Машинне навчання

3 кредити

залік

Історія Chat GPT

(Generative Pre-trained Transformer, генеративний попередньо-навчений трансформер)

- **GPT-1 2018 р. Розробник - компанія OpenAI**

Генерація наступного слова фрази – на основі пгенеративної мовної моделі з використанням нейронної мережі Transformer.

Навчання на текстах без розмітки. 70 млн параметрів.

GPT-2 2019 р. Розробник - компанія OpenAI

Навчання на основі англомовного форуму reddit .

Використані 8 млн текстів (40 GB), 1.5 млрд параметрів (6 GB).

Генерація есе на задану тему англійською мовою. В 10 разів > **GPT-1**

Історія Chat GPT

- **GPT-3 2020 р. Розробник - компанія OpenAI**

420 GB текстів різними мовами, 175 млрд параметрів, мовна модель – 700 GB.

Модель сама навчилась арифметиці і перекладу між різними мовами. Prompt – запити до **GPT.**

GPT-3.5 листопад 2022 р. Розробник - компанія OpenAI

Нейромережа донавчалась з використанням зворотного звязку для врахування морально-етичних обмежень.

Вбудована в Bing, Microsoft придбала **OpenAI**

Історія Chat GPT

GPT-4 травень 2023 р. Розробник - компанія OpenAI

Зручний інтерфейс –діалогове вікно

Модель навчається приблизно на 13 трлн токенів із різних джерел. Здебільшого це інтернет, книги та наукові статті. Щоб зменшити витрати на навчання, OpenAI використовує тензорний і конвеєрний паралелізм.

GPT-4 має 1,8 трлн параметрів на 120 рівнях, що в десять разів більше, аніж має GPT-3.

GPT-4 використовує модель MoE (Mixture of Experts) із 16 експертами, кожен із яких має приблизно 111 млрд параметрів. MoE дозволяє робити логічне виведення ефективніше, потребуючи близько 280 млрд параметрів і 560 TFLOPs.

Орієнтовна вартість навчання для GPT-4 становить близько $63 млн.

Остання версія - GPT-4.4

Сем Альтман

народженні	англ. *Samuel Harris Altman*
Народився	22 квітня 1985 (38 років) Чикаго
Країна	США
Місце проживання	Сан-Франциско
Діяльність	підприємець, програміст, блогер
Alma mater	Стенфордський університет і John Burroughs School[d]
Знання мов	англійська
Заклад	Y Combinator, Reddit Inc.[d], Loopt[d] і OpenAI[1][2][…]
Роки активності	2005 — тепер. час
Сайт	blog.samaltman.com

Машинне навчання

- Машинне навчання – це підгалузь штучного інтелекту, що базується на створенні алгоритмів, здатних вчитися на даних і приймати рішення без заздалегідь запрограмованих інструкцій.
- Машинне навчання дозволяє комп’ютерам вирішувати складні задачі, аналізуючи великі масиви даних і виявляючи закономірності.
- Машинне навчання – це потужна технологія, яка відкриває широкі можливості для автоматизації і оптимізації бізнесу. Алгоритми машинного навчання здатні аналізувати великі дані, виявляти приховані закономірності і робити точні прогнози.
- Однак застосування машинного навчання потребує обережності і розуміння його сильних та слабких сторін. Лише комплексний підхід з урахуванням етичних принципів дозволить повною мірою використати потенціал цієї технології для вирішення актуальних проблем і покращення нашого життя.

Як працює машинне навчання

Процес машинного навчання можна умовно розділити на три етапи:

1. Навчання

На цьому етапі алгоритм аналізує навчальний набір даних (тренувальна вибірка) і виявляє закономірності. Чим більше і якісніших даних, тим краще алгоритм зможе навчитися.

2. Тестування

Після навчання модель тестується на окремому наборі даних, щоб оцінити її точність. Якщо точність незадовільна, алгоритм доопрацьовується.

3. Використання

Коли модель навчилася достатньо добре, її можна застосувати для прогнозування, класифікації чи інших задач штучного інтелекту. Наприклад, розпізнавання образів, машинний переклад, автоматизація роботи чат-ботів тощо.

Основні типи машинного навчання

Існують три основні підходи в машинному навчанні:

Навчання з вчителем (контрольоване)

В алгоритм заздалегідь вводяться правильні відповіді (набір даних з мітками). Модель аналізує дані і намагається знайти зв'язок між вхідними даними і бажаним результатом.

Приклад: розпізнавання об'єктів на зображеннях.

Навчання без вчителя (не контрольоване)

Алгоритм самостійно шукає закономірності в даних без попередньо відомих відповідей.

Приклад: кластеризація даних, зниження розмірності.

Навчання з підкріпленням

Алгоритм вчиться на основі взаємодії з навколишнім середовищем шляхом випробувань і помилок, отримуючи позитивне або негативне підкріплення за свої дії.

Приклад: навчання AI-систем грати в ігри.

Переваги машинного навчання

Автоматизація складних задач. Машинне навчання дозволяє автоматизувати задачі, які раніше потребували людської участі.

Обробка великих даних. Алгоритми можуть аналізувати величезні масиви даних, виявляючи неочевидні закономірності.

Адаптивність. Моделі постійно вдосконалюються і адаптуються під нові дані.

Персоналізація. Можливість навчати моделі під конкретного користувача.

Економія. Автоматизація дозволяє знизити витрати на рутинні операції.

Сфери застосування:

- Рекомендаційні системи (YouTube, Netflix)
- Розпізнавання образів (Face ID, автопілоти)
- Обробка природної мови (Siri, Google Assistant, ChatGPT)
- Аналіз даних і прогнозування (фінанси, маркетинг)
- Кібербезпека (виявлення шахрайства)
- Медицина (діагностика захворювань)

Виклики і обмеження машинного навчання

Незважаючи на успіхи, машинне навчання має певні обмеження:

- **Якість даних.** Від якості навчальних даних залежить успіх моделі.
- **Упередженість даних.** Моделі можуть відтворювати і підсилювати людські упередження в даних.
- **“Чорний ящик”.** Важко пояснити і зрозуміти, як модель досягла певного результату.
- **Обчислювальні ресурси.** Для навчання складних моделей потрібно багато потужностей.
- **Кібербезпека.** Моделі можуть бути вразливими до хакерських атак.
- **Морально-етичні виклики.** Моделі можеть використовуватись в дискрутивних цілях

Тому розвиток машинного навчання потребує комплексного підходу, який враховує етичні аспекти та кібербезпеку.

Типи задач машинного навчання

У машинному навчанні основні типи задач зазвичай поділяють на наступні категорії:

1. **Класифікація**: Задача класифікації полягає в присвоєнні категорій (класів) вхідним даним. Модель машинного навчання намагається визначити, до якого класу належить новий приклад на основі навчальних даних.
   - **Бінарна класифікація**: Коли є лише два класи, наприклад, спам або не спам.
   - **Багатокласова класифікація**: Коли є більше ніж два класи, наприклад, класифікація зображень за категоріями об'єктів.
2. **Регресія**: Задача регресії полягає в передбаченні неперервної величини. Модель вивчає відносини між змінними та прогнозує числовий результат.
   - **Лінійна регресія**: Прогнозування значення змінної на основі лінійної залежності від інших змінних.
   - **Нелінійна регресія**: Коли залежність між змінними є нелінійною.

Типи задач машинного навчання

3. **Кластеризація**: Задача кластеризації полягає в групуванні об'єктів за певними схожими характеристиками без використання попередньо визначених міток.

   Ієрархічна кластеризація: Створення ієрархії кластерів, які можуть бути візуалізовані у вигляді деревоподібної структури.

   Нечітка кластеризація: Кожен об'єкт може належати до більше ніж одного кластера з певним ступенем приналежності.

4. **Зниження розмірності**: Задачі, які полягають у зменшенні кількості вхідних змінних, видаляючи неважливі або шумові дані, що не вносять значного вкладу в передбачувану змінну.

   Вибір особливостей: Вибір найбільш значущих особливостей (змінних) для використання в моделі.

   Витягання особливостей: Трансформація вхідних даних у новий набір особливостей, як правило, з меншою розмірністю.

5. **Аномалії (виявлення викидів)**: Задачі, які полягають у виявленні аномальних, несподіваних або рідкісних елементів у даних, які можуть вказувати на помилки або нові, раніше невідомі закономірності.

6. **Рекомендаційні системи**: Задачі, які полягають у створенні систем, що рекомендують користувачам продукти чи послуги на основі їхніх попередніх переваг та поведінки.

Алгоритми машинного навчання

Існує багато алгоритмів машинного навчання, кожен з яких підходить для різних типів задач і даних. Ось деякі з найпопулярніших алгоритмів:

1. **Лінійна регресія**: Це один з найпростіших алгоритмів, що використовується для прогнозування числових значень. Він намагається знайти найкращу лінійну відповідність між вхідними та вихідними змінними.
2. **Логістична регресія**: Цей алгоритм використовується для задач бінарної класифікації і вивчає ймовірність того, що дана вхідна змінна належить до однієї з двох категорій.
3. **Дерева рішень (Decision Trees)**: Популярний метод для класифікації та регресії, який використовує деревоподібну модель рішень.
4. **Випадковий ліс (Random Forest)**: Ансамблевий метод, який створює багато дерев рішень на підмножинах даних та використовує середнє їх прогнозів для покращення точності та контролю перенавчання.
5. **Машини опорних векторів (Support Vector Machines, SVM)**: Ефективний алгоритм класифікації, який шукає гіперплощину, яка найкраще розділяє класи даних.

Алгоритми машинного навчання

6. **k-найближчих сусідів (k-Nearest Neighbors, k-NN)**: Простий алгоритм, що класифікує об'єкти на основі найближчих у просторі особливостей тренувальних зразків.
7. **Наївний Баєсів класифікатор (Naive Bayes)**: Заснований на застосуванні теореми Баєса з припущенням про незалежність між особливостями, цей алгоритм часто використовується для класифікації текстових даних.
8. **Градієнтний бустінг (Gradient Boosting)**: Ще один ансамблевий метод, який покращує модель за допомогою послідовного додавання нових моделей, які виправляють помилки попередніх моделей.
9. **Нейронні мережі (Neural Networks)**: Велика категорія моделей, що намагаються імітувати роботу людського мозку для розв'язання складних задач, таких як розпізнавання образів, мови та інших складних візерунків у даних.
10. **Глибоке навчання (Deep Learning)**: Підмножина нейронних мереж з багатьма прихованими шарами, яка дозволяє моделювати дуже складні відносини у великих обсягах даних.

Інструменти та бібліотеки машинного навчання (Python)

У світі машинного навчання існує безліч інструментів та бібліотек, які полегшують розробку та реалізацію алгоритмів. Python є однією з найпопулярніших мов програмування для машинного навчання через свою читабельність, гнучкість та велику екосистему бібліотек. Ось деякі з ключових бібліотек та інструментів, які використовуються в Python для машинного навчання:

1. **Scikit-learn (sklearn)**: Це одна з найбільш популярних бібліотек для машинного навчання в Python. Вона містить широкий набір алгоритмів для класифікації, регресії, кластеризації та вибору змінних, а також інструменти для попередньої обробки даних, оцінки моделі та пайплайнів.
2. **TensorFlow**: Відкрита бібліотека, розроблена Google Brain Team, TensorFlow широко використовується для глибинного навчання. Вона дозволяє створювати складні архітектури нейронних мереж з можливістю масштабування на різні платформи від мобільних пристроїв до великих кластерів серверів.
3. **Keras**: Це високорівневий інтерфейс для нейронних мереж, який працює поверх TensorFlow, Theano або CNTK. Keras робить експериментування з нейронними мережами швидким та простим завдяки своїй простоті та зручності.
4. **Pandas**: Бібліотека для обробки та аналізу даних, яка надає швидкі, гнучкі та виразні структури даних, призначені для роботи з "реляційними" або "маркованими" даними, які легко і інтуїтивно роблять роботу з даними в Python.

Інструменти та бібліотеки машинного навчання

5. **NumPy**: Бібліотека для обчислень з великими масивами та матрицями. NumPy є основою для багатьох інших бібліотек машинного навчання, оскільки вона надає ефективні структури даних та операції.
6. **Matplotlib**: Бібліотека для створення статичних, інтерактивних та анімованих візуалізацій у Python. Matplotlib може використовуватися для створення графіків, гістограм, спектрів потужності тощо.
7. **Seaborn**: Надбудова над Matplotlib, яка надає більш високорівневий інтерфейс для створення статистичних графіків. Seaborn робить графіки більш привабливими та зрозумілими за замовчуванням.
8. **Plotly**: Інша бібліотека для візуалізації, яка підтримує інтерактивні графіки та дашборди.
9. **Jupyter Notebook**: Веб-додаток, який дозволяє створювати та ділитися документами, які містять живий код, рівняння, візуалізації та пояснювальний текст.

Ці інструменти та бібліотеки забезпечують потужні можливості для дослідників та розробників, що працюють у сфері машинного навчання, і вони стали стандартом у цій області.

Середовища для розробки

- **Visual Studio Code**
- **IDLE**
- **Anaconda (Spider)**
- **Colab,** або Google Colaboratory, є безкоштовним облачним сервісом від Google, який дозволяє писати та виконувати код Python через веб-браузер. Colab широко використовується для машинного навчання, аналізу даних та освіти, оскільки він надає легкий доступ до потужних обчислювальних ресурсів, включаючи графічні процесори (GPU) та тензорні процесори (TPU).

Основні особливості та переваги Colab

1. **Безкоштовне використання**: Colab доступний безкоштовно для всіх з Google обліковим записом.
2. **Немає необхідності встановлення**: Все, що вам потрібно для роботи з Colab, – це веб-браузер. Немає потреби встановлювати Python чи будь-які бібліотеки на локальний комп'ютер.
3. **Доступ до GPU та TPU**: Colab надає доступ до графічних та тензорних процесорів, що значно прискорює обчислення, особливо корисно для тренування моделей машинного навчання.
4. **Сумісність з Jupyter Notebook**: Colab ноутбуки сумісні з Jupyter Notebook, тому ви можете експортувати ноутбуки з Jupyter до Colab і навпаки.
5. **Спільна робота в реальному часі**: Colab підтримує спільну роботу над ноутбуками в режимі реального часу, подібно до Google Docs.
6. **Інтеграція з Google Drive**: Colab тісно інтегрований з Google Drive, що дозволяє легко зберігати та ділитися ноутбуками, а також доступати до даних, збережених у вашому Drive.

Для роботи з Colab, відвідайте веб-сайт https://colab.research.google.com/ і увійдіть за допомогою вашого Google облікового запису.

Основні навички, необхідні для штучного інтелекту

- **Наука про дані:** Багатодисциплінарна галузь, зосереджена на використанні даних для отримання розуміння, навички науки про дані мають вирішальне значення для ШІ. Вони можуть включати все, від програмування до математики, і вони допомагають дослідникам даних використовувати такі методи, як статистичне моделювання та візуалізація даних.
- **робототехніка:** AI забезпечує роботів комп'ютерним зором, щоб допомогти їм орієнтуватися та відчувати середовище.
- **етика:** Кожен, хто займається штучним інтелектом, повинен добре знати всі етичні наслідки такої технології. Етика є одним із головних питань, що стосуються розгортання систем ШІ.
- **Знання домену:** Маючи знання в галузі, ви краще зрозумієте галузь. Це також допоможе вам розробити інноваційні технології для вирішення конкретних проблем і ризиків, покращуючи підтримку вашого бізнесу.
- **Машинне навчання:** Щоб по-справжньому зрозуміти ШІ та застосувати його найкращим чином, ви повинні добре розуміти машинне навчання. Хоча вам, можливо, не потрібно знати кожен окремий технічний аспект розробки машинного навчання, ви повинні знати його фундаментальні аспекти.

Основні навички, необхідні для машинного навчання

- **Програмування:** Кожен фахівець з машинного навчання повинен володіти такими мовами програмування, як Java, R, Python, C++ і Javascript.
- **Математика:** Фахівці з машинного навчання багато працюють з алгоритмами та прикладною математикою, тому вони повинні володіти сильними аналітичними навичками та навичками вирішення проблем у поєднанні з математичними знаннями.
- **Архітектура нейронної мережі:** Нейронні мережі є фундаментальними для глибокого навчання, яке є підмножиною машинного навчання. Експерти з ML мають глибоке розуміння цих нейронних мереж і того, як їх можна застосовувати в різних секторах.
- **Великі дані:** Основною частиною машинного навчання є великі дані, де ці моделі аналізують масивні набори даних, щоб ідентифікувати закономірності та робити прогнози. Великі дані стосуються ефективного вилучення, управління та аналізу величезних обсягів даних.
- **Розподілені обчислення:** Розподілені обчислення, галузь інформатики, є ще однією важливою частиною машинного навчання. Це відноситься до розподілених систем, компоненти яких розташовані на різних об'єднаних в мережу комп'ютерах, які координують свої дії шляхом обміну повідомленнями.

Препроцесінг даних

- Препроцесінг даних у машинному навчанні відіграє критично важливу роль у підготовці сировинних даних для ефективного та точного моделювання.
- Цей процес включає різноманітні методи та техніки, спрямовані на очищення, нормалізацію, трансформацію та вибір ознак з даних перед їхнім використанням у алгоритмах машинного навчання.
- Препроцесінг допомагає покращити якість даних, що може значно збільшити точність та ефективність моделей машинного навчання.

Основні кроки препроцесінгу даних

1. **Очищення даних:**
   - **Видалення або коригування шуму** (неправильні або викривлені дані).
   - **Заповнення відсутніх значень** через інтерполяцію, використання середнього/медіанного значення або за допомогою моделювання.
   - **Видалення дублікатів**, щоб уникнути зайвого впливу повторюваних даних на модель.

2. **Трансформація даних:**
   - **Нормалізація та стандартизація** для приведення всіх ознак до одного масштабу, що особливо важливо для алгоритмів, чутливих до масштабу ознак.
   - **Кодування категоріальних даних** у числовий формат через one-hot encoding, label encoding тощо.
   - **Розбиття текстових даних на токени**, очищення від стоп-слів, стемінг, лематизація для подальшої роботи з текстом.

Основні кроки препроцесінгу даних

3. Вибір ознак:

- **Видалення нерелевантних ознак**, що не вносять корисної інформації для моделі.
- **Вибір найважливіших ознак** за допомогою статистичних тестів, алгоритмів вибору ознак або за допомогою моделей машинного навчання.
- **Генерація нових ознак** через комбінування або трансформацію існуючих ознак для покращення моделі.

4. Перетворення даних:

- **Кодування категоріальних ознак**
- **Розбиття тексту на слова/речення**
- **Перетворення дат/часу**

5. Розбиття даних на навчальну та тестову вибірки: Відокремлення частини даних для перевірки та оцінки моделі після навчання.

Важливість препроцесінгу:

- Препроцесінг даних є важливим, оскільки:
  - Покращує якість даних, зменшуючи шум та виправляючи помилки.
  - Допомагає забезпечити більшу точність моделей.
  - Зменшує обчислювальні витрати за рахунок видалення нерелевантних признаків.
  - Дозволяє алгоритмам машинного навчання ефективніше працювати з даними.
- Правильно виконаний препроцесінг може значно підвищити шанси на успіх проекту з машинного навчання, тоді як недостатньо якісна підготовка даних може призвести до помилкових висновків та неефективності моделей.

Лінійна регресія

Лінійна регресія є одним з найпростіших та найбільш використовуваних статистичних методів у машинному навчанні для прогнозування числових значень (тобто регресії). Цей метод моделює залежність між залежною змінною (цільовою, яку ми хочемо передбачити) і однією або кількома незалежними змінними (предикторами).

Існують типи лінійної регресії:

Проста лінійна регресія: Моделює залежність між однією залежною змінною та однією незалежною змінною. Формула простої лінійної регресії: $y = w_0 + w_1 x + eps$

де y − цільова змінна, x − незалежна змінна, w_0 − перетин (intercept), w_1− коефіцієнт нахилу (slope), eps − випадкова помилка.

Поліноміальна лінійна регресія:

$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_n x^n + eps_n$

Узагальнена лінійна регресія:

$y = w_0 + w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x) + eps_n$

Багатовимірна лінійна регресія: Моделює залежність між однією залежною змінною та кількома незалежними змінними. Формула багатовимірної лінійної регресії:

$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_k + eps$

x_i − незалежні змінні, і w_i − відповідні коефіцієнти.

Лінійна регресія

Лінійна регресія - один з найпростіших і найпоширеніших методів машинного навчання для прогнозування числових значень на основі лінійної залежності між вхідними та вихідними змінними.

Основна ідея лінійної регресії полягає в наступному:

1.Маємо набір даних, що складається з вхідних ознак (предикторів) X та цільової змінної (відгуку) y, яку потрібно спрогнозувати.

2. Припускаємо, що між x та y існує лінійна залежність:

$y = w_0 + w_1 x_1 + ... + w_n x_n$, де w_i - вагові коефіцієнти моделі.

3. За допомогою методу найменших квадратів знаходимо такі w_i, щоб мінімізувати суму квадратів відхилень між реальними значення y та спрогнозованими моделлю $\hat{y}$.

Переваги лінійної регресії: простота, інтерпретовність, масштабовність.

Недоліки: робить припущення про лінійність, схильна до перенавчання.

Лінійна регресія широко застосовується в різних сферах: фінансах, страхуванні, соціології тощо для прогнозування числових показників. Вона часто використовується як базовий алгоритм або як частина складніших моделей машинного навчання.

Алгоритм лінійної регресії

1. **Формулювання моделі**: Визначаємо структуру моделі, тобто вибираємо, які змінні будуть включені до моделі як предиктори.
2. **Оцінка параметрів**: Використовуючи метод найменших квадратів (МНК), знаходимо оцінки коефіцієнтів (w_i), мінімізуючи суму квадратів різниць між спостережуваними та прогнозованими значеннями (y).

$$y = w_O + w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x) + eps_n$$

1. **Діагностика моделі**: Перевіряємо, чи модель адекватно описує дані, за допомогою різних статистичних тестів та діагностичних графіків (наприклад, аналіз залишків).
2. **Інтерпретація результатів**: Тлумачимо отримані коефіцієнти (\beta), щоб зрозуміти взаємозв'язок між незалежними та залежною змінними.
3. **Прогнозування**: Використовуючи отриману модель, робимо прогнози для нових значень незалежних змінних.

Переваги лінійної регресії:

Простота та інтерпретовність результатів.

Швидкість обчислень.

Широке застосування в різних галузях.

Обмеження:

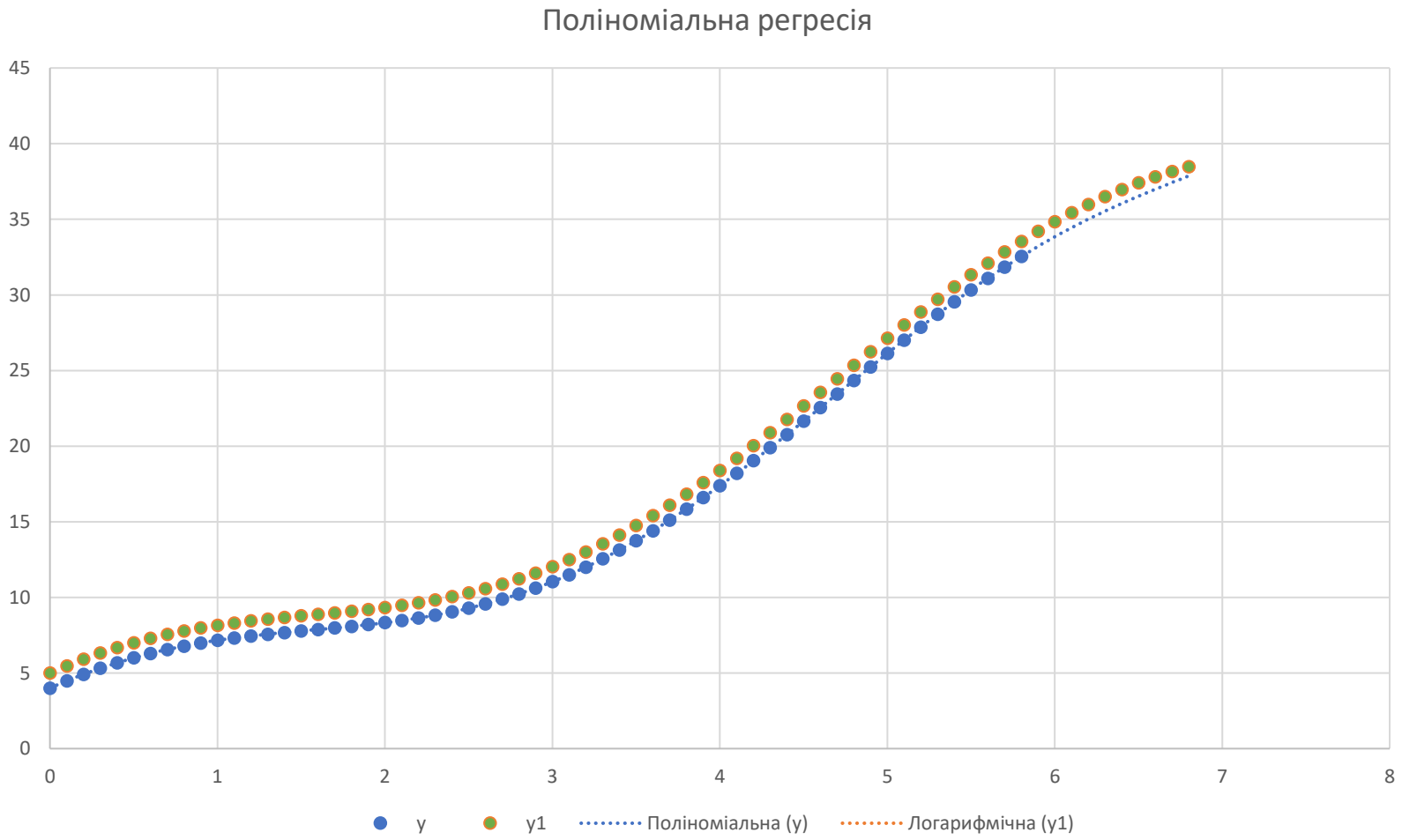
Припускає лінійний зв'язок між змінними, що може бути не завжди реалістичним.

Чутлива до викидів (outliers) та мультиколінеарності (висока кореляція між незалежними змінними).

Не може моделювати складні зв'язки без додавання поліноміальних або взаємодіючих термінів.

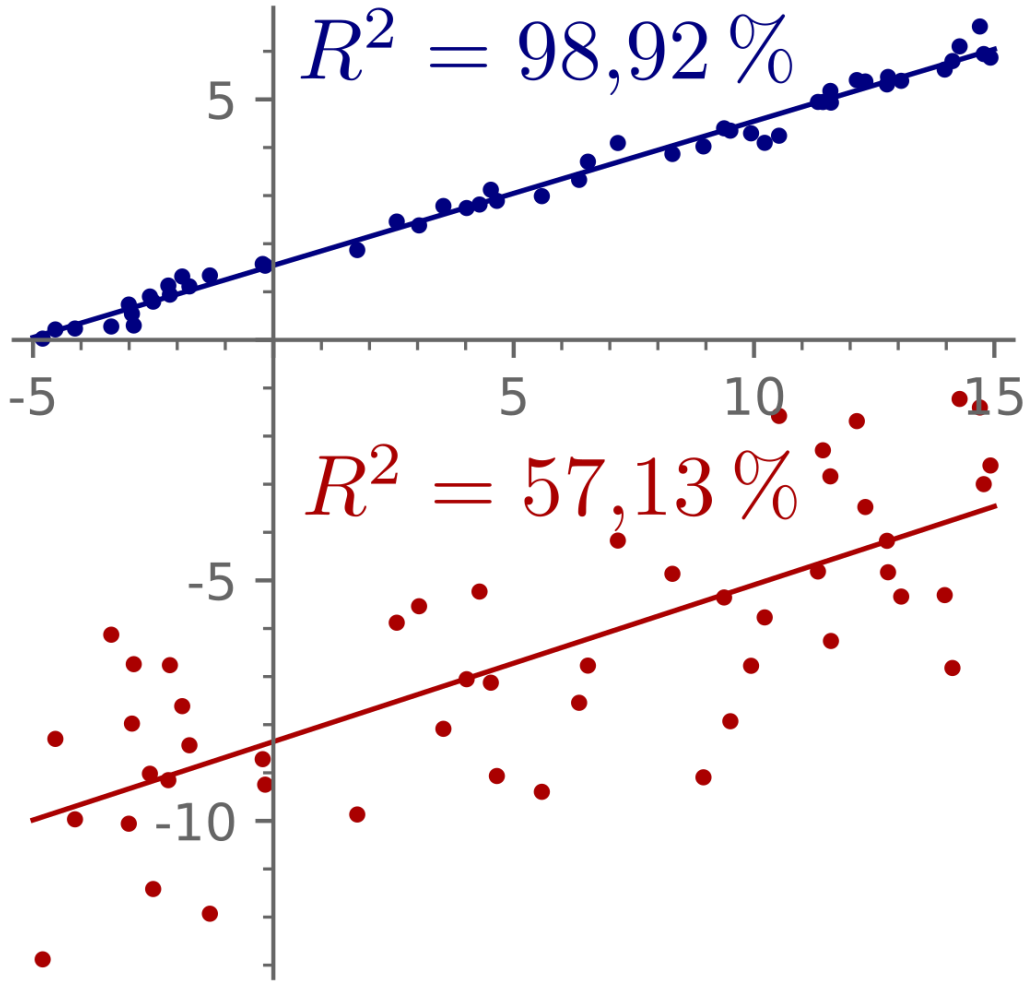
Лінійна поліноміальна регресія

- $y = w_O + w_1 x + w_2 x^2 + \ldots + w_n x^n + eps_n$
- $Y = 0{,}004x^6 - 0{,}0756x^5 + 0{,}4343x^4 - 0{,}4865x^3 - 1{,}5661x^2 + 4{,}8459x + 4{,}0108$

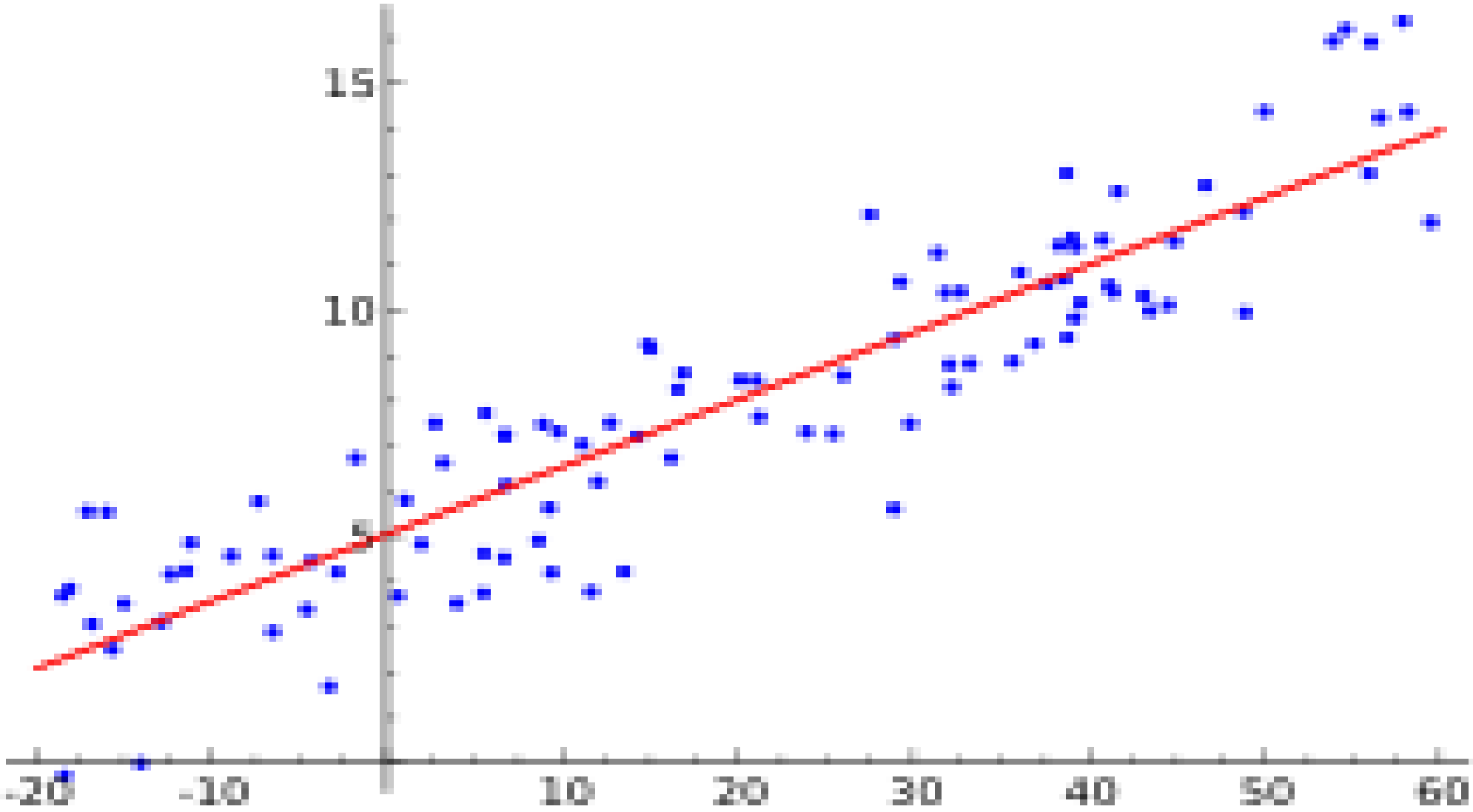


Коефіцієнт детермінації

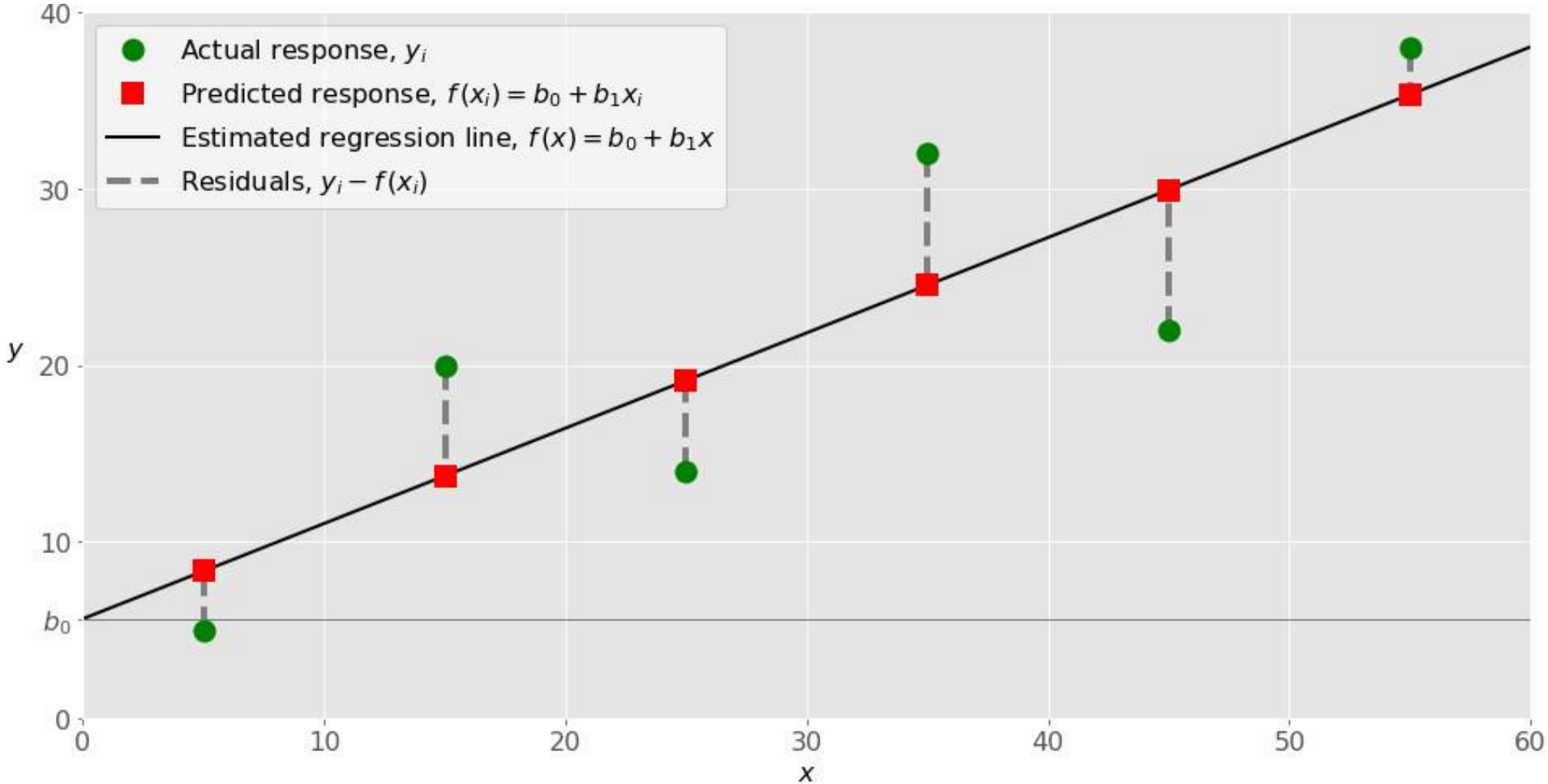
$$R^2 = \frac{\sum_{i=1}^{n} (Y_{розр} - Y_{сер})^2}{\sum_{i=1}^{n} (Y_{факт} - Y_{сер})^2}$$



Лінійна регресія



Лінійна регресія



Лінійна регресія

- Дані: x_i| y_i , i=1,…,n

Функція регресії: $y(x\ \ ) = \sum_{m=0}^{M} W_m f_m(x\ \ )$

Функція втрат:

$J(W_0, W_1, \dots, W_N) = \frac{1}{2n}\sum_{i=1}^{n}(y(x_i) - y_i)^2$ ->min

$\frac{\partial J}{\partial W_k} = \frac{1}{n}\sum_{i=1}^{n}(\sum_{m=0}^{M} W_m f_m(x_i) - y_i) \quad f_k(x_i)$

$$\sum_{m=0}^{M} W_m \frac{1}{n}\sum_{i=1}^{n} f_m(x_i)\, f_k(x_i) = \frac{1}{n}\sum_{i=1}^{n} f_k(x_i)\, y_i$$

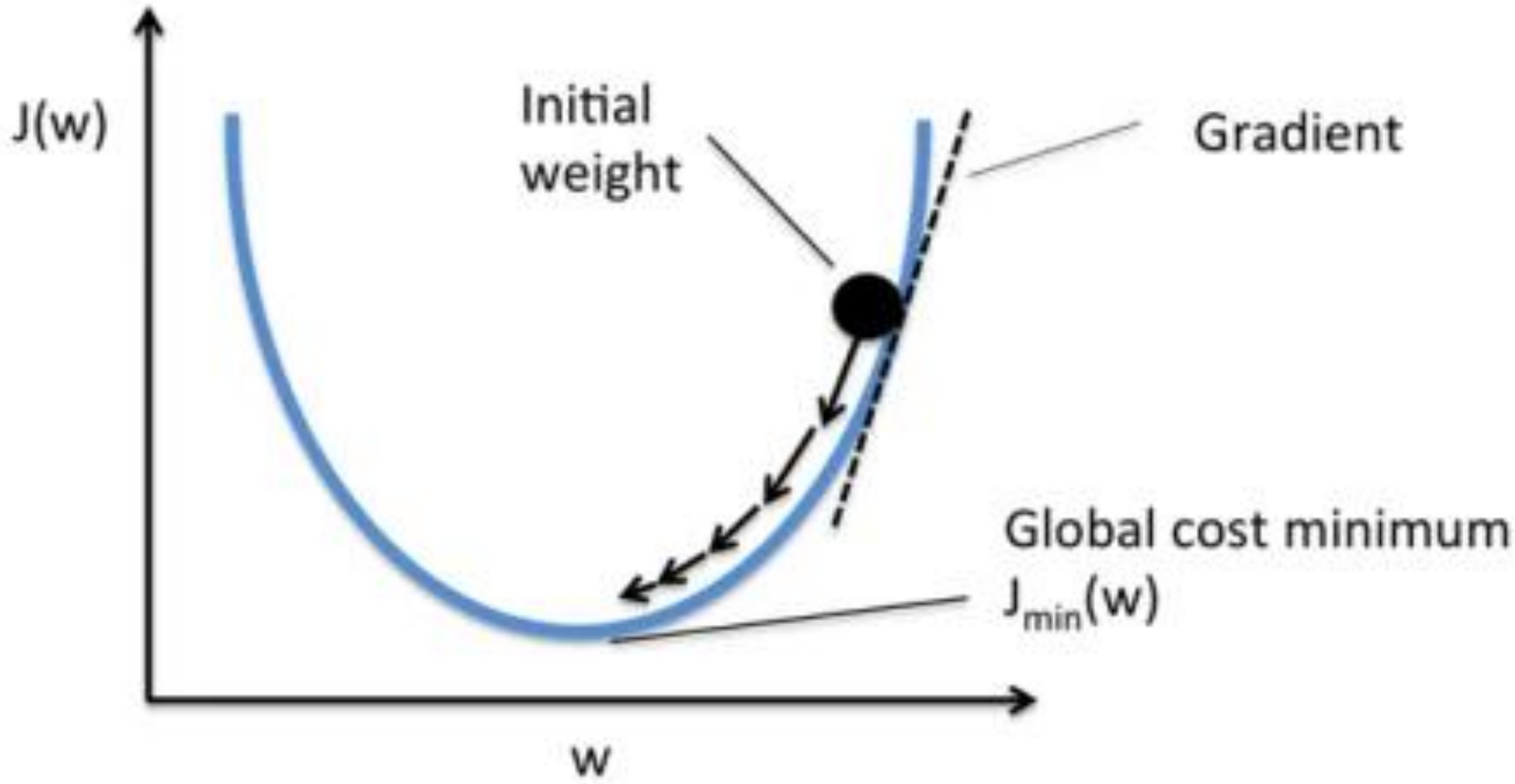
Метод градієнтного спуску

$$J(W_0, W_1, \dots, W_N) = \frac{1}{2n}\sum_{i=1}^{n}(y(x_i) - y_i)^2$$

$$y(x_i) = \sum_{m=0}^{M} W_m f_m(x_i) \quad = \sum_{m=0}^{M} W_m {x_i}^m$$

$$W_i(k+1) = W_i(k) - \eta \frac{\partial J}{\partial W_i}$$

Градієнний спуск



Стохастичний градієнтний спуск

Стохастичний градієнтний спуск (SGD) - популярний алгоритм оптимізації, який широко використовується при навчанні нейронних мереж та інших моделей машинного навчання.

Основні ідеї SGD:

Замість обчислення градієнту по всьому навчальному набору, обчислюємо його по випадковій підвибірці даних (міні-батчу)

Оновлюємо ваги моделі в напрямку обчисленого стохастичного градієнта

Повторюємо процес для багатьох ітерацій, поступово рухаючись до оптимуму

Стохастичний градієнтний спуск

Переваги:

Ефективніше масштабується на великі набори даних

Можливість онлайн-навчання по мірі надходження даних

Недоліки:

Шум в оновленнях через випадковість

Складність налаштування гіперпараметрів (швидкість навчання і т.д.)

Популярні модифікації SGD, які вирішують частину проблем: Momentum, Nesterov Accelerated Gradient, Adagrad, Adadelta, RMSprop, Adam і інші.

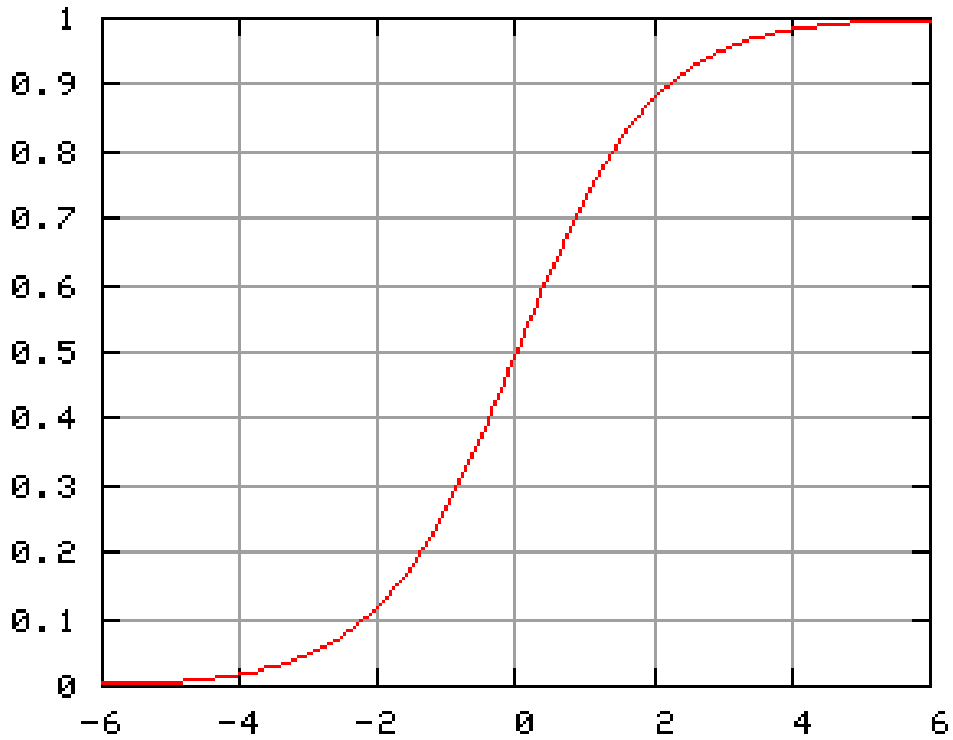
Логістична регресія (класифікація)

Відмінність логістичної регресії від лінійної полягає в тому, що вона використовує логістичну функцію (або сигмоїду) для обмеження виведеного прогнозу в інтервалі між О та 1, що дозволяє інтерпретувати його як ймовірність.

Це робить логістичну регресію особливо корисною для завдань класифікації, де потрібно визначити, до якої категорії належить спостережуваний випадок.

Логістична регресія (класифікація)

$$h(w) = \frac{1}{1+e^{-wx}}$$



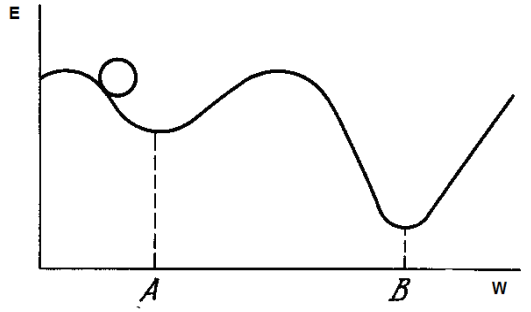
Ключові аспекти логістичного класифікатора

Бінарна класифікація: Найпростіший випадок логістичної регресії, коли вона використовується для розрізнення між двома класами (наприклад, "так" або "ні", "хворий" або "здоровий"). Вихід моделі інтерпретується як ймовірність належності до одного з цих класів.

Багатокласова класифікація: Логістичну регресію також можна адаптувати для вирішення задач багатокласової класифікації, використовуючи техніки, такі як метод "один проти всіх" (OvA) або "один проти одного" (OvO).

Оцінка ймовірності: Логістичний класифікатор не просто призначає клас спостереженню, але й оцінює ймовірність належності до кожного з можливих класів, що надає додаткову інформацію про невизначеність класифікації.

Логістична регресія (класифікація)



- Функція втрат:

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Для класифікації при заданих x:

На виході $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Функція softmax

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

Вступ до теореми Байєса

У машинному навчанні ми часто зацікавлені у виборі найкращої гіпотези (h) на основі даних (d).
У задачі класифікації наша гіпотеза (h) може бути класом, який слід призначити для нового екземпляра даних (d).
Один із найпростіших способів вибору найбільш імовірної гіпотези, враховуючи дані, які ми маємо, які ми можемо використовувати як наші попередні знання про проблему. Теорема Байєса дає змогу обчислити ймовірність гіпотези, враховуючи наші попередні знання.
Теорема Байєса формулюється так:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

Де
P(h|d) — це ймовірність гіпотези h за даними d. Це називається апостеріорною ймовірністю.
P(d|h) — це ймовірність даних d за умови, що гіпотеза h була вірною.
P(h) — це ймовірність того, що гіпотеза h є істинною (незалежно від даних). Це називається апріорною ймовірністю h.
P(d) — ймовірність даних (незалежно від гіпотези).

Вступ до теореми Байєса

Нас цікавить обчислення апостеріорної ймовірності P(h|d) з попередньої ймовірності p(h) з P(D) і
P(d|h). Після обчислення апостеріорної ймовірності для ряду різних гіпотез ви можете вибрати гіпотезу з найвищою ймовірністю. Це максимально ймовірна гіпотеза, яку формально можна назвати максимальною апостериорною (MAP) гіпотезою. Це можна записати так:

MAP(h) = max(P(h|d))

або

MAP(h) = max((P(d|h) * P(h)) / P(d))

або

MAP(h) = max(P(d|h) * P(h))

P(d) є нормалізуючим терміном, який дозволяє обчислити ймовірність. Ми можемо відмовитися від нього, коли нас цікавить найбільш імовірна гіпотеза, оскільки вона постійна і використовується лише для нормалізації. Повертаючись до класифікації, якщо ми маємо парну кількість екземплярів у кожному класі в наших навчальних даних, тоді ймовірність кожного класу (наприклад, P(h)) буде рівною. Знову ж таки, це буде постійний член у нашому рівнянні, і ми можемо його відкинути, щоб отримати:

MAP(h) = max(P(d|h))

Наївний класифікатор Байєса

Naive Bayes — це алгоритм класифікації для бінарних (двокласових) і багатокласових задач класифікації.
Техніку найлегше зрозуміти, якщо її описувати з використанням двійкових або категоріальних вхідних значень.

Його називають *наївним Байєсом* або *idiot Bayes*, тому що обчислення ймовірностей для кожної гіпотези спрощено, щоб зробити їх розрахунок доступним. Замість того, щоб намагатися обчислити значення кожного значення атрибута **P(d1, d2, d3…|h),** вони вважаються умовно незалежними з урахуванням цільового значення та обчислюються як

P(d1|h) * P(d2|H) * P(d2|H)… .

Це дуже сильне припущення, яке є малоймовірним у реальних даних, тобто те, що атрибути не взаємодіють.
Тим не менш, цей підхід напрочуд добре працює і на даних, де це припущення не виконується.

Прогноз з допомогою Naive Bayes

Враховуючи наївну модель Байєса, ви можете робити прогнози для нових даних за допомогою теореми Байєса.

MAP(h) = max(P(d|h) * P(h))

Використовуючи приклад, якщо ми маємо новий випадок із сонячною *погодою* , ми можемо обчислити:

вихід = P(погода=сонячно|клас=вихід) * P(клас=вихід)
залишайся вдома = P(погода=сонячно|клас=залишайся вдома) * P(клас=залишайся вдома)

Ми можемо вибрати клас, який має **найбільше обчислене значення.** Ми можемо перетворити ці значення на ймовірності, нормалізувавши їх таким чином:

P(виходити|погода=сонячно) = виходити / (виходити + залишатися вдома)
P(залишатися вдома|погода=сонячно) = залишатися вдома / (виходити + залишатися вдома)

Набір даних

Iris setosa

Iris versicolor

Iris virginica

Іриси Фішера				
Довжина оцвітини	Ширина чашолистка	Довжина пелюстки	Ширина пелюстки	Вид ірису
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4.0	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3.0	5.9	2.1	virginica
6.3	2.9	5.6	1.8	virginica
5.9	3.0	5.1	1.8	virginica

Дерева рішень в машинному навчанні

Дерева рішень та ансамблеві методи, такі як бустінг, бегінг і випадковий ліс (Random Forest), є потужними інструментами в машинному навчанні для розв'язання задач класифікації та регресії.

Дерева рішень

- Дерево рішень — це модель прогнозування, яка використовує набір правил для прийняття рішень.
- Кожен вузол у дереві представляє тест на атрибуті, кожна гілка — результат тесту, а кожен листовий вузол (кінцевий вузол) відповідає за прогнозоване значення.
- Дерева рішень легко інтерпретувати, але вони схильні до перенавчання, особливо при великій глибині

Алгоритм дерева рішень у машинному навчанні

- Алгоритм дерева рішень є одним з найпопулярніших методів використовуваних у машинному навчанні для розв'язання задач класифікації та регресії.

- Дерево рішень моделює рішення та їх можливі наслідки, включаючи випадкові події, витрати ресурсів та корисність. Воно складається з вузлів, гілок і листків.

- Кожен внутрішній вузол відповідає перевірці атрибута, кожна гілка відповідає результату перевірки, а кожен листок вузол (термінальний вузол) відповідає прогнозу класу або значенню.

Типологія дерев

Дерева рішень, використовувані в Data Mining, бувають двох основних типів:

- **Аналіз дерева класифікації**, коли прогнозований результат є класом, до якого належать дані;
- **Регресійний аналіз дерева**, коли прогнозований результат можна розглядати як дійсне число (наприклад, ціна на будинок, або тривалість перебування пацієнта в лікарні).

Згадані вище терміни вперше були використані Брейманом та ін. Перераховані типи мають деякі подібності, а також деякі відмінності, такі, як процедура, що використовується для визначення місця, де розбивати.

Л. Брейман, Дж. Фрідман, Р. Олшен і К. Стоун, «Дерева класифікації та регресії», Водсворт, Белмонт, Каліфорнія, 1984 р.

Деякі методи дозволяють побудувати більше одного дерева рішень

Побудова дерева рішень

Дерево рішень будується за допомогою рекурсивного поділу даних на підмножини на основі атрибутів. Цей процес продовжується, доки не буде досягнуто критерію зупинки, наприклад, коли всі елементи в підмножині належать до одного класу або коли досягнуто максимальної глибини дерева.

Основні кроки алгоритму:

- **Вибір атрибута для поділу.** На кожному кроці алгоритм вибирає найкращий атрибут для поділу даних на підмножини. Вибір атрибута заснований на метриці, такій як приріст інформації, індекс Джині або зменшення середньоквадратичної помилки.
- **Розбиття даних.** Дані розбиваються на підмножини на основі значень вибраного атрибута. Процес поділу продовжується рекурсивно для кожної підмножини.
- **Критерій зупинки.** Процес побудови дерева припиняється, коли досягнуто критерію зупинки.

Метрики для вибору атрибута

- **Приріст інформації** вимірює зменшення ентропії або невизначеності при розбитті даних на підмножини. Вибирається атрибут, що максимізує приріст інформації.
- **Індекс Джині** використовується для вимірювання ступеня нечистоти або змішування підмножин даних. Мета полягає в мінімізації індексу Джині.
- **Зменшення середньоквадратичної помилки (MSE)** зазвичай використовується при побудові дерев регресії. Вибирається поділ, що мінімізує MSE.

Метрики для вибору атрибута

Є різні способи вибирати черговий атрибут:

- **Алгоритм ID3**, де вибір атрибута відбувається на підставі приросту інформації (англ. *Gain*), або на підставі Коефіцієнту Джині.
- **Алгоритм C4.5** (поліпшена версія ID3), де вибір атрибута відбувається на підставі нормалізованого приросту інформації (англ. *Gain Ratio*).
- **Алгоритм CART** і його модифікації — IndCART, DB-CART.
- **Автоматичний детектор** взаємодії Хі-квадрат (CHAID). Виконує багаторівневий поділ при розрахунку класифікації дерев;[5]
- **MARS**: розширює дерева рішень для поліпшення обробки цифрових даних.

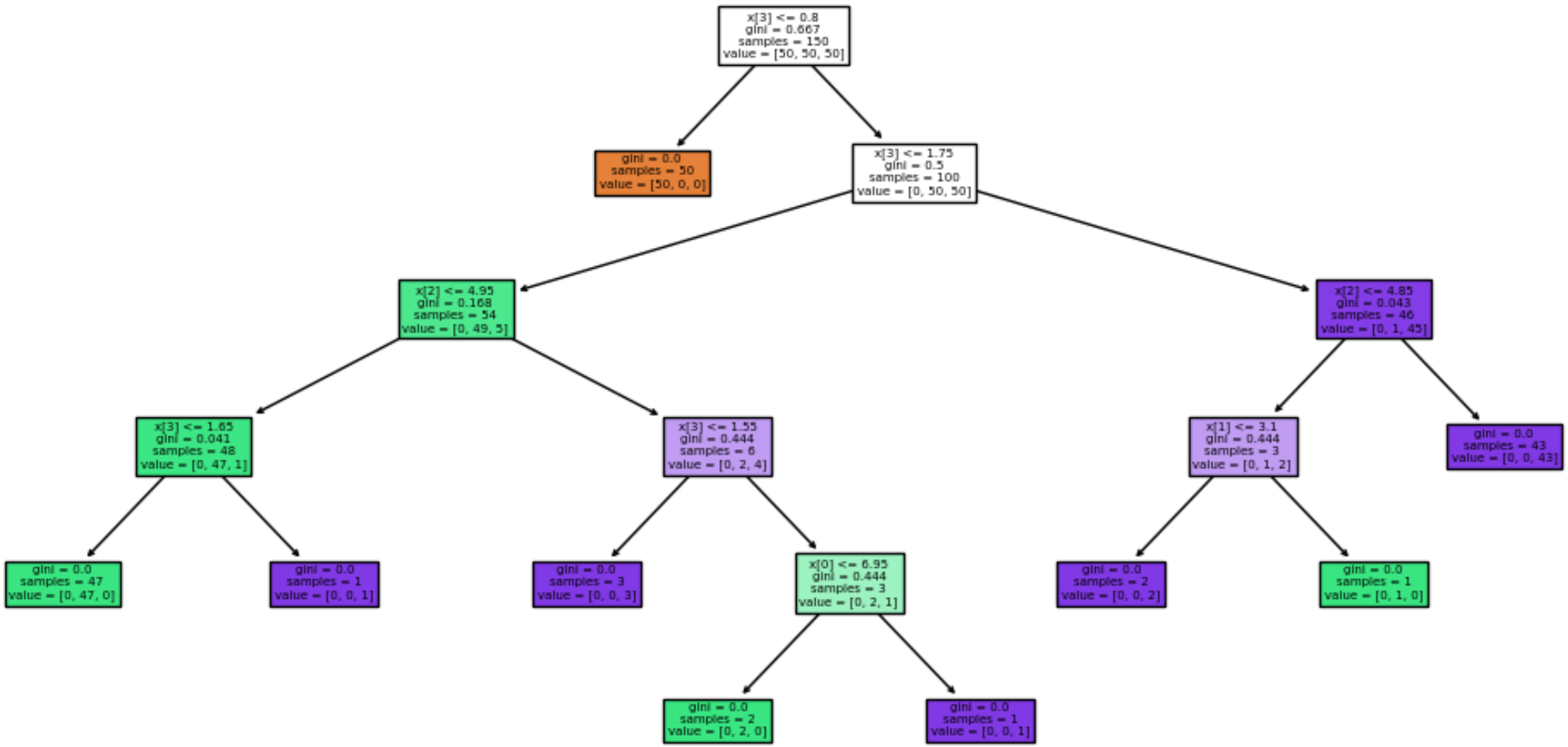
На практиці в результаті роботи цих алгоритмів часто виходять занадто деталізовані дерева, які при їх подальшому застосуванні дають багато помилок.

DecisionTreeClassifier

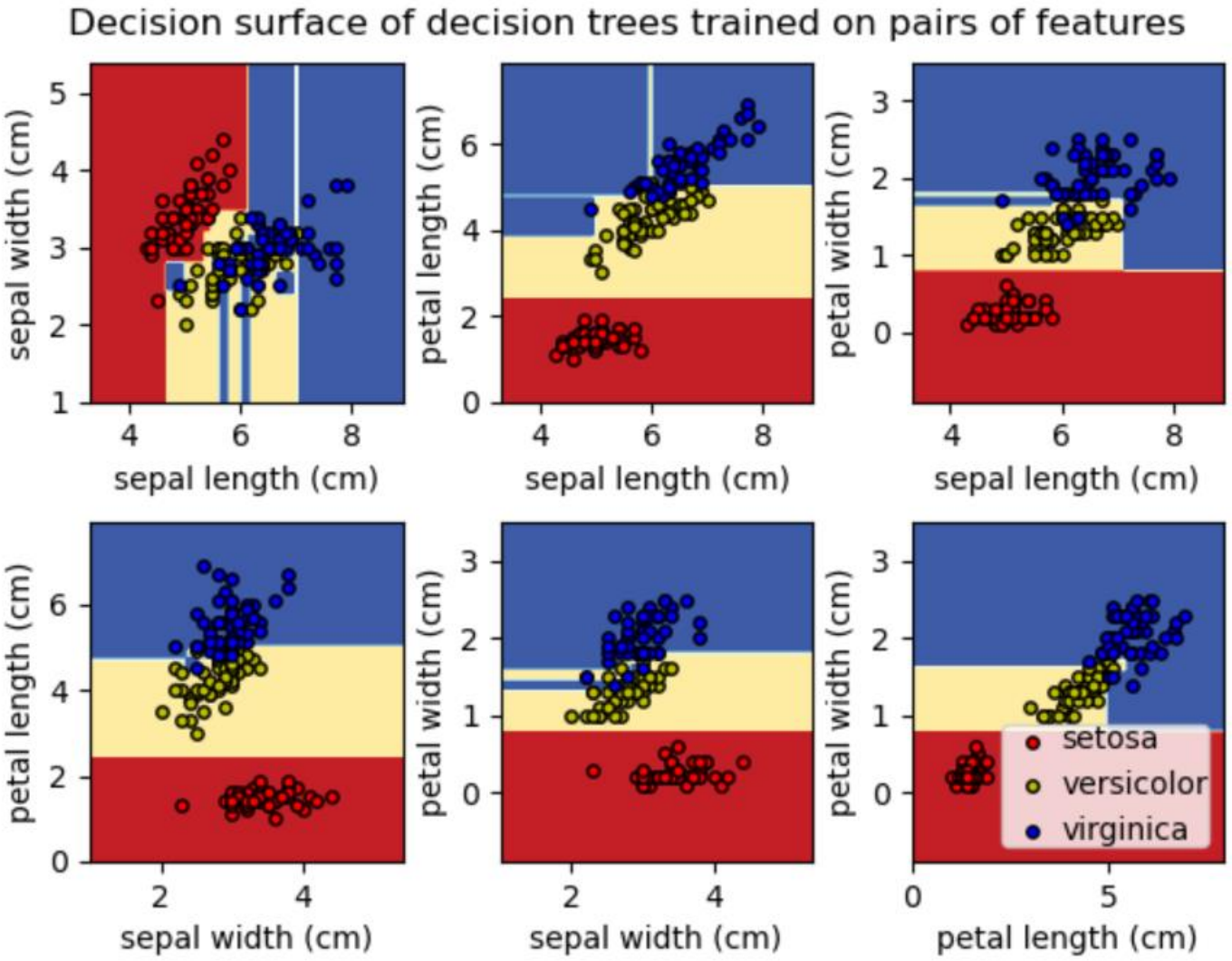
sklearn.tree. **DecisionTreeClassifier**

(criterion = 'gini' ,
splitter = 'best' ,
max_depth = None ,
min_samples_split = 2 ,
min_samples_leaf = 1 ,
min_weight_fraction_leaf = 0.0 ,
max_features = None ,
random_state = None ,
max_leaf_nodes = None ,
min_impurity_decrease = 0,0 ,
class_weight = None,
ccp_alpha = 0,0 ,
monotonic_cst = None)

Decision tree trained on all the iris features



Навчання дерева рішень для Iris



Ансамблеві методи в деревах рішень

Ансамблеві методи в деревах рішень є популярними та ефективними техніками в машинному навчанні для підвищення точності прогнозування та уникнення перенавчання. Ці методи включають комбінацію декількох моделей дерев рішень для створення більш потужної загальної моделі.

Основні ансамблеві методи, які використовують дерева рішень, включають беггінг, випадкові ліси та бустинг.

- **Переваги:**
- Зменшення перенавчання (overfitting).
- Підвищення точності прогнозування.
- Висока ефективність на багатьох даних та в задачах.
- **Недоліки:**
- Збільшення обчислювальної складності та часу тренування.
- Складність у інтерпретації моделей порівняно з одиночним деревом рішень.

Беггінг

- Беггінг (Bootstrap Aggregating) — це метод, при якому створюється кілька версій однієї тренувальної вибірки за допомогою вибірки з поверненням, і для кожної з цих вибірок будується окреме дерево рішень.

- Після тренування кінцевий прогноз формується шляхом усереднення прогнозів від окремих дерев (для задач регресії) або шляхом голосування (для задач класифікації). Цей метод допомагає зменшити варіативність моделі.

- Бегінг — це метод ансамблю, який покращує стабільність і точність машинного навчання.

- Він працює шляхом створення кількох версій тренувального набору даних за допомогою бутстрапінгу, після чого тренує окремі моделі паралельно і об'єднує їхні прогнози.

- Бегінг ефективно зменшує варіативність і допомагає уникнути перенавчання.

Випадковий ліс (Random Forest)

- Випадковий ліс — це розширення бегінгу, яке використовує дерева рішень як базові моделі.
- Воно вносить додатковий рівень випадковості під час побудови дерев, оскільки при кожному розділенні вузла вибирається випадковий піднабір ознак.
- Це робить випадковий ліс одним з найпотужніших і найпопулярніших методів машинного навчання.

Випадковий ліс

- Алгоритм випадкового лісу (Random Forest) є одним із найбільш широко використовуваних та ефективних алгоритмів машинного навчання. Ось детальний огляд цього алгоритму:

1. **Основна ідея**:
   - Випадковий ліс – це ансамблевий метод, який поєднує множину дерев рішень.
   - Кожне дерево навчається на випадковій підвибірці тренувальних даних (бутстрепінг).
   - При розбитті вузлів дерев використовується лише випадкова підмножина ознак.
   - Остаточний прогноз отримується шляхом агрегування (усереднення або голосування) прогнозів усіх дерев.
2. **Процес навчання**:
   - З тренувального набору даних створюються множинні бутстреп-вибірки.
   - На кожній бутстреп-вибірці будується окреме дерево рішень.
   - При розбитті вузлів дерева, випадково вибирається підмножина ознак (зазвичай, квадратний корінь від загальної кількості ознак для класифікації або третина для регресії).
   - Дерева будуються до максимальної глибини без обрізки гілок.
3. **Прогнозування**:
   - Для класифікації: кожне дерево дає прогноз класу, і клас, який отримує більшість голосів, стає остаточним прогнозом ансамблю.
   - Для регресії: прогнози від усіх дерев усереднюються, щоб отримати остаточний прогноз.

Бустінг

- Бустінг(Boosting) — це техніка ансамблю, яка створює сильну модель за допомогою послідовного навчання слабких моделей, де кожна нова модель намагається виправити помилки попередніх.

- Важливими видами бустінгу є AdaBoost, Gradient Boosting і XGBoost. Вони відрізняються способами побудови послідовності моделей та оцінки помилок.

- Бустінг добре працює, коли є велика кількість даних і мало шуму, але може бути схильний до перенавчання в інших умовах.

Бустінг (англ. boosting)

Бустінг (англ. boosting) — це метод ансамблю в машинному навчанні, який об'єднує кілька слабких моделей для створення однієї сильної моделі прогнозування. Ідея полягає в тому, щоб послідовно навчати моделі, при цьому кожна наступна модель намагається виправити помилки попередніх. Ось основні кроки алгоритму бустінгу:

1. Початковий набір даних

Маємо набір тренувальних даних ((x1, y1), (x2, y2), ..., (xn, yn)), де xi — це вектор вхідних ознак для i-го прикладу, а yi — відповідний цільовий вихід (наприклад, класифікація або регресія).

2. Ініціалізація ваг

Кожному прикладу тренувального набору присвоюється вага, яка спочатку може бути рівною для всіх прикладів.

3. Послідовне навчання моделей

Алгоритм послідовно навчає серію слабких моделей. Кожна модель намагається виправити помилки, зроблені попередніми моделями, шляхом зосередження на важких для прогнозування прикладах (тобто тих, для яких попередні моделі допустили помилки).

4. Оновлення ваг

Після кожної моделі ваги тренувальних прикладів оновлюються, щоб моделі, які будуть навчені наступними, зосереджувались на більш складних прикладах.

5. Комбінування моделей

В кінці всі слабкі моделі комбінуються в одну сильну модель. Це може бути зроблено, наприклад, шляхом голосування з вагою для класифікації або шляхом взяття зваженої суми для регресії.

Популярні алгоритми бустінгу:

- **AdaBoost (Adaptive Boosting)**: Один з перших алгоритмів бустінгу, який адаптивно змінює ваги прикладів, зосереджуючись на тих, які були неправильно класифіковані попередніми моделями.
- **Gradient Boosting**: Підходить до проблеми, як до оптимізації градієнта, де кожна нова модель намагається мінімізувати втрати відносно попередньої суми моделей, використовуючи градієнтний спуск.
- **XGBoost (Extreme Gradient Boosting)**: Оптимізована реалізація Gradient Boosting, яка є дуже популярною завдяки своїй швидкості та ефективності.
- **LightGBM**: Ще одна оптимізована версія Gradient Boosting, яка використовує алгоритми зростання дерев на основі гістограм, що дозволяє їй бути швидшою та ефективнішою при роботі з великими наборами даних.
- **CatBoost**: Алгоритм бустінгу, оптимізований для категорійних даних, який також показує високу ефективність на різноманітних задачах.

Кожен з цих алгоритмів має свої особливості та параметри налаштування, але основна ідея бустінгу залишається однаковою: посилювати слабкі моделі, комбінуючи їх у сильні ансамблі.

Переваги і недоліки бустингу

Переваги бустингу:

1. Висока точність: Бустінг часто дає дуже точні результати, особливо коли базові моделі є відносно слабкими.
2. Гнучкість: Бустінг може працювати з різними типами базових моделей (дерева рішень, нейронні мережі тощо) і може вирішувати як задачі класифікації, так і регресії.
3. Обробка різних типів даних: Алгоритми бустінгу можуть обробляти числові, категоріальні та пропущені дані без необхідності значної попередньої обробки.

Недоліки бустингу:

1. Схильність до перенавчання: Якщо алгоритм бустінгу не налаштований правильно або якщо базові моделі занадто складні, він може перенавчатися на тренувальних даних.
2. Обчислювальна складність: Бустінг вимагає послідовного навчання багатьох моделей, що може бути обчислювально витратним, особливо на великих наборах даних.
3. Чутливість до шуму: Бустінг може бути чутливим до шуму та викидів у даних, оскільки він зосереджується на важких прикладах.

Порівняння методів

- **Перенавчання**: Дерева рішень схильні до перенавчання, тоді як ансамблеві методи, особливо випадковий ліс і бустінг, краще з ним борються.
- **Точність**: Ансамблеві методи зазвичай показують вищу точність прогнозування порівняно з одиночними деревами рішень.
- **Швидкість навчання та прогнозування**: Дерева рішень швидше тренуються і роблять прогнози, ніж ансамблеві методи, які вимагають більше часу та ресурсів.
- **Інтерпретованість**: Дерева рішень легко інтерпретувати, в той час як ансамблеві методи можуть бути складнішими для розуміння.

Кожен з цих методів має свої переваги та недоліки, і вибір залежить від конкретної задачі, набору даних і вимог до моделі.

Метод опорних векторів SVM (Support Vector Machines)

- SVM (Support Vector Machines, укр. Метод опорних векторів) — це надзвичайно потужний та гнучкий алгоритм машинного навчання, який використовується для задач класифікації, регресії та навіть виявлення викидів (аномалій). SVM входить до категорії машинного навчання з учителем.
- Основна ідея SVM полягає в тому, щоб знайти гіперплощину (у двовимірному просторі це буде лінія, у тривимірному — площина, і так далі для вищих розмірностей), яка найкращим чином розділяє дані на класи. Гіперплощина обирається таким чином, щоб максимізувати відстань (маржу) між найближчими до неї точками обох класів, які називаються опорними векторами.

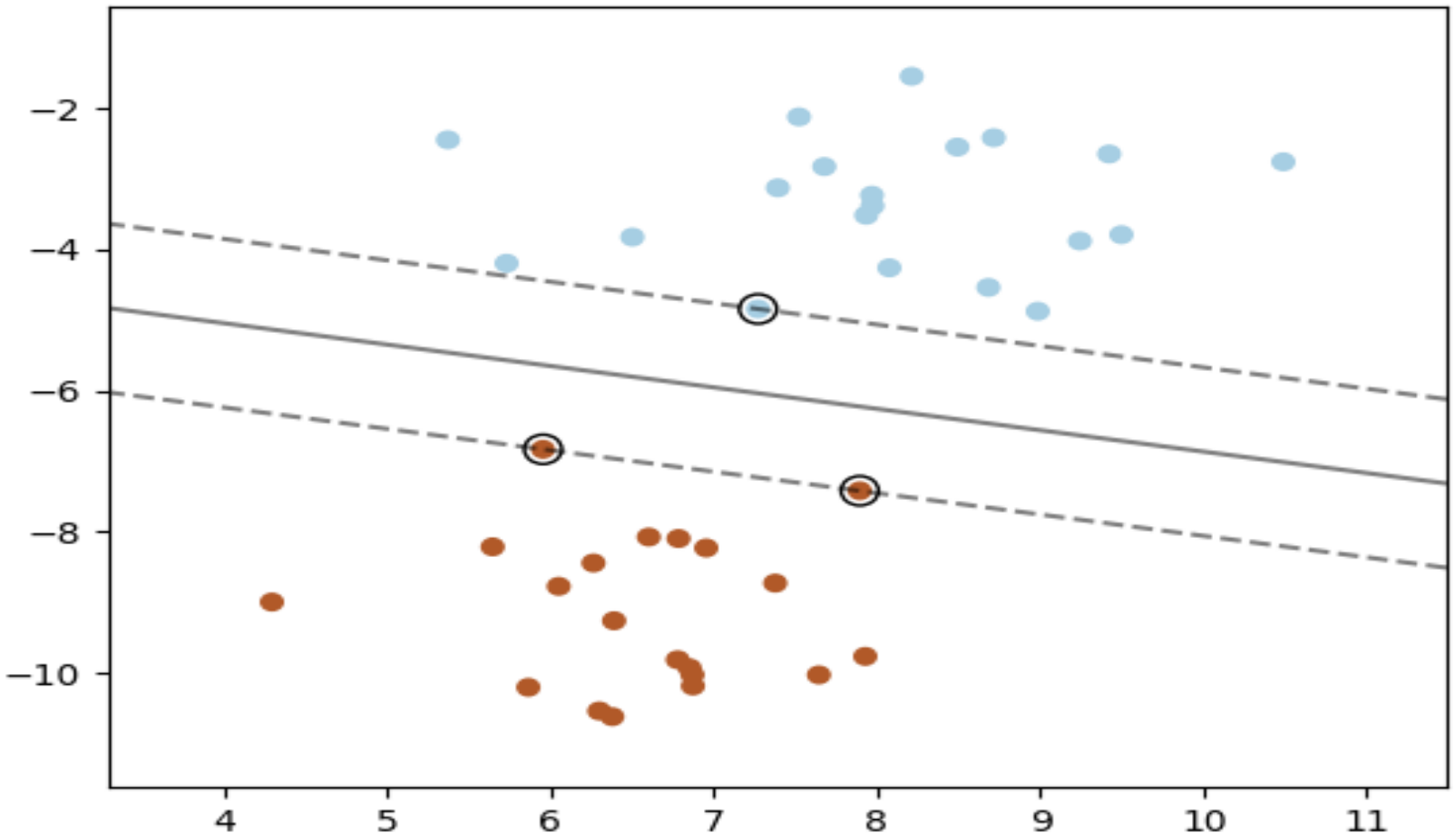
Метод опорних векторів SVM (Support Vector Machines)

Основні аспекти SVM:

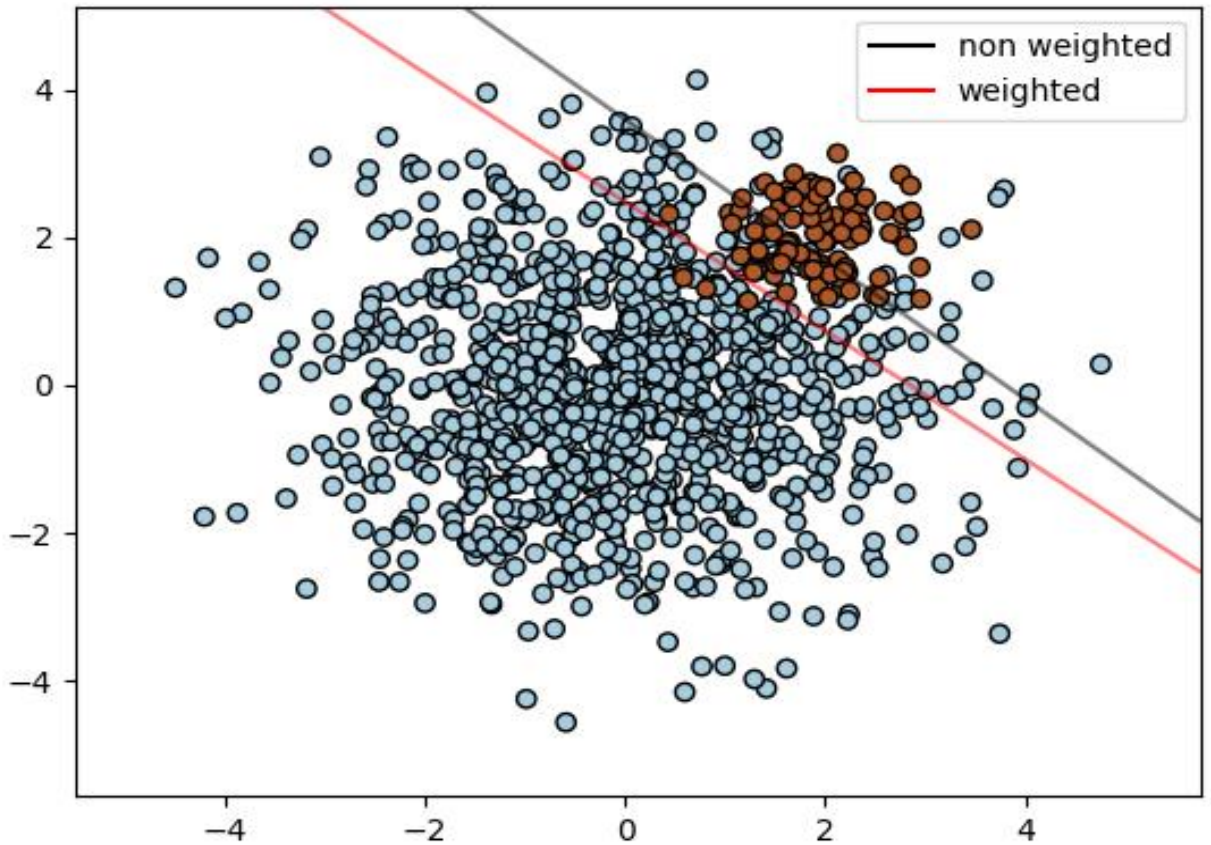
1. **Максимізація маржі**: SVM шукає гіперплощину з максимально можливою відстанню до найближчих точок обох класів, що забезпечує кращу узагальнюючу здатність моделі.
2. **Опорні вектори**: Це дані, які знаходяться найближче до гіперплощини. Вони мають вирішальне значення для визначення положення гіперплощини. Інші точки даних, які лежать далі від гіперплощини, не впливають на її положення.
3. **Ядра (Kernel tricks)**: Для випадків, коли дані не можна лінійно розділити у вихідному просторі, SVM використовує функції ядра для перетворення даних у вищий вимірний простір, де вони можуть бути лінійно розділені. Це дозволяє SVM ефективно працювати навіть із складними не лінійно роздільними даними.
4. **Регуляризація**: SVM містить параметри регуляризації, які контролюють баланс між досягненням високої точності на тренувальних даних і збереженням узагальнюючої здатності моделі на нових даних, запобігаючи перенавчанню.

SVM може використовуватися в різноманітних доменах, таких як розпізнавання образів, класифікація текстів, біоінформатика тощо, завдяки його гнучкості та ефективності

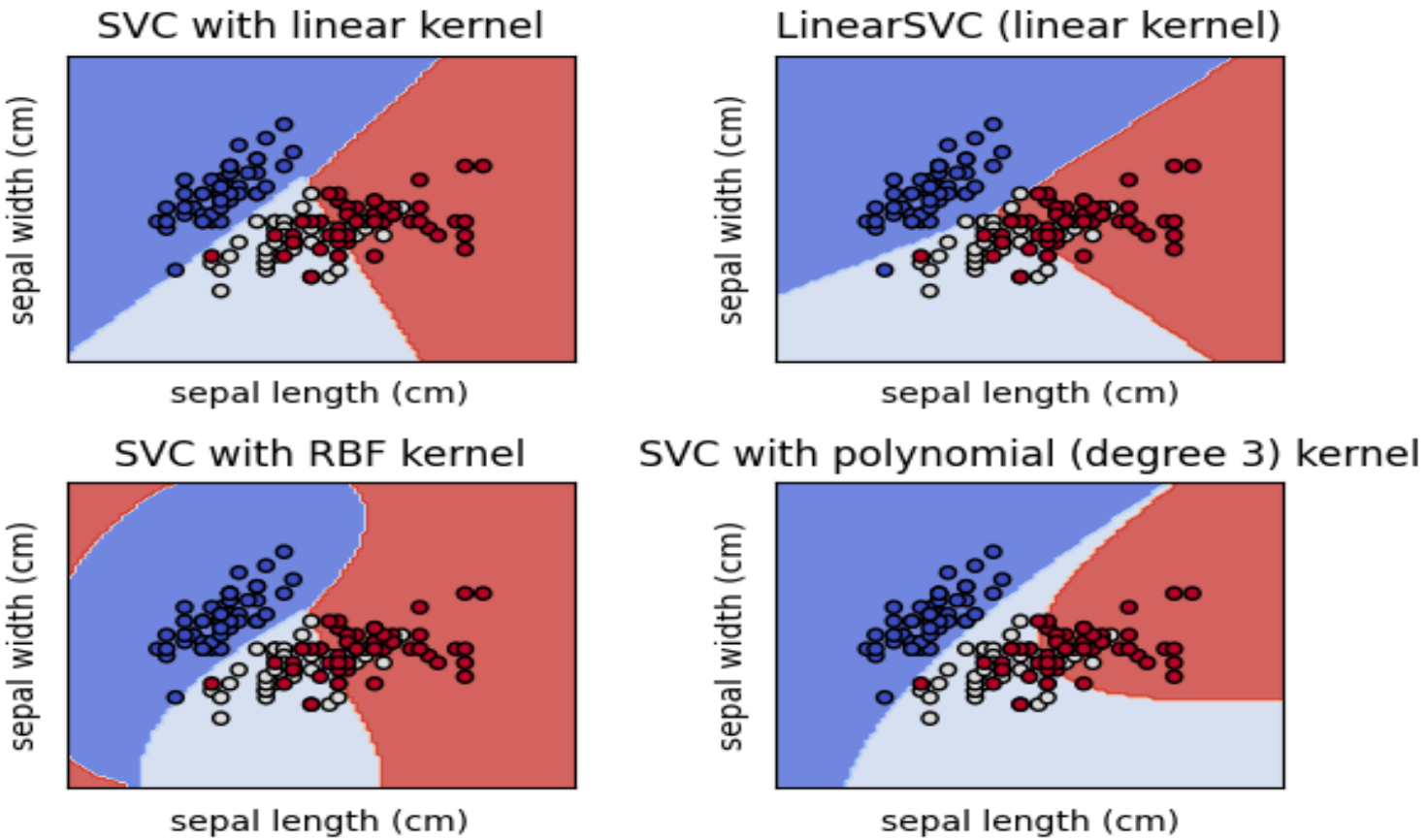
Метод опорних векторів SVM (Support Vector Machines)



Метод опорних векторів SVM (Support Vector Machines)

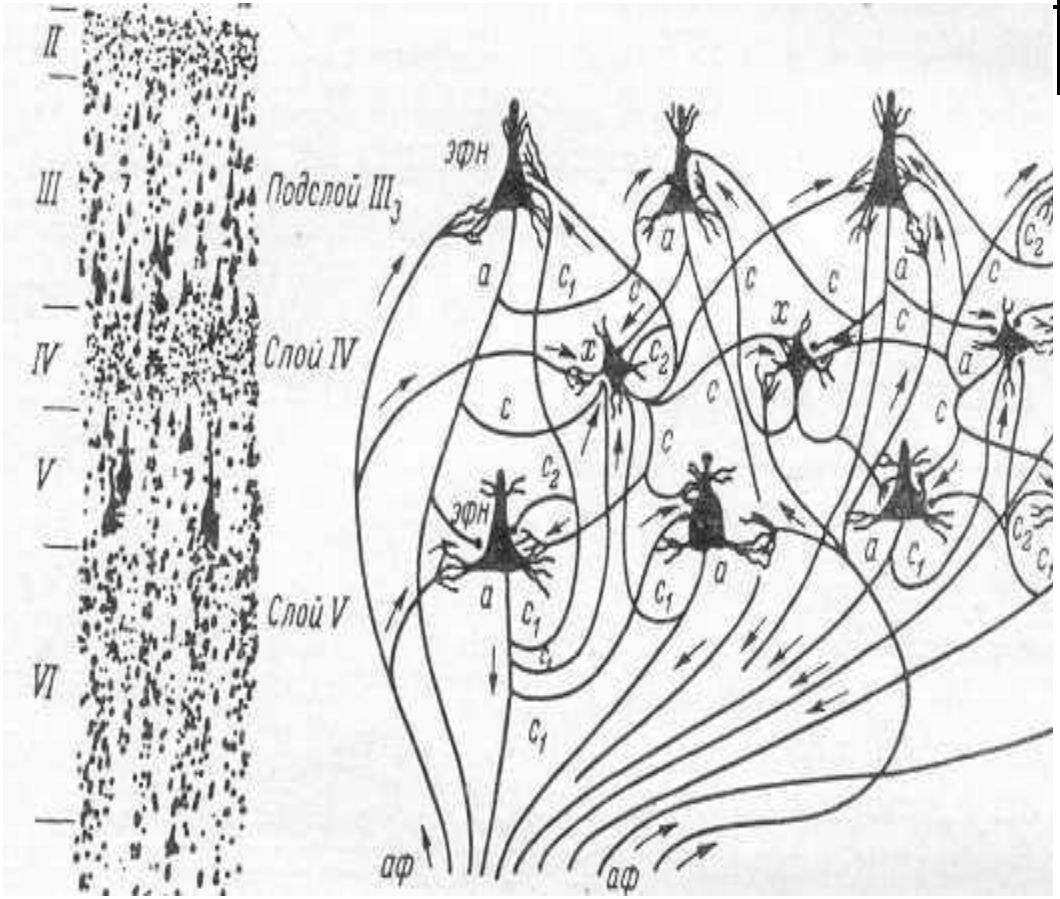


Метод опорних векторів SVM (Support Vector Machines)

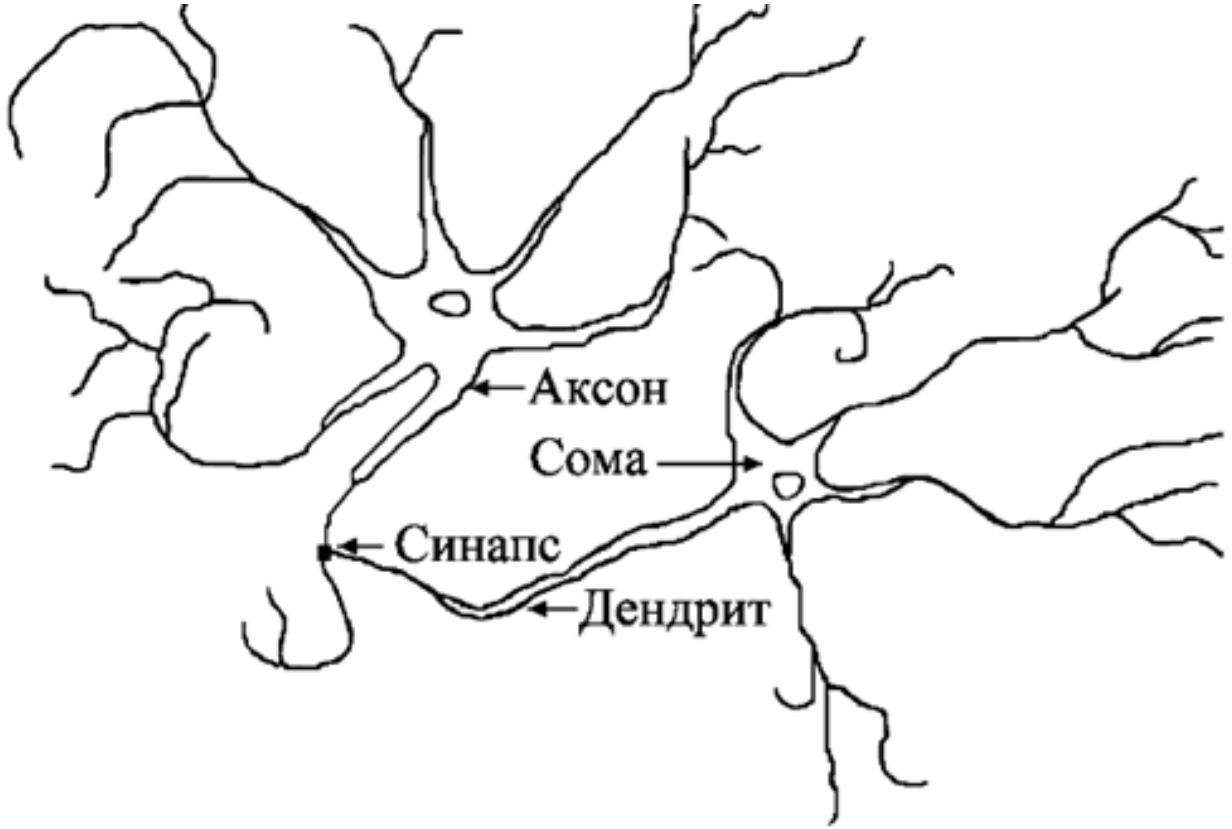


Штучні нейронні мережі

Фрагмент нервової тканини кори мозку



Біологічний прототип



Теорема Колмогорова-Арнольда

Теорема Колмогорова-Арнольда (про яку часто не підозрюють практики) служить математичною основою нейронних мереж.

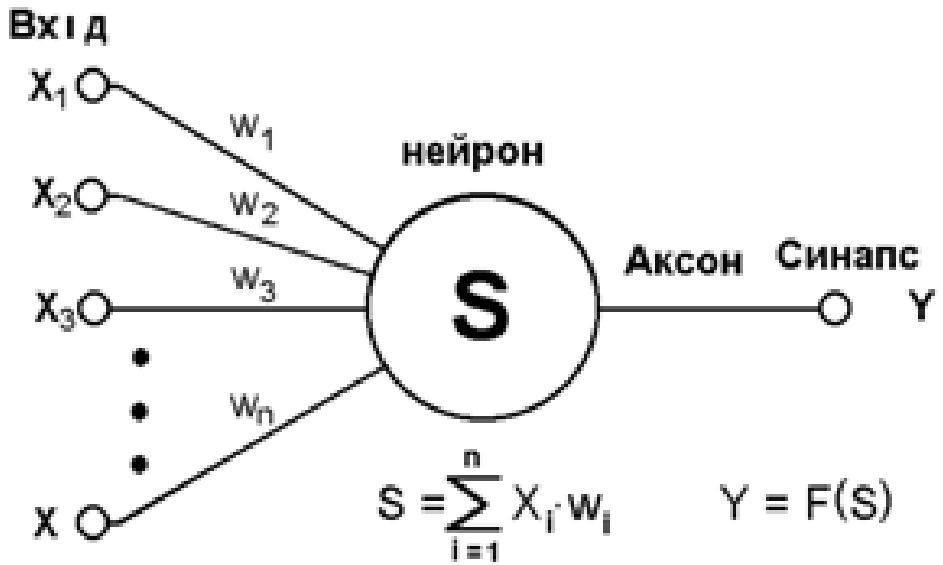
Теорема Колмогорова стверджує, що будь-яка безперервна функція *f,* визначена на n - вимірному одиничному кубі, може бути представлена у вигляді суми 2n+1 суперпозицій безперервних і монотонних відображень одиничних відрізків :

$$f(x_1,\ldots,x_n)=\sum_{q=1}^{2n+1} g_q\left(\sum_{p=1}^{n}\phi_{pq}(x_p)\right)$$

x = (x1, .., xn), 0 ≤ xi ≤ 1

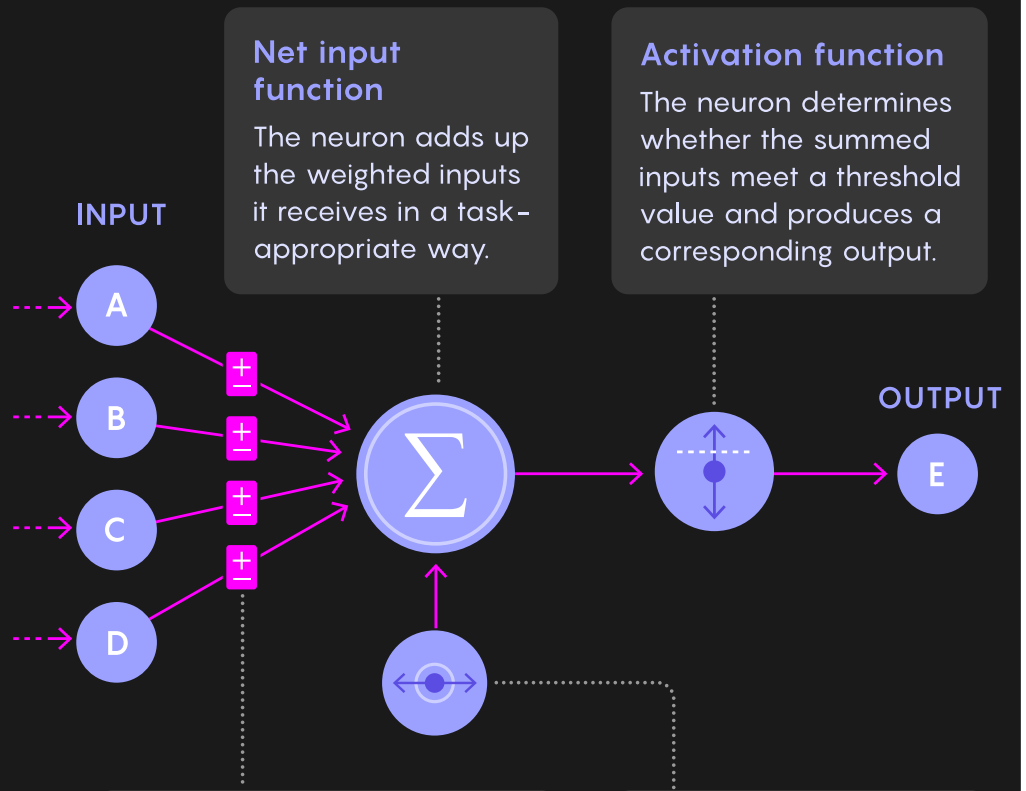
Ліворуч в цій формулі стоїть довільна неперервна функція, визначена на багатовимірному кубі, справа функції визначені на відрізках [0, 1].

Штучний нейрон



Simulating a Neuron

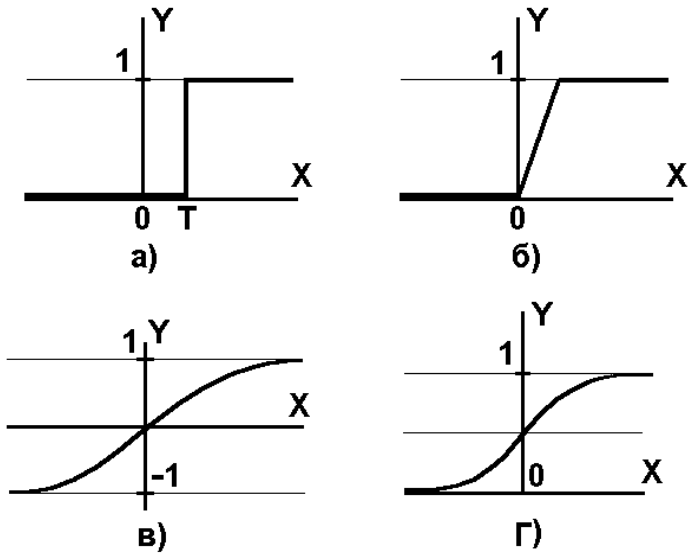
The interconnected devices in artificial neural networks conceptually resemble living neurons: Inputs from their "synapses" are weighted, tallied and translated into an output value.



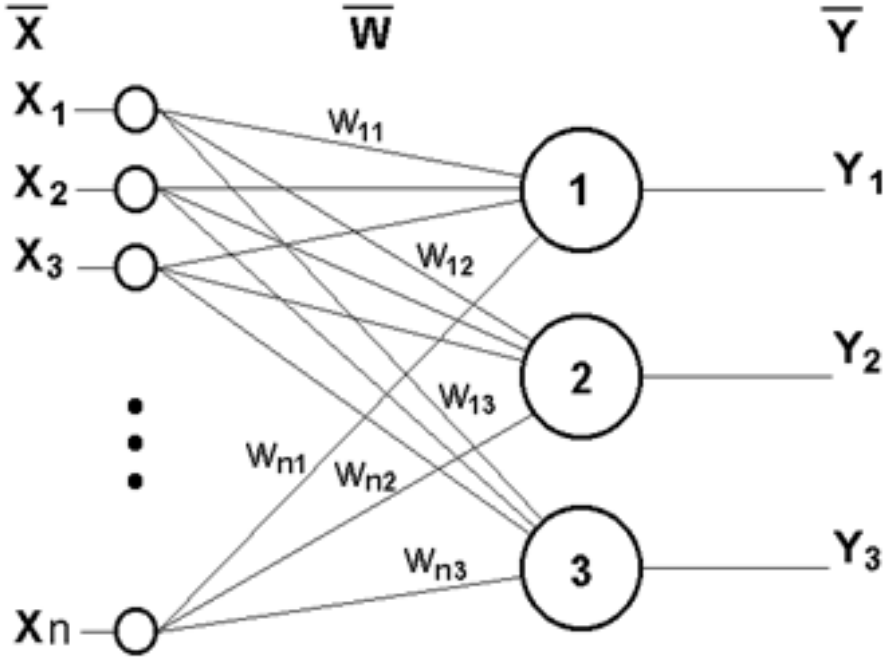
Штучний нейрон

$$S_p = \sum_{i=0}^{n} W_i \overline{X}_{pi}$$

$$F(S) = \frac{2}{1+e^{-aS}} - 1$$



Одношаровий персептрон



$$y_j = F\left[\sum_{i=1}^{n} x_i \cdot w_{ij}\right]$$

Лабораторна робота №4

Завдання: Побудувати бінарний класифікатор з допомогою моделі штучного нейрону.

Алгоритм лабораторної роботи №1

1.Обираємо задачу класифікації образів за неявними ознаками $\mathbf{A_j}$ та готуємо базу прикладів для навчання (навчальну вибірку) у вигляді таблиці:

Xpi	**A1**	**A2**	**…**	**An**	**dp**
P1	X_{11}	X_{12}		X_{1n}	$\mathbf{d_1}$
P2	X_{21}	X_{22}		X_{2n}	$\mathbf{d_2}$
…					
$\mathbf{P_M}$	X_{M1}	X_{M2}		X_{Mn}	$\mathbf{d_M}$

Де X_{pi} - числові значення ознаки **Aj** для прикладу $\mathbf{P_p}$,

d_p - ознака класу, $d_p = +1$ – I клас, $d_p = -1$ – II клас.

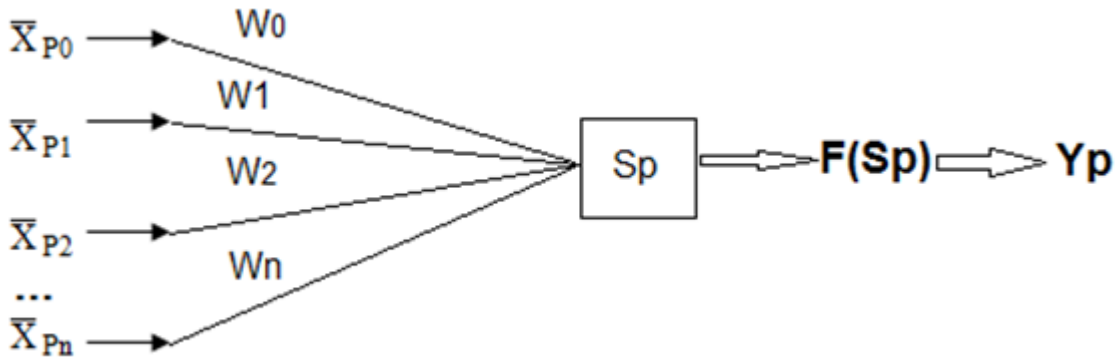
Число ознак повинно бути близько 5, а число прикладів більше 10, контрольна вибірка близько 5.

Лабораторна робота №4

2. Виконуємо природну нормалізацію ознак:

3. Ініціалізуємо нейрон:

$$\overline{X_{pi}} = \frac{X_{pi} - X_{\min i}}{X_{\max i} - X_{\min i}}$$



Лабораторна робота №4

Початкові значення вагових параметрів –невеликі, випадкові:

Wi = 0.01*rand(-1,+1).

Параметр ітерації: t=0.

Функцію активації приймаємо у вигляді:

$$F(S) = \frac{2}{1 + e^{-aS}} - 1$$

, де *a=2.*

Задаємо параметр швидкості навчання *η =0.9 і* точність навчання

ε =0.01.

p=1

Лабораторна робота №4

4. Основний цикл навчання за методом Відроу-Хебба:

a) Обираємо випадково новий приклад з навчальної вибірки з ознаками і обчислюємо зважену суму, вихідне значення і помилку:

$Yp=F(S_p)$,

$\Delta p=(Y_p - d_p)$

$$S_p = \sum_{i=0}^{n} W_i \overline{X}_{pi}$$

b) Обчислюємо вагові параметри для ітерації (t+1):

$$W_i(t+1) = W_i(t) - \eta \overline{X}_{pi} \Delta_p$$

c) Повторити b) для i=1,2,...,n.

d) Перейти до a) до вичерпання всіх прикладів навчальної вибірки.

e) Обчислюємо сумарну помилку:

$$E = \frac{1}{2} \sum_{p=1}^{M} \Delta_p$$

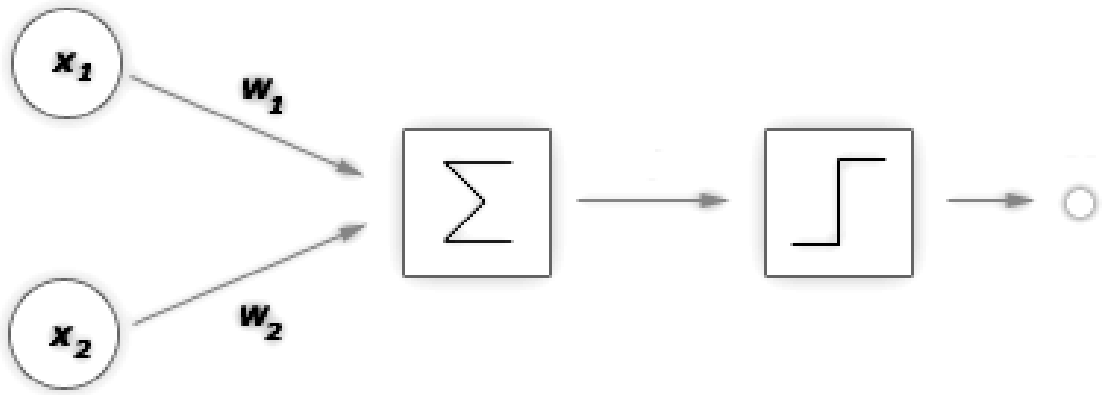
f) При умові E > ε^2 продовжуємо ітерації *t=t+1*, переходимо до a),
при умові E <= ε^2 - вихід.

Проблема лінійного розподілу в НМ

Часто, для того, щоб продемонструвати обмежені можливості одношарових персептронів при рішенні завдань удаються до розгляду так званої проблеми XOR - що виключає АБО.

Суть завдання полягають в наступному. Дана логічна функція XOR - що виключає АБО. Це функція від двох аргументів, кожен з яких може бути нулем або одиницею. Вона набуває значення, коли один з аргументів дорівнює одиниці, але не обоє, інакше . Проблему можна проілюструвати за допомогою одношарової одно нейронної системи з двома входами, показаної на малюнку нижче.

Проблема лінійного розподілу в НМ



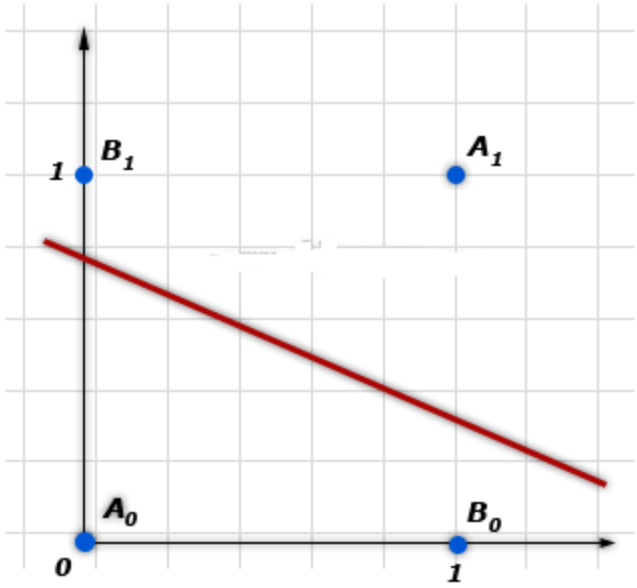
Проблема лінійного розподілу в НМ

Позначимо один вхід через X1, а інший через X2, тоді усі їх можливі комбінації складатимуться з чотирьох точок на площині.

Один нейрон з двома входами може сформувати вирішальну поверхню у вигляді довільної прямої:

$$X1*W1+X2*W2 =T$$

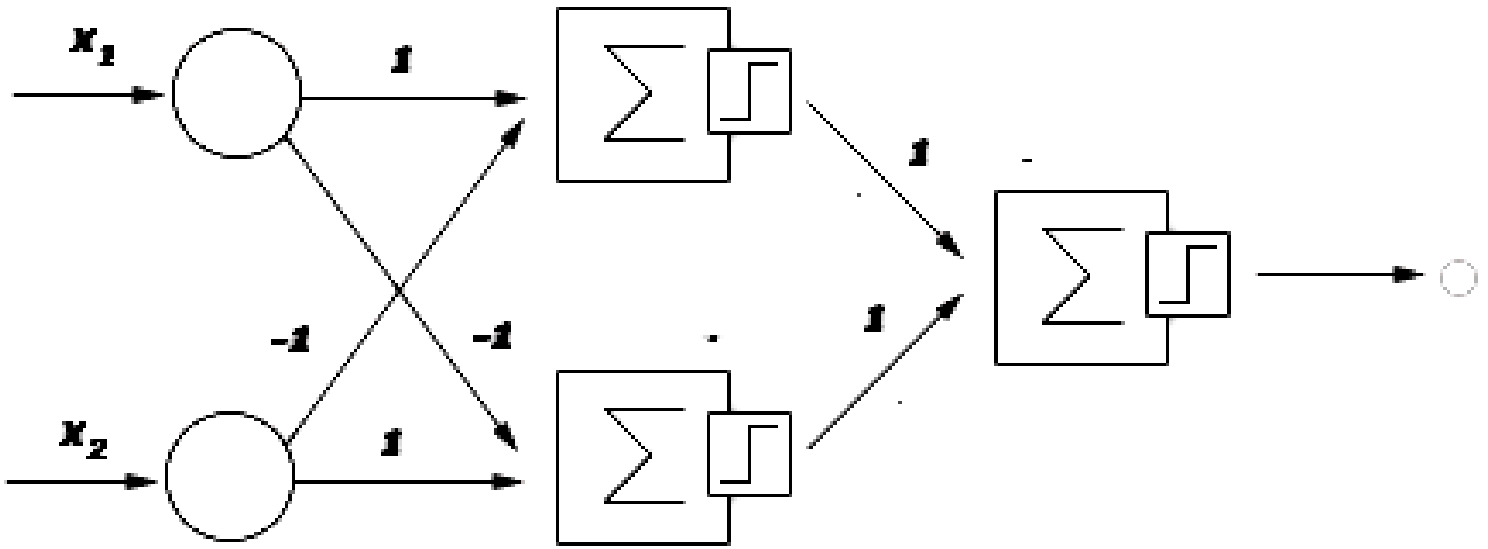
Проблема лінійного розподілу в НМ



Проблема лінійного розподілу в НМ

- Це означає, що які б значення не приписувалися вагам і порогу, одношарова нейронна мережа нездатна відтворити співвідношення між входом і виходом, потрібне для представлення функції XOR.

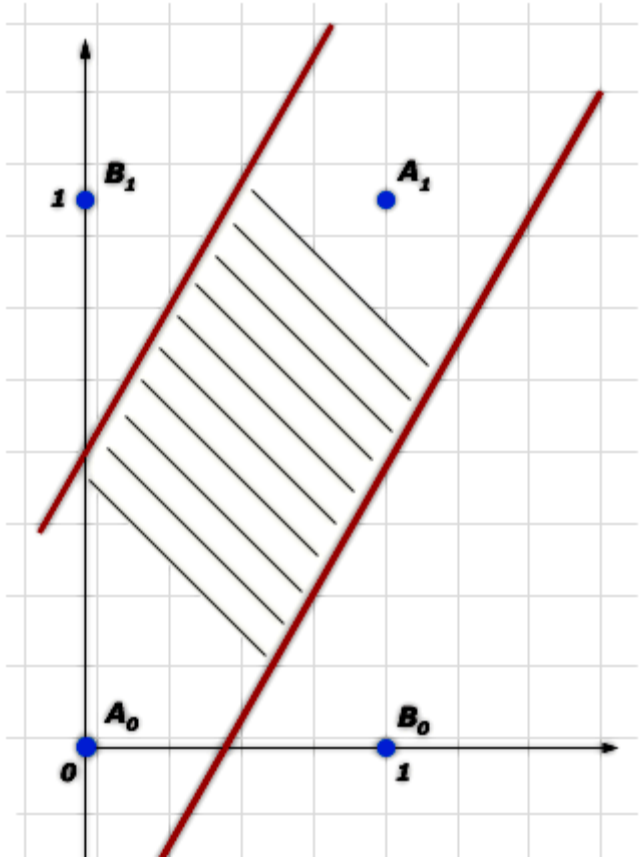
Проте функція XOR легко формується вже двошаровою мережею, причому багатьма способами. Розглянемо один з таких способів. Модернізуємо мережу, додавши ще один прихований шар нейронів


x_1
x_2
1
-1
-1
1
1
1

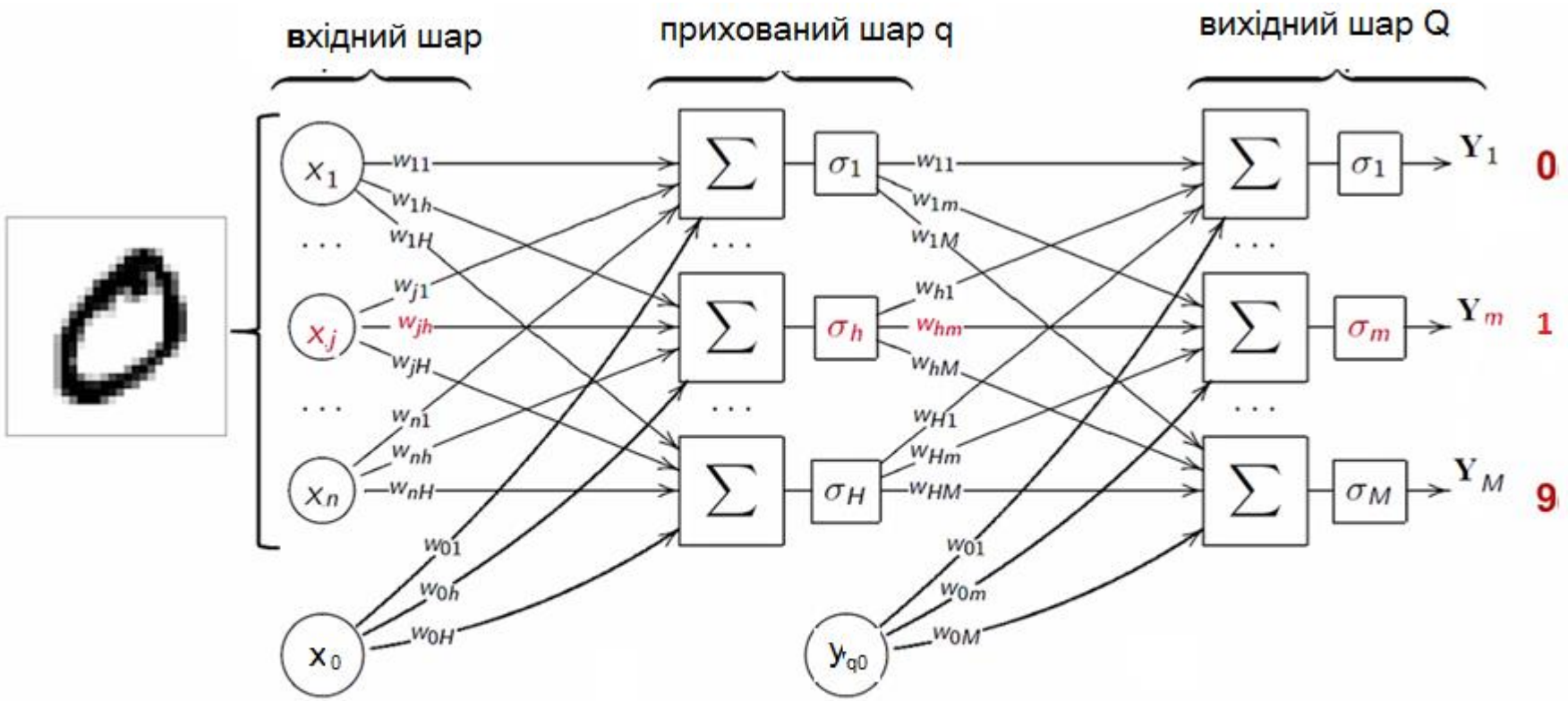
Проблема лінійного розподілу в НМ

- Кожен з двох нейрон першого шару формує вирішальну поверхню у вигляді довільної прямої (ділить площину на дві напівплощини), а нейрон вихідного шару об'єднує ці два рішення, утворюючи вирішальну поверхню у вигляді смуги, утвореної паралельними прямими нейронів першого шару,

Проблема лінійного розподілу в НМ



Багатошарова НМ



Градієнтний спуск

Цільова функція помилки

$$E(w_{ij}^{(q)}) = \frac{1}{2}\sum_{p}\sum_{j=1}^{M} (y_{j,p}^{(Q)} - d_{j,p})^2 \to \min$$

Мінімізуємо функцію помилки для кожного прикладу по черзі

Тоді $$E_p(w_{ij}^{(q)}) = \frac{1}{2}\sum_{j=1}^{M} (y_{j,p}^{(Q)} - d_{j,p})^2 = \frac{1}{2}\sum_{j=1}^{M} \Delta_{j,p}^2 \to \min$$

$$E(w_{ij}^{(q)}) = \sum_{p=1}^{P} E_p(w_{ij}^{(q)})$$

Градієнтний спуск

Згідно з методом градієнтного спуску - крок η

$$w_{ij}^{(q)}(t+1) = w_{ij}^{(q)}(t) + \Delta w_{ij}^{(q)} = w_{ij}^{(q)}(t) - \eta \cdot \frac{\partial E_p(w_{ij}^{(q)}(t))}{\partial w_{ij}^{(q)}}$$

Так як $$\Delta_{j,p} = (y_{j,p}^{(Q)} - d_{j,p})$$

То при *q=Q* $$\frac{\partial E_p}{\partial w_{ij}^{(Q)}} = \frac{\partial E_p}{\partial \Delta_{j,p}} \cdot \frac{\partial \Delta_{j,p}}{\partial y_{j,p}^{(Q)}} \cdot \frac{\partial y_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}}$$

$$\frac{\partial E_p}{\partial \Delta_{j,p}} = \Delta_{j,p}; \frac{\partial \Delta_{j,p}}{\partial y_{j,p}^{(Q)}} = 1;$$

Оскільки $$y_{j,p}^{(Q)} = F(\sum_{i=0}^{n_{Q-1}} w_{ij}^{(Q)} y_{i,p}^{(Q-1)}) = F(s_{j,p}^{(Q)})$$

Градієнтний спуск

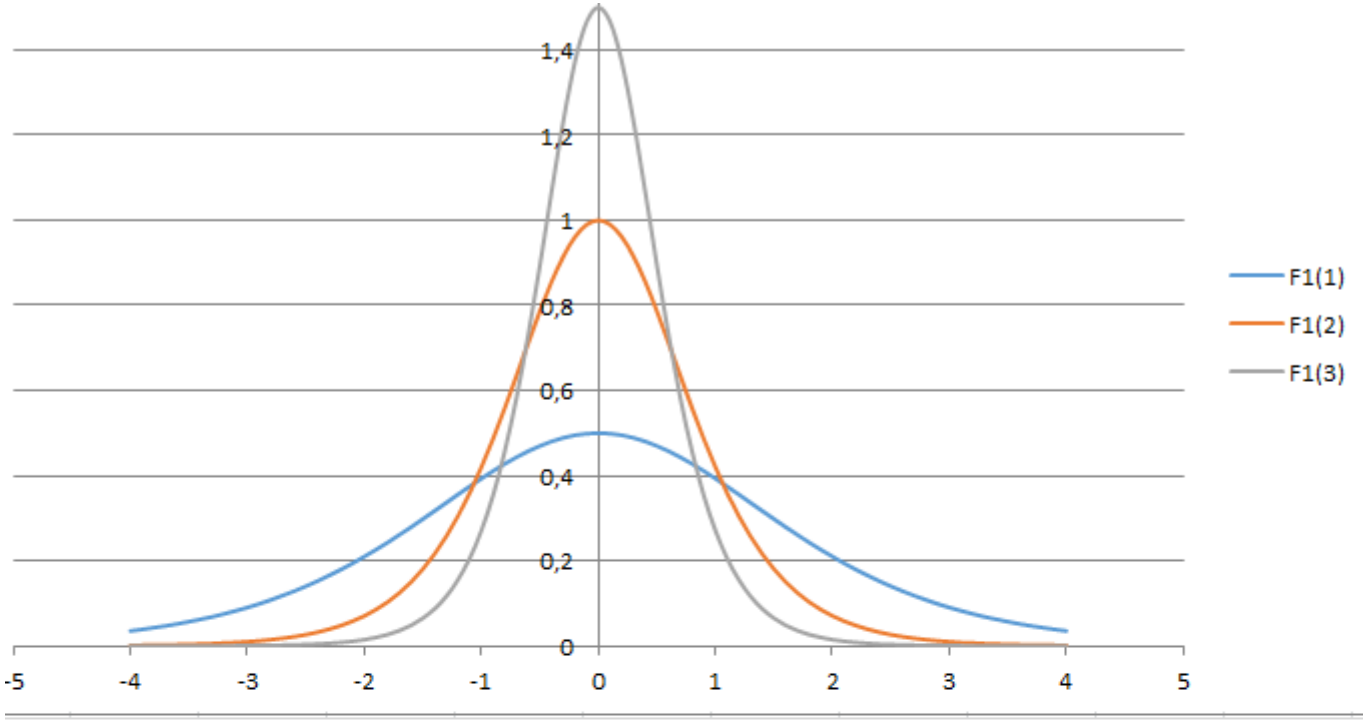
то

$$\frac{\partial y_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}} = F'(s_{j,p}^{(Q)}) \cdot \frac{\partial s_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}} = F'(s_{j,p}^{(Q)}) \cdot y_{i,p}^{(Q-1)}$$

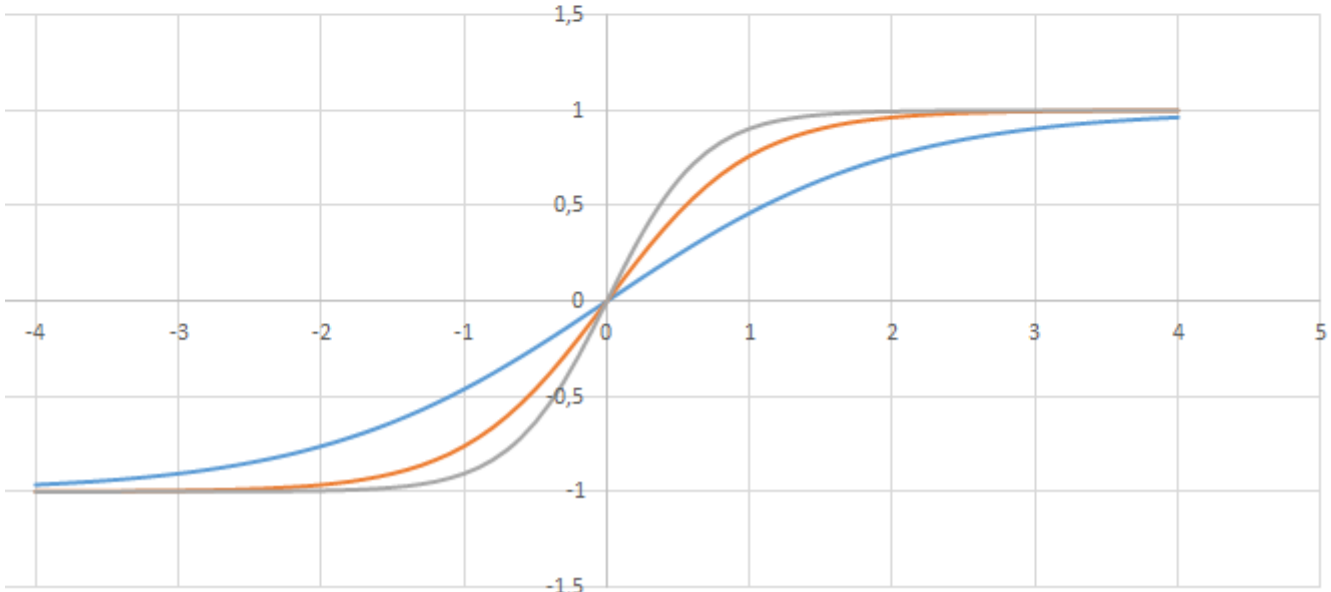
Оскільки множник $\frac{\partial y_{j,p}^{(Q)}}{\partial w_{ij}^{(Q)}}$ є похідною функції по її аргументу, з цього виходить, що похідна активаційної функція має бути визначена на усій осі абсцис, тому застосовуються такі функції, як гіперболічний тангенс або класичний сигмоїд з експонентою.

$$F(s) = \frac{2}{1+e^{-a \cdot s}} - 1 = th(\frac{a \cdot s}{2})$$

$$F'(s) = \frac{a}{2 \cdot ch^2(\frac{a \cdot s}{2})} = \frac{a}{2}(1 - th^2(\frac{a \cdot s}{2})) = \frac{a}{2}(1 - F^2(s))$$


1,4
1,2
1
0,8
0,6
0,4
0,2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
F1(1)
F1(2)
F1(3)

Сігмоїд


1,5
1
0,5
0
-0,5
-1
-1,5
-4
-3
-2
-1
0
1
2
3
4
5

Градієнтний спуск

При одношаровій мережі Q=1, $y^{(0)}=x_i$

$$\frac{\partial E_p}{\partial w_{ij}^{(Q)}} = \Delta_{j,p} \cdot F'(s_{j,p}^{(Q)}) \cdot x_{i,p}$$

Формула для уточнення $w_{ij}^{(Q)}$ матиме вигляд

$$w_{ij}^{(Q)}(t+1) = w_{ij}^{(Q)}(t) - \eta \cdot (\Delta_{j,p} \cdot F'(s_{j,p}^{(Q)})) \cdot x_{i,p}$$

І процес навчання буде повторювати процедуру Відроу-Хебба, тобто уточнення

Для багатошарових НМ невідомі помилки у проміжних шарах

Алгоритм Back propagation error

Запропонований Дэвидом И. Румельхартом, Дж. Е. Хинтоном и Рональдом Дж. Вильямсом в работе

***Rumelhart D.E., Hinton G.E., Williams R.J.*, Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing, vol. 1, pp. 318—362. Cambridge, MA, MIT Press. 1986.**

Згідно з методом найменших квадратів, цільовою функцією помилки НС, що мінімізується, є величина:

$$E(w) = \frac{1}{2}\sum_{j,p}(y_{j,p}^{(Q)} - d_{j,p})^2 \qquad (2.1)$$

де $y_{j,p}^{(N)}$ - реальний вихідний стан нейрона j вихідного шару N нейронної мережі при подачі на її входи p -го образу;

d_{jp} - ідеальний (бажане) вихідний стан цього нейрона.

Підсумовування ведеться по усіх нейронах вихідного шару і по усіх оброблюваних мережею образах.

Алгоритм Back propagation error

Мінімізація ведеться методом градієнтного спуску, що означає підстроювання вагових коефіцієнтів таким чином:

$$w_{ij}^{(q)}(t+1) = w_{ij}^{(q)}(t) + \Delta w_{ij}^{(q)} = w_{ij}^{(q)}(t) - \eta \cdot \frac{\partial E_p}{\partial w_{ij}^{(q)}} \qquad (2)$$

Тут $\boldsymbol{w_{ij}}$ - ваговий коефіцієнт синаптичного зв'язку, що сполучає і –й нейрон шару n – 1 з j –м нейроном шару n,

η - коефіцієнт швидкості навчання, 0< η ≤1,

Очевидно що,

$$\frac{\partial E}{\partial w_j^{(q)}} = \frac{\partial E}{\partial y_j^{(q)}} \cdot \frac{dy_j^{(q)}}{ds_j^{(q)}} \cdot \frac{\partial s_j^{(q)}}{\partial w_j^{(q)}} \qquad (3)$$

Тут під y_j, як і раніше, мається на увазі вихід нейрона j, а під s_j – зважена сума його вхідних сигналів, тобто аргумент активаційної функції.

$$\frac{dy}{ds} = F'(s) = \frac{1}{2}(1 - y^2) \qquad (4)$$

Алгоритм Back propagation error

Третій множник ds_j/dw_{ij}), очевидно, дорівнює виходу нейрона попереднього шару: $y_i^{(q-1)}$.

$$s_j^{(q)} = \sum_{i=0}^{n_q} y_i^{(q-1)} \cdot w_{ij}^{(q)} \qquad s_j^{(q+1)} = \sum_{i=0}^{n_q} y_i^{(q)} \cdot w_{ij}^{(q+1)}$$

Що стосується першого множника в (3), він легко розкладається таким чином:

$$\frac{\partial E}{\partial y_j^{(q)}} = \sum_{i=0}^{n_{q+1}} \frac{\partial E}{\partial y_i^{(q+1)}} \cdot \frac{dy_i^{(q+1)}}{ds_i^{(q+1)}} \cdot \frac{\partial s_i^{(q+1)}}{\partial y_i^{(q+1)}} = \sum_{i=0}^{n_{q+1}} \left(\frac{\partial E}{\partial y_i^{(q+1)}} \cdot \frac{dy_i^{(q+1)}}{ds_i^{(q+1)}}\right) \cdot w_{ji}^{(q+1)} \tag{5}$$

Тут підсумовування по і виконується серед нейронів шару *n+1*.

Ввівши нову змінну

$$\delta_j^{(q)} = \frac{\partial E}{\partial y_j^{(q)}} \cdot \frac{dy_j^{(q)}}{ds_j^{(q)}} \tag{6}$$

отримаємо рекурсивну формулу для розрахунків величин (j(n) шару n з величин (k(n+1) більше старшого шару n+1.

$$\delta_j^{(q)} = \left[\sum_{i=0}^{n_{q+1}} \delta_k^{(q+1)} \cdot w_{jk}^{(q+1)}\right] \cdot F'(s_j^{(q)}) \tag{7}$$

Для вихідного ж шару

$$\delta_j^{(Q)} = (y_j^{(Q)} - d_j) \cdot F'(s_j^{(Q)})$$

Алгоритм Back propagation error

(8)

Тепер ми можемо записати (2) в розкритому виді:

$$\delta_j^{(q)} = \left[\sum_{i=0}^{n_{q+1}} \delta_k^{(q+1)} \cdot w_{jk}^{(n+1)} \right] \cdot F'(s_j^{(q)}) \qquad (9)$$

Іноді для надання процесу корекції ваг деякої інерційності, що згладжує різкі скачки при переміщенні по поверхні цільової функції, (9) доповнюється значенням зміни ваги на попередній ітерації

$$\Delta w_{ij}^{(q)} = -\eta \cdot \delta_j^{(q)} \cdot y_i^{(q-1)},$$

$$\Delta w_{ij}^{(q)}(t+1) = -\eta \cdot (\mu \cdot \Delta w_{ij}^{(q)}(t) + (1-\mu) \cdot \delta_j^{(q)} \cdot y_i^{(q-1)}) \qquad (10)$$

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

Реалізація Backpropogation ERROR

Алгоритм навчання багатошарової НМ за допомогою процедури зворотного поширення помилки :

0. Зформувати навчальну вибірку {$\mathbf{x}_p$, $\mathbf{d}_p$}, p=1…P

ініціалізувати початкові значення вагових параметрів

$$w_{ij} = 0.01*rand(-1,+1).$$

1. Подати на входи мережі один прикладів навчальної вибірки і в режимі звичайного функціонування НМ, коли сигнали поширюються від входів до виходів, розрахувати значення останніх. Нагадаємо, що

$$s_j^{(q)} = \sum_{i=0}^{n_{q-1}} y_i^{(q-1)} \cdot w_{ij}^{(q)}$$

де n_{q-1} - число нейронів в шарі q-1 з урахуванням нейрона з постійним вхідним станом +1, що задає зміщення;

$$y_j^{(q)} = F(s_j^{(q)}),$$

де F() – функція активації

$$y_i^{(0)} = x_i$$

де x_i - і -та компонента вектору вхідного образу.

2. Розрахувати для вихідного шару по формулі $\delta_j^{(Q)} = (y_j^{(Q)} - d_j) \cdot F'(s_j^{(Q)})$

Реалізація Backpropogation

3. Розрахувати по формулах відповідні для усіх інших шарів,

q=(Q – 1),..1.

$$\delta_j^{(q)} = \left[\sum_{i=0}^{n_{q+1}} \delta_k^{(q+1)} \cdot w_{jk}^{(q+1)} \right] \cdot F'(s_j^{(q)})$$

4. Розрахувати по формулі

$$\Delta w_{ij}^{(q)} = -\eta \cdot \delta_j^{(q)} \cdot y_i^{(q-1)}$$

зміни ваг шару q.

5. Скоригувати усі ваги в НМ

$$w_{ij}^{(q)}(t) = w_{ij}^{(q)}(t-1) + \Delta w_{ij}^{(q)}(t)$$

6. Перейти на крок 1 до вичерпання всіх прикладів навчальної вибірки.

7. Якщо функція помилки мережі E для всіх M прикладів навчальної вибірки істотна, перейти на крок 1. Інакше - кінець.

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{n_Q} \Delta_{jp}^2$$

де $\Delta_{jp} = (y_j^{(Q)} - d_j)$

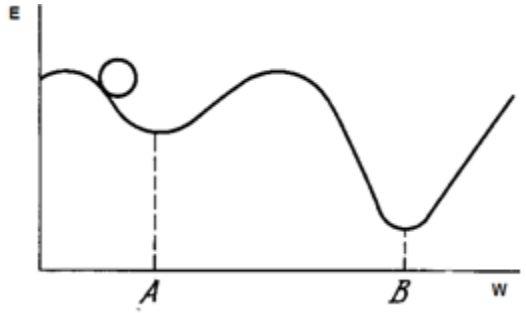
Мережі на кроці 1 поперемінно у випадковому порядку пред'являються усі тренувальні приклади.

Налаштування НМ для вирішення задач

- Локальні мінімуми.
- Параліч НМ
- Розмір кроку.
- Тимчасова нестійкість.
- Попередня обробка вхідних даних.
- Згасання помилки у алгоритмі Backprop.

Локальні мінімуми

BackProp використовує різновид градієнтного спуску, тобто здійснює спуск вниз по поверхні помилки, безперервно налаштовуючи ваги у напрямі до мінімуму. Поверхня помилки складної мережі дуже порізана і складається з пагорбів, долин, складок і ярів в просторі високої розмірності.



Мережа може потрапити в локальний мінімум (неглибоку долину), коли поруч є набагато глибший мінімум. У точці локального мінімуму усі напрями ведуть вгору, і мережа нездатна з нього вибратися.

Статистичні методи навчання можуть допомогти уникнути цієї пастки, але вони повільні. Застосовується також метод «струшування», тобто зміна деяких вагових параметрів для виходу з локального мінімуму і продовження процесу навчання.

Параліч мережі

В процесі навчання мережі значення ваг можуть в результаті корекції стати дуже великими величинами. Це може привести до того, що усі або більшість нейронів функціонуватимуть при дуже великих значеннях S, в області, де похідна функції дуже мала. Оскільки помилка пропорційна цій похідній, то процес навчання може практично завмерти.

Різні евристики використовувалися для оберігання від паралічу або для відновлення після нього, наприклад тимчасове зменшення параметра сигмоїду **a**, оскільки причиною паралічу НМ часто є нульове значення похідної від активаційної функції $F'(S) = \boldsymbol{\alpha}(1-F^2(S))$ для великих значень S і це зупиняє процес навчання. Після відновлення навчання значення параметра $\boldsymbol{\alpha}$ повертається.

Розмір кроку

Уважний розбір збіжності показує, що корекції ваг передбачаються нескінченно малими. Ясно, що це нездійсненно на практиці, оскільки веде до нескінченного часу навчання.

Розмір кроку **η** повинен братися скінченним, і в цьому питанні доводиться спиратися тільки на досвід.

Якщо розмір кроку **η** дуже малий, то збіжність занадто повільна, якщо ж дуже великий, то може виникнути параліч або постійна нестійкість.

Автоматичне налаштування кроку може здійснюватись по формулі: $\eta = \eta_0/(1+b*t)$

t - номер епохи навчання,
η_0 і **b –** обираються експериментально.

Тимчасова нестійкість

Якщо мережа вчиться розпізнавати букви, то немає сенсу учити "Б", якщо при цьому забувається "А". Процес навчання має бути таким, щоб мережа навчалася на усій навчальній множині без пропусків того, що вже вивчене.

Необхідні зміни ваг повинні обчислюватися **на усій множині прикладів,** а це вимагає додаткової пам'яті; після ряду таких навчальних циклів вони зійдуться до мінімальної помилки.

Цей метод може виявитися даремним, якщо мережа знаходиться в зовнішньому середовищі, що постійно міняється, так що другий раз один і той же вектор може вже не повторитися.

В цьому випадку процес навчання може ніколи не зійтися, безцільно блукаючи або сильно осцилюючи. У цьому сенсі зворотне поширення не схоже на біологічні системи.

Попередня обробка вхідних даних.

- Процес навчання можна суттєво покращити попередньою обробкою навчальної вибірки.
- Приклади для кожного класу повинні бути типовими проте різними. При майже однакових прикладах спостерігається синдром «перенавчання», коли нейронна мережа у процесі функціонування добре розпізнає образи близькі до прикладів і не розпізнають ся віддалені образи.
- Разом з тим, дуже «зашумлені» приклади у навчальній вибірці не дозволяють навчити НС взагалі.
- Суттєво може прискорити навчання попередня нормалізація навчальної вибірки, наприклад методом природної чи стандартної нормалізації.

Функції активації

- Логістичний сигмоїд $F(s)=\frac{1}{1+e^{-a\cdot s}}$ $F'(s)=a\cdot F(s)\cdot(1-F^2(s))$
- Гіперболічний тангенс $F(s)=th(as)$ $F'(s)=a(1-F^2(s))$
- SoftSign $F(s)=\frac{a\cdot s}{1+|a\cdot s|}$ $F'(s)=\frac{1}{(1+|a\cdot s|)^2}$
- SoftPlus $F(s)=\ln(1+e^{a\cdot s})$ $F'(s)=\frac{a}{1+e^{-a\cdot s}}$
- ReLU(rectified linear units) $F(s)=\max(0,s)$

$F'(s)=\max(0,1)$

Згасання помилки у алгоритмі BackProp.

Спостерігається згасання помилки при зворотному розповсюдженні, що зупиняє процес навчання.

Виходом можуть бути застосування інших методів навчання, зокрема, чисельних методів оптимізації чи обмеженої машини Больцмана.

Лабораторна робота № 6

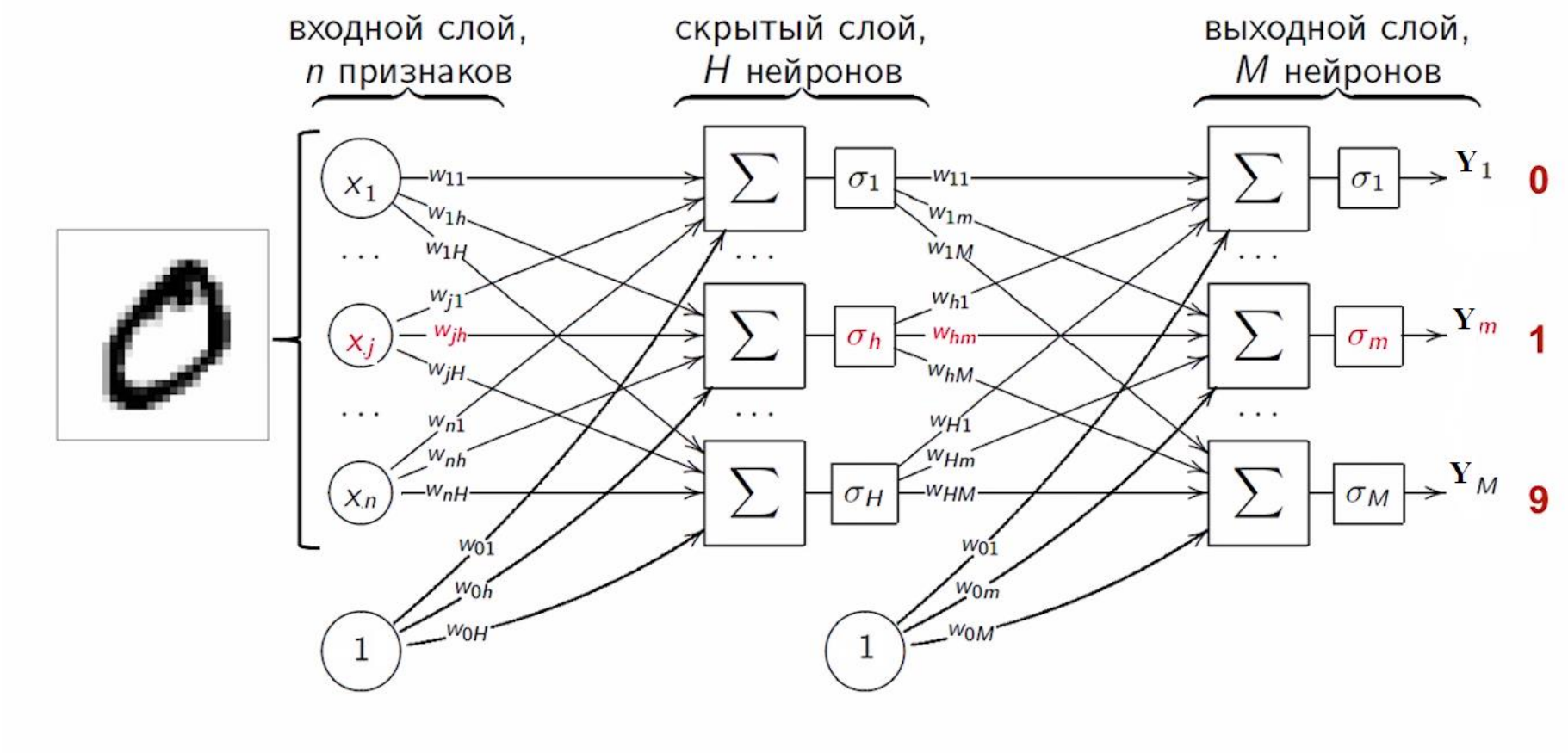
Створити нейро комп’ютерну систему розпізнавання рукописних символів, використавши алгоритм зворотного поширення помилки для навчання багатошарової НМ.

Алгоритм лабораторної роботи №3

1. Створити графічний інтерфейс для введення прикладів і розпізнавання символів.

2. Створити навчальну вибірку рукописних символів або використати стандартний набір MNIST з Internet

Лабораторна робота № 6



Лабораторна робота №6

3.Спроектувати нейронну мережу, задаючись на вході достатньою кількістю рецепторів для відображення символів через мега пікселі двовимірної сітки, а на виході НС достатньою кількістю нейронів для кодування символів (кількість різних символів <=2^m,

де m – кількість нейронів на виході).

4.Кількість нейронів у проміжних шарах визначити експериментально.

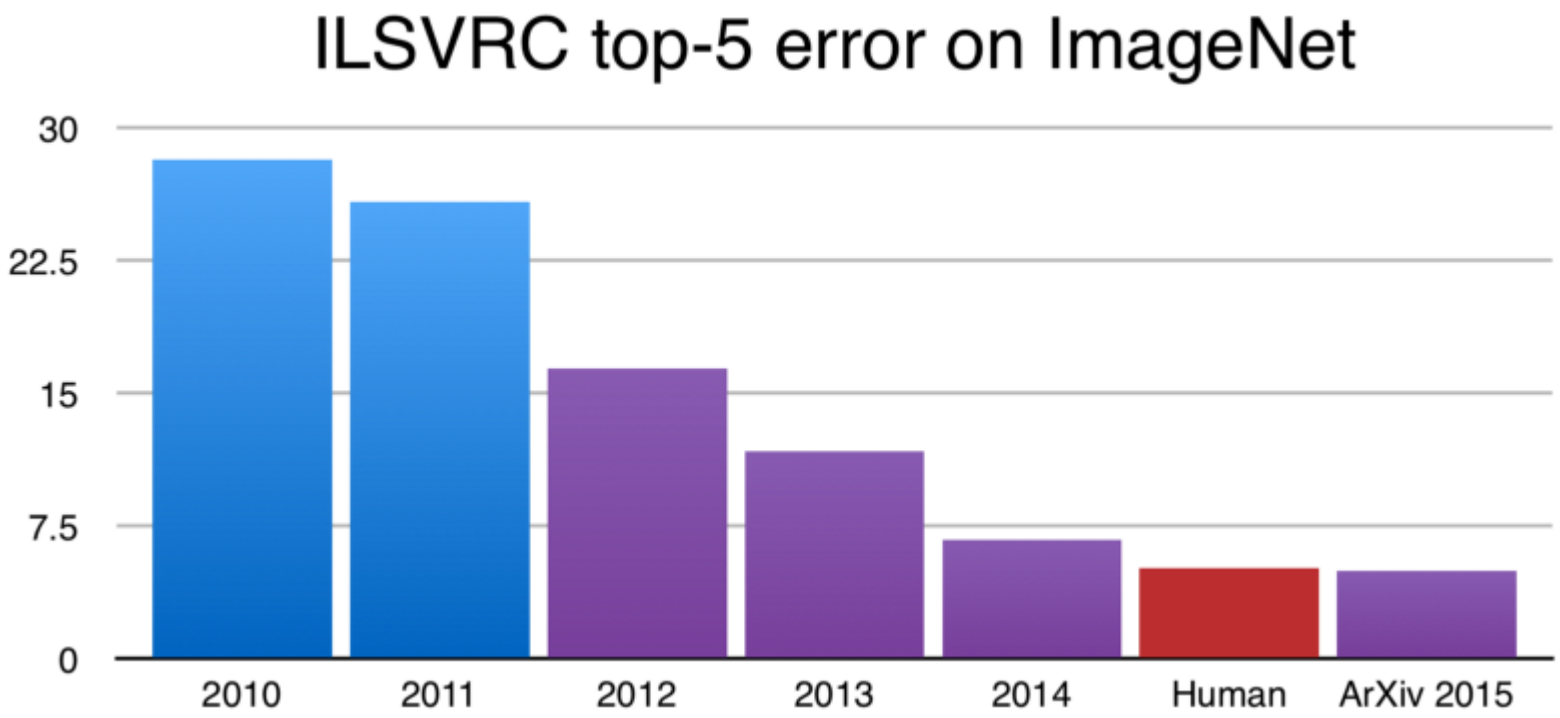
5.Навчити НМ згідно з алгоритмом BackProp.

6. Протестувати роботу НС на контрольній частині навчальної вибірки і при необхідності донавчити НС.

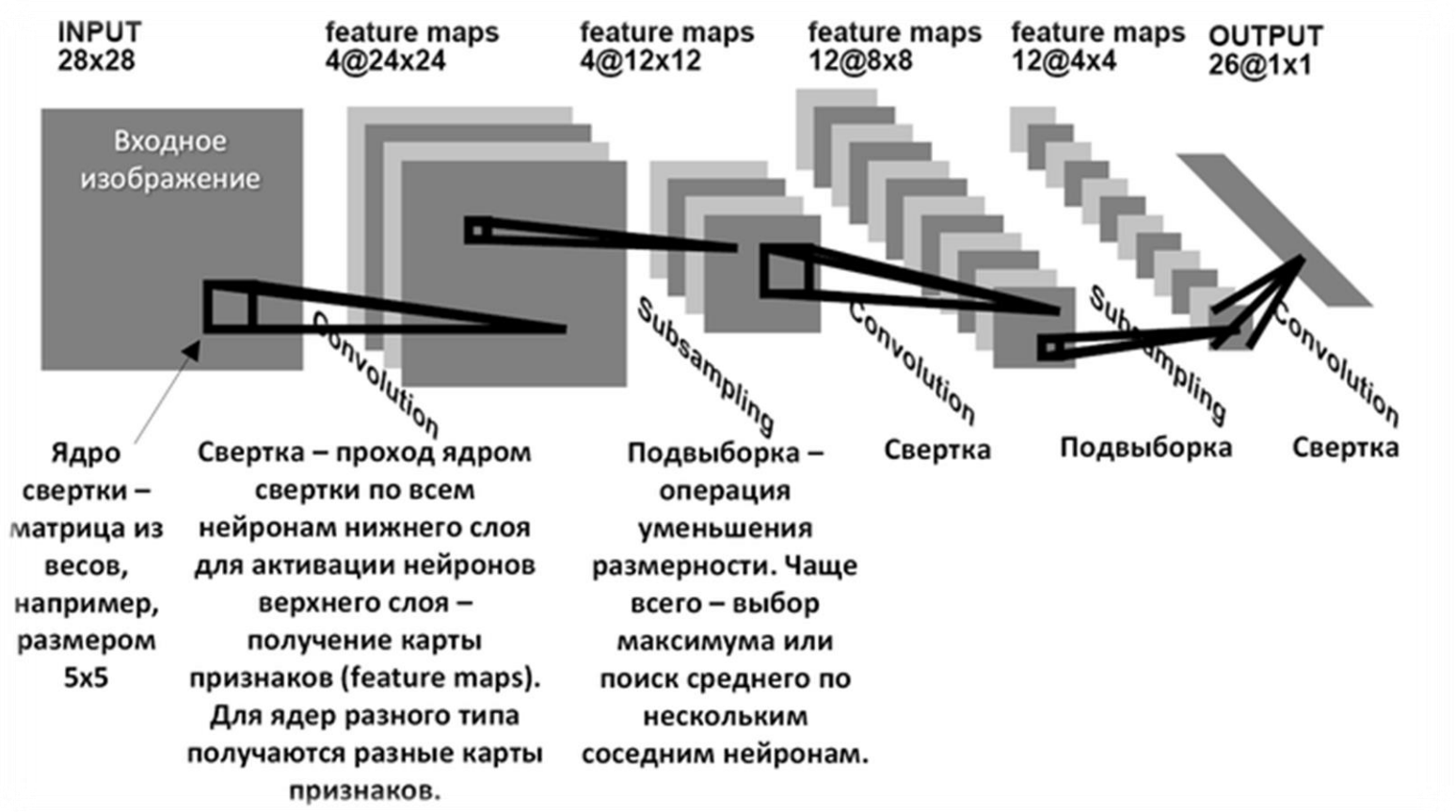
Згорткові нейронні мережі
Convolution Neural Network

Yann LeCun's http://yann.lecun.com/

Professor The Courant Institute of Mathematical Sciences
New York University

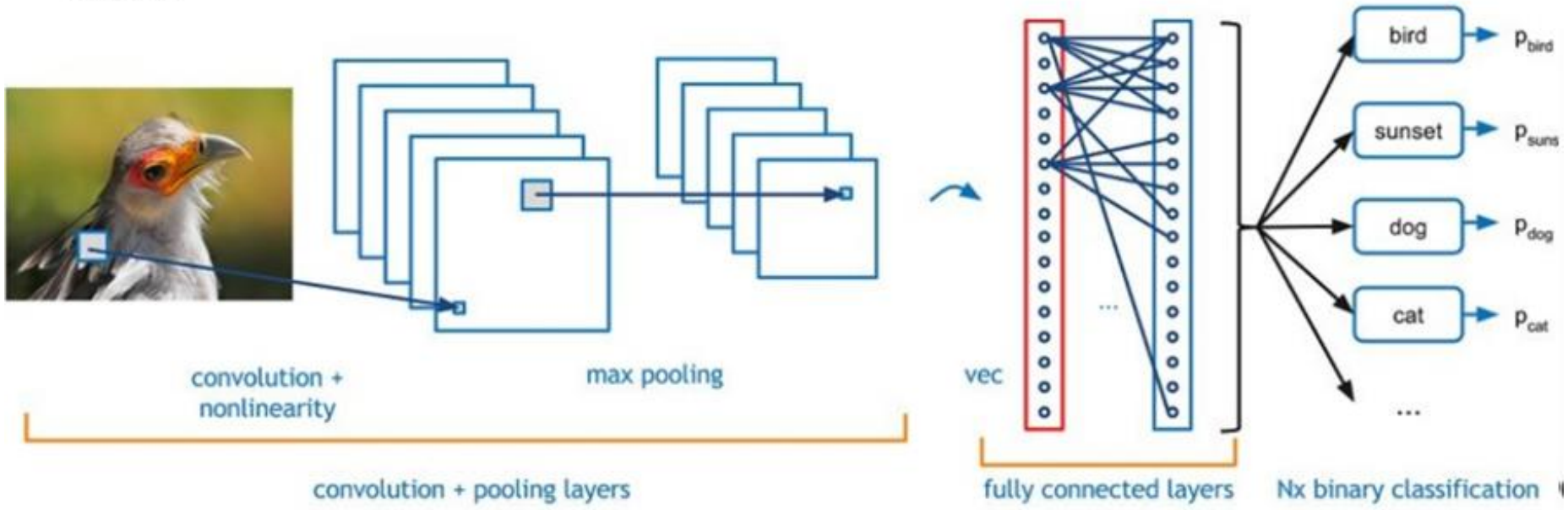

ILSVRC top-5 error on ImageNet
30
22.5
15
7.5
0
2010
2011
2012
2013
2014
Human
ArXiv 2015

Згорткові нейронні мережі



Згорткова нейронна мережа

Свёрточная нейросеть (CNN) — это Feed-Forward сеть специального вида:



У загорткових НМ менше параметрів

CNN

- вход картинка 100x100
- два свёрточных слоя по 100 плоскостей каждый (conv 5x5 и subsampling 2)
- полносвязный слой на 100 нейронов
- выход: 10 классов
- число параметров примерно **70K** (25*100 + 25*100 + 25*25*100 + 100*10)

FNN

- вход: картинка 100x100
- три скрытых слоя по 100 нейронов
- выход: 10 классов
- число параметров примерно **1M** (10000*100 + 100*100 + 100*100 + 100*10)

Згорткові нейронні мережі

Згорткові нейронні мережі працюють на основі фільтрів, які займаються розпізнаванням певних характеристик зображення (наприклад, прямих ліній).

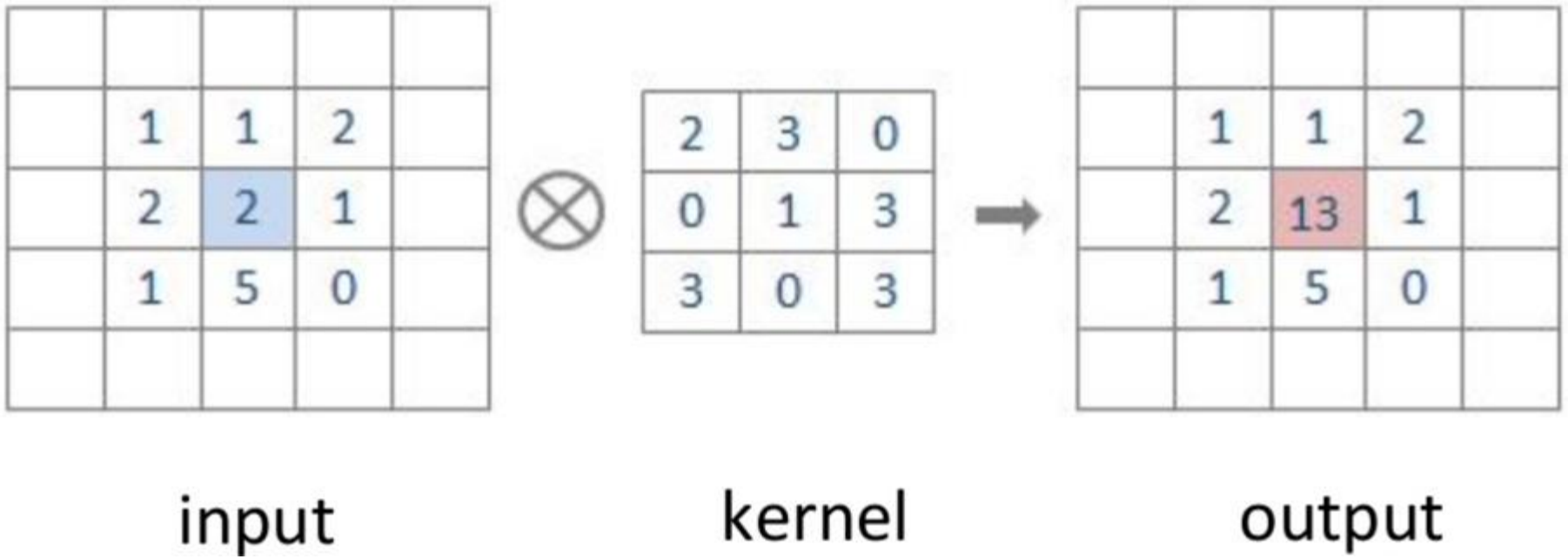
Фільтр — це колекція кернелов; іноді у фільтрі використовується один кернел.

Кернел — це звичайна матриця чисел, званих вагами, які "навчаються" (підлаштовуються, якщо вам так зручніше) з метою пошуку на зображеннях певних характеристик.

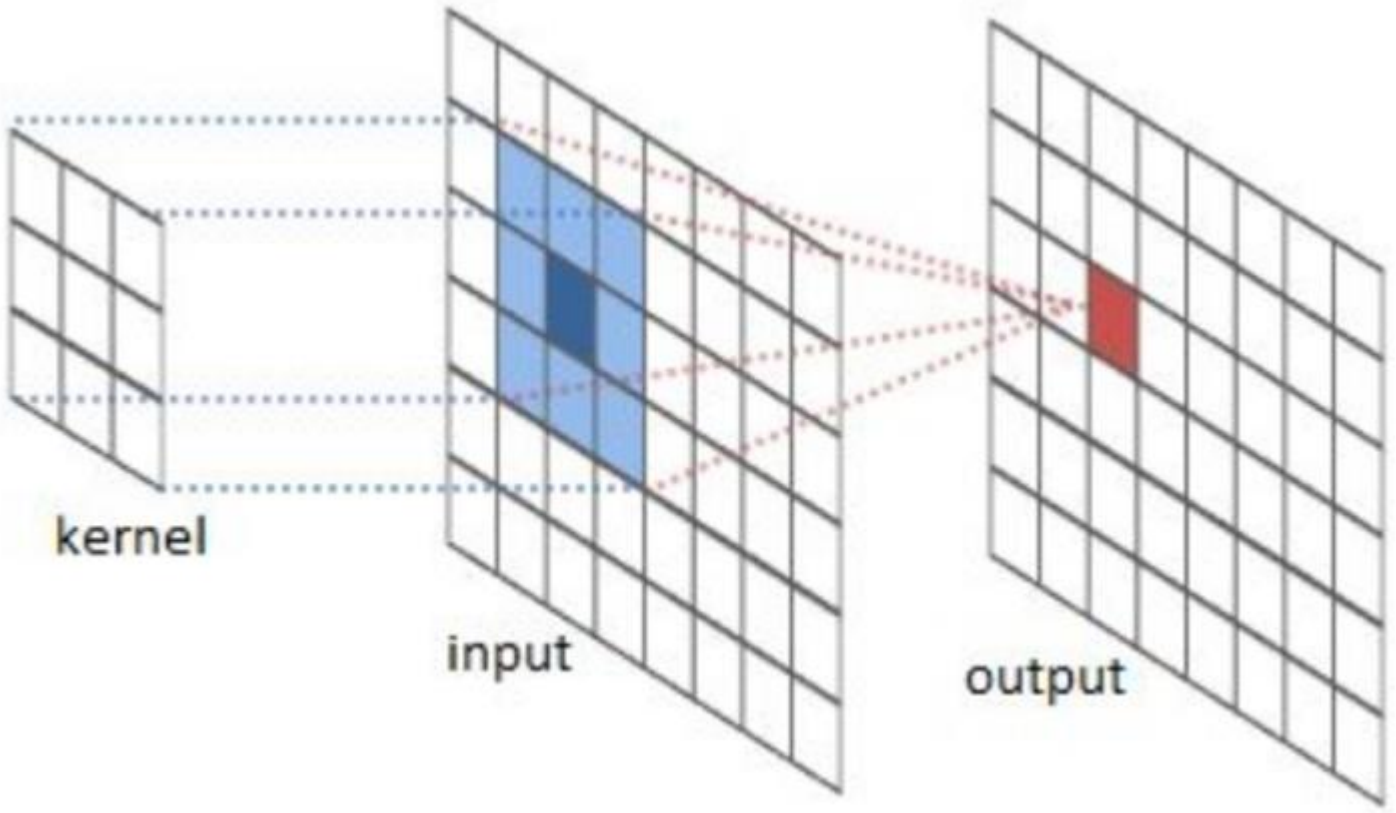
Фільтр переміщається уздовж зображення і визначає, чи є присутньою деяка шукана характеристика в конкретній його частині.

Для отримання відповіді такого роду здійснюється операція свертки, яка є сумою творів елементів фільтру і матриці вхідних сигналів.

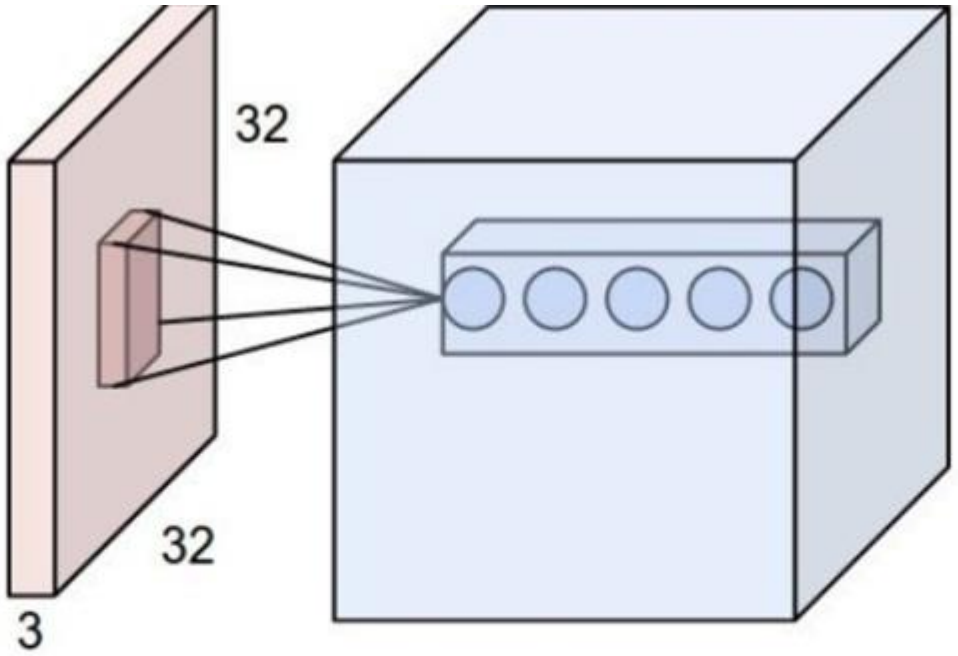
Візуалізація згортки



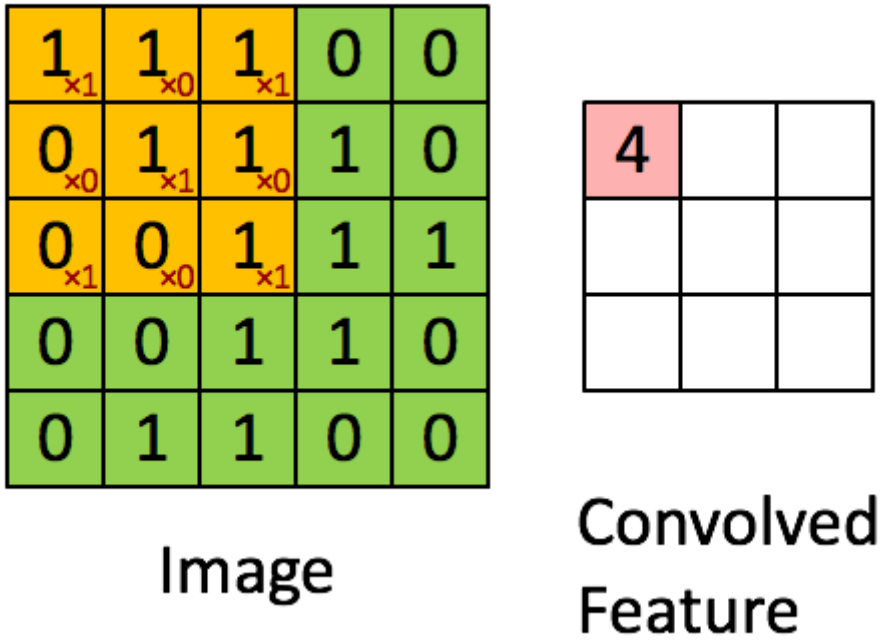
Згортка



Згортковий шар 5 нейронів



Веса нейронов — это коэффициенты ядра свёртки. Каждая “обучаемая” свёртка выделяет одинаковые локальные признаки во всех частях изображения.


1×1 1×0 1×1 0 0
0×0 1×1 1×0 1 0
0×1 0×0 1×1 1 1
0 0 1 1 0
0 1 1 0 0
Image
4
Convolved Feature

Згорткові нейронні мережі

- Фільтр може переміщатися уздовж матриці вхідних сигналів з кроком, відмінним від одиниці. Крок переміщення фільтру називається **страйдом (stride).**
- **Страйд визначає, на яку кількість пікселів повинен зміститися фільтр за один крок.**

$$n_{out} = floor(\frac{n_{in} - f}{s}) + 1$$

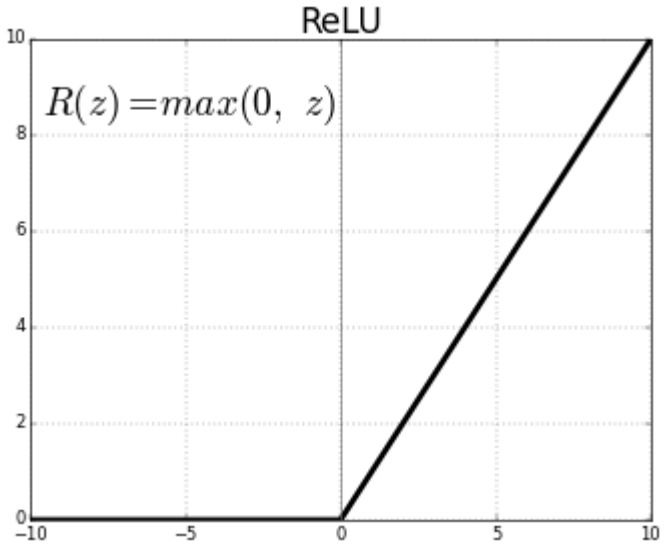
- Кількість вихідних значень після операції згортки може бути розрахована по формулі 1.
- Де n_in — к-ть вхідних пікселів,
- f — к-ть пікселів у фільтрі,
- s — страйд.
- Для прикладу на мал дану формулу слід застосувати таким чином: (25-9)/2+1=3.

Згорткові нейронні мережі

- Для того, щоб навчання ваг, що знаходяться в кернелах, було ефективним, в результати згорток слід ввести деяке зміщення (bias) і нелінійність.
- Зміщення — це статична величина, на яку слід "змістити" вихідні значення. За своєю суттю це звичайна операція складання кожного елементу вихідної матриці з величиною зміщення.
- Якщо пояснювати дуже поверхнево, це треба для того, щоб вивести нейронну мережу з тупикових ситуацій, що мають суто математичні причини.
- Нелінійність представляє з себе **функцію активації**. Завдяки ній картина, що формується за допомогою операції згортки, отримує деяке спотворення, що дозволяє нейронній мережі ясніше оцінювати ситуацію.

Згорткові нейронні мережі

- Цю необхідність дуже грубо можна порівняти з необхідністю людям із слабким зором носити контактні лінзи.
- А взагалі-то така необхідність пов'язана з тим, що вхідні дані за своєю природою нелінійні, тому треба умисне спотворювати проміжні результати, щоб відповідь нейронної мережі була такою, що вирішує проблему.
- Часто як функцію активації використовують ReLU (Rectified Linear Unit).

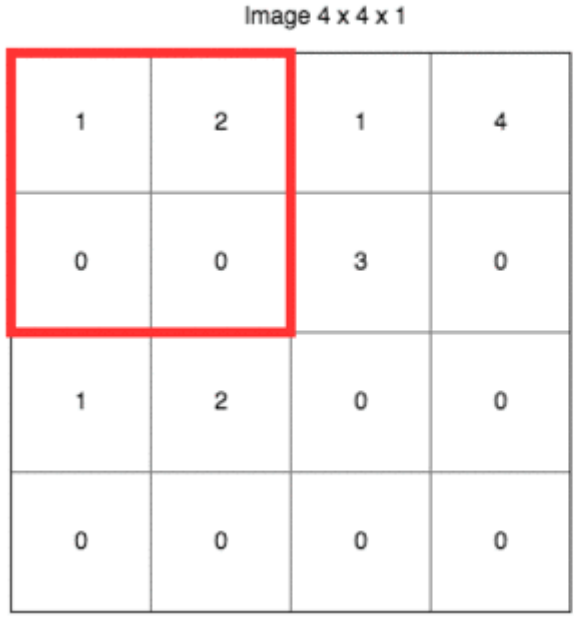


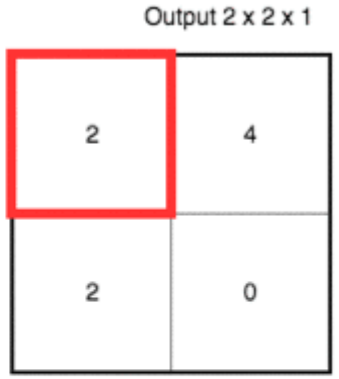
Даунсемплінг

- З метою прискорення процесу навчання і зменшення споживання обчислювальних ресурсів роблять даунсемплинг початкових і проміжних даних
- . Існує декілька способів зробити це. У цьому курсі розглядатимемо найпростіший і поширеніший з них — **максимальне об'єднання (max pooling).**
- Операція максимального об'єднання полягає в тому, що уздовж даних переміщається так зване *вікно просіювання.* З пікселів, що потрапляють в його поле зору, відбирається максимальний і переміщається в результуючу матрицю.
- Страйд для цього вікна може бути відмінним від одиниці; усе залежить від того, якого розміру вихідну матрицю треба отримати, і з якою мірою треба "стиснути" дані.

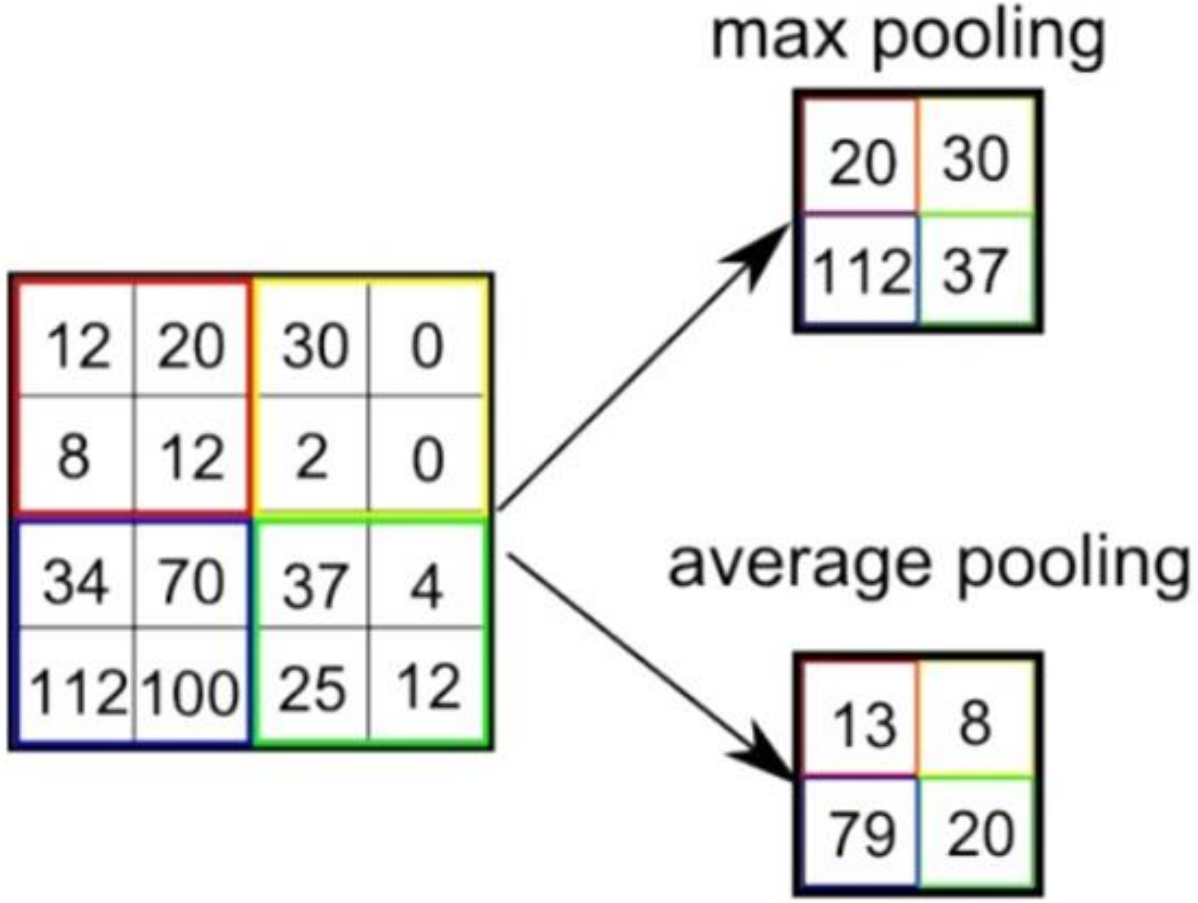
Даунсемплинг

- Подивіться уважно на малюнок. На ньому наочно зображена операція максимального об'єднання : вікно розмірністю 2×2 переміщається уздовж матриці; страйд має значення 2.
- Страйд визначає крок, на який переміщається вікно за один етап. На кожному етапі відбирається максимальне значення. Початкова матриця нібито просіюється через сито, яке видає назовні тільки характерні пікселі.

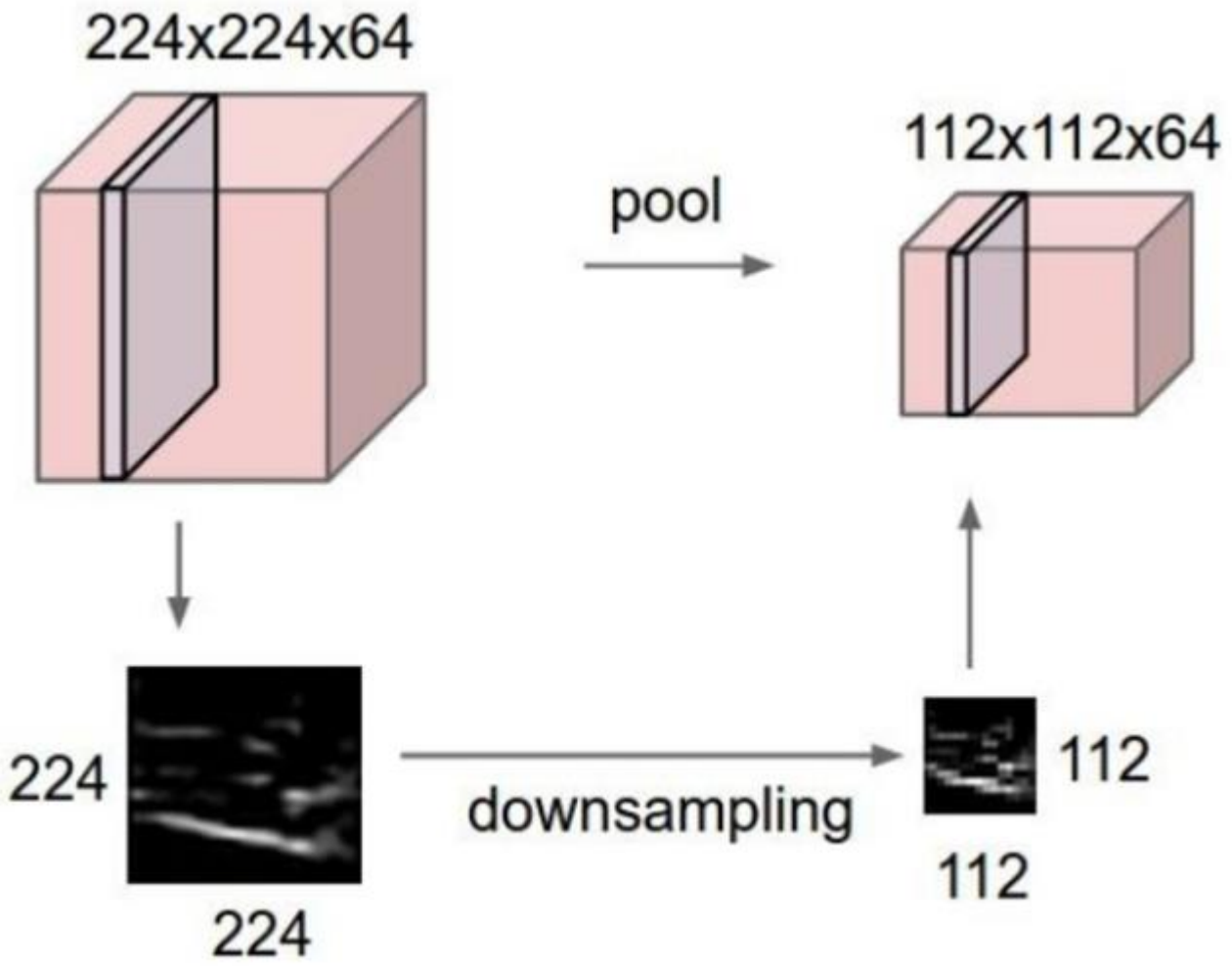




Pooling

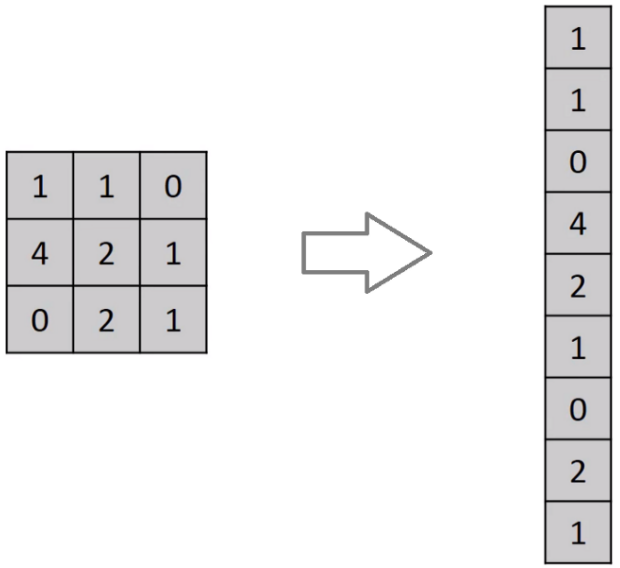

max pooling
20 30
112 37
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
average pooling
13 8
79 20

Downsampling



Повнозв'язний шар

- На цьому етапі перетворення даних тривимірна матриця сигналів буде розгорнута у вектор, який буде пропущений через повнозв'язну нейронну мережу. Її завдання полягає в тому, щоб повідомити нам вірогідність того, яку цифру представляє вхідний образ. Мал. демонструє цю операцію.

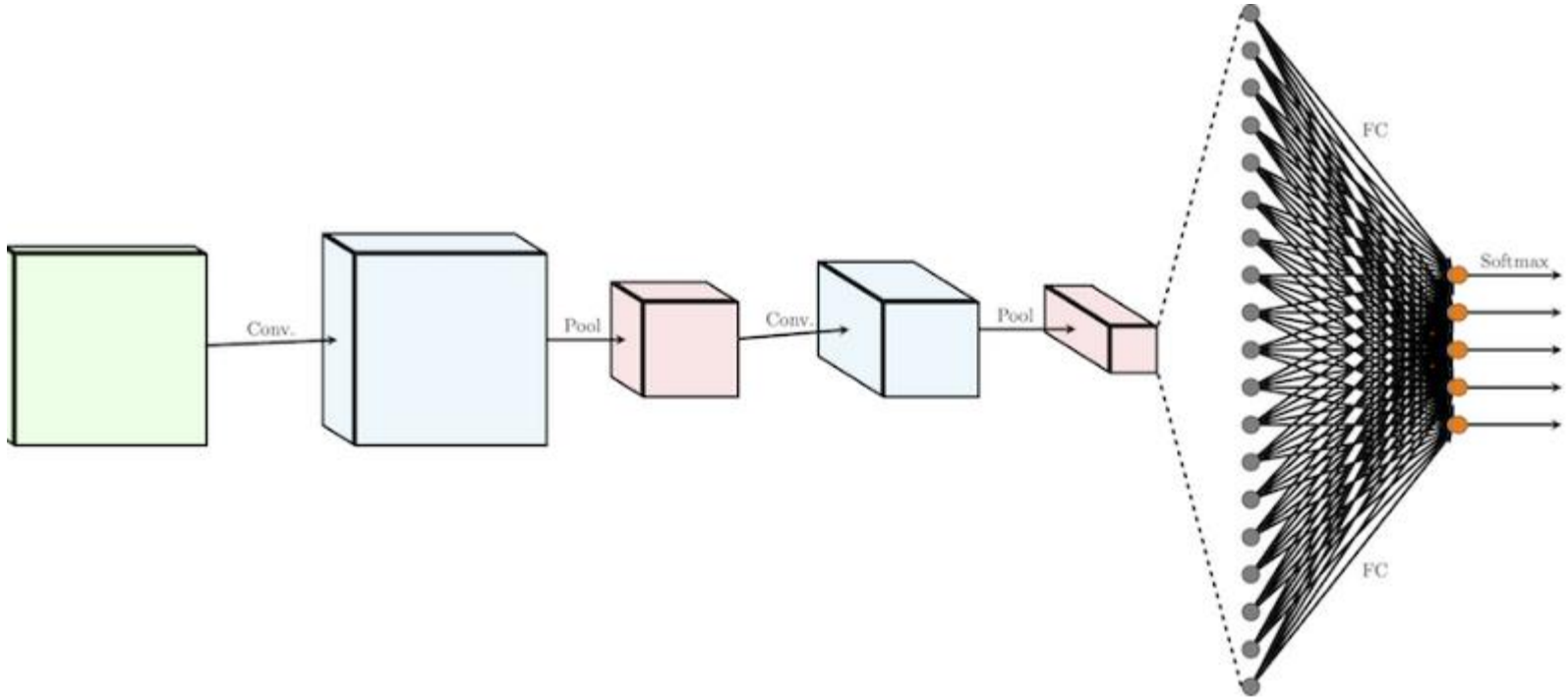


Повнозв'язний шар

- З малюнка видно, що операція розгортання полягає в "склеюванні" рядків в єдиний — величезної довжини — числовий ряд. Це буде воістину великий вектор, який треба буде ще і перетворити за допомогою багатошарової мережі з повними зв'язками!
- Якщо ви не знаєте, як працює повнозв'язний шар, ось простий опис механізму : кожен елемент вектору множиться на вагу зв'язку, ці твори далі підсумовуються між собою і з деяким зміщенням, після чого результат піддається перетворенню за допомогою функції активації. На мал. описане представлене в наочній формі.
- Варто відмітити, що Ян ЛеКун в цьому посту на Фейсбуці сказав, що "в згортальних нейронних мережах немає такого поняття, як повнозв'язний шар". І він абсолютно правий!

Повнозв'язний шар

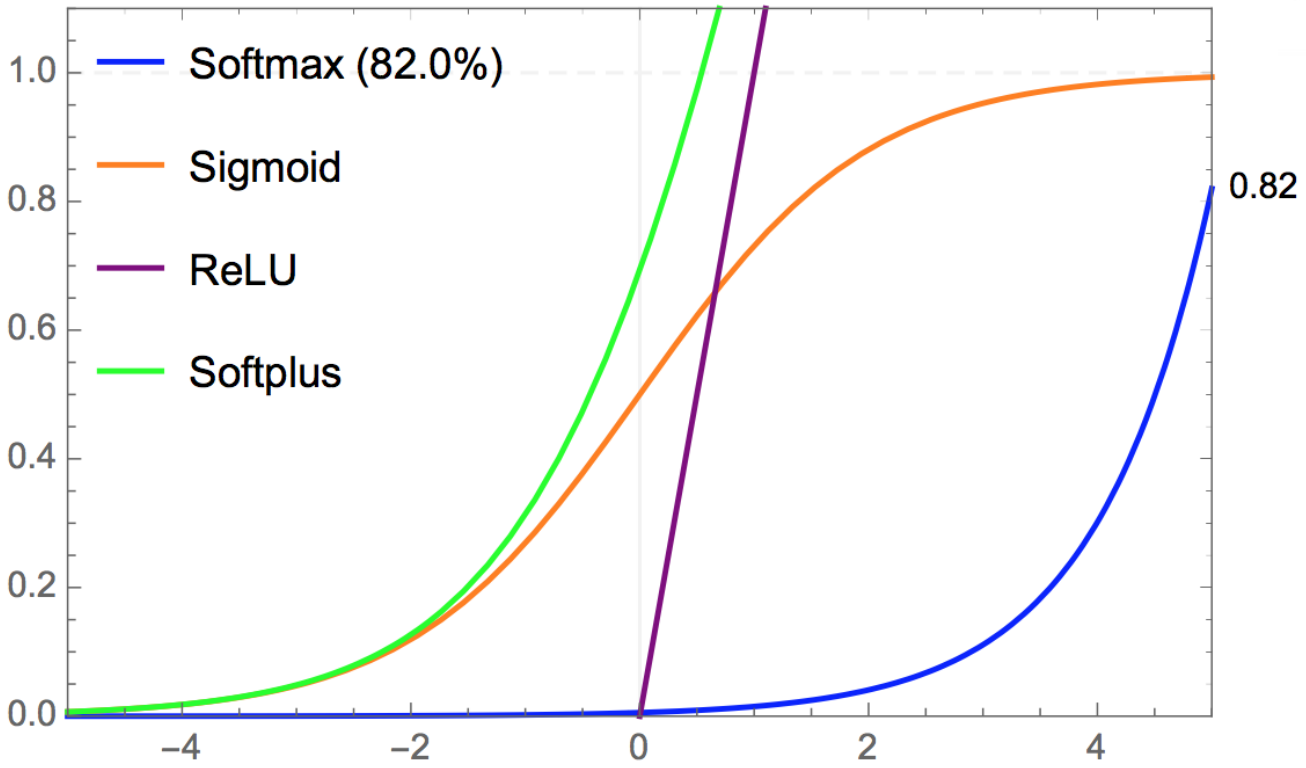
Якщо уважно придивитися, то стане абсолютно очевидно, що принцип роботи повнозв'язного шару аналогічний тому, що має місце в згортальному шарі з кернелом розмірністю 1×1. Тобто, якщо в нашому розпорядженні 128 фільтрів розмірністю n×n, які взаємодіятимуть із зображенням розмірністю n×n, на виході ми отримаємо вектор, в якому буде 128 елементів.



Вихідний шар

- Вихідний шар відповідає за формування вірогідності приналежності вхідного образу тому або іншому класу (деякому числу). Щоб добитися цього, вихідний шар повинен містити кількість нейронів, що відповідають кількості класів. Зважені і підсумовані сигнали далі модифікуються за допомогою **функції активації використовуватимемо softmax**

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$



Функція помилки (втрат)

Щоб визначити, наскільки точно нейронна мережа визначає написані від руки цифри, ми використовуватимемо **функцію втрат**.

Відповіддю функції втрат є дійсне число, що характеризують якість відповіді нейронної мережі. Зазвичай для оцінки якості класифікаторів застосовують категоріальну крос-энтропийну функцію втрат (Categorical Cross - Entropy Loss Function, або CCELF).

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

У формулі

$\hat{y}$ — *фактична відповідь нейронної мережі,*

y — бажана відповідь нейронної мережі.

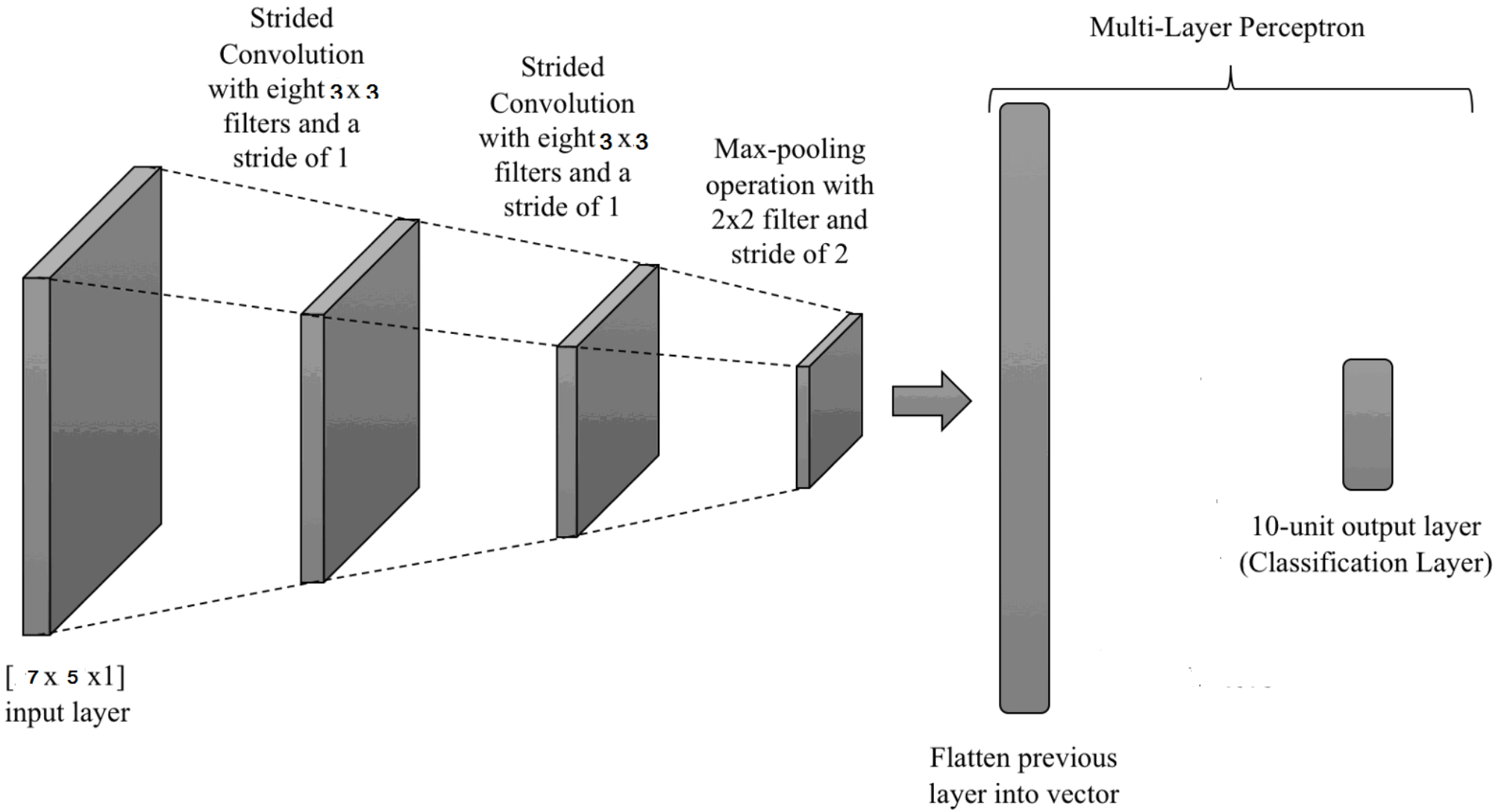
Відповіддю в нашому випадку є деяке число; у ширшому сенсі відповідь представляє категорію. Для отримання загального показника втрат, характерного для усіх категорій в цілому, беруть середнє від значень по кожній категорії.

Приклад CNN

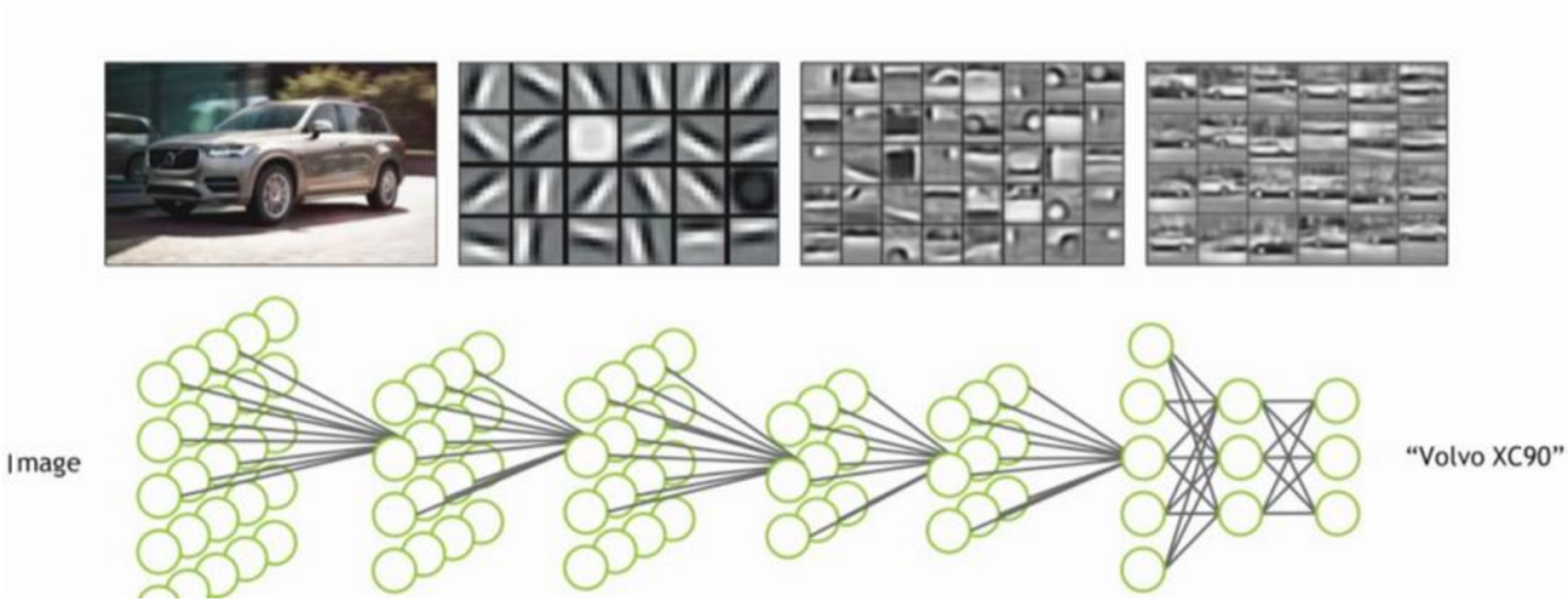
Розглянемо згорткову НМ для розпізнавання цифр.

У нашому розпорядженні відносно невелика кількість класів (10, якщо бути точним). Розмір зображень в навчаній множині складає 7x5 пікселів. З цих причин архітектура нейронної мережі буде досить простою:

- два згортальні шари і один шар максимального об'єднання (max pooling layer) для витягання характеристик початкового зображення;
- багатошаровий перцептрон (Multi - Layer Perceptron, MLP), завданням якого є класифікація отриманих раніше характеристик зображення.

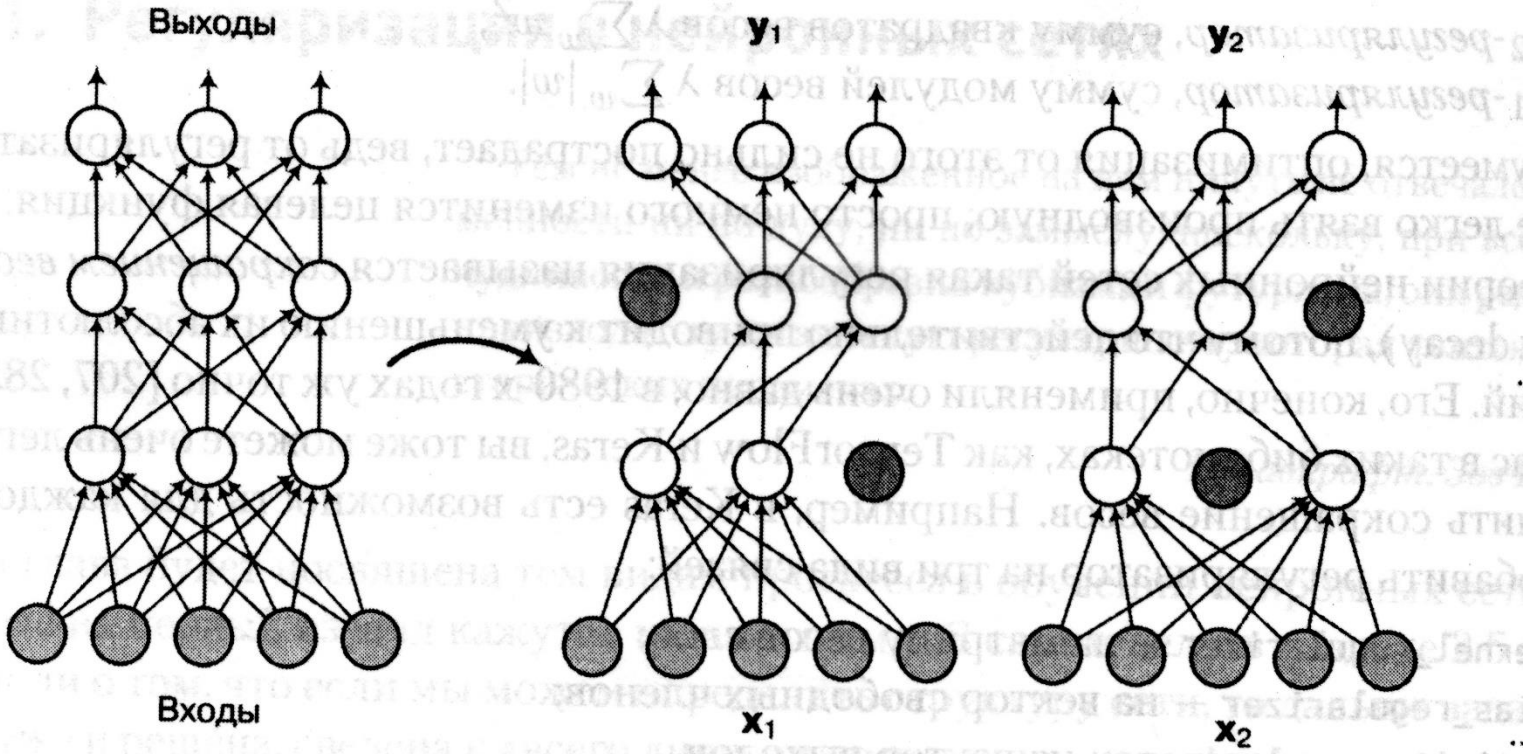

Strided Convolution with eight 3 x 3 filters and a stride of 1
Strided Convolution with eight 3 x 3 filters and a stride of 1
Max-pooling operation with 2x2 filter and stride of 2
Multi-Layer Perceptron
[7 x 5 x1] input layer
Flatten previous layer into vector
10-unit output layer (Classification Layer)

Свёрточные слои учат иерархические признаки для изображений,
а spatial pooling даёт некоторую инвариантность к перемещениям.



Регуляризація

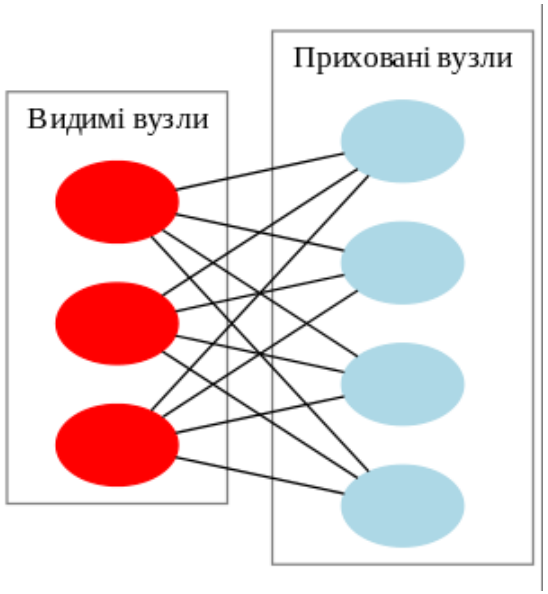
- Дропаут



Попереднє навчання CNN

1. Навчання з допомогою обмеженої машини Больцмана

Restricted Boltzmann machine, RBM (2006)



Для глибокого навчання починаємо із вхідних значень, а потім після навчання значення прихованих вузлів стають вхідними для наступного шару. Процес закінчується на вихідних вузлах.

Навчання відбувається без учителя.

Значення вагових параметрів зв'язків є початковими для подальшого навчання НМ з учителем.

Попереднє навчання CNN

2. Навчання з допомогою ініціалізації Ксав’є

Автори: Ксав’є Глоро і Йошуа Бенджі *(2010)*

Метод застосовується тільки для антисиметричних функцій активації, типу tanh(s).

Формула ініціалізації для чергового шару

$$W_{ij} \approx U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$

U- рівномірний розподіл випадкових чисел

n_{in} *– кількість нейронів у попередньому шарі;*

n_{out} *– кількість нейронів у наступному шарі.*

Попереднє навчання CNN

3. Навчання з допомогою ініціалізації Хе

Автор: Каймінг Хе *(2015)*

Метод застосовується тільки для несиметричних функцій активації, типу ReLU*(s).*

Формула ініціалізації для чергового шару

$$W_{ij} \approx N\left[0, \frac{\sqrt{2}}{\sqrt{n_{in}^{(1)}}}\right]$$

N - нормальний розподіл випадкових чисел

$n_{in}^{(1)}$ *– кількість нейронів у вхідному шарі;*

Попередньо навчені нейронні мережі

Попередньо навчені нейронні мережі (pre-trained neural networks) є потужним інструментом у галузі машинного навчання та штучного інтелекту. Вони надають базу моделей, які вже були навчені на великих наборах даних, часто на задачах, що вимагають значних обчислювальних ресурсів.

Ці моделі можуть бути використані як вихідний пункт для подальшого навчання або як основа для трансферного навчання (transfer learning), де знання, отримані під час одного завдання, адаптуються до іншого, схожого завдання.

Ключові аспекти попередньо навчених мереж:

- **Швидке розгортання**: Можливість використовувати вже навчені моделі дозволяє розробникам швидко впроваджувати рішення без необхідності витрачати час і ресурси на тривале навчання моделей з нуля.
- **Економія ресурсів**: Навчання глибоких нейронних мереж вимагає значних обчислювальних ресурсів, особливо коли йдеться про великі набори даних. Використання попередньо навчених моделей може значно скоротити ці витрати.
- **Трансферне навчання**: Попередньо навчені моделі часто використовуються як основа для трансферного навчання, де знання з одного завдання переносяться на інше, часто з деякими модифікаціями в кінцевих шарах мережі для адаптації до нового завдання.

Приклади попередньо навчених моделей

1. ImageNet навчені моделі:

- ***VGG16, VGG19***: Ці моделі були розроблені в Oxford і показали високі результати на змаганнях ImageNet. Вони часто використовуються як основа для різних задач обробки зображень.
- **ResNet-50, ResNet-101**: Мережі з "глибокими залишковими" з'єднаннями, які дозволяють ефективно навчати дуже глибокі нейронні мережі.
- **Inception v3, Xception**: Моделі, що використовують модулі "Inception", які дозволяють зменшити кількість параметрів при збереженні або підвищенні точності класифікації.

Приклади попередньо навчених моделей

2. **NLP моделі**:

- **BERT (Bidirectional Encoder Representations from Transformers)**: Модель від Google, яка встановила нові стандарти в області обробки природної мови, зокрема в задачах розуміння тексту.
- **GPT (Generative Pre-trained Transformer)**: Розроблена OpenAI, ця модель показала вражаючі результати в генерації тексту.

3. Моделі для аналізу аудіо:

- **DeepSpeech**: Модель від Mozilla, яка використовується для перетворення мови в текст.

- Використання попередньо навчених моделей може значно прискорити розробку і впровадження систем штучного інтелекту, а також підвищити їхню доступність і ефективність.

Нормалізація на міні-батчах CNN

Міні- батч – це випадкова частина навчальної вибірки, за якою уточнюється градієнт на кожному кроці навчання. Цей процес називається стохастичним градієнтним спуском. Він займає менше часу і ресурсів, чим навчання по всьому дата сетові.

Виявилось, що таке навчання суттєво прискорюється, якщо проводити нормалізацію вхідних і проміжних даних по міні-батчах.

Сергій Йоффе, Крістіан Сегеді (2015).

- **Формули нормалізації:**

$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

Навчання на міні-батчах CNN

Міні- батч – це випадкова частина навчальної вибірки, за якою уточнюється градієнт на кожному кроці навчання. Цей процес називається стохастичним градієнтним спуском. Він займає менше часу і ресурсів, чим навчання по всьому дата сетові.

Для {$\mathbf{x}_p$, $\mathbf{d}_p$}, p=1…p1, p1<P виконуємо матричні операції:

1. Прямий хід

W(q) = k(q)*rand(-1,+1).

$y^{(q)} = F(s^{(q)})$, q=1…Q

$$y^{(0)} = x \qquad s^{(q)} = y^{(q-1)} \cdot w^{(q)}$$

2. Зворотний хід

3. Уточнення ваг

$$\delta_j^{(Q)} = (y^{(Q)} - d\ ) \cdot F'(s^{(Q)}) \qquad \delta^{(q)} = \delta^{(q+1)} \cdot w^{(q+1)} F'(s^{(q)})$$

$$\Delta w_{ij}^{(q)} = -\eta \cdot \delta_j^{(q)} \cdot y_i^{(q-1)} \qquad w^{(q)}(t+1) = w^{(q)}(t) + \Delta w^{(q)}$$

4. Новий випадковий міні-бач

$$E_p(w_i^{(q)}) = \frac{1}{2}\sum_{j=1}^{M} (y_{j,p}^{(Q)} - d_{j,p})^2 < \varepsilon$$

Адаптивний градієнтний спуск

Адаптивний градієнтний спуск дозволяє налаштовувати крок для окремих вагових параметрів

$$w_{ij}^{(q)}(t+1) = w_{ij}^{(q)}(t) + \Delta w_{ij}^{(q)} = w_{ij}^{(q)}(t) - \eta \cdot \frac{\partial E_p(w_{ij}^{(q)}(t))}{\partial w_{ij}^{(q)}}$$

$$g_{ij}^{(q)} = \frac{\partial E_p(w_{ij}^{(q)}(t))}{\partial w_{ij}^{(q)}} = \delta_j^{(q)} \cdot y_i^{(q-1)}$$

- **Формули для зміни загального кроку:**

$$\eta = \frac{\eta_0}{1+\beta \cdot t} \qquad \eta = \eta_0(1-\frac{t}{T}) \qquad \eta = \eta_0 e^{\frac{t}{T}}$$

- **Метод Adagrad**

$$\eta = \frac{\eta_0}{\sqrt{G_{ij}^{(q)}(t)+\varepsilon}} \qquad G_{ij}^{(q)}(t+1) = G_{ij}^{(q)}(t) + (g_{ij}^{(q)})^2$$

Адаптивний градієнтний спуск

- **Метод Adadelta**

$$\eta = \frac{\sqrt{R_{ij}^{(q)}(t) + \varepsilon}}{\sqrt{G_{ij}^{(q)}(t) + \varepsilon}}$$

$$R_{ij}^{(q)}(t+1) = \rho \cdot R_{ij}^{(q)}(t) + (1-\rho) \cdot (\Delta w_{ij}^{(q)}(t))^2$$

$$0 < \rho < 1$$

- **Метод Adam**

$$\eta = \frac{\eta_0}{\sqrt{G_{ij}^{(q)}(t) + \varepsilon}}$$

$$G_{ij}^{(q)}(t+1) = \beta_2 \cdot r_{ij}^{(q)}(t) + (1-\beta_2) \cdot (g_{ij}^{(q)})^2$$

$$r_{ij}^{(q)}(t+1) = \beta_1 \cdot r_{ij}^{(q)}(t) + (1-\beta_1) \cdot g_{ij}^{(q)}$$

$$0 < \beta_1 < 1 \qquad \beta_1 = 0.9 \qquad 0 < \beta_2 < 1 \qquad \beta_2 = 0.999$$

Рекурентні нейронні мережі

Рекуре́нтні нейро́нні мере́жі (англ. *recurrent neural networks, RNN*) — це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф орієнтований у часі.

На відміну від нейронних мереж прямого поширення, *RNN* можуть використовувати свою внутрішню пам'ять для обробки довільних послідовностей на входах, що змінюються в часі.

Що таке набір даних MNIST?

Набір MNIST — це велика колекція **рукописних цифр.** Це дуже популярний набір даних у сфері обробки зображень. Його часто використовують для порівняльного аналізу алгоритмів машинного навчання.

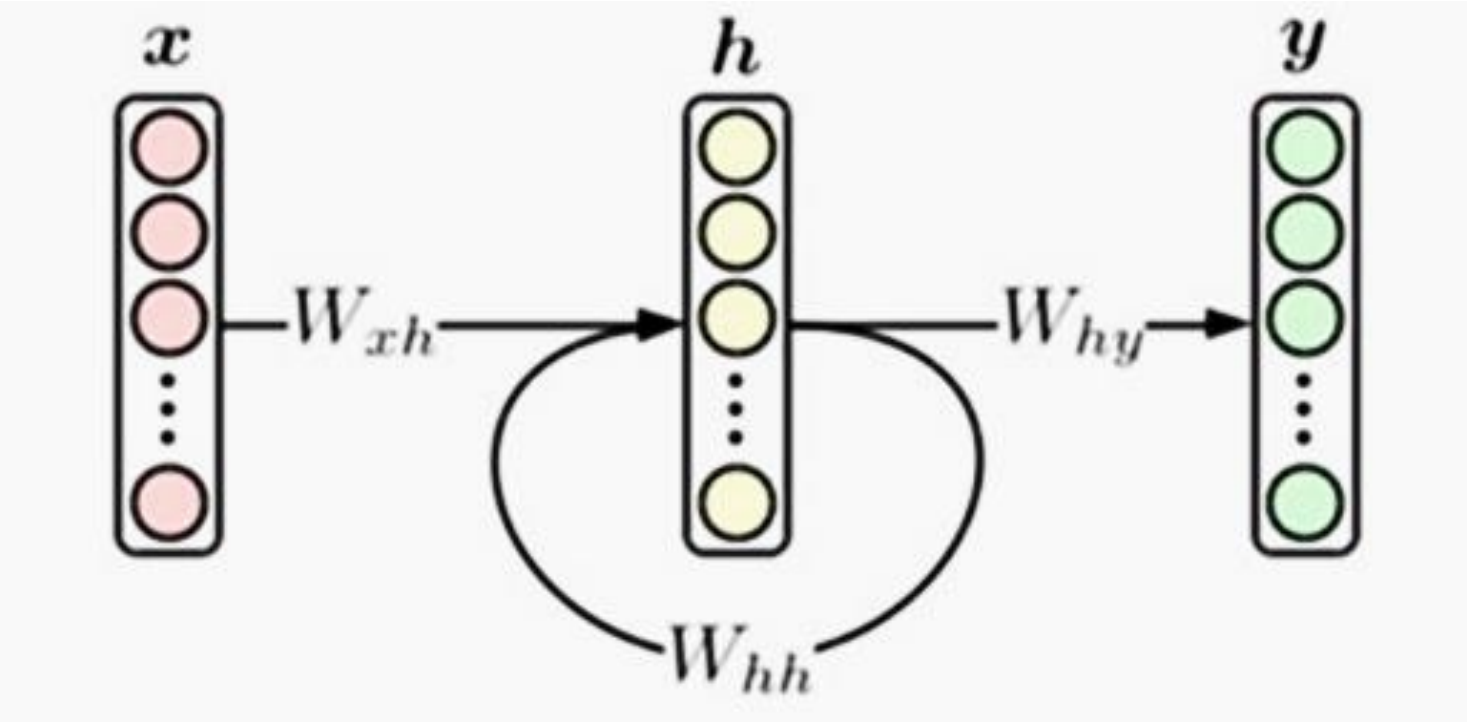
- ***MNIST*** *є* скороченням від ***Modified National Institute of Standards and Technology database.***

- MNIST містить колекцію з **70 000 зображень 28 x 28** рукописних цифр від **0 до 9.**

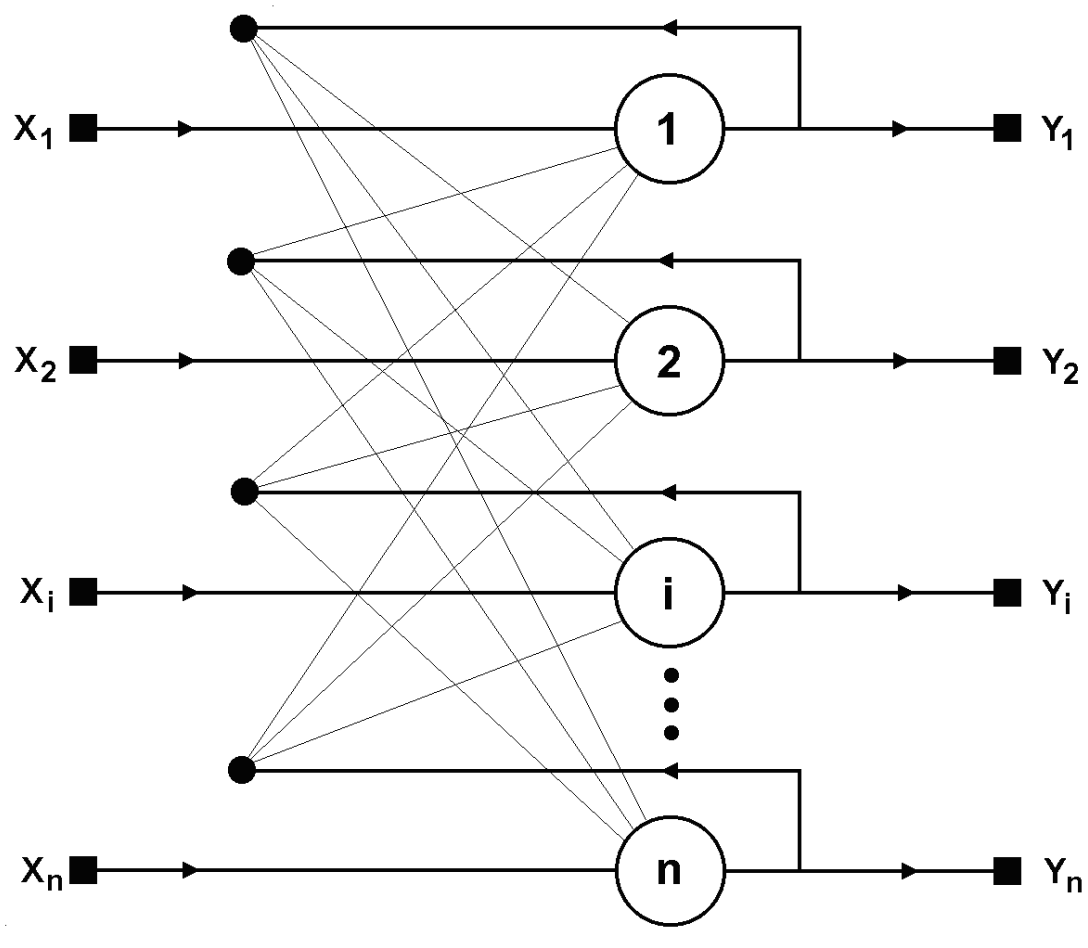
- Набір даних уже розділений на набори для навчання та тестування 60000 І 10000 данних.

Рекурентні нейронні мережі

Основна відмінність рекурентних НМ– наявність циклічних зв’язків



Мережа Хопфілда

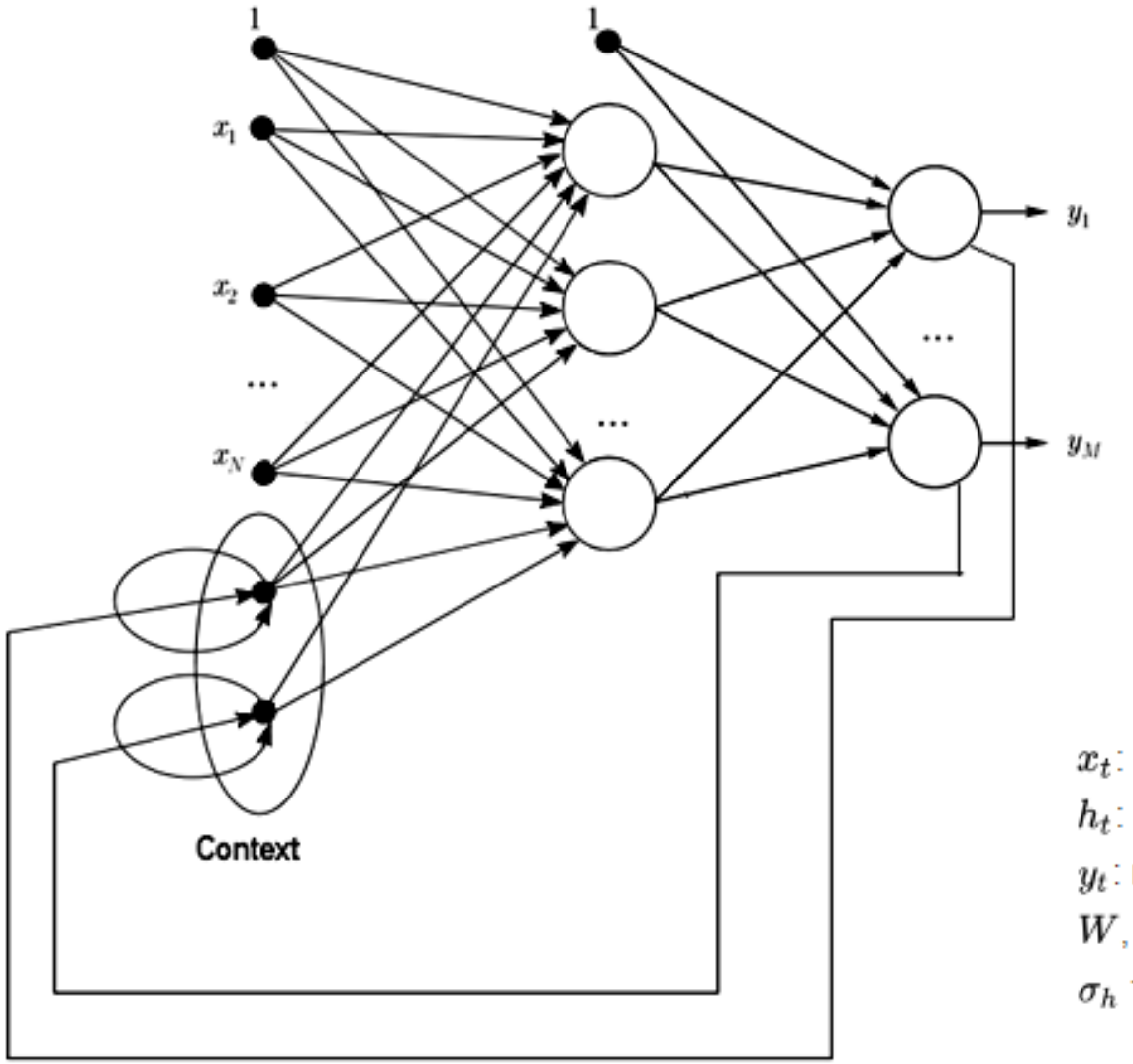


Мережа Джордана

Мережа Майка Джордана - частково рекурентна мережа (Jordan networks are partially recurrent networks).

- Її можна розглядати як мережу прямого поширення з додатковими нейронами контексту у вхідному шарі.
- Ці контекстні нейрони приймають вхід від себе (прямий зворотний зв'язок, direct feedback) і з вихідних нейронів.
- Контекстні нейрони зберігають поточний стан мережі. У мережі Джордана кількість контекстних і вихідних нейронів має бути однаковою.

Мережа Джордана



Мережа Джордана

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

x_t: входовий вектор

h_t: вектор прихованого шару

y_t: виходовий вектор

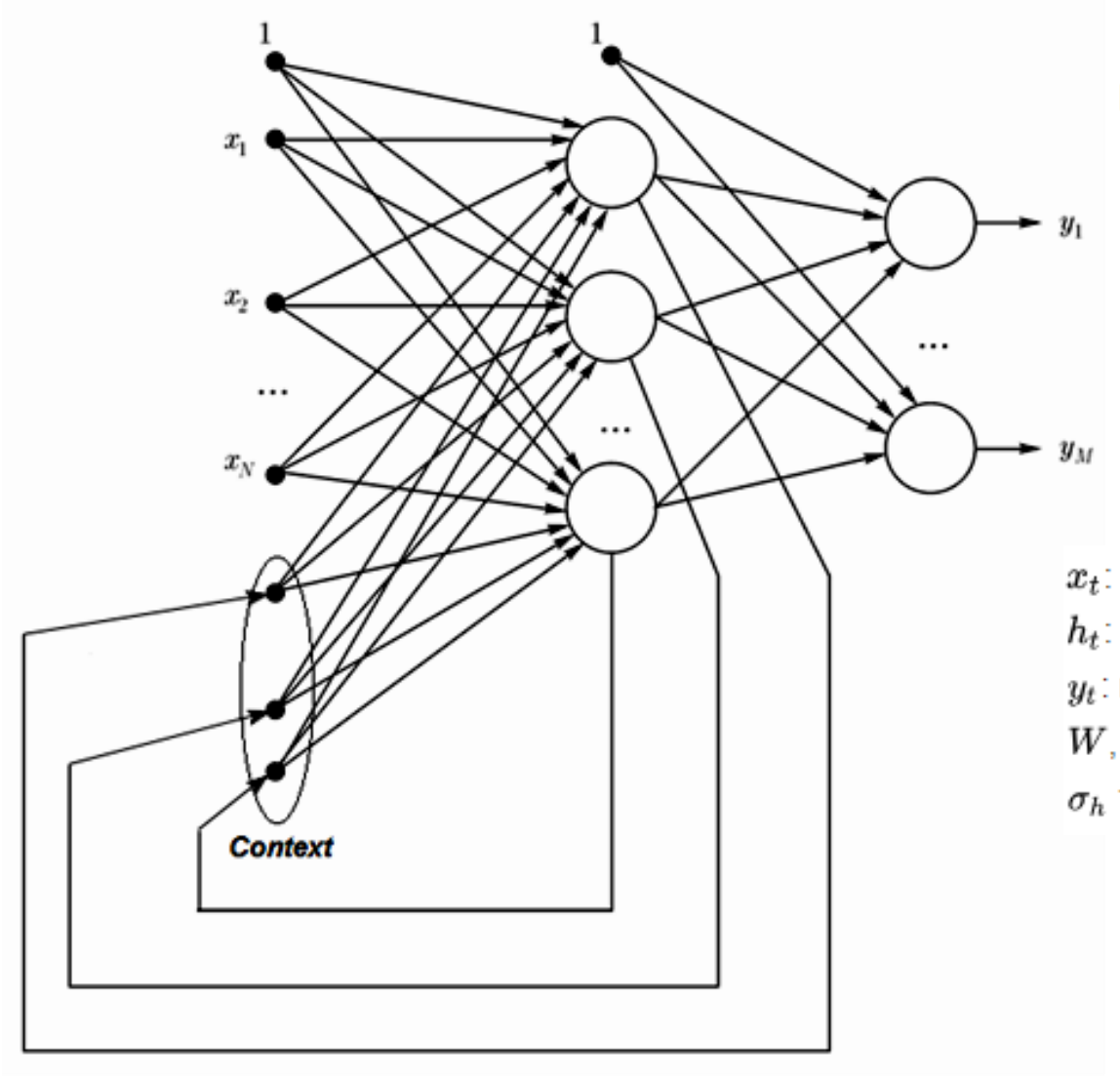
W, U та b: матриці та вектор параметрів

σ_h та σ_y: функції активації

Мережа Елмана

- **Мережа Джеффа Елмана -** частково рекурентна мережа, подібна до мереж Джордана (Elman networks are partially recurrent networks and similar to Jordan networks).
- Різниця між мережею Елмана і Джордана в тому, що в мережі Елмана контекстні нейрони беруть вхід не від вихідних нейронів, а від прихованих. Крім того, в контекстних нейронах немає ніякого зворотного зв'язку.
- У мережі Елмана число контекстних і прихованих нейронів має бути однаковим. Головна перевага мереж Елмана полягає в тому, що число контекстних нейронів визначається не розмірністю виходу а кількістю прихованих нейронів, що робить її гнучкішою. Можна легко додати або прибрати приховані нейрони, на відміну від кількості виходів.

Мережа Елмана



Мережа Елмана

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

x_t: входовий вектор

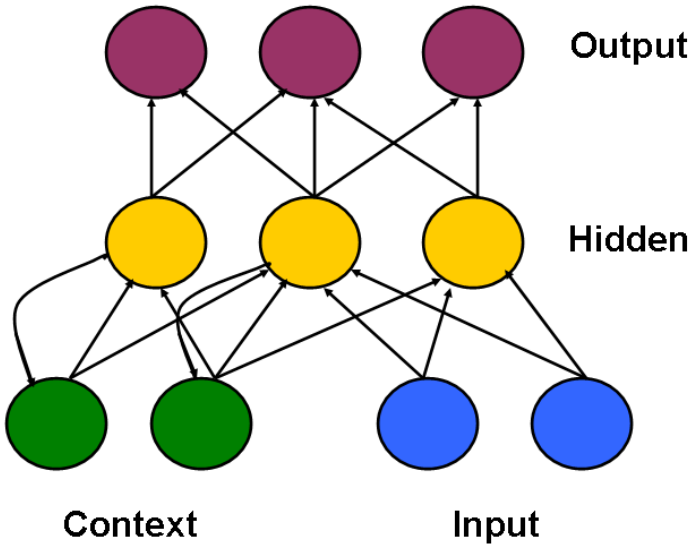
h_t: вектор прихованого шару

y_t: виходовий вектор

W, U та b: матриці та вектор параметрів

σ_h та σ_y: функції активації

Мережа Елмана

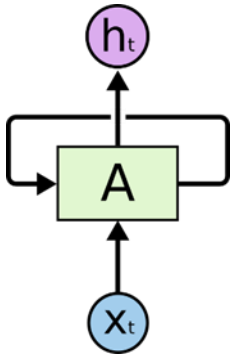


Мережа Елмана[10]

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Рекурентна нейронна мережа

Рекурентні нейронні мережі (RNN) - це клас вдосконаленої штучної нейронної мережі (ANN), містять зворотні зв'язки і дозволяють зберігати інформацію

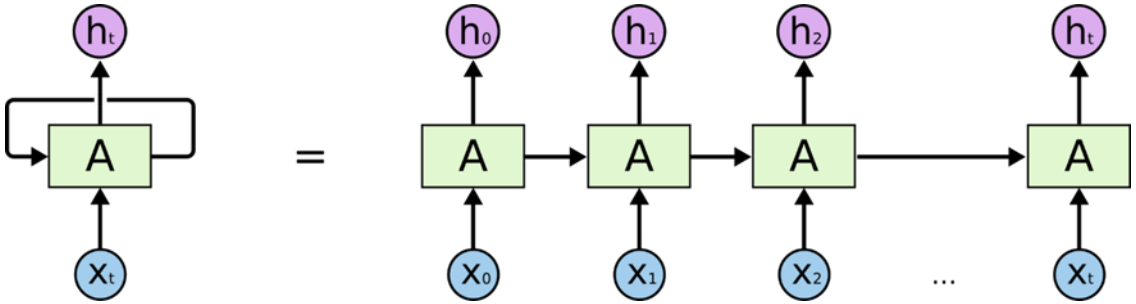


На схемі вище фрагмент нейронної мережі A приймає вхідне значення x_ti повертає значення h_t. Наявність зворотного зв'язку дозволяє передавати інформацію від одного кроку мережі до іншого.

Рекурентна нейромережа

Рекурентную мережу можна розглядати, як кілька копій однієї і тієї ж мережі, кожна з яких передає інформацію подальшої копії.

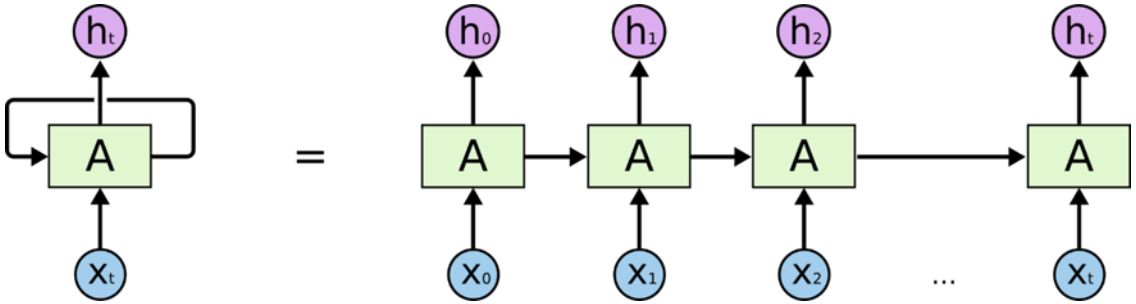
Ось, що станеться, якщо розгорнемо зворотний зв'язок, представлено нижче



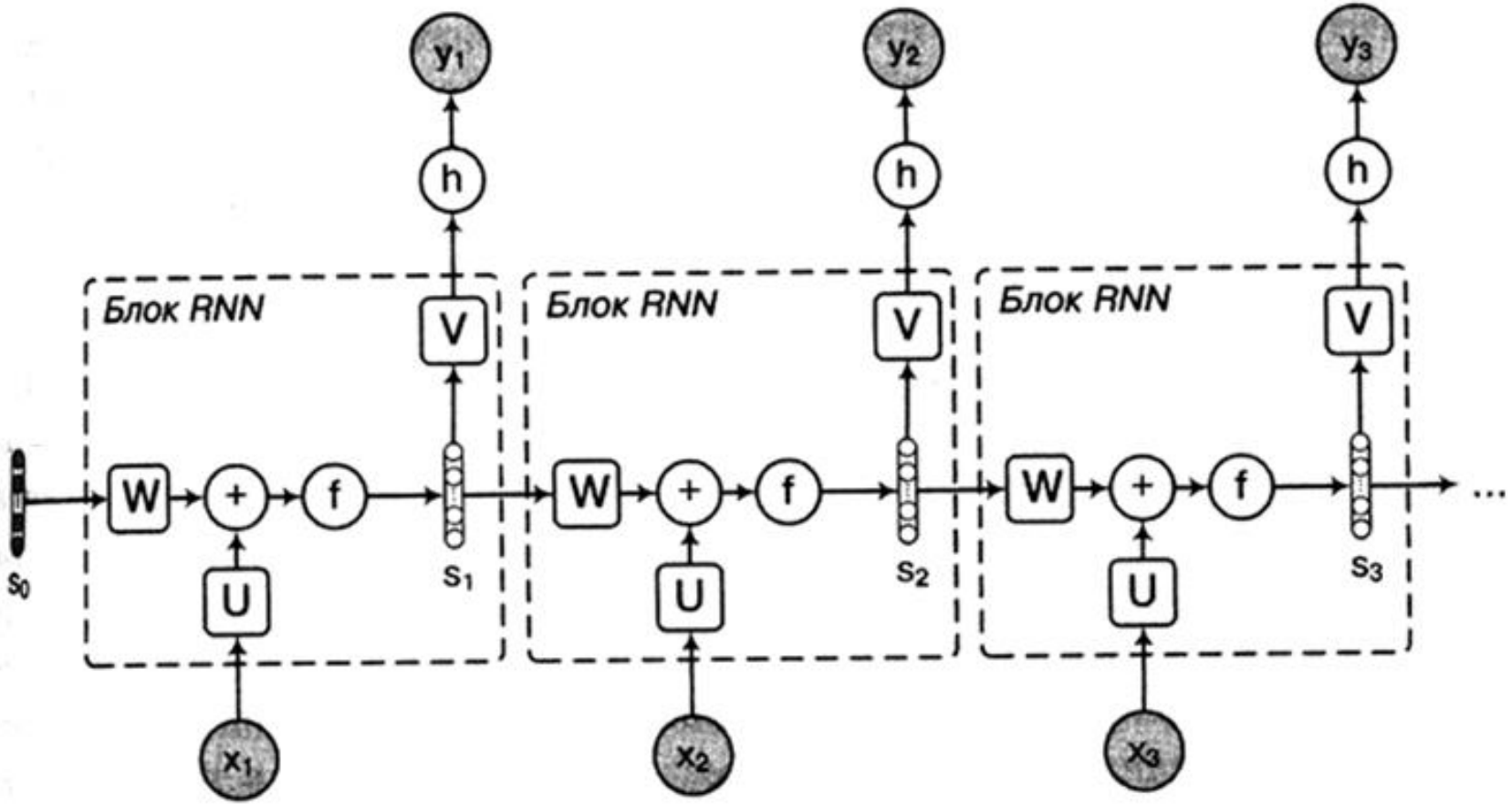
Рекурентна нейромережа

На жаль, у міру зростання цієї відстані, RNN втрачають здатність зв'язувати інформацію

Цю проблему детально досліджували Зепп Хохрайтер (Sepp Hochreiter, 1991) і Іошуа Бенджі (Yoshua Bengio) зі співавторами (1994). Вихід - LSTM



Розгортання RNN



Рекурентні співвідношення RNN

- $a_t = b + \mathbf{W} s_{t-1} + \mathbf{U} x_t$
- $s_t = f(a_t)$
- $o_t = c + \mathbf{V} s_t$
- $Y_t = h(o_t)$

Довга короткострокова пам'ять (Long short-term memory; LSTM)

LSTM - особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довготривалим залежностям.

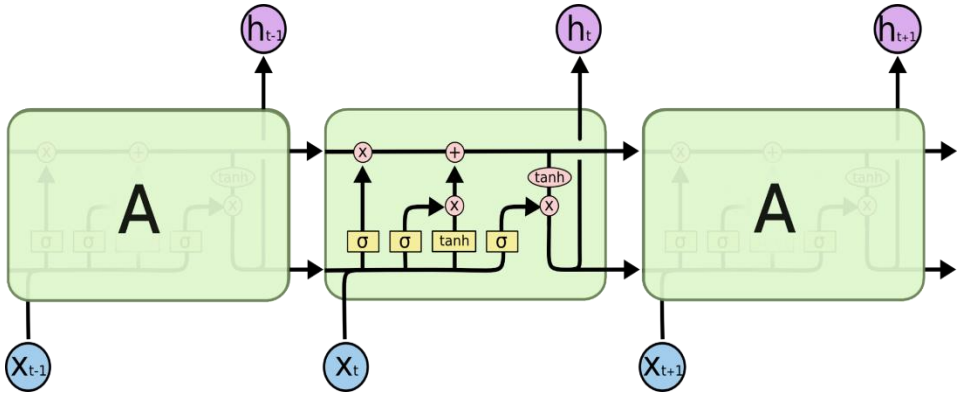
Вони були представлені Зеппом Хохрайтером і Юргеном Шмідхубером (Jürgen Schmidhuber) в 1997 році, а потім вдосконалені і популярно викладені в роботах багатьох інших дослідників.

Вони прекрасно вирішують цілий ряд різноманітних завдань і в даний час широко використовуються.

LSTM розроблені спеціально, щоб уникнути проблеми довготривалої залежності. Запам'ятовування інформації на довгі періоди часу - це їх звичайна поведінка, а не щось, чого вони намагаються навчитися.

LSTM

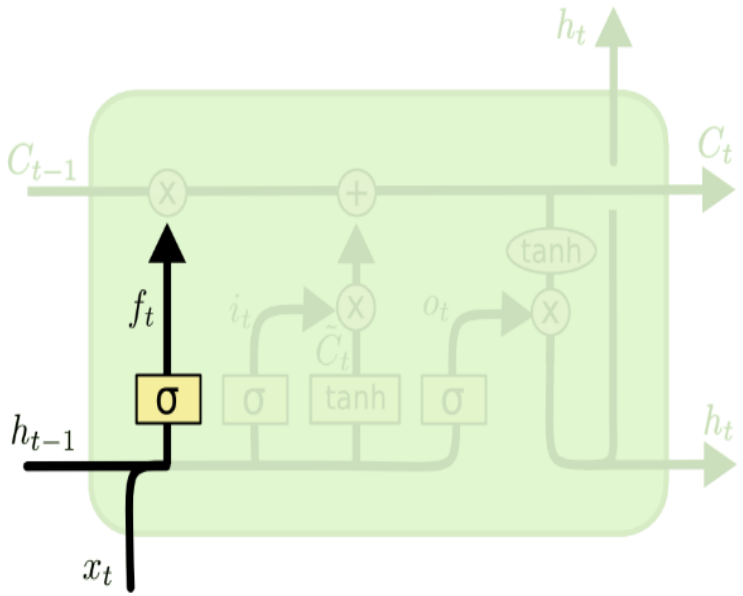
Ключовий компонент LSTM - це стан осередку (cell state) - горизонтальна лінія, що проходить по верхній частині схеми.



Стан осередку нагадує конвеєрну стрічку, зображений на рисунку LSTM може видаляти інформацію зі стану осередку; цей процес регулюється структурами, званими фільтрами (gates).

Фільтри дозволяють пропускати інформацію на підставі деяких умов. Вони складаються з шару сигмоїдної нейронної мережі і операції поточечного множення.

Перший крок в LSTM - визначити, яку інформацію можна викинути зі стану осередку, зображений на рисунку 1.13. Це рішення приймає сигмоїдальна шар, званий «шаром фільтра забування» (forget gate layer).

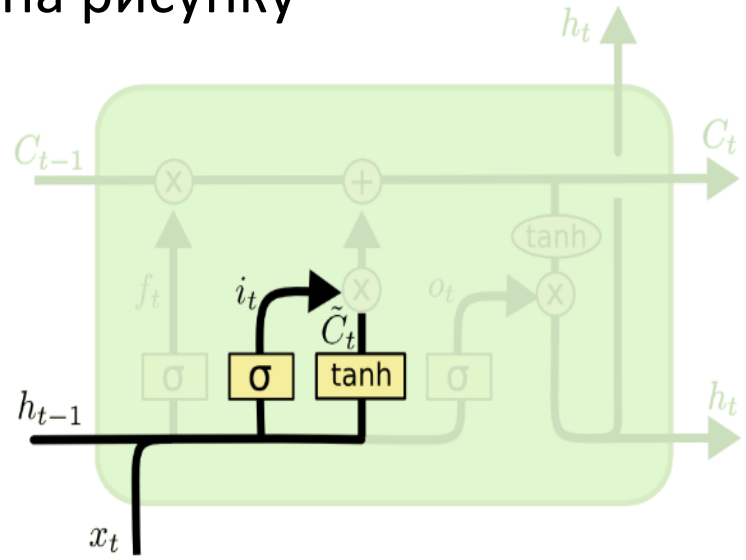


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Він дивиться на h_{t-1} і x і повертає число від 0 до 1 для кожного числа зі стану осередку C_{t-1}. 1 означає «повністю зберегти», а 0 – «повністю викинути».

LSTM

Другий крок - вирішити, яка нова інформація буде зберігатися в стані осередку, зображений на рисунку



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Цей етап складається з двох частин.

Спочатку сигмоїдний шар під назвою «шар вхідного фільтра» (input layer gate) визначає, які значення слід оновити.

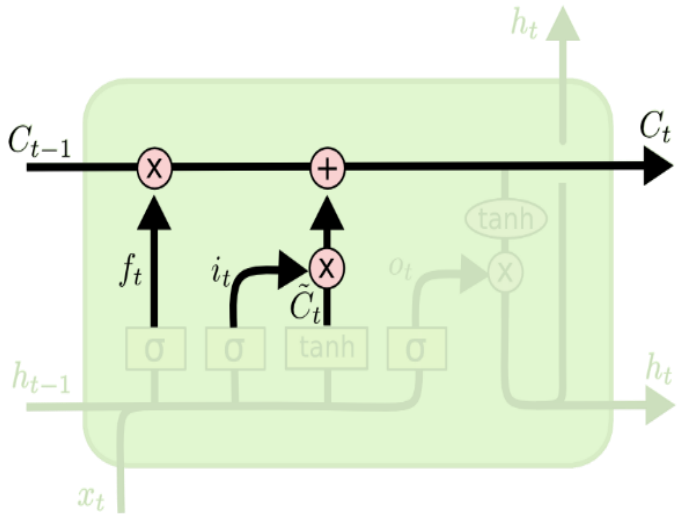
Потім tanh-шар будує вектор нових значень-кандидатів C_t, які можна додати в стан осередку.

LSTM

Третій крок: множимо старий стан на f, забуваючи те, що вирішили забути.

Потім додаємо $i_t * C_t$.

Це нові значення-кандидата, помножені на t - на скільки хочемо оновити кожне з значень стану

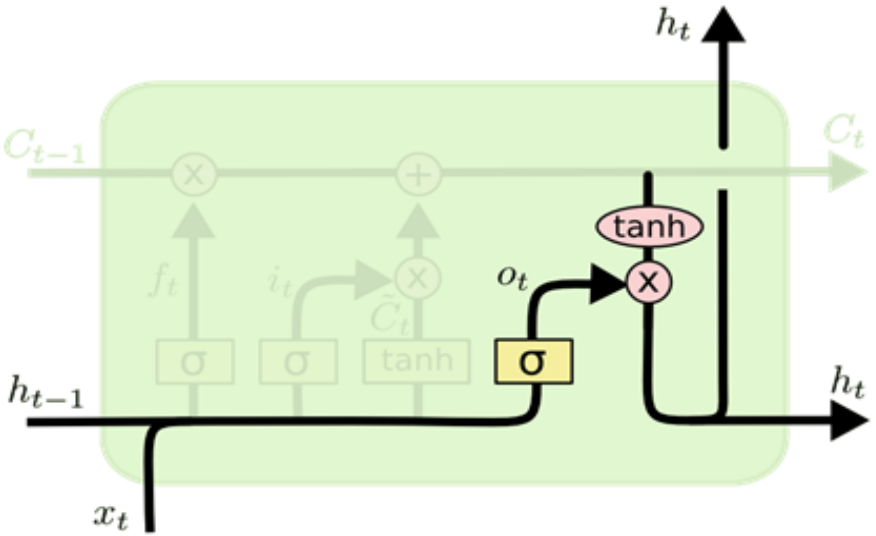


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM

Четвертий крок - вихідні дані будуть засновані на стані осередку і до них будуть застосовані деякі фільтри. Спочатку ми застосовуємо сигмоїдний шар, який вирішує, яку інформацію зі стану осередку ми будемо виводити.

Потім значення стану осередку проходять через tanh-шар, щоб отримати на виході значення з діапазону від -1 до 1, і перемножуються з вихідними значеннями сигмоїдного шару, що дозволяє виводити тільки необхідну інформацію.

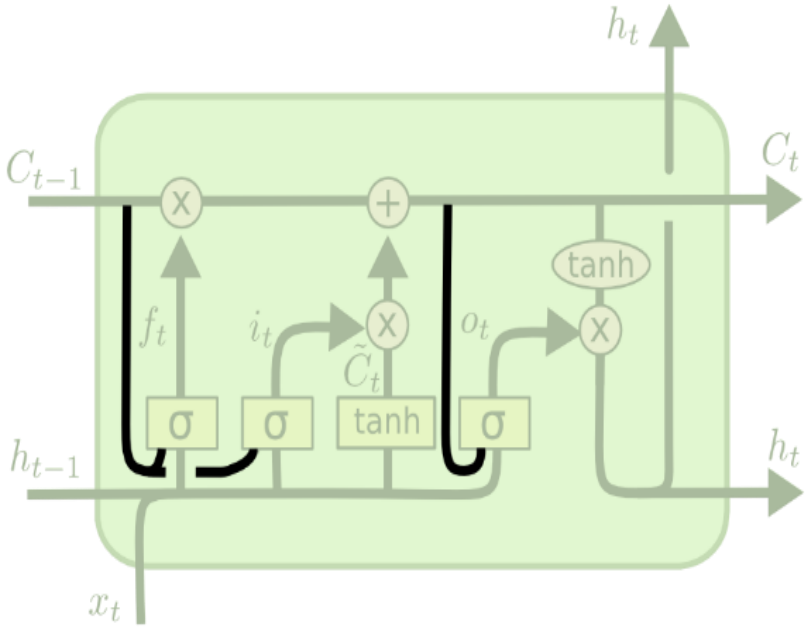


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM

Одна з популярних варіацій LSTM, запропонована Герсом і Шмідхубером (Gers & Schmidhuber, 2000), характеризується додаванням так званих «оглядових вічок» («peephole connections»), зображена нижче на рисунку 1.17. З їх допомогою шари фільтрів можуть бачити стан осередку.

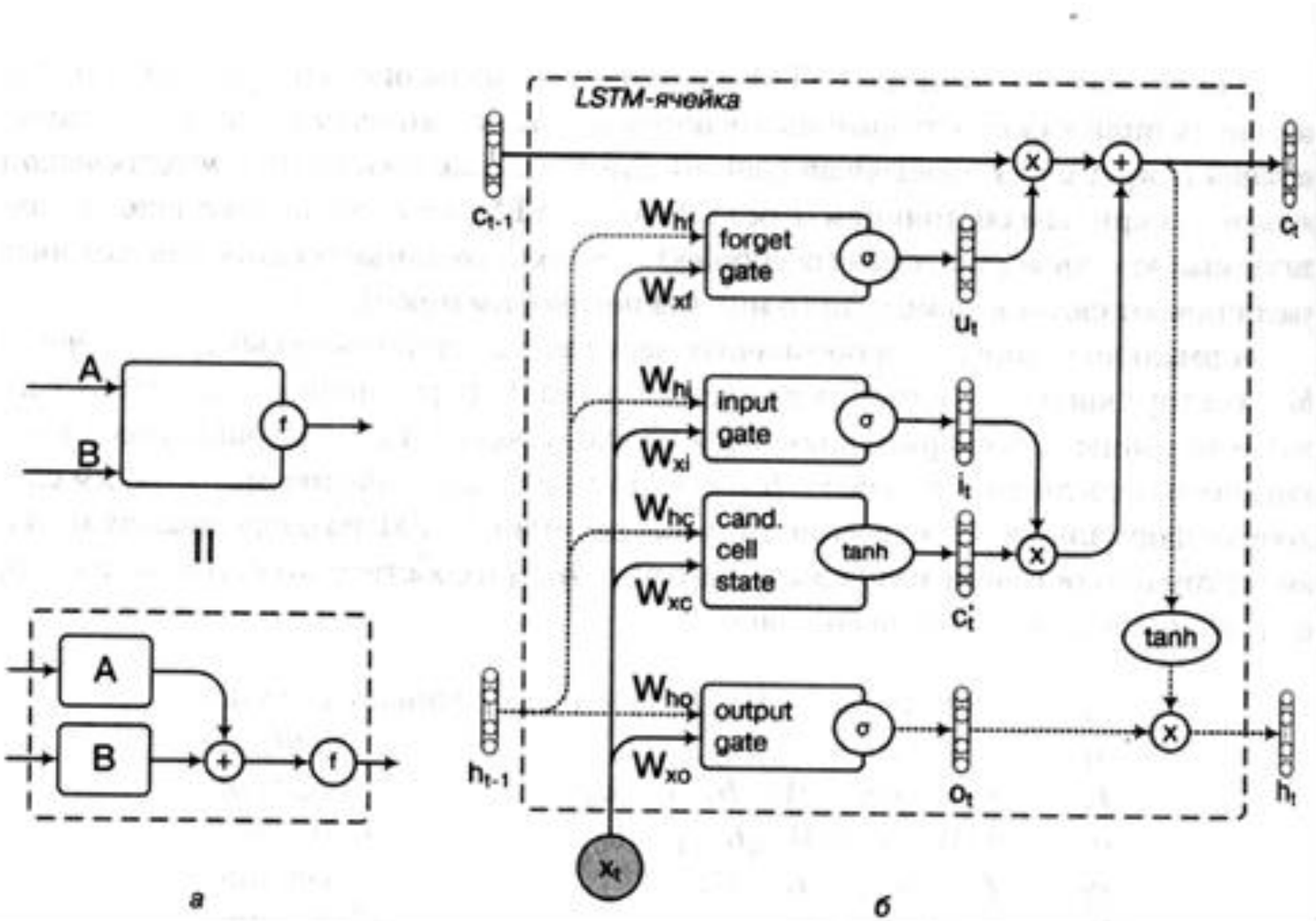


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$

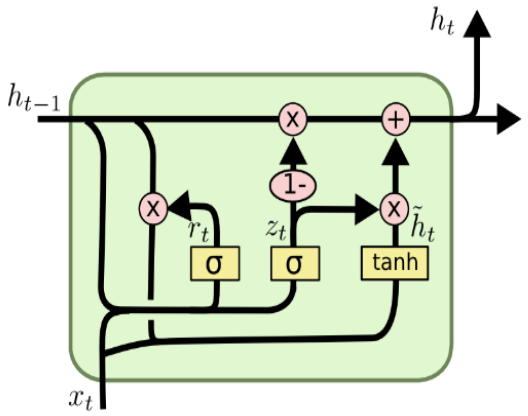
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

LSTM


A
B
f
=
A
B
+
f
а
LSTM-ячейка
c_{t-1}
W_{hf}
W_{xf}
forget gate
σ
u_t
W_{hi}
W_{xi}
input gate
σ
i_t
W_{hc}
W_{xc}
cand. cell state
tanh
c'_t
W_{ho}
W_{xo}
output gate
σ
o_t
h_{t-1}
x_t
tanh
c_t
h_t
б

GRU

Керовані рекурентні нейрони (Gated recurrent units, GRU), вперше описані в роботі Cho, et al (2012). У ній фільтри «забування» і входу об'єднують в один фільтр «оновлення» (update gate). Крім того, стан комірки об'єднується з прихованим станом, є й інші невеликі зміни. Побудована в результаті модель простіша, ніж стандартна LSTM, і популярність її неухильно зростає, зображена нижче на рисунку :



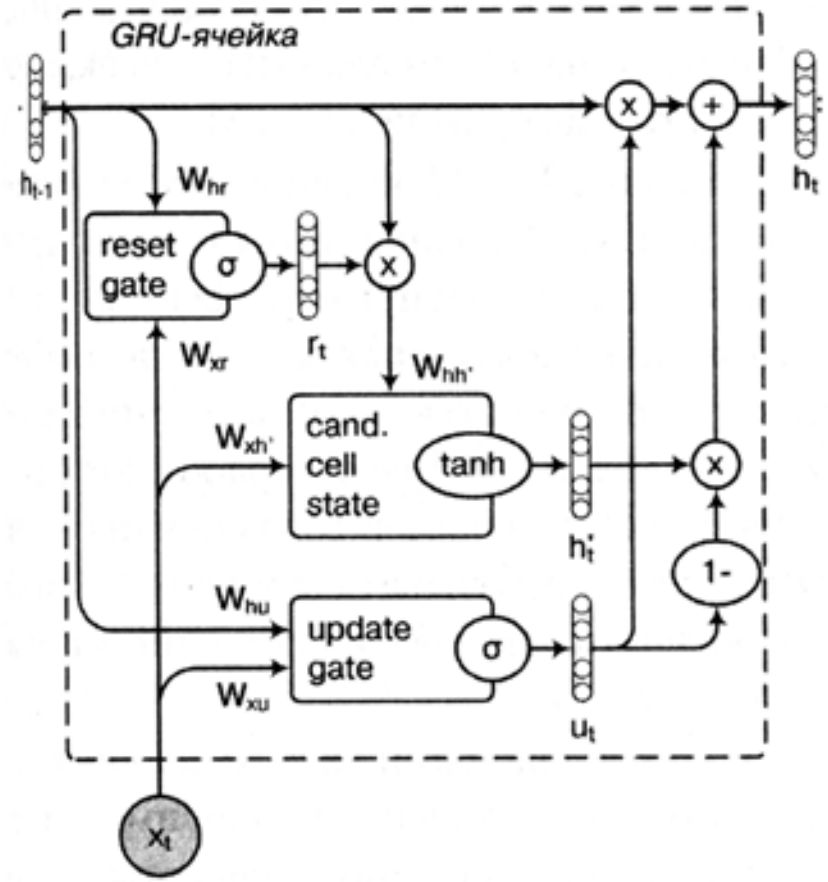
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

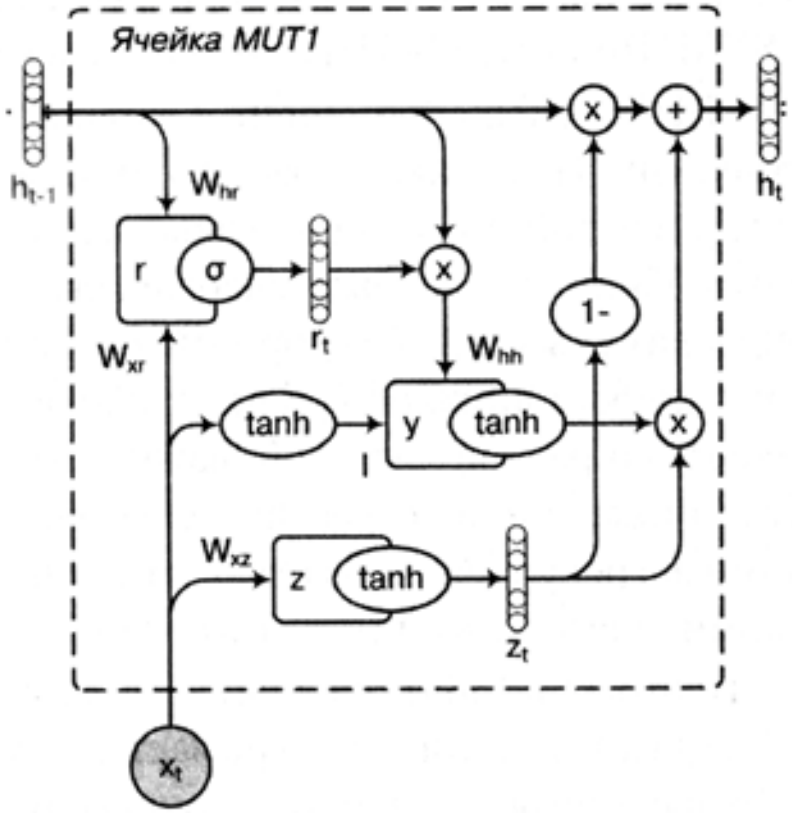
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU MUT


GRU-ячейка
h_{t-1}
W_{hr}
reset gate
σ
r_t
W_{xr}
$W_{hh'}$
$W_{xh'}$
cand. cell state
tanh
h'_t
W_{hu}
update gate
σ
W_{xu}
u_t
x
+
1-
h_t
x_t


Ячейка MUT1
h_{t-1}
W_{hr}
r
σ
r_t
W_{xr}
W_{hh}
tanh
y
tanh
l
W_{xz}
z
tanh
z_t
1-
x
+
h_t
x_t

Бібліотеки нейронних мереж

Бібліотеки нейронних мереж — це спеціалізовані інструменти, які дозволяють розробникам ефективно створювати, тренувати та використовувати нейронні мережі для різноманітних завдань машинного навчання та штучного інтелекту.

Ці бібліотеки надають різні інструменти та API для різних рівнів складності та потреб у глибокому навчанні, дозволяючи розробникам вибирати найбільш підходящий інструмент для своїх проектів.

Ось декілька популярних бібліотек:

1. **TensorFlow** - одна з найбільш популярних бібліотек для глибокого навчання, розроблена Google. Вона підтримує широкий спектр інструментів для створення складних архітектур нейронних мереж і використовується як в академічних, так і в комерційних цілях.
2. **Keras** - високорівнева нейромережева API, яка може працювати на основі TensorFlow, Theano або CNTK. Keras розроблена з метою швидкого експериментування та прототипування, що робить її ідеальною для початківців.

Бібліотеки нейронних мереж

3. PyTorch - бібліотека від Facebook, яка здобула велику популярність завдяки своїй гнучкості та інтуїтивно зрозумілому інтерфейсу. PyTorch особливо популярний у дослідницькій спільноті через легкість експериментування та динамічне створення графів.

4. Theano - це одна з перших бібліотек глибокого навчання, яка дозволяє ефективно реалізовувати математичні операції з масивами. Хоча розробка Theano була припинена, вона все ще використовується в академічних колах.

5. Microsoft Cognitive Toolkit (CNTK) - це бібліотека від Microsoft, яка дозволяє легко масштабувати та інтегрувати глибоке навчання в продукти Microsoft.

6. MXNet - це бібліотека, яка підтримує як імперативне, так і символьне програмування, що дозволяє користувачам гнучко керувати своїми моделями. MXNet є частиною ініціативи Amazon для глибокого навчання.

Бібліотека TensorFlow

TensorFlow — це потужна бібліотека для чисельних обчислень, яка особливо ефективна для великих масштабів і паралельної обробки даних. Розроблена Google Brain Team, TensorFlow широко використовується для задач машинного навчання та глибокого навчання. Ось деякі з основних можливостей TensorFlow:

Гнучка система створення моделей:

- TensorFlow дозволяє легко створювати складні архітектури нейронних мереж, включаючи конволюційні нейронні мережі (CNN), рекурентні нейронні мережі (RNN), LSTM (Long Short-Term Memory) та багато інших.
- Підтримка керованих, некерованих, посиленого навчання та інших парадигм машинного навчання.

Виконання на різних пристроях:

- TensorFlow може використовуватися на різних пристроях: від мобільних пристроїв і одноплатних комп'ютерів до масштабних розподілених систем на серверах з GPU або TPU (Tensor Processing Units).

Бібліотека TensorFlow

Автоматичне диференціювання:

- TensorFlow автоматично вираховує градієнти, що є ключовим елементом для тренування нейронних мереж, за допомогою backpropagation. Це робить бібліотеку дуже зручною для розробки складних алгоритмів глибокого навчання.

Масштабованість:

- TensorFlow підтримує горизонтальне масштабування, що дозволяє обробляти великі набори даних і складні моделі з використанням розподілених обчислень.

Інструменти для візуалізації:

- TensorBoard — це інструмент для візуалізації, який дозволяє візуалізувати графи обчислень, показники тренування, розподіли параметрів та багато іншого, що може допомогти в аналізі та відладці моделей.

Бібліотека TensorFlow

Гнучкість і модульність:

- TensorFlow дозволяє легко модифікувати існуючі моделі та швидко експериментувати з новими ідеями. Це включає можливість змішувати низькорівневі та високорівневі API.

Екосистема і спільнота:

- TensorFlow має велику та активну спільноту розробників та дослідників, які постійно доповнюють екосистему новими інструментами, бібліотеками та ресурсами. Це включає інтеграцію з іншими платформами, такими як Keras для високорівневого API для нейронних мереж.

Підтримка мов програмування:

- Хоча основною мовою для роботи з TensorFlow є Python, бібліотека також підтримує інтерфейси для C++, Java, Go та інших мов, що робить її доступною для широкого спектру розробників.

Ці можливості роблять TensorFlow однією з найпопулярніших бібліотек для розробки та дослідження в галузі машинного навчання та штучного інтелекту.

Бібліотека Keras

Keras — це високорівнева нейронна мережева бібліотека, написана на Python. Вона була розроблена з метою забезпечення швидкого експериментування з глибинними нейронними мережами. Ось деякі з основних можливостей Keras:

1. **Інтеграція з TensorFlow**:
   - З випуском TensorFlow 2.0, Keras стала інтегрованою частиною TensorFlow, використовуючи його як свій основний бекенд. Це забезпечує щільну інтеграцію і оптимізацію для вирішення задач глибокого навчання.
2. **Вбудовані засоби для обробки даних**:
   - Keras надає утиліти для обробки даних, такі як зображення та текст, що дозволяє легко підготувати дані для тренування моделей.

Ці можливості роблять Keras дуже популярним вибором серед дослідників та розробників, які займаються машинним навчанням і глибоким навчанням, особливо для тих, хто цінує швидкість розробки та простоту використання.

Бібліотека PyTorch

PyTorch — це відкрита бібліотека машинного навчання, що базується на бібліотеці Torch. Вона розроблена головним чином командою штучного інтелекту Facebook. PyTorch широко використовується для досліджень та розробок у галузі штучного інтелекту, зокрема в глибинному навчанні.

Основні характеристики PyTorch:

Динамічний граф обчислень: На відміну від інших бібліотек, які використовують статичний граф обчислень (наприклад, TensorFlow), PyTorch використовує динамічний граф обчислень, що дозволяє більш гнучко модифікувати архітектуру моделі під час виконання програми.

Інтуїтивно зрозумілий інтерфейс: PyTorch має дуже чистий та зрозумілий API, що робить його популярним серед дослідників та розробників.

Підтримка GPU: PyTorch підтримує обчислення на графічних процесорах (GPU), що значно прискорює процес тренування моделей.

Багатий набір інструментів: PyTorch має широкий спектр інструментів та бібліотек, таких як TorchVision для роботи з зображеннями, TorchText для обробки тексту, і TorchAudio для аудіофайлів.

Сумісність з іншими бібліотеками Python: PyTorch добре інтегрується з бібліотеками Python, такими як NumPy та SciPy, а також з іншими інструментами для візуалізації, наприклад, Matplotlib.

Спільнота та підтримка: PyTorch має активну спільноту користувачів та розробників, яка постійно розширюється. Багато університетів та дослідницьких інститутів використовують

Бібліотека ML.NET

ML.NET – це безкоштовна, крос-платформна та відкрита бібліотека машинного навчання для розробки на мові C# та платформі .NET. Вона дозволяє додавати можливості машинного навчання в .NET застосунки. Ось деякі ключові особливості ML.NET:

1. Підтримка різних задач ML: класифікація, регресія, кластеризація, виявлення аномалій, рекомендаційні системи тощо.
2. Гнучкість у створенні ML конвеєрів: можливість комбінувати різні алгоритми, трансформації даних та моделі в єдиний конвеєр.
3. Автоматизоване машинне навчання (AutoML): автоматичний підбір найкращої моделі та гіперпараметрів для даної задачі.
4. Трансформації та обробка даних: широкий набір вбудованих трансформацій для попередньої обробки та підготовки даних.
5. Збереження та завантаження моделей: можливість зберігати навчені моделі та використовувати їх для передбачень.
6. Інтеграція з іншими фреймворками: сумісність з TensorFlow, ONNX, тощо для використання моделей з інших бібліотек.

Розгортання моделей ML.NET

Розгортання навчених моделей у реальних застосунках .NET.
Для початку роботи з ML.NET потрібно встановити пакет Microsoft.ML через NuGet. Потім можна створювати конвеєри машинного навчання, які включають в себе:

- Завантаження та трансформацію даних
- Вибір та налаштування алгоритму
- Навчання моделі
- Оцінку якості моделі
- Використання моделі для передбачень

ML.NET надає зручний API для всіх цих кроків та дозволяє швидко додавати можливості машинного навчання у .NET застосунки. Це потужний інструмент, який спрощує використання ML для .NET розробників.

Платформа ONNX

ONNX (Open Neural Network Exchange) – це відкритий формат для представлення моделей машинного навчання. Він був розроблений для забезпечення сумісності між різними фреймворками та інструментами машинного навчання.

Основна мета ONNX – дозволити розробникам та дослідникам використовувати моделі, навчені в одному фреймворку, в іншому фреймворку без необхідності повторного навчання або внесення значних змін у модель. Це спрощує процес переносу моделей між різними середовищами та платформами.

Ключові особливості ONNX:

Сумісність: ONNX підтримується багатьма популярними фреймворками машинного навчання, такими як PyTorch, TensorFlow, scikit-learn, Keras, та інструментами, такими як MATLAB і SAS.

Портативність: Моделі, представлені у форматі ONNX, можуть бути розгорнуті на різних платформах і пристроях, включаючи сервери, мобільні пристрої та навіть браузери.

Оптимізація: ONNX дозволяє оптимізувати моделі для конкретних середовищ виконання, що може покращити продуктивність та ефективність.

Версіонування: ONNX підтримує версіонування моделей, що дозволяє відстежувати зміни та забезпечувати сумісність між різними версіями.

Використання ONNX

Процес використання ONNX зазвичай включає наступні кроки:

- Навчання моделі в одному з підтримуваних фреймворків (наприклад, PyTorch або TensorFlow).
- Експорт навченої моделі у формат ONNX.
- Імпорт моделі ONNX в цільовий фреймворк або середовище виконання.
- Використання моделі для висновку або подальшого навчання в цільовому середовищі

ONNX відіграє важливу роль у спрощенні процесу розгортання та обміну моделями машинного навчання між різними платформами та інструментами, сприяючи співпраці та відтворюваності в галузі машинного навчання.