів, передбачення попиту.
4. **Обробка сигналів:** Розпізнавання мови, аналіз звуку.

Отже багатошаровий перцептрон є базовим типом нейронних мереж, який служить основою для більш складних моделей, таких як згорткові мережі (CNN) чи рекурентні мережі (RNN). Він поєднує простоту реалізації з потужними можливостями моделювання.

### 6.3. Згорткові нейронні мережі (Convolutional Neural Networks, CNN)

**Опис:** Мережі, що спеціалізуються на обробці даних із просторовою структурою, наприклад, зображень.

**Ключові елементи:**

- **Згорткові шари:** Виділення локальних ознак.
- **Шари об'єднання (Pooling):** Зменшення розмірності даних.
- **Задачі:** Розпізнавання зображень, обробка відео, медична діагностика.

### 6.4. Рекурентні нейронні мережі (Recurrent Neural Networks, RNN)

**Опис:** Мережі, що враховують попередній контекст завдяки зворотним зв'язкам.

**Задачі:** Аналіз послідовностей, обробка текстів, розпізнавання мовлення.

**Обмеження:** Проблеми з тривалими залежностями через ефект “зникання градієнта”.

### 6.5. Удосконалені рекурентні мережі (LSTM і GRU)

**LSTM (Long Short-Term Memory):**

Мережа з додатковими “комірками пам’яті”, що дозволяє зберігати тривалі залежності. Використовується для прогнозування часових рядів, машинного перекладу.

**GRU (Gated Recurrent Unit):**

Спрощена версія LSTM із меншою кількістю параметрів.

### 6.6. Генеративно-змагальні мережі (Generative Adversarial Networks, GANs)

**Опис:** Система з двох нейронних мереж (генератора і дискримінатора), які “змагаються”.

- Генератор створює дані, схожі на реальні.
- Дискримінатор розпізнає, чи є дані реальними.

**Задачі:** Генерація зображень, відео, глибока фейкова графіка.

### 6.7. Автоенкодери (Autoencoders)

**Опис:** Мережі для зниження розмірності даних або їхнього відновлення.

Склад:

**Енкодер:** Зводить вхідні дані до меншого представлення.

**Декодер:** Відновлює дані з цього представлення.

**Задачі:** Зниження шуму, виявлення аномалій, стиснення даних.

### 6.8. Трансформери (Transformers)

**Опис:** Нейронні мережі, які базуються на механізмі уваги (attention) і відмовляються від рекурентних зв’язків.

**Особливості:** Паралельна обробка послідовностей, висока ефективність.

**Задачі:** Обробка природної мови (GPT, BERT), машинний переклад, створення текстів.

### 6.9. Радіальні базисні функції (RBF Networks)

**Опис:** Мережі, що використовують радіальні базисні функції як активаційні.

**Задачі:** Інтерполяція, регресія, класифікація.

### 6.10. Самоорганізуючі карти (Self-Organizing Maps, SOM)

**Опис:** Нейронні мережі для навчання без учителя, які проєктують багатовимірні дані в двовимірну площину.

**Задачі:** Візуалізація, кластеризація.

### 6.11. Резервуарні мережі (Reservoir Computing, RC)

**Опис:** Використовують великий фіксований резервуар для обробки послідовностей, де лише вихідні з'єднання навчаються.

**Приклад:** Echo State Networks (ESN).

**Задачі:** Аналіз часових рядів.

### 6.12. Гібридні мережі

**Опис:** Поєднують різні види нейромереж для досягнення кращих результатів.

**Приклад:** CNN+RNN для обробки відео, де CNN аналізує кадри, а RNN моделює послідовність кадрів.

Отже різні види нейронних мереж використовуються для вирішення різноманітних задач — від класифікації й обробки зображень до прогнозування та генерації даних. Вибір типу нейромережі залежить від специфіки проблеми, структури даних і мети аналізу.

## 7. Основні нейромережні парадигми

**Основні нейромережні парадигми** визначають підходи до навчання та функціонування нейронних мереж. Їх вибір залежить від типу задачі, наявних даних і очікуваних результатів. Виділяють три головні парадигми:

**1. Навчання з учителем (Supervised Learning)**

У цій парадигмі нейромережа навчається на мічених даних, де кожному вхідному зразку відповідає відомий вихід (мітка або значення).

**Основні характеристики:**

- Мережа намагається знайти залежність між входами та виходами.
- Під час навчання використовуються функції втрат (наприклад, MSE, крос-ентропія), які мінімізуються за допомогою алгоритмів оптимізації (наприклад, градієнтний спуск).

**Задачі:**

**Класифікація:** Присвоєння входу певного класу (розпізнавання облич, визначення категорії зображення).

**Регресія:** Прогнозування числових значень (прогноз погоди, ціни на акції).

**Приклади моделей:**

- Багатошаровий перцептрон (MLP).
- Згорткові нейронні мережі (CNN) для аналізу зображень.
- Рекурентні нейронні мережі (RNN) для роботи з послідовностями.

**2. Навчання без учителя (Unsupervised Learning)**

Ця парадигма працює з немічеными даними, де вихідні значення невідомі. Мета полягає у виявленні прихованих структур у даних.

**Основні характеристики:**

- Мережа кластеризує дані, знаходить аномалії або знижує розмірність.
- Виходи визначаються внутрішньою структурою даних.

**Задачі:**

**Кластеризація:** Групування схожих зразків (поділ клієнтів на сегменти).

**Зниження розмірності:** Зменшення кількості змінних для зручнішої візуалізації чи аналізу (наприклад, PCA або автоенкодери).

**Виявлення аномалій:** Знаходження незвичайних зразків у даних (шахрайські транзакції).

**Приклади моделей:**

- Автоенкодери (Autoencoders).
- Кластеризація k-means.
- Самоорганізуючі карти (Self-Organizing Maps, SOM).

**3. Навчання з підкріпленням (Reinforcement Learning, RL)**

У цій парадигмі агент (нейромережа) взаємодіє із середовищем і навчається на основі зворотного зв'язку у вигляді винагороди чи штрафу.

**Основні характеристики:**

- Агент приймає рішення на кожному кроці, які впливають на середовище.
- Мета — максимізувати сукупну винагороду за серію дій.
- Навчання базується на алгоритмах, таких як Q-навчання, політика градієнта.

**Задачі:**

**Робототехніка:** Оптимізація рухів роботів.

**Ігри:** Навчання агента вигравати в шахи, го, або відеоігри (AlphaGo, Dota 2).

**Управління:** Оптимізація роботи складних систем (енергомережі, виробництво).

**Приклади моделей:**

- Deep Q-Networks (DQN).

• Політика градієнта (Policy Gradient).

• Actor-Critic моделі.

**4. Напівконтрольоване навчання (Semi-supervised Learning)**

Поєднує риси навчання з учителем і без нього, використовуючи як мічені, так і немічені дані.

*Приклад:* Навчання моделі розпізнавання облич, де мічених прикладів мало, але є велика кількість немічених.

**5. Самонавчання (Self-supervised Learning)**

Мережа генерує мітки з власних даних і використовує їх для навчання. Цей підхід популярний у великих мовних моделях (GPT, BERT).

**6. Генеративне навчання (Generative Learning)**

Мета — створення нових даних, схожих на навчальні.

*Приклад:* Генеративно-змагальні мережі (GANs) для створення зображень або відео.

**Порівняння парадигм**

| Парадигма | Тип даних | Мета | Приклади застосування |
|---|---|---|---|
| **Навчання з учителем** | Мічені дані | Прогнозування / класифікація | Розпізнавання обличь, прогнозування погоди |
| **Навчання без учителя** | Немічені дані | Виявлення структур | Кластеризація клієнтів, пошук аномалій |
| **Навчання з підкріпленням** | Зворотній зв’язок | Оптимізація дій | Ігри, управління роботами, автоматичне водіння |

Отже, кожна з нейромережних парадигм має свої переваги та використовується для вирішення певних класів задач. Вибір парадигми залежить від типу даних, доступності міток і специфіки проблеми, яку потрібно розв’язати.

## 8. Програмне забезпечення для інтелектуального аналізу даних і машинного навчання

### 8.1. KNIME Analytics Platform

KNIME Analytics Platform — це потужний, гнучкий інструмент для аналізу даних, побудови моделей машинного навчання та автоматизації бізнес-процесів. KNIME (Konstanz Information Miner) створений як програмне забезпечення з відкритим вихідним кодом і широко

використовується в задачах обробки даних завдяки простоті візуального програмування.

**Основні характеристики KNIME**

**1. Візуальне програмування:**

- Робота в KNIME здійснюється через **інтуїтивний графічний інтерфейс**.
- Користувач створює робочі процеси шляхом з'єднання вузлів (**nodes**), які відповідають за конкретні дії (імпорт даних, трансформація, моделювання тощо).
- Не потрібно володіти глибокими знаннями програмування.

**2. Відкритий вихідний код:**

- KNIME є безкоштовною платформою з відкритим кодом, що робить її доступною для широкого кола користувачів.
- Можливість інтегрувати власні скрипти та модулі.

**3. Широка бібліотека вузлів:**

- Містить сотні готових вузлів для обробки даних, статистичного аналізу, візуалізації, машинного навчання та інтеграції з іншими інструментами.

**4. Підтримка розширень:**

- Інтеграція з Python, R, TensorFlow, H2O, Spark, та іншими платформами.
- Можливість додавання кастомних вузлів.

**5. Масштабованість:**

- KNIME підтримує аналіз великих даних через інтеграцію з Apache Spark та іншими платформами для роботи з big data.

**6. Візуалізація даних:**

- Різноманітні інструменти для створення графіків, таблиць, інтерактивних звітів.

**Функціонал KNIME:**

***1. Обробка даних:***

- Завантаження даних з різних джерел (SQL-бази, Excel, CSV, JSON, API).
- Очищення, нормалізація та об'єднання даних.

***2. Аналіз даних:***

- Моделювання та аналіз даних за допомогою статистичних методів.
- Виявлення трендів, кластеризація, кореляційний аналіз.

***3. Машинне навчання:***

• Побудова моделей класифікації, регресії, кластеризації.

• Алгоритми: дерева рішень, нейронні мережі, SVM, k-means тощо.

***4. Автоматизація процесів:***

• Побудова та автоматизація робочих процесів для регулярного аналізу.

***5. Великі дані:***

• Підтримка Hadoop та Spark для обробки великих наборів даних.

***6. Інтеграція з іншими системами:***

• Підтримка різних мов програмування (Python, R).

• API для розробки кастомних рішень.

**Переваги KNIME**

- **Простота у використанні:** Інтуїтивний інтерфейс з вузловою структурою.
- **Розширюваність:** Інтеграція зі сторонніми інструментами та бібліотеками.
- **Спільнота:** Велика база користувачів, активний форум, навчальні матеріали.
- **Кросплатформеність:** Доступність для Windows, macOS, Linux.

**Недоліки KNIME**

- **Вимогливість до ресурсів:** Для обробки великих даних може знадобитися потужне обладнання.
- **Крива навчання:** Для початківців можуть виникати складнощі з налаштуванням вузлів або інтеграцією.
- **Обмеженість для дуже специфічних задач:** Деякі вузли вимагають додаткового програмування або розширень.

**Застосування KNIME**

***1. Бізнес-аналітика:***

• Аналіз продажів, прогнозування попиту, сегментація клієнтів.

2. Біоінформатика:

• Аналіз геномних даних, дослідження лікарських сполук.

***3. Фінанси:***

• Виявлення шахрайства, кредитний скоринг, управління ризиками.

***4. Маркетинг:***

• Аналіз кампаній, поведінки споживачів, прогнозування конверсії.

***5. Наукові дослідження:***

• Обробка великих масивів даних, тестування гіпотез.

Отже KNIME Analytics Platform — це потужний інструмент для обробки, аналізу та візуалізації даних, який поєднує зручність використання з високою функціональністю. Завдяки своїй відкритій архітектурі та широким можливостям інтеграції KNIME підходить як для початківців, так і для досвідчених фахівців у галузі науки про дані.

## 8.2. Фреймворки Python

Python пропонує багато фреймворків для створення нейронних мереж і роботи з машинним навчанням. Вони відрізняються за функціональністю, продуктивністю та рівнем складності. Ось найпопулярніші фреймворки:

**1. TensorFlow**

**Розробник:** Google Brain.

**Опис:**

• Один із найпопулярніших фреймворків для машинного навчання.

• Забезпечує підтримку як високорівневих API (Keras), так і низькорівневого контролю.

• Підтримує створення нейронних мереж будь-якої складності, включаючи згорткові (CNN), рекурентні (RNN), трансформери тощо.

**Особливості:**

• Підтримка GPU та TPU.

• Інструменти для деплойменту моделей у мобільних і веб-додатках.

• TensorFlow Lite для мобільних пристроїв і TensorFlow.js для браузерів.

**Використання: Глибоке навчання, обробка зображень, тексту, великих даних.**

**2. PyTorch**

**Розробник:** Facebook AI Research (FAIR).

**Опис:**

• Гнучкий і інтуїтивний фреймворк для побудови та тренування нейронних мереж.

• Активно використовується в наукових дослідженнях і реальних додатках.

**Особливості:**

- Динамічне обчислення графів (граф будується під час виконання програми).
- Підтримка роботи з GPU.
- Вбудовані бібліотеки для обробки зображень і тексту (Torchvision, Torchtext).

**Використання: Дослідження, створення експериментальних моделей, нейронні мережі для NLP і CV.**

**3. Keras**

**Розробник:** Спільнота open-source.

**Опис:**

- Високорівнева бібліотека для створення нейронних мереж, яка працює поверх TensorFlow.
- Орієнтована на швидкий прототип моделей.

**Особливості:**

- Простий у використанні API.
- Можливість швидкого налаштування шарів і моделей.
- Інтеграція з TensorFlow для розгортання моделей.

**Використання:** Для початківців, які хочуть створювати прості й ефективні моделі.

**4. Scikit-learn**

**Розробник:** Спільнота open-source.

**Опис:**

- Бібліотека машинного навчання для класичних алгоритмів (SVM, дерева рішень, кластеризація тощо).
- Підтримка простих нейронних мереж (MLPClassifier, MLPRegressor).

**Особливості:**

- Інтеграція з бібліотеками для обробки даних, такими як Pandas і NumPy.
- Простота використання для базових задач машинного навчання.

**Використання:** Класичне машинне навчання, аналіз даних, побудова простих нейромереж.

**5. Theano**

**Розробник:** Université de Montréal.

**Опис:**

- Один із перших фреймворків для роботи з нейронними мережами.
- Використовується для обчислення математичних виразів із багатовимірними масивами.

**Особливості:**

- Висока продуктивність завдяки використанню GPU.
- Орієнтований на математичні обчислення.

**Використання:** Створення базових моделей нейронних мереж. Використовується як основа для інших бібліотек (наприклад, Keras).

**6. MXNet**

**Розробник:** Apache Software Foundation.

**Опис:**

- Гнучкий і масштабований фреймворк для створення нейронних мереж.
- Використовується в Amazon Web Services (AWS).

**Особливості:**

- Підтримка імперативного та декларативного програмування.
- Підтримка кількох мов програмування (Python, Julia, R).

**Використання:** Аналіз великих даних, хмарні обчислення.

**7. Caffe**

**Розробник:** Berkeley Vision and Learning Center (BVLC).

**Опис:**

- Спеціалізується на згорткових нейронних мережах для комп'ютерного зору.

**Особливості:**

- Швидкість навчання та висока продуктивність.
- Використовується переважно для аналізу зображень.

**Використання:** Обробка зображень, розпізнавання об'єктів.

**Порівняння**

| **Фреймовр** | **Рівень** | **Головна перевага** | **Основне застосування** |
|---|---|---|---|
| **TensorFlow** | Середній/ Високий | Масштабованість та інструменти | Універсальний |
| **PyTorch** | Середній | Гнучкість і зручність | Дослідження, |

| | | | |
|---|---|---|---|
| | | | NLP, CV |
| **Keras** | Початковий | Простота використання | Прототипування |
| **Scikit-learn** | Початковий | Класичне ML | Аналіз даних, базові моделі |
| **MXNet** | Середній | Масштабованість | Велики дані, хмари |
| **Caffe** | Середній | Висока продуктивність | Комп'ютерний зір |

## 9. Лабораторна робота 1 KNIME

**The KNIME Workflow**

KNIME не працює з скриптами, він працює з робочими процесами.

Робочий процес - аналіз потоку, який є послідовністю кроків аналізу, необхідних для досягнення заданого результату.

Робочий процес - це алгоритм аналізу даних, який на традиційній мові програмування буде виконуватися ряд інструкцій та викликів до функцій. KNIME реалізує його графічно. Це графічне подання полегшує огляд процесу аналізу та його документування.

Вузол ("нод") - це єдиний блок обробки робочого процесу.

Вузол приймає набір даних як вхідний, обробляє його і робить його доступним на вихідному порті. Дія "обробки" вузла коливається від моделювання, як-от вузол Artificial Neural Network Learner, до маніпуляцій з даними, як перенесення матриці вхідних даних з графічних інструментів, таких як сюжет розсіювання, на операції читання / запису.

Кожен вузол KNIME має три стани:

Неактивний і ще не налаштований червоне світло світлофора.

Налаштований, але ще не виконаний жовте світло світлофора.

Виконано успішно зелене світло світлофора.

Якщо вузол виконується з помилками, його статус залишається на жовтому кольору світлофора.

Вузли, що містять інші вузли, називаються мета-вузлами.

**Node Repository**

Репозиторій вузлів містить всі вузли KNIME, упорядковані за категоріями. Категорія може містити іншу категорію, наприклад, категорія Читання є підкатегорією категорії IO. Вузли додаються зі сховища до редактора робочого процесу шляхом перетягування їх у редактор робочого процесу.

Вибір категорії відображає всі вузли, що містяться в представленні опису вузла; Вибір вузла відображає довідку для цього вузла.

Якщо відома назва вузла, можна ввести частину імені у поле пошуку сховища вузлів.

**Вузли зчитування даних:**

У вузлах зчитування даних є можливість змінювати типи стовпців.

**Excel Reader (XLS)**

Цей вузол читає електронну таблицю і забезпечує її в її вихідному порту. Він зчитує тільки дані з одного аркуша в даний момент. Він може зчитати тільки числові дані, дати, булеві значення, рядкові дані, але, ніяких діаграм, малюнків або інших об'єктів.

В даний час KNIME підтримує наступні типи - String, час and дати (Time and Date), числа з плавуючою точкою (Double), Boolean, і цілочисельні (Integer).

**CSV Reader**

Вузол читає файли з розширенням CSV. Використання цього вузла відбувається, якщо робочий процес використовується в серверному або пакетному середовищі, а структура вхідних файлів змінюється між різними викликами. Зокрема, це включає в себе змінну кількість стовпців введення. Після виконання вузла буде перевірятися вхідний файл, щоб визначити число і типи стовпців та вивести таблицю з автоматично вгаданою структурою.

**File Reader**

Цей вузол може використовуватися для читання даних з файлу ASCII або URL-адреси. Він може бути налаштований на читання різних форматів.

Коли відкривається діалогове вікно налаштування вузла та надане ім'я файлу, він намагається вгадати налаштування зчитування, аналізуючи вміст файлу. Потім необхідно перевірити результати цих налаштувань у таблиці попереднього перегляду. Якщо дані відображені неправильно або виявлено помилку, є можливість налаштувати параметри вручну.

**Вузли перегляду:**

**Interactive Table**

Відображення даних у вигляді таблиці. Якщо кількість рядків невідома, під час перегляду підраховується кількість рядків при відкритті. Крім того, рядки можуть бути виділені та відтворені.

**Box Plot**

Box Plot показує надійні статистичні параметри: мінімальний, нижню чверть (квартель), медіани, верхню чверть (квартель) і максимум. Ці параметри називаються надійними, оскільки вони не чутливі до крайніх викидів.

**Scatter Plot**

Створює розбивку з двох виділених атрибутів. Тоді кожна база даних відображається як точка у відповідному місці, залежно від її значень

вибраних атрибутів. Точки відображаються у кольорі, визначеному диспетчером кольорів, розміром, визначеним диспетчером розмірів, і формою, визначеною керівником Форми.

**Вузли аналітики:**

**Statistics**

Цей вузол розраховує статистичні моменти, такі як мінімальне, максимальне, середнє, стандартне відхилення, дисперсія, медіана, загальна сума, кількість відсутніх значень та кількість рядків у всіх цифрових стовпчиках, а також підраховуються всі номінальні значення разом з їх появою.

**Вузли маніпуляції для рядків. Фільтри**

**Nominal Value Row Filter**

Фільтрує рядки на основі вибраного значення номінального атрибута. Можна виділити номінальний стовпець та один або більше номінальних значень цього атрибута.

**Вузли маніпуляції для рядків**

**Concatenate**

Цей вузол об'єднує дві таблиці. Таблиця у inport 0 вказується як перша вхідна таблиця (верхній вхідний порт), таблиця в inport 1 - друга таблиця, resp. Колонки з однаковими іменами об'єднані (якщо типи стовпчиків відрізняються, тип стовпця є загальним базовим типом обох вхідних стовпців). Якщо в таблиці вводу містяться імена стовпців, які не мають іншої таблиці, стовпці можуть бути заповнені відсутніми значеннями або відфільтровані, тобто вони не будуть в таблиці виводу.

**Row Filter**

Вузол дозволяє фільтрувати рядки відповідно до певних критеріїв. Він може включати або виключати: певні діапазони (за номером рядка), рядки з певним ідентифікатором рядка та рядки з певним значенням у вибраному стовпці (атрибут).

**Вузли маніпуляції для стовпців. Фільтри**
**Column Filter**
Цей вузол дозволяє відфільтрувати стовпці з вхідної таблиці, що передаються до вихідної таблиці. У діалоговому вікні стовпці можна переміщувати між списком Include і Exclude.

**Вузли маніпуляції для стовпців. Конвертація та зміна**

**Column Rename**

Перейменування назви стовпців або змінювання їхніх типів. У діалоговому вікні можна змінити назву окремих стовпців, відредагувавши текстове поле або змінюючи тип стовпця, вибравши один із можливих типів у комбінаторі. Конфігурація з червоним кольором показує, що налаштований стовпець більше не існує.

**Вузли аналітики Weka – Класифікація алгоритмів**

**NaiveBayes**

Клас для наївного класифікатора Байєса за допомогою класів оцінок

**Вузли аналітики Weka – Передбачення**

**Weka Predictor**

Weka Predictor приймає модель, сформовану в вузлі weka, і класифікує дані тесту в inport.

**Практичні завдання**

Для виконання завдань необхідно встановити програмне забезпечення KNIME Analytics Platform.

Для виконання деяких вузлів, необхідно встановити додаткові розширення, так як у базовій версії не всі вузли наявні.

Потім необхідно встановити додаткові розширення File – Install KNIME Extentions. Далі обрати всі розширення і встановити.

**Пояснення до таблиць**

БАЗА ДАНИХ СТУДЕНТІВ містить дані, які були випадковим чином створені для вступу на освітній ступінь Магістр.

1. Прізвище
2. Ім’я
3. По-батькові
4. Стать
5. Астрономія : оцінка
6. Адміністрування інформаційних систем : оцінка
7. Історія та методологія інформатики : оцінка
8. Вибіркові обстеження в педагогіці, психології та соціології : оцінка
9. Методи досліджень та аналіз даних в освіті : оцінка
10. Методика застосування комп'ютерної техніки при викладанні предметів шкільного курсу : оцінка
11. Організація дистанційної освіти у навчальному закладі : оцінка
12. Педагогічна творчість : оцінка

13. Практикум з розв'язування задач з інформатики : оцінка
14. Теоретичні основи інформатики : оцінка
15. Курсова робота з інформаційних технологій : оцінка
16. Педагогічна практика : оцінка
17. Атестаційний екзамен : оцінка
18. Вступ : Чи вступив до магістратури?

**Виконання**

**Створіть робоче середовище з назвою Pract1_1**

**1. Завантажте набір даних «Студенти.xls» за допомогою вузла «XLS Reader»**

A. Виберіть файл xls за допомогою кнопки «Configure»

B. Встановіть, що таблиця містить імена стовпців у рядку 1

C. Перевірте прапорець the Read entire data sheet, or…

D. Перевірте прапорець «The Skip empty columns and skip empty row checkboxes»

E. Натисніть кнопку «ОК» і виконайте вузол

F. Перегляньте таблицю

**2. Завантажте набір даних «Студенти.cvs» за допомогою вузла «CSV Reader»**

A. Виберіть файл csv за допомогою кнопки «Configure»

B. Встановіть Column delimiter на «;», String «Delimiter на» \ n «,» Char «*», Comment Char «#»

C. Поставте прапорець поруч із the Has Column Header та зніміть прапорець біля заголовка « the Has Row Header checkbox «

D. Натисніть кнопку «ОК» і виконайте вузол

E. Перегляньте таблицю файлів вузла

F. Відкрийте вікно налаштування

G. Прочитайте перші 5 рядків у вкладці Limit rows

H. Натисніть кнопку «ОК» і виконайте вузол

I. Перегляд таблиці файлів

**3. Завантажте набір даних «Студенти.cvs» за допомогою вузла «File Reader»**

A. Виберіть файл csv за допомогою кнопки «Configure»

B. Перевірте заголовки стовпчиків читання та ігноруйте прапорці біля панелей і вкладок та зніміть прапорці інші

C. Встановіть Column delimiter на «;»

D. Натисніть на стовпчик «Астрономія», оберіть тип Number (double).

E. Натисніть на стовпчик на «Студенти» та змініть назву на «Прізвище»

F. Натисніть кнопку «ОК» і виконайте вузол

G. Перегляньте таблицю файлів

H. Відкрийте вікно Configure

I. Натисніть кнопку Advanced та оберіть панель « the Limit Rows «

J. Прочитайте лише перші 10 рядків з файлу

K. Натисніть кнопку «ОК»

L. Натисніть кнопку «ОК» ще раз

M. Виконайте вузол

N. Відкрийте таблицю

O. Поверніть все назад до пункту J.

**4. Використовуючи вузол File Reader підключіть до нього вузол Interactive Table.**

A. Використайте File Reader та підключіть до нього вузол Interactive Table

B. Виконайте вузол Interactive Table

C. Перегляньте виконання View: Table View

5. Використовуючи вузол XLS Reader підключіть до нього вузол Statistics

A. Використайте вузол XLS Reader і підключіть до нього вузол Statistics

B. Відкрийте вікно налаштування

C. Включіть всі атрибути

D. Перевірте наявність прапорця «the Manual Selection»

E. Перевірте наявність прапорця «the Calculate median»

F. Нехай інші параметри задаються за замовчуванням

G. Натисніть кнопку «ОК»

H. Виконайте вузол Statistics

I. Виберіть параметр View: Statistics View

J. Виберіть вкладку «Nominal»

K. Виберіть вкладку «Top/bottom tab»

**6. Використовуючи вузол File Reader підключіть до нього вузол Box Plot**

A. Використайте вузол XLS Reader і підключіть до нього вузол Box Plot

B. Виконати вузол the Box Plot

C. Виберіть параметр View: Box Plot

D. Перейдіть на вкладку Appearance.

E. Встановіть прапорець Normalize, потім зніміть

F. Виберіть вкладку вибір колонки

G. Включіть лише атрибут віку

H. Закрийте вікно the Box Plot

I. Виберіть параметр «the Robust Statistics»

J. Закрийте вікно «the Robust Statistics»

**7. Використовуючи вузол File Reader підключіть до нього вузол Scatter Plot**

A. Використайте вузол XLS Reader і підключіть до нього вузол Scatter Plot

B. Виконайте вузол Scatter Plot

C. Виберіть параметр View: Scatter Plot

**8. Використовуючи вузол File Reader підключіть до нього вузол Scatter Matrix**

A. Використайте вузол XLS Reader і підключіть до нього вузол Scatter Matrix

B. Виконайте вузол Scatter Matrix

C. Виберіть параметр View: Scatter Matrix

D. Виберіть вкладку вибір колонки

E. Включіть наступні атрибути: Методи досліджень та аналіз даних в освіті, Педагогічна творчість, Теоретичні основи інформатики

F. Закрийте вікно Scatter Matrix

**9. Завантажте набір даних «Студенти пропущені значення.cvs» за допомогою вузла «File Reader»**

A. Створіть вузл «File Reader»

B. Використайте кнопку "browse", щоб вибрати файл "Студенти пропущені значення.cvs"

C. Перейменуйте значення «Col16» на «Вступ до магістратури»

D. Закрийте вікно налаштування

E. З'єднайте вузол Statistics з вузлом File Reader

F. Відкрийте вікно налаштування вузла Statistics

E. Перевірте значення обчислення медіанних значень, перемикач ручного вибору (the Manual selection) і включайте всі атрибути

H. Закрийте вікно «Configure» та запустіть вузол Statistics

I. Відкрийте вікно View: Statistics View

J. Виберіть вкладку «Nominal»

K. Виберіть вкладку "Top/bottom"

L. Закрийте вікно

**10. Використовуючи вузол File Reader підключіть до нього вузол Missing Value**

A. Використайте вузол File Reader і підключіть до нього вузол відсутніх значень (Missing Value)

B. Відкрийте вікно налаштування та скористайтеся вкладкою «Default»

C. Установіть прапорець на атрибути для значень медіани – числа з плаваючою точкою, і найбільш часто використовуваних для рядків стовпців.

D. Закрийте вікно «Налаштування» та запустіть вузол

E. Підключіть вузол Statistics до вузла Missing Value

F. Відкрийте вікно налаштування вузла Statistics

Г. Перевірте значення обчислення медіанних значень, перемикач ручного вибору і включіть всі атрибути

H. Закрийте вікно налаштування та запустіть вузол Statistics

I. Відкрийте вікно View: Statistics View

J. Виберіть вкладку "Top/bottom" та порівняйте її вихід з виходом з верхньої / нижньої вкладки, пов'язаної з вузлом статистики, який ви розробили в 1 п., порівняйте кількість відсутніх значень.

K. Закрийте вікна

**11. Використовуючи вузол File Reader підключіть до нього вузол Meta.**

A. Використовуйте вузол File Reader і підключіть до нього вузол Meta з одним вхідним портом даних і одним вихідним портом даних

B. Відкрийте вузол Meta і підключіть порт вхідних даних до двох вузлів Nominal Value Row

C. Відкрийте вікно налаштування першого вузла Nominal Value Row

D. Встановіть атрибут «Вступ до магістратури», виключіть значення «так» і включіть значення «ні»

E. Закрийте вікно налаштувань і виконайте вузол

F. Відкрийте вікно налаштувань другого вузла Nominal Value Row

G. Встановіть атрибут «Вступ до магістратури», виключіть значення «так» і включіть значення «ні»

H. Закрийте вікно налаштувань та виконайте вузол

I. Підключіть вузли Missing Value до кожного вузла Nominal Value Row

J. Відкрийте вікно налаштування вузлів Missing Value

K. Установіть перемикач на атрибути Most Frequent for Integer, для Most frequent for String Columns

L. Закрийте вікна

M. Підключіть два порожніх значення до загального вузла конкатенації (Concatenate)

N. Підключіть порт вихідних даних вузла Concatenate до вихідного порту Meta вузла

O. Закрийте вікно Meta вузла та виконайте його

П. Підключіть порт вихідних даних мета-вузла до вузла Statistics

Q. Відкрийте вікно налаштування вузла Statistics

R. Перевірте значення обчислення медіанних значень, перемикач ручного вибору та включіть всі атрибути

S. Закрийте вікно налаштувань та виконайте вузол Statistics

T. Відкрийте вікно View: Statistics View

U. Виберіть вкладку "Top/bottom" та порівняйте її вихід з виходом з верхньої / нижньої вкладки, пов'язаної з вузлом статистики, який ви розробили в 1 п., порівняйте кількість відсутніх значень.

**Індивідуальне завдання**

Створіть робоче середовище Lab1, наступного вигляду (рис.1):

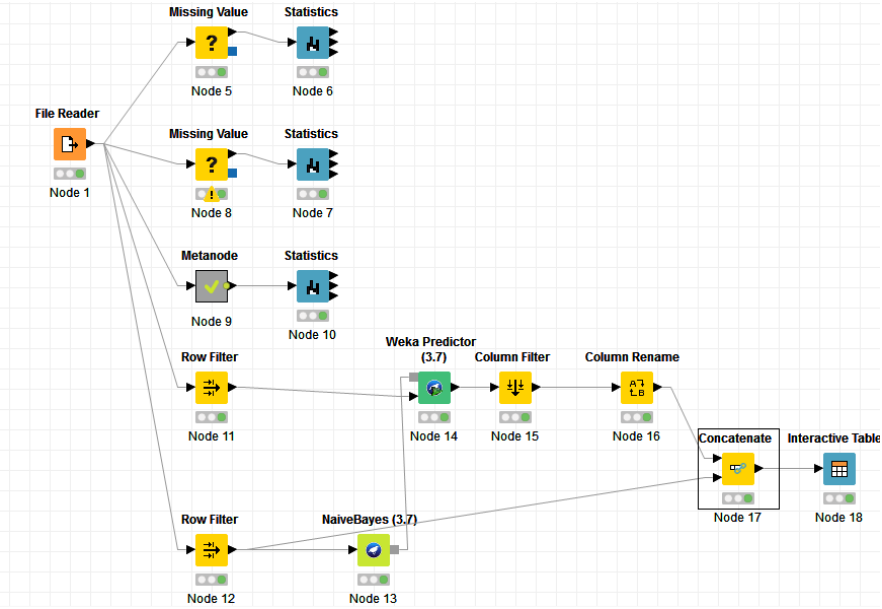


Рис. 1

Підключіть таблицю з даними «Студенти пропущені значення.xls».

## 10. Лабораторна робота 2 KNIME

**Виконання**

Спочатку нам потрібно завантажити дані із таблиці Студенти.csv. Для цього ми виберемо вузол Excel Reader.

Завантажені дані - це таблиця, структура якої описана вище. Рядки цієї таблиці ми розіб'ємо на 2 групи у відношенні 40% до 60%.

До вузла Excel Reader приєднаємо вузол Column Filter, у якому ми повинні виключити стовпці Прізвища, Ім'я та По-батькові.

Нам необхідно розбити таблицю на 2 групи у відношенні 40:60 %. Для цього ми до вузла Excel Reader приєднаємо 1 вузол Partitioning. Він відсортує 40% записів для тренування нашої ШНМ. А друга частина записів буде використана для тестування ШНМ.

Встановимо вузли, що власне є ШНМ. Це вузли MultiLayerPerceptron Predictor, тому що це один із найпопулярніших ШНМ,

Підключаємо 1-й вузол RProp MLP Learner, який буде використаний для тренування і 2-й вузол MultiLayerPerceptron Predictor, який буде використаний для передбачення.

Зв’язуємо ці вузи із вхідними даними та між собою.

Навчальний вузол MultiLayerPerceptron Predictor ми приєднуємо до однієї частини вузла Partioning, а саме тієї яка є тренувальною. А вузол передбачувача приєднуємо до другої частини Partioning, яка відповідає за передбачення.

Приєднаємо до вузла MultiLayerPerceptron Predictor вузол Interactive Table і перевіримо наш передбачувач.

Для того, щоб порівняти отримані результати передбачення, приєднаємо вузол Scorer до вузла MultiLayerPerceptron Predictor.

## 11. Фреймворк PyTorch

PyTorch — це один із найпопулярніших фреймворків для розробки та навчання моделей глибокого навчання. Він надає потужний і гнучкий інструментарій для створення нейронних мереж і роботи з ними, забезпечуючи зручний та інтуїтивно зрозумілий API.

**Розробник**

PyTorch був створений **Facebook AI Research (FAIR)** і випущений у 2016 році як open-source проєкт. З того часу він став одним із провідних інструментів у галузі штучного інтелекту.

**Особливості PyTorch**

**1. Динамічні обчислювальні графи:**

- PyTorch використовує динамічний граф обчислень, який створюється в процесі виконання коду.
- Це дозволяє змінювати структуру моделі “на льоту”, що дуже корисно для досліджень і експериментів.

**2. Підтримка GPU:**

- PyTorch забезпечує легку інтеграцію з GPU, дозволяючи значно пришвидшити обчислення.
- Простий перехід між CPU і GPU через функцію .to(device).

**3. Автоматичне диференціювання:**

Завдяки модулю **Autograd**, PyTorch автоматично обчислює градієнти, що спрощує реалізацію алгоритму зворотного поширення помилки (backpropagation).

**4. Модульна структура:**

- PyTorch має модульний дизайн, що дозволяє легко створювати моделі будь-якої складності.

• Модуль **torch.nn** містить базові блоки для створення нейронних мереж.

**5. Підтримка бібліотек:**

**Torchvision**: для роботи із зображеннями.

**Torchtext**: для обробки тексту.

**Torchaudio**: для роботи зі звуковими даними.

**6. Активна спільнота та документація:**

PyTorch має широку спільноту розробників і дослідників, що сприяє швидкому оновленню і підтримці.

**Переваги PyTorch**

**1. Гнучкість:**

Завдяки динамічним графам, PyTorch ідеально підходить для експериментів і швидкого створення прототипів.

**2. Інтуїтивність:**

Його API схожий на NumPy, що робить PyTorch легким для вивчення та використання.

**3. Підтримка навчання з підкріпленням:**

Широко використовується в задачах навчання з підкріпленням завдяки інтеграції з бібліотекою OpenAI Gym.

**4. Популярність у дослідницькій спільноті:**

PyTorch є основним вибором для багатьох дослідницьких робіт, включаючи моделі NLP (GPT, BERT) і CV (згорткові мережі).

**5. Інтеграція з виробництвом:**

Завдяки **TorchServe**, PyTorch легко розгортати моделі у виробничих середовищах.

**Основні компоненти PyTorch**

**1. Tensor:**

• Основна структура даних у PyTorch.

• Аналог масивів NumPy з підтримкою обчислень на GPU.

**2. Autograd:**

• Інструмент для автоматичного обчислення градієнтів.

• Полегшує реалізацію навчання моделей.

**3. Module:**

• Базовий клас для створення нейронних мереж.

- Надає інструменти для визначення шарів і функцій вперед/назад проходу.

4. **Optimizer:**
   - Клас для реалізації алгоритмів оптимізації (SGD, Adam, RMSprop тощо).
5. **DataLoader:**
   - Інструмент для ефективного завантаження і обробки даних під час навчання.

**Недоліки PyTorch**

1. **Використання ресурсів:**
   Навчання моделей із PyTorch може бути ресурсоємним і вимагати потужного обладнання.
2. **Менша підтримка для виробничих середовищ (раніше):**
   Хоча зараз це вирішується за допомогою **TorchServe**, TensorFlow мав більш розвинені інструменти для розгортання на виробництві.
3. **Конкуренція з TensorFlow:**
   TensorFlow іноді вибирають через більший набір інструментів для інтеграції та розгортання.

**Типові задачі, для яких використовується PyTorch**

1. **Обробка зображень:**
   - Реалізація згорткових нейронних мереж (CNN).
   - Завдання класифікації, сегментації, детекції об’єктів.
2. **Обробка природної мови (NLP):**
   - Розробка трансформерів, таких як BERT або GPT.
3. **Навчання з підкріпленням (Reinforcement Learning):**
   - Наприклад, створення агентів для ігор або робототехніки.
4. **Генеративні моделі:**
   - Генеративно-змагальні мережі (GANs) або автокодери.
5. **Мультизадачність:**
   - Побудова моделей для одночасного виконання кількох задач.

**Корисні посилання**

- Офіційний сайт PyTorch: https://pytorch.org
- Документація: https://pytorch.org/docs/stable/index.html
- GitHub: https://github.com/pytorch/pytorch

## 12. Бібліотеки, що використовуються разом з PyTorch

### 12.1. NumPy

**NumPy** (Numeric Python) — це бібліотека Python для роботи з багатовимірними масивами та виконання математичних і статистичних обчислень. Вона є основою для багатьох інших бібліотек у сфері обробки даних, машинного навчання та наукових обчислень, таких як Pandas, TensorFlow і PyTorch.

**Ключові можливості NumPy**

1. **Масиви (ndarray):**
   - Основна структура даних NumPy — це багатовимірний масив, який називається **ndarray**.
   - Масиви дозволяють працювати з великими наборами даних швидше й ефективніше, ніж стандартні списки Python.
2. **Швидкість:**
   - NumPy написаний на C, що забезпечує високу швидкість виконання обчислень.
3. **Математичні операції:**
   - Бібліотека надає функції для лінійної алгебри, матриць, тригонометрії, статистики, а також операцій з комплексними числами.
4. **Обробка даних:**
   - Зручне зчитування, перетворення й обробка великих обсягів даних.
5. **Інтеграція:**
   - NumPy добре інтегрується з іншими науковими бібліотеками (SciPy, Pandas, Matplotlib).

**Встановлення NumPy**

Встановити NumPy можна через менеджер пакетів pip:

```
pip install numpy
```

**Основні компоненти NumPy**

1. **ndarray (N-dimensional array):**
   - Основна структура для зберігання числових даних.
   - Підтримує багатовимірні масиви будь-якого розміру.
2. **Функції створення масивів:**
   - numpy.array(): створення масиву з наявних даних.

- numpy.zeros(): створення масиву, заповненого нулями.
- numpy.ones(): створення масиву, заповненого одиницями.
- numpy.arange(): створення послідовності чисел.
- numpy.linspace(): створення рівномірно розподілених значень.

**3. Математичні операції:**

- Операції додавання, віднімання, множення та ділення масивів.
- Обчислення статистичних величин: середнє, медіана, стандартне відхилення.

**4. Індексація та зрізи:**

- NumPy підтримує складну індексацію та нарізку масивів для вибірки даних.

**5. Лінійна алгебра:**

- Множення матриць, обчислення визначників, знаходження власних чисел.

**6. Обробка даних:**

- Функції сортування, фільтрації та агрегації даних.

**Приклади використання NumPy**

**1. Створення масиву:**

```
import numpy as np
# Створення одновимірного масиву
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

**2. Масштабування масиву:**

```
arr = np.array([1, 2, 3, 4])
scaled = arr * 2
print(scaled)  # [2, 4, 6, 8]
```

**3. Операції над масивами:**

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
print(a + b)  # [5, 7, 9]
```

**4. Обчислення статистик:**

```
data = np.array([10, 20, 30, 40])
print(np.mean(data))  # Середнє значення: 25.0
```

```
print(np.std(data))    # Стандартне відхилення
```

**5. Лінійна алгебра:**

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 3]])
product = np.dot(A, B)  # Множення матриць
print(product)
```

**Переваги NumPy**

**1. Швидкість:**

- Виконує операції швидше, ніж стандартні списки Python, завдяки використанню низькорівневих мов програмування (C).

**2. Зручність:**

- Інтуїтивний синтаксис та гнучкі інструменти для роботи з масивами.

**3. Широка підтримка:**

- NumPy інтегрується з іншими бібліотеками, що робить його основою екосистеми Python для наукових обчислень.

**4. Масштабованість:**

- Підтримка роботи з великими обсягами даних.

**Недоліки NumPy**

**1. Робота тільки з однорідними даними:**

- Масиви NumPy повинні містити елементи одного типу (наприклад, тільки числа).

**2. Висока вартість пам'яті для малих масивів:**

- Використання NumPy для невеликих наборів даних може бути менш ефективним порівняно зі списками Python.

**3. Обмежені можливості для роботи з нерегулярними структурами даних:**

- Для таких задач зазвичай використовують Pandas.

**Застосування NumPy**

**1. Наукові обчислення:**

Використовується для роботи з числовими даними в фізиці, хімії, біології.

**2. Машинне навчання:**

NumPy є основою для фреймворків, таких як TensorFlow, PyTorch, і Scikit-learn.

**3. Обробка зображень:**

Використовується для роботи з піксельними значеннями зображень.

**4. Аналіз даних:**

Застосовується для агрегації та обробки великих наборів числових даних.

## 12.2. Pandas

**Pandas** — це потужна бібліотека Python для аналізу та обробки даних. Вона забезпечує гнучкі інструменти для роботи зі структурованими даними, такими як таблиці, часові ряди або бази даних. Pandas дозволяє ефективно зчитувати, фільтрувати, змінювати, агрегувати й аналізувати дані.

**Ключові можливості Pandas**

**1. Двома основними структурами даних є:**

- **Series**: одномірна структура даних (аналог списку або стовпця таблиці).
- **DataFrame**: двовимірна структура даних (аналог таблиці).

**2. Обробка пропущених значень:**

Pandas дозволяє легко обробляти відсутні дані, замінюючи їх, видаляючи або заповнюючи значеннями.

**3. Маніпуляції з даними:**

Фільтрація, сортування, агрегація, об'єднання та групування даних.

**4. Підтримка різних джерел даних:**

Pandas може працювати з файлами **CSV**, **Excel**, **SQL**, **JSON**, та іншими форматами.

**5. Робота з часовими рядами:**

Зручні інструменти для роботи з датами й часовими інтервалами.

**6. Інтеграція з іншими бібліотеками:**

Легко поєднується з бібліотеками NumPy, Matplotlib, Scikit-learn для складнішого аналізу та візуалізації.

**Встановлення Pandas**

```
pip install pandas
```

**Основні структури даних Pandas**

**1. Series:**

Одновимірний об'єкт, подібний до списку, але з можливістю індексації.

```
import pandas as pd
# Створення Series
s = pd.Series([10, 20, 30, 40], index=["a", "b", "c",
"d"])
print(s)
```

**2. DataFrame:**

Двовимірна таблиця даних із рядками та стовпцями, кожен із яких може мати власну назву.

```
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "Salary": [50000, 60000, 70000]
}
df = pd.DataFrame(data)
print(df)
```

**Основні функції Pandas**

**1. Читання даних:**

`pd.read_csv()`: читання даних із CSV-файлів.

`pd.read_excel()`: читання даних із Excel.

`pd.read_sql()`: зчитування даних із бази даних SQL.

**2. Маніпуляції з даними:**

• Фільтрація:

```
df[df['Age'] > 25]
```

• Додавання нових стовпців:

```
df['Bonus'] = df['Salary'] * 0.1
```

• Видалення пропущених значень:

```
df.dropna()
```

**3. Групування та агрегація:**

• Групування за категоріями:

```
grouped = df.groupby('Age')['Salary'].sum()
print(grouped)
```

**4. Індексація та зрізи:**

• Індексація за іменем стовпця:

```
df['Name']
```

• Зрізи:

```
df.iloc[0:2, 1:3]
```

***5. Злиття та об'єднання:***

• Злиття таблиць:

```
pd.merge(df1, df2, on="common_column")
```

**6. Статистика:**

• Розрахунок базових статистик:

```
df.describe()
```

**Приклад обробки даних**

```
import pandas as pd
# Завантаження даних із CSV
df = pd.read_csv("data.csv")

# Перегляд перших рядків
print(df.head())
# Вибір стовпців і фільтрація
filtered = df[df['Age'] > 30][['Name', 'Salary']]
print(filtered)
# Групування та обчислення середньої зарплати за категоріями
average_salary = df.groupby('Department')['Salary'].mean()
print(average_salary)
```

**Переваги Pandas**

**1. Простота використання:**

Інтуїтивний синтаксис для маніпуляції таблицями та структурованими даними.

**2. Широкі можливості:**

Підтримка багатьох форматів даних та інструментів для їхнього аналізу.

**3. Висока продуктивність:**

Pandas базується на бібліотеці NumPy, що забезпечує швидкі операції над великими наборами даних.

**4. Підтримка пропущених значень:**

Зручна робота з NaN і пропусками в даних.

**Недоліки Pandas**

**1. Пам'яттєвикористання:**

Pandas менш ефективний із великими наборами даних порівняно з системами розподіленої обробки (наприклад, Dask або Apache Spark).

**2. Складність для великих даних:**

При обробці масивів, які не поміщаються в пам'ять, Pandas може бути недостатньо ефективним.

**3. Крута крива навчання:**

Новачкам може бути складно одразу зрозуміти всі функції та можливості Pandas.

**Застосування Pandas**

**1. Аналіз даних:**

Підготовка, очищення та аналіз великих наборів даних.

**2. Фінансовий аналіз:**

Робота з часовими рядами, обчислення статистичних показників.

**3. Підготовка даних для машинного навчання:**

Pandas використовується для попередньої обробки даних перед подачею в моделі машинного навчання.

**4. Візуалізація даних:**

Інтеграція з Matplotlib і Seaborn для створення графіків.

## 12.3. Бібліотека random

Бібліотека random у Python надає функції для генерації випадкових чисел, вибірки елементів зі списків, перетасування послідовностей і виконання інших операцій, пов'язаних із випадковістю. Це стандартна бібліотека, яка входить до складу Python, і для її використання не потрібно додаткової установки.

**Основні можливості random**

**1. Генерація випадкових чисел:**

• Генерація чисел із різних діапазонів та розподілів (рівномірний, нормальний, гамма тощо).

**2. Робота зі списками:**

• Випадковий вибір елемента.

• Перетасування елементів.

**3. Контрольованість генерації:**

- Можливість встановлення початкового значення генератора випадкових чисел за допомогою seed().

**Основні функції бібліотеки random**

1. Генерація випадкових чисел

Генерує випадкове число із діапазону [0.0, 1.0).

```
random()
```

Генерує випадкове число з рівномірним розподілом у діапазоні [a, b].

```
uniform(a, b)
```

Генерує ціле випадкове число з діапазону range(start, stop, step).

```
randint(a, b)
```

Генерує випадкове число з діапазону range(start, stop, step).

```
randrange(start, stop, step)
```

**2. Робота зі списками**

```
Повертає випадковий елемент із послідовності seq
  (список, кортеж, рядок тощо).
```

```
choice(seq)
```

Повертає список із k випадкових елементів із послідовності з можливістю задати ваги, наприклад:

```
choices(population, weights=None, k=1)
```

Повертає список із k унікальних випадкових елементів (без повторень).

```
items = ['apple', 'banana', 'cherry']
print(random.choices(items,  weights=[1,  2,  3],
  k=2))  # Наприклад, ['cherry', 'banana']
```

```
sample(population, k)
```

Перетасовує елементи послідовності на місці.

```
shuffle(seq)
```

**3. Робота з розподілами**

Генерує випадкове число з нормального розподілу з середнім значенням mu і стандартним відхиленням sigma.

```
normalvariate(mu, sigma)
```

Альтернативна функція для нормального розподілу (швидша для великих обсягів даних).

```
gauss(mu, sigma)
```

Генерує число з експоненціального розподілу з параметром lambd.

```
expovariate(lambd)
```

Генерує число з бета-розподілу.

```
betavariate(alpha, beta)
```

**4. Контроль випадковості**

Ініціалізує генератор випадкових чисел певним початковим значенням (a). Це дозволяє відтворювати однакові результати для певного значення seed.

```
seed(a=None)
```

**Приклад використання random**

```
import random
# Генерація випадкових чисел
print("Випадкове число (0-1):", random.random())
print("Випадкове ціле число:", random.randint(1, 100))
# Робота зі списком
names = ['Alice', 'Bob', 'Charlie']
print("Випадкове ім'я:", random.choice(names))
# Перетасування списку
deck = list(range(1, 11))
random.shuffle(deck)
print("Перетасований список:", deck)
# Використання розподілів
print("Число з нормального розподілу:",
random.normalvariate(0, 1))
```

**Недоліки random**

**1. Псевдовипадковість:**

Генератор базується на алгоритмах, що створюють псевдовипадкові числа, які визначаються початковим значенням (seed).

**2. Не підходить для криптографії:**

Для криптографічної безпеки слід використовувати модуль secrets.

**Застосування бібліотеки random**

**1. Ігри:**

Створення ігрових механік (напр., випадкове розташування ворогів).

**2. Тестування:**

Генерація випадкових даних для перевірки алгоритмів.

**3. Моделювання:**

Створення випадкових сценаріїв для моделювання процесів.

**4. Машинне навчання:**

Випадковий вибір даних для навчання та тестування моделей.

### 12.4. Бібліотека scikit-learn (sklearn)

scikit-learn (або sklearn) — це популярна Python-бібліотека для машинного навчання, що забезпечує простий і ефективний інтерфейс для реалізації найпоширеніших алгоритмів аналізу даних і побудови моделей. Вона широко використовується в наукових дослідженнях, розробці продуктів і навчанні завдяки простоті та зручності.

**Основні можливості бібліотеки:**

**1. Алгоритми машинного навчання:**

- Класифікація: LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, SVM тощо.
- Регресія: LinearRegression, Ridge, Lasso, SVR тощо.
- Кластеризація: KMeans, DBSCAN, AgglomerativeClustering.
- Зниження розмірності: PCA, TSNE.
- Ансамблі: RandomForest, GradientBoosting, Bagging, AdaBoost.

**2. Обробка даних:**

- Масштабування: StandardScaler, MinMaxScaler.
- Обробка категоріальних даних: OneHotEncoder, LabelEncoder.
- Обробка відсутніх значень: SimpleImputer.
- Розбиття даних: train_test_split, KFold.

**3. Моделювання та оцінка якості:**

- Метрики: accuracy_score, mean_squared_error, roc_auc_score, f1_score.
- Перехресна перевірка: cross_val_score, GridSearchCV, RandomizedSearchCV.
- Вибір моделей: model_selection.

**4. Інтеграція з Python-бібліотеками:**

- Може використовуватися разом із NumPy, Pandas, Matplotlib і seaborn.

**Структура бібліотеки:**

**1. sklearn.datasets — Генерація або імпорт тестових наборів даних.**

```
from sklearn.datasets import load_iris
```

```
data = load_iris()
print(data.keys())
```

**2. sklearn.model_selection — Інструменти для розбиття та перехресної перевірки.**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

**3. sklearn.preprocessing — Нормалізація, кодування даних.**

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

**4. sklearn.linear_model — Лінійні моделі (регресії).**

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

**5. sklearn.tree — Дерева рішень.**

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
```

**6. sklearn.ensemble — Ансамблеві моделі.**

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

**7. sklearn.metrics — Метрики оцінки моделей.**

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

**Приклад простої програми класифікації:**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
# Завантаження даних
data = load_iris()
X, y = data.data, data.target
# Розбиття на тренувальні і тестові дані
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
# Масштабування
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Модель
model = RandomForestClassifier()
model.fit(X_train, y_train)
# Оцінка
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

**Переваги scikit-learn:**

1. **Універсальність:**
   - Містить алгоритми для більшості задач машинного навчання.
2. **Простий інтерфейс:**
   - Більшість алгоритмів мають однакові методи: .fit(), .predict().
3. **Гнучкість:**
   - Підходить як для новачків, так і для досвідчених розробників.
4. **Широка підтримка:**
   - Активно підтримується спільнотою, регулярно оновлюється.

**Недоліки:**

- Не оптимізована для роботи з великими наборами даних (використовує однопоточні алгоритми).
- Більшість реалізацій відносно прості (не підходить для глибоких нейронних мереж).

## 13. Тензори в PyTorch

В PyTorch тензори (torch.Tensor) є основною структурою даних, яка використовується для зберігання числових даних і виконання операцій з ними. Тензори є узагальненням скалярів, векторів і матриць, і можуть мати довільну кількість вимірів.

**Основні операції з тензорами**

**1. Створення тензорів**

PyTorch пропонує різноманітні методи для створення тензорів:

```
import torch
# Створення порожнього тензора (без ініціалізації значень)
tensor_empty = torch.empty(3, 3)
# Тензор з нулями
tensor_zeros = torch.zeros(3, 3)
# Тензор з одиницями
tensor_ones = torch.ones(3, 3)
# Тензор зі випадковими значеннями
tensor_random = torch.rand(3, 3)
# Тензор з фіксованими значеннями
tensor_fixed = torch.tensor([[1, 2], [3, 4]])
# Тензор із рівномірно розподіленими значеннями
tensor_linspace = torch.linspace(0, 1, steps=5)  # 5 значень від 0 до 1
```

**2. Властивості тензорів**

Тензор має кілька важливих властивостей:

```
tensor = torch.randn(3, 4)  # Випадковий тензор 3x4
print(tensor.shape)       # Розмірність (torch.Size([3, 4]))
print(tensor.size())     # Те саме, що й .shape
print(tensor.ndimension())  # Кількість вимірів
print(tensor.dtype)       # Тип даних (за замовчуванням float32)
print(tensor.device)           # Пристрій, на якому зберігається тензор (CPU/GPU)
```

**3. Операції з тензорами**

PyTorch підтримує широкий набір математичних операцій:

```
# Арифметичні операції
a = torch.tensor([1, 2, 3])
b = torch.tensor([4, 5, 6])
```

```
print(a + b)  # Сума
print(a * b)  # Помноження елементів
print(a @ b)  # Скалярний добуток (dot product)
# Транспонування
matrix = torch.randn(3, 4)
print(matrix.T)  # Транспонована матриця
# Агрегатні операції
print(matrix.sum())       # Сума всіх елементів
print(matrix.mean())      # Середнє значення
print(matrix.max())       # Максимальне значення
# Індексація і зрізи
print(matrix[0, :])       # Перший рядок
print(matrix[:, 1])       # Другий стовпець
```

**4. Зміна розміру та форми тензора**

```
tensor = torch.randn(2, 3, 4)
# Перетворення розміру (reshape)
reshaped = tensor.view(6, 4)  # Перетворення на форму
(6, 4)
# Розгортання в 1D
flattened = tensor.flatten()
# Додавання осі
unsqueezed = tensor.unsqueeze(0)  # Додати вісь на
позиції 0
squeezed = unsqueezed.squeeze()  # Видалити осі з
розміром 1
```

**5. Підтримка CUDA (GPU)**

PyTorch дозволяє виконувати обчислення на GPU:

```
# Перенесення тензора на GPU
device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')
tensor = torch.randn(3, 3).to(device)
# Повернення на CPU
tensor_cpu = tensor.to('cpu')
```

**6. Градієнти та автодиференціюванн**

**я**

Тензори з підтримкою обчислення градієнтів використовуються для навчання нейронних мереж:

```
x = torch.tensor([2.0, 3.0], requires_grad=True)
y = x ** 2 + 3 * x + 4
y.sum().backward()  # Обчислення градієнтів
print(x.grad)  # Вивід градієнтів
```

**7. Злиття, розбиття та комбінування тензорів**

```
a = torch.randn(2, 3)
b = torch.randn(2, 3)
# Конкатенація
concat = torch.cat((a, b), dim=0)  # Уздовж осі 0
# Розбиття
split = torch.chunk(concat, chunks=2, dim=0)  # На 2
частини
# Стекування
stacked = torch.stack((a, b), dim=0)  # Додавання нової
осі
```

**8. Перетворення між NumPy і PyTorch**

```
import numpy as np
# PyTorch → NumPy
tensor = torch.randn(3, 3)
numpy_array = tensor.numpy()
# NumPy → PyTorch
numpy_array = np.array([[1, 2], [3, 4]])
tensor = torch.from_numpy(numpy_array)
```

## 14. Проста програма з використанням PyTorch

Наведемо приклад найпростішої програми на Python із використанням PyTorch. Ця програма створює тензори, виконує над ними математичні операції та демонструє базові можливості PyTorch.

```
import torch
# Створення двох тензорів
a = torch.tensor([1.0, 2.0, 3.0])  # Тензор із числами
b = torch.tensor([4.0, 5.0, 6.0])  # Інший тензор

# Арифметичні операції
```

```
sum_result = a + b             # Сума
product_result = a * b         # Покомпонентне множення
dot_product = torch.dot(a, b) # Скалярний добуток
# Вивід результатів
print("Tensor a:", a)
print("Tensor b:", b)
print("Sum (a + b):", sum_result)
print("Element-wise product (a * b):", product_result)
print("Dot product (a · b):", dot_product)
# Використання GPU, якщо доступний
if torch.cuda.is_available():
      print("CUDA is available! Moving tensors to
GPU...")
    a = a.to('cuda')
    b = b.to('cuda')
    sum_result = a + b
    print("Sum on GPU (a + b):", sum_result)
else:
    print("CUDA is not available.")
```

**Що робить цей код:**

1. **Створює два тензори** a **та** b з трьома числами кожен.
2. Виконує прості арифметичні операції:
   - Сума тензорів.
   - Покомпонентне множення.
   - Скалярний добуток (dot product).
3. Виводить результати операцій у консоль.
4. Перевіряє доступність GPU (torch.cuda.is_available()) і, якщо можливо, переносить тензори на GPU та виконує обчислення на ньому.

**Результат виконання:**

Якщо запустити цей код на комп'ютері без GPU, консоль видасть приблизно таке:

```
Tensor a: tensor([1., 2., 3.])
Tensor b: tensor([4., 5., 6.])
Sum (a + b): tensor([5., 7., 9.])
Element-wise product (a * b): tensor([ 4., 10., 18.])
Dot product (a · b): tensor(32.)
CUDA is not available.
```

### 15. Приклад найпростішої програми для класифікації в PyTorch

Ця програма демонструє, як створити модель, підготувати дані, навчити її на синтетичних даних і зробити передбачення.

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# 1. Генерація синтетичних даних
X, y = make_classification(n_samples=1000,
n_features=2, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
# Нормалізація даних
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Перетворення в тензори
X_train_tensor = torch.tensor(X_train,
dtype=torch.float32)
y_train_tensor = torch.tensor(y_train,
dtype=torch.long)
X_test_tensor = torch.tensor(X_test,
dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

# 2. Визначення простої моделі
class SimpleClassifier(nn.Module):
    def __init__(self):
        super(SimpleClassifier, self).__init__()
         self.fc = nn.Linear(2, 2)  # Вхідні ознаки: 2,
вихідні класи: 2
    def forward(self, x):
        return self.fc(x)
model = SimpleClassifier()
# 3. Визначення функції втрат та оптимізатора
criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.SGD(model.parameters(), lr=0.01)
# 4. Навчання моделі
epochs = 100
for epoch in range(epochs):
    # Прямий прохід
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    # Зворотний прохід і оновлення ваг
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 10 == 0:
          print(f"Epoch [{epoch + 1}/{epochs}], Loss:
{loss.item():.4f}")
# 5. Тестування моделі
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    _, predicted = torch.max(test_outputs, 1)
             accuracy      =      (predicted      ==
y_test_tensor).float().mean()
    print(f"Test Accuracy: {accuracy:.4f}")
```

**Опис:**

**1. Генерація даних**

- Використовується sklearn.datasets.make_classification для створення синтетичних двокласових даних із двома ознаками.
- Дані розбиваються на навчальну і тестову вибірки за допомогою train_test_split.

**2. Модель**

- Модель складається з одного повнозв'язного шару nn.Linear(2, 2).
- Вхід (2 ознаки).
- Вихід (2 класи).

**3. Функція втрат**

- Використовується nn.CrossEntropyLoss, яка підходить для задач класифікації з кількома класами.

**4. Оптимізатор**

- Використовується стохастичний градієнтний спуск (SGD) із навчальною швидкістю 0.01.

**5. Навчання**

- Дані передаються через модель.
- Обчислюється втрата.
- Виконується зворотне поширення помилки (loss.backward) і оновлення ваг (optimizer.step).

**6. Тестування**

- Модель оцінюється на тестовому наборі. Передбачення порівнюються з фактичними мітками, і обчислюється точність.

**Результат виконання:**

При виконанні коду на синтетичних даних ви побачите щось на зразок:

```
Epoch [10/100], Loss: 0.6501
Epoch [20/100], Loss: 0.6005

...

Epoch [100/100], Loss: 0.4012

Test Accuracy: 0.8500
```

## 16. Лабораторна робота 3. Задача класифікації ірісів

Імпортуємо необхідні бібліотеки: torch та random і numpy

```
import torch
import random
import numpy as np
import pandas as pd
from torch.utils.data import random_split, DataLoader,
TensorDataset
```

Перш ніж приступати до реалізації, давайте поговоримо про випадкові числа, які виникають в результаті роботи нейронної мережі, і як вони відтворюються.

Якщо ми візьмемо один і той же код і запустимо його кілька разів, то ми отримаємо різні результати. Це відбувається тому, що кожен раз у нас ініціалізуються заново ваги, відбувається по-різному навчання нейронної мережі, і це все залежить від того, як відпрацював генератор випадкових чисел. Можливо, ми хочемо щоб експерименти були відтворені: щоб, взявши один і той же python файл і виконавши його, ми отримали б той же самий результат, як і раніше. Наприклад, це потрібно для того щоб розуміти: а чи

дійсно ті зміни, які ми робимо з нейромережею, покращують наші результати, або це результат певної випадковості.

Є таке поняття як random seed, це можна інтерпретувати, як номер послідовності випадкових чисел, яку видає нам випадковий генератор, якщо його попросити видати нам послідовність. Випадкових генераторів є кілька: є випадковий генератор модуля random в Python, є випадковий генератор модуля numpy.random (вже бібліотеки numpy), є також випадкові генератори бібліотеки PyTorch, і інші. Давайте зафіксуємо їх всі. Для цього візьмемо random seed і поставимо його в конкретне значення. random.seed(0)

```
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True
```

Тобто, ми завжди будемо використовувати нульову послідовність при виклику випадкового генератора бібліотеки random (це бібліотека мови python). Також нам потрібно зафіксувати seed в numpy і в PyTorch, випадкові seed, які відповідають за обрахування і CPU і на GPU – вони різні, відповідно потрібно ще зафіксувати випадковий seed підмодуля CUDA. І якщо ви виконуєте обчислення за допомогою відеокарти, ви використовуєте бібліотеку cudnn, і вона може виконуватися в детерміністичному режимі, або в недетерміністичному. Недетерміністичний режим набагато швидший, але якщо ми хочемо встановити детерміністичний режим, щоб була відтворюваність, то нам потрібно виставити цей параметр в "True". Виставивши всі ці параметри, ми можемо практично гарантувати, що експерименти будуть відтворені. Якщо це не так, то швидше за все потрібно шукати помилку в коді.

Нам потрібно завантажити датасет із файлу iris2.csv і розбити його на дві частини: частина для тренування (train part), на якій ми будемо навчатися, і на тестову (test part, можливо додати ще validation part, але кількість даних в нас замала), на якій ми будемо рахувати метрики. **Це потрібно зробити самостійно!**

Після цього ми всі "фолди" (отримані частини датасетів): X_train, X_test, Y_train і Y_test переведемо до torch тензорів (якщо число з плаваючою комою – обернемо у float тензор, якщо ні (наша ціль) – обернемо у long тензор).

```
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
```

```
y_test = torch.LongTensor(y_test)
```

Реалізуємо клас IrisNet, це буде наша нейромережа для класифікації. Використовуємо успадкований клас від torch.nn.Module, у цей раз організуємо два прихованих шари, тобто наша нейромережа буде складатися з трьох шарів і два з них будуть прихованими.

1) Перший шар – це fully connected (повнозв'язний) шар, з 4 входів (у нас 4 колонки для кожної квітки), на виході N прихованих нейронів;
2) Далі активація: використаємо функцію Sigmoid, можна поставити будь-яку іншу, якщо хочете.
3) Після цього – прихований шар, який складається з N нейронів, перетворює їх теж в N нейронів;
4) Знову активація Sigmoid.
5) Після цього знову повнозв'язний шар, який видає нам три нейрона, кожен нейрон буде відповідати за свій клас. Тобто на виході цих трьох нейронів будуть деякі числа, які після цього ми передаємо в у функцію Softmax, і отримаємо ймовірності класів.
6) Далі ініціалізується Софтмакс.

Функція "forward" буде реалізовувати граф нашої нейронної мережі. Ми передаємо вхідний тензор в перший fullyconnected шар, після цього в першу активацію, в другій fully connected шар, в другу активацію, в третій fully connected шар, у якого три виходи. Для того, щоб розрахувати ймовірності, напишемо також окрему функцію inference, яка буде викликати функцію "forward", і пропускати її через softmax. Ініціалізуємо нашу нейромережу з кількістю прихованих нейронів = 10. Нехай вона буде мати назву iris_net.

```
class IrisNet(torch.nn.Module):
      def __init__(self, n_hidden_neurons):
        super(IrisNet, self).__init__()
        self.fc1 = torch.nn.Linear(4, n_hidden_neurons)
        self.activ1 = torch.nn.Sigmoid()
           self.fc2 = torch.nn.Linear(n_hidden_neurons,
n_hidden_neurons)
        self.activ2 = torch.nn.Sigmoid()
        self.fc3 = torch.nn.Linear(n_hidden_neurons, 3)
        self.sm = torch.nn.Softmax(dim=1)
    def forward(self, x):
        x = self.fc1(x)
        x = self.activ1(x)
```

```
        x = self.fc2(x)
        x = self.activ2(x)
        x = self.fc3(x)
        return x
    def inference(self, x):
        x = self.forward(x)
        x = self.sm(x)
        return x
iris_net = IrisNet(10)
```

Нам залишилося тільки задати функцію втрат (функція, яка характеризує втрати при неправильному прийнятті рішень на основі спостережених даних. Тобто це метод оцінки того, наскільки добре алгоритм моделює вказаний набір даних, наскільки гарно алгоритм працює з заданим набором) – бінарну крос-ентропію (У контексті машинного навчання крос-ентропія — це міра похибки для задачі багатокласової класифікації) – torch.nn.CrossEntropyLoss, яка використовує не виходи після softmax, а виходи нейронної мережі, що не пропущені ще через softmax.

Також нам потрібен optimizer – той метод, який буде використовуватися для обчислення градієнтних кроків. У optimizer ми передаємо всі параметри нейронної мережі. Параметри нейронної мережі – це ваги. Це ті приховані значення, які перебувають в наших нейронах, які ми хочемо підбирати:

```
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(iris_net.parameters(),
lr=1.0e-3)
```

**Функція Adam у PyTorch** — це реалізація оптимізатора Adam (Adaptive Moment Estimation), який використовується для оновлення параметрів нейронної мережі під час навчання. Вона доступна в модулі torch.optim і є однією з найпопулярніших через свою ефективність та адаптивність.

**Переваги Adam:**

- Адаптивність кроку навчання для кожного параметра.
- Швидка збіжність навіть для великих моделей.
- Підходить для різних задач: класифікація, регресія, генеративні моделі тощо.

В реальному житті навряд чи поміститься в пам'яті весь датасет. Тобто, навчання в реальному житті відбувається по частинах даних – вони

називаються Батч (batch, або пакет). Ми повинні відрізати деякий шматочок наших даних, порахувати по ньому втрати, порахувати по ньому градієнтний крок, зробити градієнтний крок, взяти наступний шматочок, і так далі. Відповідно, одна епоха, тобто ітерація перегляду всього датасету, б'ється на багато маленьких частин. Нам знадобиться функція numpy.random.permutation – вона дає випадково перемішані значення. Якщо ми сюди підставимо розмір нашого тренувального датасета, то отримаємо деякі індекси, у випадковому порядку. Якщо ми від нашого датасета візьмемо ці індекси, то отримаємо "перемішаний" (shuffle – перемішувати) датасет. Кожну епоху ми будемо перемішувати датасет, і потім різати його на частини. Скажімо, що ці частини будуть розміром десять елементів (batch_size). Можна взяти будь-яке інше значення. Отже, кожну епоху ми будемо перемішувати наш датасет, у нас є змінна "order", яка визначається кожну епоху, яка визначає порядок індексів, який потрібно застосувати до датасета. З нього ми будемо вирізати ділянки довжиною batch_size. Тобто, кожну епоху ми будемо робити перемішування нашого датасета, визначати змінну order, яка відповідає за порядок елементів.

Після цього порахуємо втрати (loss) на виходах нейронної мережі і реальних значеннях, і порахуємо зворотній хід (backward), тобто у результаті виконання loss функції, ми порахуємо похідну. А результат цієї похідної, тобто градієнти, які вийшли, підуть у оптимізатор (optimizer), тому що він перераховує всі ваги нейронної мережі.

Далі optimizer може зробити step, тобто крок градієнтного спуску.

```
batch_size = 10
for epoch in range(500):
    order = np.random.permutation(len(X_train))
        for start_index in range(0, len(X_train),
batch_size):
        optimizer.zero_grad()
                                        batch_indexes =
order[start_index:start_index+batch_size]
        x_batch = X_train[batch_indexes]
        y_batch = y_train[batch_indexes]
        preds = iris_net.forward(x_batch)
        loss_value = loss(preds, y_batch)
        loss_value.backward()
        optimizer.step()
    if epoch % 100 == 0:
        test_preds = iris_net.forward(X_test)
        test_preds = test_preds.argmax(dim=1)
        print((test_preds == y_test).float().mean())
```

Крім того, кожні 100 епох ми будемо обчислювати метрики на тестовому датасеті, щоб подивитися, навчається у нас нейромережа, чи ні. А саме, кожні 100 епох ми викликаємо функцію forward за тестовими даними, отримуємо тестові передбачення (prediction), і обчислюємо, який вихід був максимальний. Знову ж щоб зрозуміти, який клас передбачає нейромережа, не обов'язково обчислювати softmax, не обов'язково обчислювати ймовірності. Нам досить подивитися: а який вихід був найбільший, і він же і буде згодом виходом з максимальною ймовірністю. Отже нам потрібно порахувати функцію argmax виходів нейронної мережі, це буде номер нейрона, і порівняти його з тим номером класу, який знаходиться в Y_test. Після цього ми хочемо порахувати: а яка частка цього збігу, коли у нас нейрон з максимальним виходом збігся з реально правильним класом. Нам потрібно порахувати середнє значення, але середнє значення ми не можемо порахувати у цілочисельному тензорі, який виходить в результаті цього порівняння, тому ми спочатку його перетворимо до тензора з плаваючою комою і викличемо у нього метод mean().

Нарешті, порівняємо передбачення з фактичними даними:

```
for i in range(test_split):
    print(y_test[i],test_preds[i])
```

**Додаткове завдання**

Переробити описану вище програму для задачі про передбачення вступу студента до магістратури.

## 17. Лабораторна робота 4. Задача про класифікацію рукописних чисел повнозв’язною мережею

У цій задачі необхідно навчитися класифікувати зображення. Ми зробимо це за допомогою повнозв’язної нейронної мережі. Почнемо з того, що як і минулого разу, зробимо ініціалізацію random seed та завантажимо необхідні бібліотеки.

```
import torch
import random
import numpy as np
random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True
```

Крім того, нам знадобиться датасет. Найпростіший (класичний та найпоширеніший з навчальної точки зору) датасет для класифікації зображень – це набір даних MNIST, він містить в собі рукописні цифри від 0 до 9, і його можна завантажити за допомогою бібліотеки torchvision.datasets (він вже окремо має train і test), і отримаємо відповідно mnist_train, mnist_test:

```
import torchvision.datasets
MNIST_train      =      torchvision.datasets.MNIST('./',
download=True,
train=True)
MNIST_test       =      torchvision.datasets.MNIST('./',
download=True,
train=False)
X_train = MNIST_train.train_data
y_train = MNIST_train.train_labels
X_test = MNIST_test.test_data
y_test = MNIST_test.test_labels
```

Синтаксис завантаження даних може трохи відрізнятися у різних версіях PyTorch: зауважте, наприклад, що у новій версії train_data та test_data – це просто data; train_labels та test_labels – це targets.

Подивимося, що ми маємо за дані і їх мітки – спочатку подивимося які типи даних ми маємо:

```
print(X_train.dtype, y_train.dtype)
```

Бачимо що X_data, тобто самі картинки, мають тип dtype "unsigned int8", а ось мітки мають тип "int64".

Для більшої точності під час розрахунків, краще, щоб дані були в числах з плаваючою комою. Відповідно, ми відразу перетворимо X_train і X_test у float.

```
X_train = X_train.float()
X_test = X_test.float()
```

Подивимося на розмірності датасетів, які ми завантажили:

```
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

Бачимо, що X_train і X_test мають розмірності 60 тисяч зображень і 10 тисяч відповідно, і самі зображення розміром 28 на 28 – тобто, це дуже маленькі картинки, тому ми і можемо застосовувати повнозв’язну нейронну мережу для такого завдання. А мітки y_train і y_test мають відповідний розмір

і у них немає додаткової розмірності. Це одновимірний тензор, і тому ми позначили X_train з великої літери а y_train – з маленької, припускаючи, що X_train – це багатовимірний тензор, y_train – це одновимірний.

Коли ми працювали з датасетом про іріси, у нас кожна квітка описувалася 4-ма ознаками, і це був деякий одновимірний тензор, який відповідав квітці, і, відповідно, Батч був двовимірним тензором. А тут кожна картинка описується двовимірним тензором. Щоб це вирішити, можна розтягнути цю картинку в один довгий вектор, і тоді кожен піксель знайде своє місце в цьому довгому векторі. Звичайно, загубиться деяка інформація про те, які пікселі були поруч, які перебували далеко, але в принципі нам цього вистачить. Розтягнути наші картинки допоможе функція reshape:

```
X_train = X_train.reshape([-1, 28 * 28])
X_test = X_test.reshape([-1, 28 * 28])
```

Отже, у нас був тензор X_train, це тривимірний тензор, а ми хочемо двомірний тензор, де перша розмірність буде незмінною – 60 000 зображень (-1), а друга розмірність розтягнеться – було 28x28, а вийде одна розмірність: 784 пікселя. Якщо ми застосуємо цю функцію і до train, і до test, у нас вийдуть вже двомірні тензори X_train і X_test, і їх уже можна передавати в нашу нейронну мережу.

**Модель**

Давайте створимо таку нейронну мережу: вона буде дуже схожою на ту, яку ми створювали для ірисів, вона також буде складатися з декількох повнозв'язних шарів. Тут буде їх два: fc1 – це (fully connected) шар, на вхід якого приходить 28 на 28 = 784 пікселя. Далі вони передаються в N hidden neurons, який ми можемо як завгодно ставити в залежності від того, скільки нам потрібно інформації в прихованому шарі. Після цього у нас буде сигмоїдна активація, щоб додати нелінійність, і після активації, результат буде передаватися в ще один повнозв'язний шар. На вході у нього N hidden neurons, а на виході 10 (тому, що у нас 10 класів): це цифри від 0 до 9. Тобто відбувається класифікація на 10 класів. Крім того потрібно написати функцію forward, яка пропускає тензор X через всі ці шари послідовно: fully connected, активація, fully connected другий, і видає результуючий тензор, який являє собою виходи з другого повнозв'язного шару розміром 10.

Давайте створимо таку нейронну мережу, назвемо її MNISTnet, скажімо що у неї всередині 100 прихованих нейронів:

```
class MNISTNet(torch.nn.Module):
    def __init__(self, n_hidden_neurons):
        super(MNISTNet, self).__init__()
```

```
          self.fc1    =    torch.nn.Linear(28    *    28,
n_hidden_neurons)
          self.ac1 = torch.nn.Sigmoid()
          self.fc2  =  torch.nn.Linear(n_hidden_neurons,
10)
     def forward(self, x):
          x = self.fc1(x)
          x = self.ac1(x)
          x = self.fc2(x)
          return x
mnist_net = MNISTNet(100)
```

Крім того, нам знадобиться функція втрат, як і минулого разу, буде крос-ентропія, тому що це функція втрат, яка використовується в класифікації. Потрібно підкреслити, що функція CrossEntropyLoss на вхід приймає не ймовірність, а ті виходи, які були ще до застосування функції Softmax, тобто функція forward, яку ми написали, не містила софтмакс, тому що ми хочемо трошки прискорити наші обчислення, уникнувши застосування функції Softmax. Для прискорення обчислень можна софтмакс і крос-ентропію з'єднати однією функцією, і тоді будуть трошки швидше і стабільніше виконуватися обчислення. Ще нам потрібен оптимізатор, тобто деякий метод градієнтного спуску, нехай це буде – Adam, learning rate = 1e-3. На вхід оптимізатору передаються, як і в минулий раз, всі параметри нейронної мережі. Тобто, оптимізуються параметри нейронної мережі – це її ваги, тобто ті значення, які зберігаються в нейронах. Тому ми передаємо саме їх всередину оптимізатора.

```
loss = torch.nn.CrossEntropyLoss()
optimizer   =   torch.optim.Adam(mnist_net.parameters(),
lr=1.0e-3)
```

**Навчання моделі**

Як і минулого разу, ми будемо навчатися батчевим або стохастичним градієнтним спуском. Тобто, ми будемо ділити наш набір даних на маленькі частини (так звані Батчі), передавати ці Батчі в нейронну мережу за допомогою функції forward. Після цього ми будемо викликати loss-функцію, яка скаже нам розмір помилки. Після цього ми зможемо порахувати градієнт за допомогою функції backward, і далі ми зможемо зробити градієнтний крок, викликавши optimizer.step. Нехай у нас буде batch_size розміром 100 (це число, яке вибрано навмання), поставимо 10 000 епох. Щоразу ми будемо перемішувати набір даних, на кожній епосі і виділяти звідти послідовні ділянки, таким чином, щоб усередині однієї епохи кожен елемент, кожна картинка була показана всього один раз.

Крім того, нам би хотілося побачити, як росте наша якість після кожної епохи навчання, а також зростання цієї якості на тестовому наборі даних – на тих даних, які нейромережа не бачила. Тому будемо викликати mnistnet.forward ще й на тестовому датасеті, на X_test. Ми будемо робити forward по всьому тестовому набору даних, припускаючи, що він не дуже великий, а батчевий градієнтний спуск нам потрібен всього з двох причин.

По-перше, тому що таким чином сам процес оптимізації відбувається швидше – тобто, краще зробити 10 градієнтних кроків на епоху, ніж зробити один. А по-друге, тому що у нас можуть не поміститися Батчі в GPU (тобто на відео карту), якщо ми будемо робити їх досить великими. Тому ми обмежили batch_size значення 100 на тренуванні, а в тесті ми припустили, що все буде нормально і тому ми можемо зробити forward по всьому набору даних. Крім того, нам потрібно порахувати частку правильних відповідей – accuracy і, відповідно, нам потрібно зрозуміти, а який же клас передбачила нейронна мережа. Нейронна мережа передбачає клас, для якого вона видає максимальне значення. Тобто, після цього це значення передається у функцію softmax, якщо потрібно дізнатися його ймовірність, але вже перед цим ми можемо зрозуміти – а який найбільш ймовірний клас передбачила нейронна мережа. Це буде той нейрон, у якого максимальний вихід. Відповідно, нам потрібно зробити argmax (argmax віддає номер нейрона, одного з десяти, у якого максимальний вихід), і порівняти це з y_test. Це значення accuracy ми будемо кожен раз виводити на екран.

```
batch_size = 100
# test_accuracy_history = []
# test_loss_history = []
# X_test = X_test.to(device)
# y_test = y_test.to(device)
for epoch in range(10000):
    order = np.random.permutation(len(X_train))
    for start_index in range(0, len(X_train),
batch_size):
        optimizer.zero_grad()
        batch_indexes =
order[start_index:start_index+batch_size]
        X_batch = X_train[batch_indexes] #.to(device)
        y_batch = y_train[batch_indexes] #.to(device)
        preds = mnist_net.forward(X_batch)
        loss_value = loss(preds, y_batch)
        loss_value.backward()
        optimizer.step()
    test_preds = mnist_net.forward(X_test)
```

```
    #          test_loss_history.append(loss(test_preds,
y_test))
     accuracy     =     (test_preds.argmax(dim=1)     ==
y_test).float().mean()
    # test_accuracy_history.append(accuracy)
    print(accuracy)
```

Крім того, можна проаналізувати як проходить навчання, записавши accuracy, і loss. Будемо зберігати наші втрати на тесті, в списку "test_loss_history", а наші accuracy, відповідно, в списку "test_accuracy_history".

## 18. Лабораторна робота 5. Розпізнавання рукописних чисел згортковою нейронною мережею

У минулій роботі ми класифікували набір даних MNIST (рукописні цифри) за допомогою повнозв'язної нейронної мережі. Зараз спробуємо поліпшити наш результат, використавши згорткові нейронні мережі.

Згорткові нейронні мережі (ЗНМ) – це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовується до аналізу візуальних зображень. ЗНМ використовують різновид багатошарових перцептронів, розроблений так, щоби вимагати використання мінімального обсягу попередньої обробки. Вони відомі також як інваріантні відносно зсуву або просторово інваріантні штучні нейронні мережі, виходячи з їхньої архітектури спільних ваг та характеристик інваріантності відносно паралельного перенесення.

Розглянемо архітектуру LeNet, вона найперша в світі згорткових нейронних мереж (рис. 2). Відповідно, вона не найкраща, але вона досить логічна.

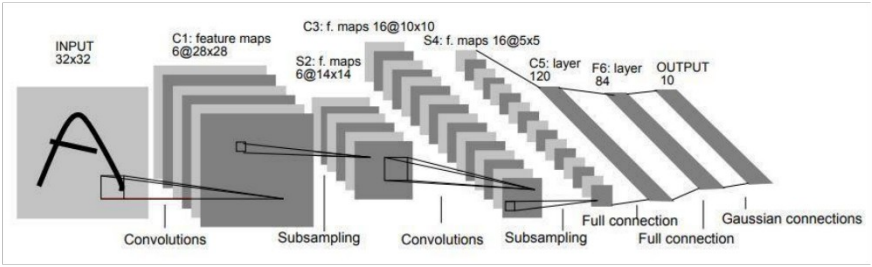


Рис. 2

| Layer | Feature map | Size | Kernel size | Stride | Activation |
|---|---|---|---|---|---|

| Input | Image | 1 | 32x32 | - | - | - |
|---|---|---|---|---|---|---|
| 1 | Convolution | 6 | 28x28 | 5x5 | 1 | tanh |
| 2 | Average Pooling | 6 | 14x14 | 2x2 | 2 | tanh |
| 3 | Convolution | 16 | 10x10 | 5x5 | 1 | tanh |
| 4 | Average Pooling | 16 | 5x5 | 2x2 | 2 | tanh |
| 5 | Convolution | 120 | 1x1 | 5x5 | 1 | tanh |
| 6 | FC | - | 84 | - | - | tanh |
| Output | FC | - | 10 | - | - | softmax |

Отже, спершу подається зображення розміром 32 на 32. У MNIST зображення 28 на 28, необхідно буде зробити деяку попередню обробку зображень. Це зображення проходить через згортку 5 на 5, з шістьма вихідними каналами, значить – у нас згортка 5 на 5 проходить по всьому зображенню. У неї нульові паддінги (padding, або відступи), тобто вона не виходить за межі зображення, і з цієї причини вона обрізає два пікселя з кожного боку зображення, і вихідне зображення з розміру 32 на 32 перетворюється до зображення 28 на 28. Але тепер це вже не зображення а тензор глибиною 6, тому що ми використали згортку з шістьма вихідними каналами. Після цього тензор розміром 28 на 28 на 6 передається в average pooling (тобто усереднене об’єднання; зараз вже майже всюди використовують max pooling), який обчислює середнє значення по квадрату 2 на 2 зі страйдом (stride тобто крок) 2 (тобто він бере непересічні ділянки 2 на 2), і обчислює середнє значення пікселів або чисел, які виявляються в цьому тензорі. І на виході маємо один піксель, відповідно, все зображення стискується у два рази: було 28 на 28 на 6, стало 14 на 14 на 6.

Далі повторюється точно така ж згортка, як в перший раз, тільки тепер у неї кількість вхідних каналів – 6, а вихідних каналів – 16. При цьому, з зображення 14 на 14, виходить зображення 10 на 10, тому що у нас по два пікселя «з'їлося» з кожного боку, тому що ми застосовували згортки без padding – без виходу за межі зображення. Далі знову пулінг (точно такий же, 2 на 2 average pooling), і далі у нас є два варіанти, що робити (в принципі вони аналогічні):

1) У оригінальному LeNet – беруть згортку 5 на 5, (тому що у нас виходить тензор 5 на 5 на 16) з кількістю вхідних каналів 16, а вихідних – 120. І ця згортка, вона перетворює глибокий тензор в один вектор розміру 1 на 1 на 120.
2) А ми зробимо трошки по-іншому: спочатку цей тензор 5 на 5 на 16 ми розвернемо в один вектор. Таким чином, у нас є готовий вектор, до якого

можна застосувати повнозв’язний шар, який на виході буде мати 120 нейронів. Таким чином, в повнозв’язному шарі буде 120 нейронів, при цьому кількість ваг буде однаковою.

Після того, як ми отримали вектор довжиною 120 (неважливо, яким способом), ми застосовуємо до нього два повнозв’язних шари. Перший шар з 120 нейронів отримує 84 нейрона, а другий нам віддає відповідь, тобто він віддає нам 10 нейронів, які потім перетворюються в одну з цифр.

Після кожної згортки та пулінгу будемо застосовувати активацію – візьмемо як і в LeNet – гіперболічні тангенси (зараз наприклад – ReLU було б актуальніше; точно так же сьогодні ми б не застосовували згортки п'ять на п'ять – напевно, ми б застосовували скрізь згортки 3 на 3). Після останнього повнозв’язного шару який утискає розмір на виході до 10 нейронів застосовується софтмакс, який ми застосовувати не будемо, тому що нам потрібен тільки максимальний нейрон по активації. А якщо нам потрібні були б ймовірності, то ми б використовували софтмакс і отримали б деякі ймовірності впевненості мережі.

**Імпортування бібліотек та датасета**

Як і в минулій практичній роботі, ми імпортуємо ті ж самі бібліотеки, фіксуємо random seed, імпортуємо бібліотеку для завантаження набору даних. Імпортуємо точно так же MNIST, і отримуємо X_train, Y_train, X_test, Y_test.

Перша відмінність полягає в тому, що, на відміну від повнозв'язної мережі, яка бачила картинку як один довгий вектор, ми хочемо в згорткову мережу передавати картинку як тривимірний тензор, де перший канал – це глибина картинки, в чорно-білій картинці це 1 канал з яскравістю сірого пікселя. А в RGB зображенні будуть RGB канали. Відповідно, ми повинні нашу картинку, яка прийшла на вхід (вона розміром 28 на 28), розтиснути до 1 на 28 на 28. Це ми робимо за допомогою функції unsqueeze – X_train.unsqueeze(тут ставимо індекс: в якому ж вимірі ми хочемо розтиснути його). X_train у нас – тензор з 60 000 картинок 28 на 28, а ми хочемо щоб було 60 000 на 1 на 28 на 28, і те ж саме ми робимо з тестом:

```
import torch
import random
import numpy as np
random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True
import torchvision.datasets
```

```
MNIST_train      =      torchvision.datasets.MNIST('./',
download=True,
train=True)
train=False)
MNIST_test       =      torchvision.datasets.MNIST('./',
download=True,
X_train = MNIST_train.train_data
y_train = MNIST_train.train_labels
X_test = MNIST_test.test_data
y_test = MNIST_test.test_labels
print(len(y_train), len(y_test))
import matplotlib.pyplot as plt
plt.imshow(X_train[0, :, :])
plt.show()
print(y_train[0])
X_train = X_train.unsqueeze(1).float()
X_test = X_test.unsqueeze(1).float()
print(X_train.shape)
```

**Модель**

Далі запрограмуємо нейронну мережу. Назвемо її LeNet5, тому що так називалася мережа у ЛеКуна. LeNet5 – вона називається тому, що там 5 шарів для навчання – три згортки і два повнозв'язних шари, у нас буде дві згортки і три повнозв'язних шари.

Точно так же, як у ЛеКуна, у нас буде перший шар, який приймає один канал на вхід, тому що картинка у нас одномірна (там один канал) – не RGB, а grayscale, і на виході буде 6 каналів. Але на відміну від ЛеКуна, до якого приходила картинка 32 на 32, у нас картинка 28 на 28. І якщо ми зробимо точно так же, як в оригінальній мережі, тобто застосуємо згортку 5 на 5 без паддінгу, у нас вийде картинка 28 - 4 = 24 пікселя на 24. Ми б хотіли, щоб після першої згортки картинка була б 28 на 28. Якщо ви не хочете втрачати розмірність картинки, то вам потрібно встановити певний паддінг, щоб згортка виходила за межі картинки і тензор, або зображення, на виході були такого ж розміру. При використанні фільтрів 5 на 5 згортки повинні *виходити* на 2 пікселя за розмір зображення, тоді підсумковий тензор матиме такий же розмір. Якби у нас були згортки 3 на 3, то нам потрібно було б виходити на один піксель, і тоді б кількість цих згорток, які ми прикладаємо до зображення, була б розміру зображення.

Поставимо паддінг 2, щоб зберегти розмір. Отже, нам потрібно застосувати наш перший згортковий шар, він називається "Conv2d", тому що він двомірний. Якби у нас були якісь тривимірні зображення, наприклад, мозку людини, і ми б мали третю координату, то у нас було б "Conv3d", який в

PyTorch теж є. Але у нас картинка плоска, відповідно ми будемо ходити в двомірному просторі:

```
self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, padding=2)
```

У цьому Conv2d є багато параметрів, ми повинні використовувати кількість вхідних каналів – обов'язково проставити, що їх буде 1. Вихідних каналів буде 6. Розмір ядра згортки – 5, якщо у вас раптом згортка не симетрична (бувають розтягнуті згортки), то тут можна вказати кортеж (tuple) з двох чисел, але у нас згортка буде квадратна, 5 на 5, відповідно, ми просто зазначимо 5. І далі ми повинні вказати паддінг 2, тому що у нас згортка повинна виходити за межі зображення.

Далі ми повинні застосувати активізацію. Активації в LeNet – гіперболічні тангенси, відповідно ми застосовуємо "torch.nn.Tanh". Після цього – перший пулінг, називається "pool1", це "average pooling". В середині мережі зазвичай використовується max pooling, але в силу традицій тут теж ми використовуємо average pooling 2d:

```
self.pool1 = torch.nn.AvgPool2d(kernel_size=2, stride=2)
```

У нього "kernel_size" – 2, тому що це пулінг 2 на 2, і "stride" – 2, тому що він застосовується без перетинів. Цей пулінг нам стисне зображення від 28 на 28 до 14 на 14.

Далі робимо приблизно все те ж саме: Conv2d – знову згортка, вхідних каналів 6, тому що було тут вихідних каналів 6, і вихідних каналів у цій згортці 16. Так було в оригінальній архітектурі – точно такий же kernel size 5, padding 0. Знову активація тангенсом, знову пулінг. З зображення, яке було 14 на 14, згортка зробить зображення 10 на 10, тому що з'їсть по 2 пікселя з кожного боку, а пулінг зробить зображення 5 на 5.

Далі розгортуємо зображення в один вектор (це ми робимо в функції forward, а поки ми уявимо, що ми вже розтягли все зображення в один вектор), і далі нам потрібні три повнозв'язних шари. Перший повнозв'язний шар на вхід приймає зображення розміром 5 на 5 і глибиною 6. Перемножуємо 5 на 5 на 6, отримуємо 400 – це розмір нашого вектора, і на виході ми хочемо вектор розміру 120.

```
self.fc1 = torch.nn.Linear(5 * 5 * 16, 120)
```

Знову тангенс, і знову повнозв'язний шар 120 на 84 (self.fc2 =torch.nn.Linear(120, 84)), знову тангенс, і нарешті fully-connected шар, який складається з 84 нейронів і робить 10 з відповідями (self.fc3 =torch.nn.Linear(84, 10)). А в функції forward ми повторюємо всю цю логіку, але тепер застосовуємо ці шари до деякого вхідного тензора X.

Вхідний тензор X – це, насправді, Батч з картинок. Застосовуємо згортку, активацію, пулінг, згортку, активацію, пулінг, і виконавши x

=x.view(x.size(0), x.size(1) * x.size(2) * x.size(3)) розгортаємо тензор, який чотиривимірний, тому що перша розмірність відповідає за розмірність Батч. У PyTorch-тензорів є функція view, яка тензор перетворює до потрібної розмірності. Перша розмірність буде x.size [0] – це розмір Батч, а далі тензор буде одновимірний, відповідно три наступні розмірності ми повинні просто перемножити і отримати 400. Далі – перший повнозв'язний шар, активація, другий повнозв'язний, активація, повнозв'язний шар.

Останнє це ініціалізація мережі, вона без параметрів (lenet5 =LeNet5()):

```
class LeNet5(torch.nn.Module):
     def __init__(self):
          super(LeNet5, self).__init__()
          self.conv1 = torch.nn.Conv2d(
                    in_channels=1,        out_channels=6,
kernel_size=5, padding=2)
          self.act1 = torch.nn.Tanh()
          self.pool1 = torch.nn.AvgPool2d(kernel_size=2,
stride=2)
          self.conv2 = torch.nn.Conv2d(
                    in_channels=6,       out_channels=16,
kernel_size=5, padding=0)
          self.act2 = torch.nn.Tanh()
          self.pool2 = torch.nn.AvgPool2d(kernel_size=2,
stride=2)
          self.fc1 = torch.nn.Linear(5 * 5 * 16, 120)
          self.act3 = torch.nn.Tanh()
          self.fc2 = torch.nn.Linear(120, 84)
          self.act4 = torch.nn.Tanh()
          self.fc3 = torch.nn.Linear(84, 10)
     def forward(self, x):
          x = self.conv1(x)
          x = self.act1(x)
          x = self.pool1(x)
          x = self.conv2(x)
          x = self.act2(x)
          x = self.pool2(x)
          x = x.view(x.size(0), x.size(1) * x.size(2) *
x.size(3))
          x = self.fc1(x)
          x = self.act3(x)
          x = self.fc2(x)
```

```
        x = self.act4(x)
        x = self.fc3(x)
        return x
lenet5 = LeNet5()
```

**Навчання**

Процес навчання, як і в минулий раз, буде йти по Батчах. Батч буде розміру 100, кожну епоху ми будемо друкувати accuracy і накопичувати loss на даній епосі. А всередині кожного Батч будемо спочатку обнуляти градієнти, після цього обчислювати – які ж картинки підуть в поточний Батч, переносити поточний Батч на device (на CUDA). Після цього пропускати Батч через мережу за допомогою функції forward, яку реалізували вище, після цього на передбачення (prediction) мережі рахувати функцію втрат loss (вона без софтмаксу, тому що крос-ентропія приймає виходи без софтмаксу). Далі будемо рахувати градієнти і робити крок градієнтного спуску. Після закінчення розрахунків на Батчі можемо порахувати якість на відкладеній вибірці, на X_test. Тут ми знову робимо обрахування якості на X_test цілком.

Точно так само ми можемо розрахувати точність (accuracy) на кожному з Батчів, а потім підсумувати та підрахувати сумарну точність.

І ще один момент, дуже важливий при використанні відеокарти: може статися виток пам'яті – ми цього не помітили, а от на даному прикладі, так як у нас мережа стає трохи більше, ми починаємо відчувати, що пам’яті на відеокарті не вистачає – в якийсь момент обрахування може впасти. Чому так відбувається? Тому що результат обчислення втрат на тестових даних ми клали в минулий раз безпосередньо в list – об'єкт, який зберігав весь граф обчислень, було test_loss_history.append(loss(test_preds, y_test).cpu())).

Виходить, що в loss зберігається не просто число, а весь граф обчислень, який нам допомагає потім обрахувати градієнти. Так як, по-перше, це все зберігається на GPU (бо у нас loss зберігається на GPU, якщо ми його спеціально на CPU не перенесемо), і весь цей loss тепер зберігається у векторі – відповідно, не очищається пам'ять. Тобто на GPU накопичуються ці графи, за якими при бажанні можна обчислити градієнти, але насправді нам ці графи не потрібні. Нам потрібні самі числа, які відповідають нашій функції втрат: її чисельне представлення. Тому ми можемо взяти, і викинути всю інформацію про граф обчислення похідної, для цього можна просто сказати їй ".data", і залишиться тільки одне число – скаляр. Після цього потрібно ще її, по-хорошому, відправити на CPU, щоб вона не займала нам пам'ять.

Далі можемо порахувати якість. Точно так же, як в минулий раз, ми беремо той нейрон, у якого вихід найбільший. Порівнюємо номер цього нейрона (ми беремо спеціально argmax, щоб у нас був номер нейрона) з y_test, а там знаходиться номер тієї цифри яку ми хочемо передбачити. Далі

перетворюємо відповідь до float, тому що саме у float можна взяти середнє (mean).

Беремо mean у цього float, після цього теж переводимо все в одне число (на всякий випадок), і беремо від нього ".cpu" (тобто, переміщаємо його на CPU). І далі accuracy, точно так же, складуємо в test_accuracy_history:

```
device = torch.device('cuda:0' if
torch.cuda.is_available() else 'cpu')
lenet5 = lenet5.to(device)
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(lenet5.parameters(),
lr=1.0e-3)
batch_size = 100
test_accuracy_history = []
test_loss_history = []
X_test = X_test.to(device)
y_test = y_test.to(device)
for epoch in range(10000):
    order = np.random.permutation(len(X_train))
    for start_index in range(0, len(X_train),
batch_size):
        optimizer.zero_grad()
        batch_indexes =
order[start_index:start_index+batch_size]
        X_batch = X_train[batch_indexes].to(device)
        y_batch = y_train[batch_indexes].to(device)
        preds = lenet5.forward(X_batch)
        loss_value = loss(preds, y_batch)
        loss_value.backward()
        optimizer.step()
    test_preds = lenet5.forward(X_test)
    test_loss_history.append(loss(test_preds,
y_test).data.cpu())
    accuracy = (test_preds.argmax(dim=1) ==
            y_test).float().mean().data.cpu()
    test_accuracy_history.append(accuracy)
    print(accuracy)
lenet5.forward(X_test)
# plt.plot(test_accuracy_history)
plt.plot(test_loss_history)
```

**Список використаних джерел**


1. Глибовець М.М., Олецький О.В. Штучний інтелект: Підручн. для студ. вищ. навч. закладів, що навчаються за спец. «Компютерні науки» та «Прикладна математика». – К.: Вид. дім «КМ Академія», 2002. – 366 с.
2. Горошко Ю.В. Інформаційне моделювання у підготовці учителів математики та інформатики. Монографія.- Чернігів: Видавець Лозовий В.М., 2012.- 368 с.
3. Горошко Ю.В., Цибко Г.Ю., Костюченко А.О.. Технології опрацювання великих даних у навчанні інформатичних дисциплін // Вісник Національного університету «Чернігівський колегіум» імені Т. Г. Шевченка. Вип. 12 (168) /Національний університет «Чернігівський колегіум»імені Т. Г. Шевченка ; голов. ред. М. О. Носко. Чернігів : НУЧК, 2021. 324 с. (Серія: Педагогічні науки). - с. 8-17
4. Троцько В.В. Методи штучного інтелекту: навчально-методичний і практичний посібник. – Київ: Університет економіки та права «КРОК», 2020 – 86 с.
5. Федоін І.В. Методи та технології обчислювального інтелекту. Практикум. Навчальний посібник.-К.: КПІ ім. Ігоря Сікорського, 2022.– 317 с.
6. Chat GPT.
7. Wikipedia.