# ЛЕКЦІЯ 2

# МОДЕЛІ НЕЙРОННИХ МЕРЕЖ



## *1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА НЕЙРОННИХ МЕРЕЖ*

Нейронні мережі є ядром сучасного штучного інтелекту. Вони виникли як спроба імітувати роботу біологічного мозку, де мільярди нейронів утворюють складні мережі взаємодій. Математична модель штучного нейрона дозволила побудувати системи, здатні до навчання, узагальнення знань та адаптації.

Моделі нейронних мереж можна розглядати з кількох точок зору:

- **біологічна мотивація** – уподібнення до нервової системи;
- **математична модель** – формалізація вузлів і зв’язків у вигляді рівнянь;
- **алгоритмічний аспект** – методи навчання та оптимізації;
- **архітектурний аспект** – типи організації мережі.

## 3. Основні типи моделей нейронних мереж

### 3.1. Перцептрон Розенблатта (1958)

- Найпростіша модель: лінійна комбінація входів + порогова функція.

- Використовується для задач лінійної класифікації.
- Має обмеження (не розв’язує задачі XOR).

**Що таке задачі XOR?**

**XOR** – це скорочення від англ. **“exclusive OR”** (виключне АБО). Це логічна операція між двома бінарними змінними (0 або 1), яка визначається так:

| **Вхід A** | **Вхід B** | **Вихід A XOR B** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Тобто XOR дає **1 лише тоді, коли входи різні**, і 0, якщо вони однакові.

**Чому XOR важлива у нейронних мережах?**

1. **Перцептрон Розенблатта (одношаровий перцептрон)** може розв’язувати лише **лінійно роздільні задачі**.
    - Лінійно роздільна задача – це така, де можна провести пряму лінію (у 2D) чи площину (у nD), яка відокремлює класи.
    - Наприклад, AND або OR можна розділити прямою.
2. **XOR не є лінійно роздільною задачею**:
    - Якщо спробувати намалювати точки (0,0), (0,1), (1,0), (1,1) на площині, то неможливо провести одну пряму, яка відокремлює точки з виходом 1 від точок з виходом 0.
3. **Висновок:**
    - Одношаровий перцептрон **не може навчитися обчислювати XOR**.
    - Для розв’язання XOR потрібен **багатошаровий перцептрон (MLP)** з прихованим шаром і нелінійною активацією.

**3.2. Багатошаровий перцептрон (MLP)**

- Складається з шарів: *вхідного, прихованих і вихідного.*
- Використовує нелінійні *активаційні функції* (**sigmoid, ReLU, tanh**).
- Навчання – алгоритм зворотного поширення помилки (**backpropagation**).
- Використовується для класифікації, регресії, розпізнавання образів.

### 3.3. Рекурентні нейронні мережі (RNN)

- Мають зворотні зв’язки, що дозволяє враховувати часову динаміку.
- Застосування: обробка послідовностей (текст, мова, часоряд).
- Модифікації: LSTM, GRU – для подолання проблеми «згасання градієнта».

### 3.4. Конволюційні нейронні мережі (CNN)

- *Моделюють зорову кору мозку.*
- Використовують згортки та підвибірку (pooling).
- *Надзвичайно ефективні для зображень, відео, медичної діагностики.*

### 3.5. Самоорганізовувальні карти Кохонена (SOM)

- *Нейронні мережі без вчителя.*
- Формують багатовимірні кластери даних.
- Використовуються для візуалізації та кластеризації.

### 3.6. Глибинні нейронні мережі (Deep Neural Networks, DNN)

- *Мають багато прихованих шарів.*
- Використовують сучасні методи оптимізації (Adam, RMSprop).
- Є базою сучасного deep learning.

### 3.7. Генеративні моделі

- **Автокодери (Autoencoders)** – стискання та відновлення даних.
- **Варіаційні автокодери (VAE)** – генерація нових зразків.

- **Генеративно-змагальні мережі (GANs)** – протиставлення генератора та дискримінатора, створення реалістичних зображень, текстів, музики.

### 3.8. Трансформери

- Використовують механізм **self-attention**.
- Дають прорив у обробці природної мови (GPT, BERT).
- Застосовуються у чат-ботах, перекладах, генерації коду.

## 4. Класифікація моделей за принципом навчання

1. **З учителем (supervised learning)** – навчання на мічених даних.
2. **Без учителя (unsupervised learning)** – кластеризація, зменшення розмірності.
3. **З підкріпленням (reinforcement learning)** – агент взаємодіє із середовищем і навчається через винагороди.

## 5. Сучасні напрями розвитку моделей

- **Нейроморфні обчислення** – апаратні аналоги мозку.
- **Гібридні моделі** – комбінація нейромереж із класичними алгоритмами.
- **Обчислювально ефективні архітектури** – компактні моделі для мобільних пристроїв (MobileNet, TinyML).
- **Мультимодальні мережі** – здатність працювати одночасно з текстом, зображенням, аудіо.

## 6. Висновок

Моделі нейронних мереж пройшли шлях від простих перцептронів до глибинних і трансформерних архітектур, що змінюють науку, промисловість і повсякденне життя. Вибір моделі завжди залежить від задачі: класифікація, прогнозування, генерація чи управління.

# *2. ПЕРСЕПТРОН, БАГАТОШАРОВІ МЕРЕЖІ*

**Перцептрон (Perceptron)** — це найпростіша модель штучного нейрона, запропонована **Френком Розенблаттом у 1958 році**.

- Має **кілька входів** ($x_1$, $x_2$, …, xn).
- Кожен вхід має свою **вагу** ($w_1$, $w_2$, …, wn).
- Обчислюється **зважена сума**:

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

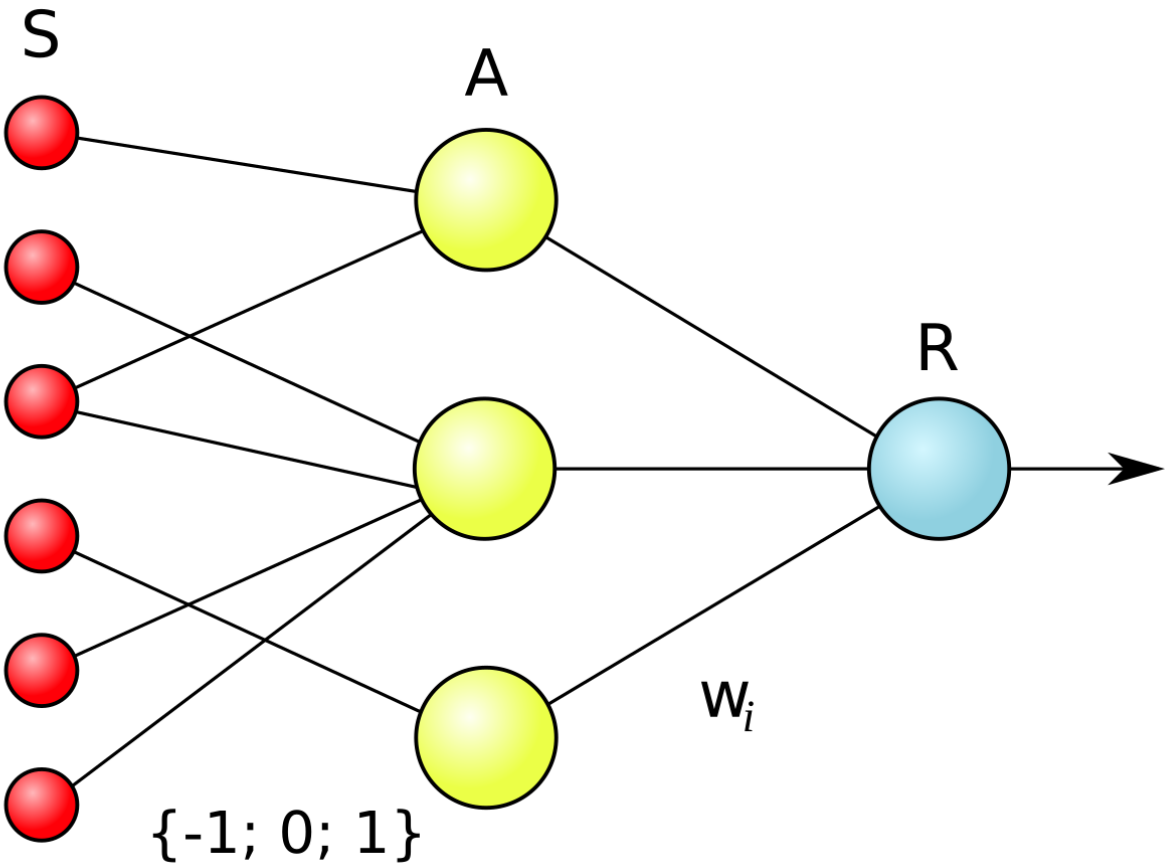
де $b$ — порогове значення (bias).

- До результату застосовується **функція активації** (наприклад, ступінчаста), яка вирішує, чи нейрон "вмикається".

$$y = f(u)$$

- Використовується для **лінійної класифікації** (поділ об’єктів на 2 класи).

Перцептрон може розв’язувати лише **лінійно роздільні задачі.**




Логічна схема елементарного перцептрону. Ваги зв'язків S-A можуть мати значення −1, 1 або 0 (тобто відсутність зв'язку). Ваги зв'язків A-R W можуть мати будь-яке значення. Якщо вага $w = -1$, це означає, що сигнал з входу **пригнічує** активацію нейрона.

*Елементарний перцептрон* складається з елементів трьох типів: **S-елементів**, **A-елементів** та *одного* **R-елементу**. **S-елементи** — це шар сенсорів, або рецепторів. У фізичному втіленні вони відповідають, наприклад, світлочутливим клітинам сітківки ока або фоторезисторам матриці камери. Кожен рецептор може перебувати в одному з двох станів — *спокою* або *збудження*, і лише в останньому випадку він передає одиничний сигнал до наступний шару, асоціативним елементам.

**A-елементи** називаються асоціативними, тому що кожному такому елементові, як правило, відповідає цілий набір (асоціація) **S-елементів**. **A-елемент** активізується, щойно кількість сигналів від S-елементів на його вході перевищує певну величину θ.

Сигнали від збуджених А-елементів, своєю чергою, передаються до суматора R, причому сигнал від і-го асоціативного елемента передається з коефіцієнтом $w_i$. Цей коефіцієнт називається *вагою* А-R зв'язку.

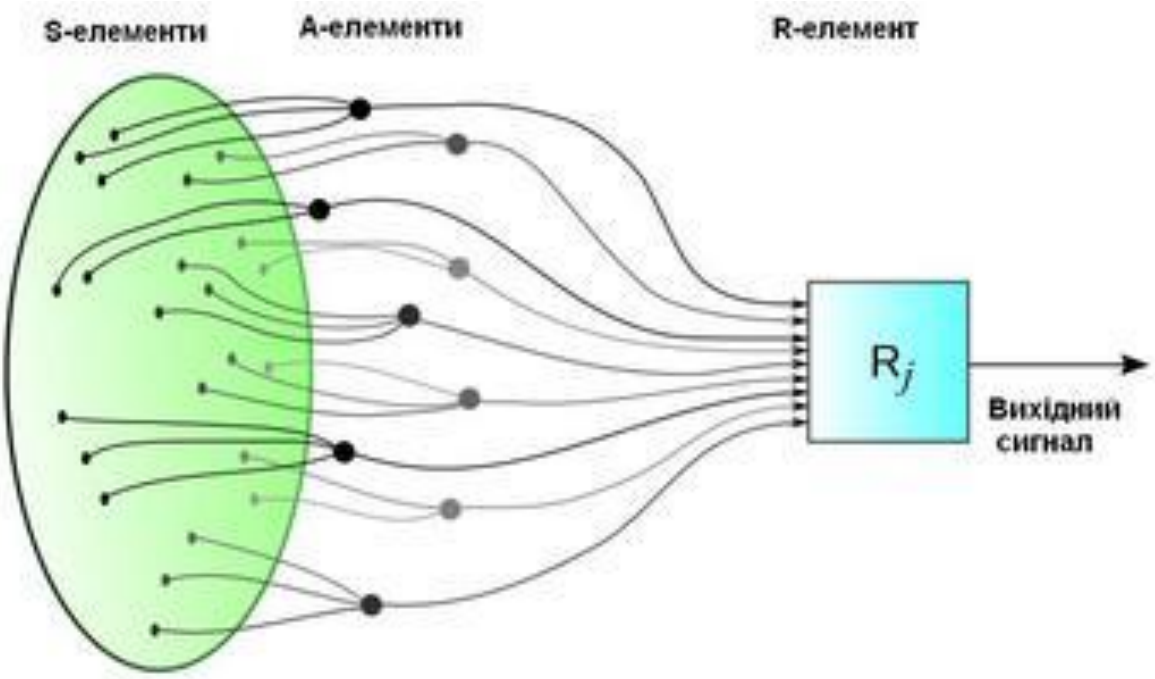
Так само як і **А-елементи**, **R-елемент** підраховує суму значень вхідних сигналів, помножених на ваги (лінійну форму). **R-елемент**, а разом з ним і елементарний перцептрон, видає «1», якщо лінійна форма перевищує поріг θ, інакше на виході буде «-1». Математично, функцію, що реалізує R-елемент, можна записати так:

$$f(x) = sign(\sum_{i=1}^{n} w_i x_i - \theta)$$

*Навчання елементарного перцептрона* полягає у *зміні вагових коефіцієнтів зв'язків А-R.* Ваги зв'язків S-A (які можуть приймати значення (-1; 0; 1)) і значення порогів **А-елементів** вибираються випадковим чином на самому початку і потім не змінюються.

Після навчання перцептрон готовий працювати в режимі *розпізнавання* або *узагальнення*. У цьому режимі перцептрону пред'являються раніше невідомі йому об'єкти, й він повинен встановити, до якого класу вони належать. Робота перцептрона полягає в наступному: при пред'явленні об'єкта, збуджені **А-елементи** передають сигнал **R-елементу**, що дорівнює сумі відповідних коефіцієнтів $w_i$ . *Якщо ця сума позитивна, то ухвалюється рішення, що даний об'єкт належить до першого класу, а якщо вона негативна — то до другого.*

Фізичний аналог перцептрона




Надходження сигналів із сенсорного поля до розв'язувальних блоків елементарного перцептрона в його фізичному втіленні.

**Висновок:**

Перцептрон — це "цеглинка", основа, з якої почався розвиток штучних нейронних мереж. Простий перцептрон може розв’язувати, зокрема, задачі типу AND. Що це означає на практиці?

Простий (одношаровий) перцептрон — це модель, яка отримує кілька входів $x_1, x_2, \ldots, x_n$, зважує їх коефіцієнтами $w_i$, додає зсув $b$ і застосовує **поріг (активацію)**.

$$y = \begin{cases} 1, & \text{якщо } \sum w_i x_i + b \geq 0 \\ 0, & \text{інакше} \end{cases}$$

Суть логічної операції **AND (логічне І)** така:

- Результат дорівнює **1 (істина)** тільки тоді, коли **обидва** вхідні значення дорівнюють 1.

- У всіх інших випадках результат **0 (хибність)**.

Можна представити це у вигляді **таблиці істинності**:

| $x_1$ | $x_2$ | $x_1$ AND $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Як перцептрон це робить?**

Виберемо ваги:

- $w_1 = 1$
- $w_2 = 1$
- поріг (або bias) $b = -1.5$

Тоді отримаємо:

$$u = w_1 x_1 + w_2 x_2 + b$$

Перевіримо:

- Для (0,0): $u = 0 + 0 - 1.5 = -1.5 < 0 \Rightarrow y = 0$
- Для (0,1): $u = 0 + 1 - 1.5 = -0.5 < 0 \Rightarrow y = 0$
- Для (1,0): $u = 1 + 0 - 1.5 = -0.5 < 0 \Rightarrow y = 0$
- Для (1,1): $u = 1 + 1 - 1.5 = 0.5 \geq 0 \Rightarrow y = 1$

---

Отримали правильну логіку AND.

Аналогічно для логіки OR:

| $x_1$ | $x_2$ | $x_1 \vee x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

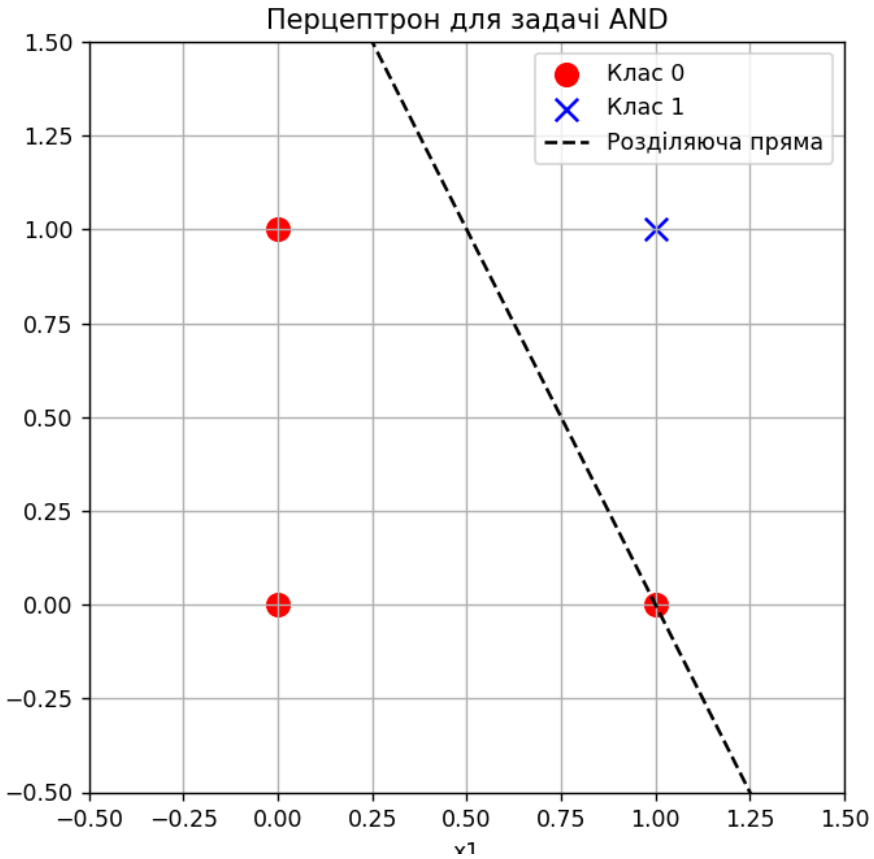
## [AI PR 2-1]

Програма демонструє роботу простого перцептрона на прикладі задачі AND

```
import numpy as np
import matplotlib.pyplot as plt

# Функція активації
def step_function(x):
    return 1 if x >= 0 else 0

# Клас перцептрона
class Perceptron:
    def __init__(self, input_size,
learning_rate=0.1):
        self.weights = np.zeros(input_size + 1)   #
+1 для bias
        self.learning_rate = learning_rate

    def predict(self, x):
        x = np.insert(x, 0, 1)  # додаємо bias=1
        weighted_sum = np.dot(self.weights, x)
        return step_function(weighted_sum)

    def train(self, X, y, epochs=10):
        for _ in range(epochs):
            for xi, target in zip(X, y):
                xi = np.insert(xi, 0, 1)   #
додаємо bias
```

```
                prediction =
step_function(np.dot(self.weights, xi))
                error = target - prediction
                self.weights += self.learning_rate
* error * xi

# Дані (логічна операція AND)
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([0, 0, 0, 1])

# Створюємо і тренуємо перцептрон
p = Perceptron(input_size=2)
p.train(X, y, epochs=10)

# Вивід результатів
print("Результати після навчання:")
for xi in X:
    print(f"{xi} -> {p.predict(xi)}")

# Візуалізація
plt.figure(figsize=(6,6))

# Точки
for xi, label in zip(X, y):
    if label == 0:
        plt.scatter(xi[0], xi[1], color='red',
marker='o', s=100, label="Клас 0" if "Клас 0" not
in plt.gca().get_legend_handles_labels()[1] else
"")
    else:
        plt.scatter(xi[0], xi[1], color='blue',
marker='x', s=100, label="Клас 1" if "Клас 1" not
in plt.gca().get_legend_handles_labels()[1] else
"")

# Побудова розділяючої прямої
x_vals = np.linspace(-0.5, 1.5, 100)
# рівняння: w0*1 + w1*x1 + w2*x2 = 0 => x2 = -(w0
+ w1*x1)/w2
w = p.weights
if w[2] != 0:  # щоб уникнути ділення на нуль
```

```
    y_vals = -(w[0] + w[1]*x_vals) / w[2]
    plt.plot(x_vals, y_vals, 'k--',
label="Розділяюча пряма")

plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Перцептрон для задачі AND")
plt.xlim(-0.5, 1.5)
plt.ylim(-0.5, 1.5)
plt.legend()
plt.grid(True)
plt.show()
```



*БАГАТОШАРОВІ МЕРЕЖІ (Multi-Layer Perceptrons, MLP).*

Одношаровий перцептрон (той, що ми вже розглянули) може розв'язувати лише **лінійно роздільні задачі** (наприклад, AND, OR). Але він не може навчитися задачам типу **XOR**, де немає однієї прямої, яка розділить класи.

Вирішення — використати **декілька шарів нейронів**, тобто багатошарову мережу. *Багатошарові мережі (MLP) стали першим реальним інструментом для вирішення складних задач машинного навчання.*

**Переваги MLP:**

- Може розв'язувати задачі класифікації, регресії, розпізнавання образів, прогнозування.
- Подолала обмеження простого перцептрона.

**Недоліки:**

- Для глибоких мереж потрібні великі дані й обчислювальні ресурси.
- Схильність до перенавчання.

**Порівняння**

| Ознака | Перцептрон | Багатошарова мережа (MLP) |
|---|---|---|
| Архітектура | Один шар | Кілька шарів |
| Функції активації | Лінійна/ступінчаста | Нелінійні (sigmoid, ReLU тощо) |
| Складність задач | Лінійно роздільні | Складні, нелінійні |
| Приклади | простий класифікатор | розпізнавання цифр, мови, прогнозування |

## Архітектура MLP

Типова багатошарова мережа складається з:

1. **Вхідний шар**
    - приймає дані (наприклад, 2 входи для задачі XOR).
2. **Приховані шари**
    - один або більше; кожен нейрон виконує зважену суму та нелінійну активацію.
    - саме вони дозволяють моделі «захоплювати» складні залежності.
3. **Вихідний шар**
    - видає результат (наприклад, класифікація 0 або 1).

## Активаційні функції

Без них багатошарова мережа поводилась би як один «великий лінійний перцептрон».
Популярні активації:

**Sigmoid**:

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

**tanh**:

$$f(x) = \tanh(x)$$

**ReLU**:

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

*Активаційна функція* — це *нелінійне перетворення,* яке застосовується після обчислення зваженої суми вхідних сигналів нейрона:

$$y = f(\sum_i w_i x_i + b)$$

**Основні причини:**

1. **Внесення нелінійності**
   - Якщо в MLP **не застосовувати активаційні функції**, то вся мережа (навіть із багатьма шарами) зводиться до одного лінійного перетворення:

     $$W_2(W_1 x) = W_2 W_1 X$$

     Тобто результат можна було б отримати одразу без прихованих шарів.

   - Нелінійність дозволяє мережі моделювати **складні залежності**.
2. **Можливість розв'язувати складні задачі**
   - Без активації перцептрон може реалізувати лише **лінійно відокремлювані функції** (AND, OR).
   - Завдяки активаційним функціям MLP може навчитися логіці XOR, розпізнаванню образів, прогнозуванню тощо.
3. **Нормування та стабілізація**
   - Деякі функції (наприклад, **sigmoid** або **tanh**) стискають вихід у діапазон [0,1][0,1][0,1] або [−1,1][-1,1][−1,1].
   - Це допомагає уникати надто великих значень і робить навчання стабільнішим.
4. **Біологічна аналогія**
   - У мозку нейрони також спрацьовують нелінійно: або «збуджуються», або «пригнічуються» після досягнення певного порогу.

Функція $f(x) = \max(0, x)$ бере два числа — 0 та x — і вибирає більше:

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

Це "ламана лінія", яка йде вздовж осі x до нуля, а потім — піднімається вгору під кутом 45°.

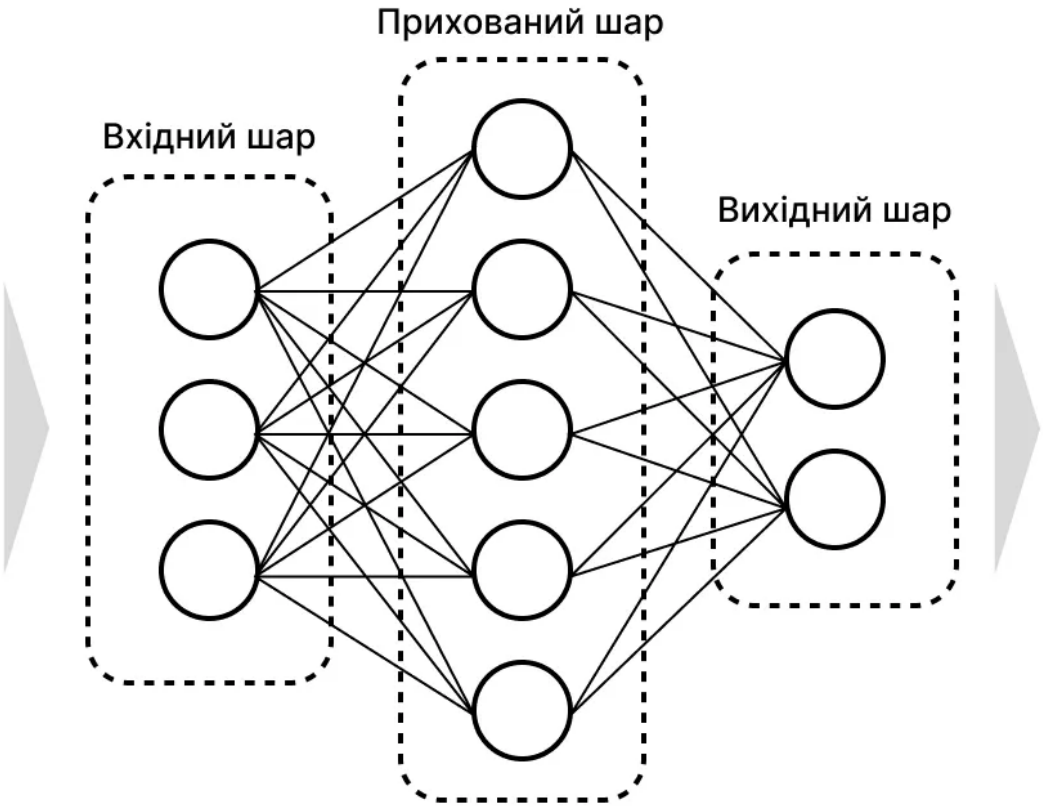
## Навчання MLP

Метод називається **зворотне поширення помилки (backpropagation)**:

1. **Прямий прохід**: дані проходять через усі шари → отримуємо прогноз.
2. **Обчислення похибки**: різниця між прогнозом і правильним значенням.
3. **Зворотний прохід**: похибка поширюється назад через шари, коригуючи ваги за допомогою *градієнтного спуску.*

MLP складається щонайменше з трьох шарів взаємопов’язаних вузлів, які називаються нейронами:

- **Вхідний шар**: Отримує початкові дані, наприклад, пікселі зображення або слова речення.
- **Прихований шар (шари)**: це робочі конячки мережі, де відбуваються складні обчислення та вилучення ознак. Прихованих шарів може бути один або декілька, кожен з яких має різну кількість нейронів.
- **Вихідний шар**: Видає остаточний прогноз або результат, заснований на обробці, виконаній на попередніх шарах.




## Приклад: XOR

Одношаровий перцептрон не розв’яже XOR.

Але MLP із **одним прихованим шаром із 2 нейронами** вже може!

Таблиця істинності XOR:

| $x_1$ | $x_2$ | $x_1$ **XOR** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

***Навіщо потрібен прихований шар (Hidden layer) в MLP?***

**1. Подолання обмежень одношарового перцептрона**

- Одношаровий перцептрон може розділити тільки **лінійно роздільні задачі** (наприклад, AND, OR).

- Але задачі на кшталт **XOR**, розпізнавання образів, мовлення, прогнозування часу — **нелінійні**.
- Прихований шар вводить **нелінійність** і дозволяє будувати складні залежності.

**2. Перетворення ознак**

Приховані нейрони створюють **нові представлення даних**.

- Вхід: «сирі» дані (пікселі зображення, значення сенсорів тощо).
- Прихований шар: комбінує їх у більш осмислені патерни (наприклад, контури, форми).
- Вихідний шар: працює вже з цими узагальненими ознаками.

**3. Приклади**

- **XOR**: немає однієї прямої, що розділить класи. Прихований шар фактично "розгортає" простір так, що класи стають роздільними.
- **Зображення**: приховані шари першими вчаться знаходити прості лінії, потім — контури, і лише потім — складні об’єкти.

**4. Формулювання**

Можна сказати так:

- **Без прихованого шару** — мережа ≈ проста лінійна модель.
- **З прихованими шарами** — мережа ≈ універсальний апроксиматор, здатний відобразити будь-яку функцію.

**[AI LEC 2-2]**

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score

# 1. Створимо простий датасет (кола vs квадрати)
# ознаки: [розмір, округлість]
X = np.array([
    [5, 1],    # маленьке коло
    [10, 1],   # велике коло
    [6, 0],    # маленький квадрат
    [12, 0],   # великий квадрат
    [7, 0.9], # майже коло
    [8, 0.1], # майже квадрат
])

# мітки: 0 = квадрат, 1 = коло
y = np.array([1, 1, 0, 0, 1, 0])

# 2. Розбиваємо на тренувальні та тестові дані
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=42)

# 3. Створюємо MLP
mlp = MLPClassifier(hidden_layer_sizes=(5,),
activation='relu', max_iter=1000, random_state=42)

# 4. Навчання
mlp.fit(X_train, y_train)

# 5. Перевірка
y_pred = mlp.predict(X_test)
print("Передбачення:", y_pred)
print("Точність:", accuracy_score(y_test, y_pred))
```

*РЕЗУЛЬТАТ*

*Передбачення: [1 0]*
*Точність: 0.5*

# *3. ТРАНСФОРМЕРИ*

**Трансформери (Transformers)**

**Трансформер** — *це архітектура нейронних мереж*, запропонована в статті *"Attention is All You Need"* (Vaswani et al., 2017). Вона замінила рекурентні (RNN, LSTM) та *згорткові (CNN)* підходи для обробки послідовностей і стала основою сучасних моделей, як-от **GPT, BERT, T5, LLaMA**.

**Основні ідеї**

1. **Механізм уваги (Attention)**
   - Дозволяє моделі "зосередитися" на різних частинах вхідної послідовності.
   - Наприклад, у реченні *“The cat sat on the mat”* — слово *“cat”* пов’язане із *“sat”*, а не з *“mat”*.
   - Ключова формула **Scaled Dot-Product Attention**:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

де $Q$ – запити (queries), $K$ – ключі (keys), $V$ – значення (values).

2. **Self-Attention**
   - Кожне слово в реченні аналізується у контексті всіх інших.
   - Це дозволяє враховувати далекі залежності (на відміну від RNN, які "забувають").

3. **Паралельна обробка**
    - На відміну від RNN, трансформери обробляють усю послідовність одразу, а не покроково.
    - Це різко підвищує швидкість навчання.
4. **Механізм "Encoder–Decoder"**
    - **Encoder** (кодувальник) — перетворює вхідні дані у приховане представлення.
    - **Decoder** (декодувальник) — генерує вихід (наприклад, переклад).

**Переваги трансформерів**

- Можуть моделювати довгі залежності.
- Легко масштабуються на великі дані та обчислювальні кластери.
- Лягли в основу **великих мовних моделей (LLM)**, включаючи ChatGPT.

**Приклади застосувань**

- **Обробка природної мови (NLP):** переклад, чат-боти, резюмування текстів.
- **Комп'ютерний зір:** візуальні трансформери (ViT) для класифікації зображень.
- **Біоінформатика:** моделювання білкових структур (AlphaFold).
- **Мультимодальні системи:** поєднання тексту, зображень, звуку (наприклад, GPT-5).

**Приведемо програму, як використовує *згорткової нейронної мережі (CNN).***
***Конкретно ця програма виконує такі дії:***

- *будує **згорткову нейронну мережу (CNN)**,*
- *навчає її на MNIST (5 епох достатньо для точності >99%),*
- *показує графіки точності/втрат,*
- *будує матрицю плутанини,*

- *будує зображення цифри.*

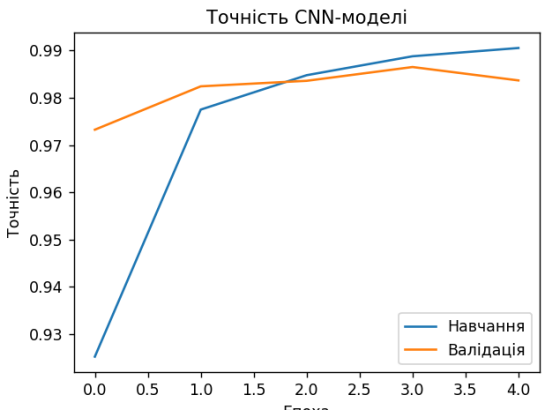
***Для розпізнавання об’єктів у даній програмі використовується Modified National Institute of Standards and Technology database (MNIST)*** *— це базовий набір зображень рукописних цифр, який дозволяє швидко навчити та перевірити алгоритми комп’ютерного зору.*

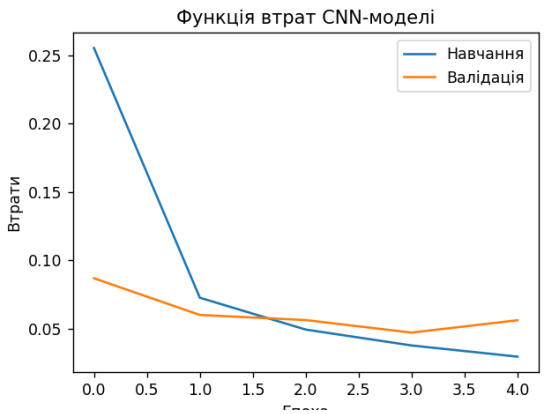
**[KMM LW 1-1]**

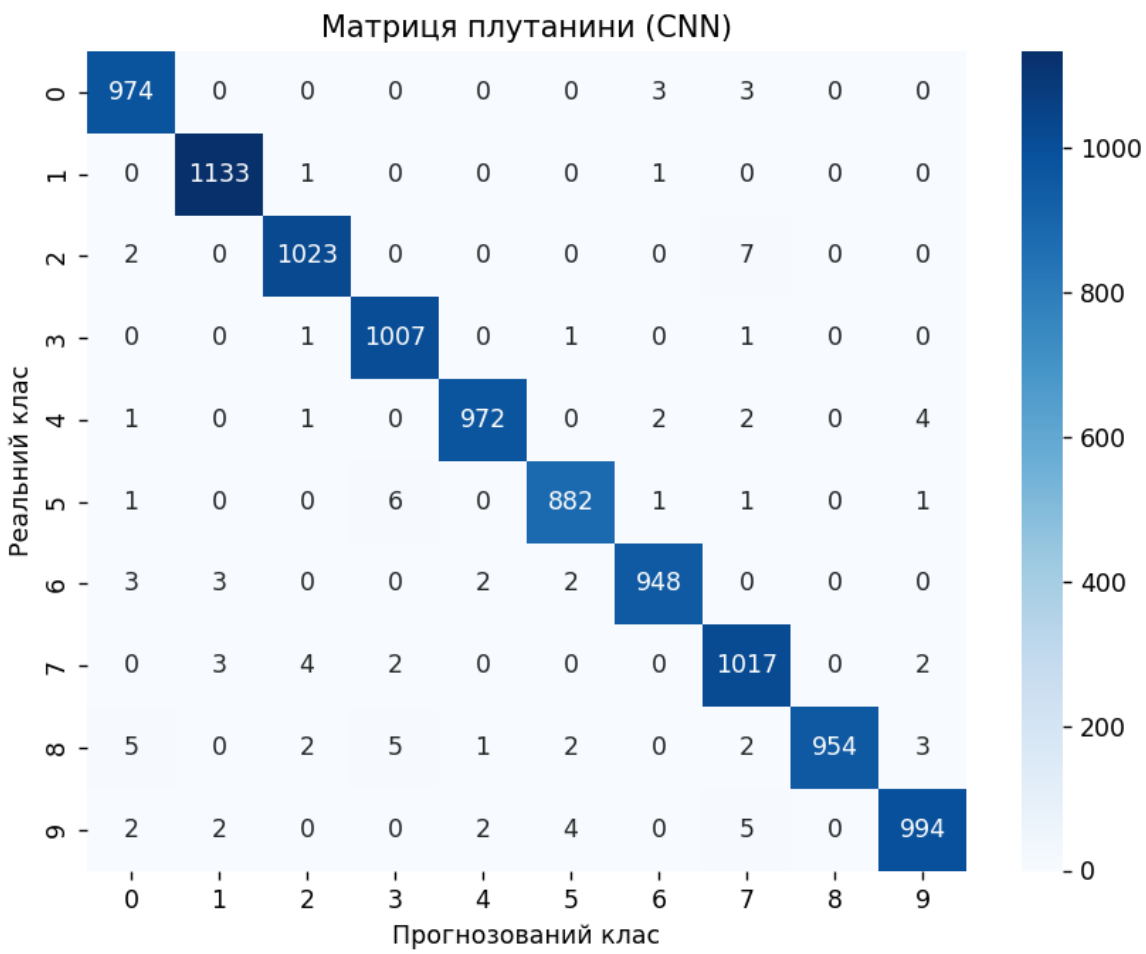
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from tensorflow import keras
from tensorflow.keras import layers

# 1. Завантаження датасету MNIST
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()

# Масштабування у [0,1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Перетворення у формат (кількість, висота, ширина,
канали)
x_train = np.expand_dims(x_train, -1)    # (60000, 28, 28,
1)
x_test = np.expand_dims(x_test, -1)      # (10000, 28, 28,
1)

# One-hot кодування міток
y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)

# 2. Побудова CNN-моделі
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3),
activation="relu", input_shape=(28, 28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3),
activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

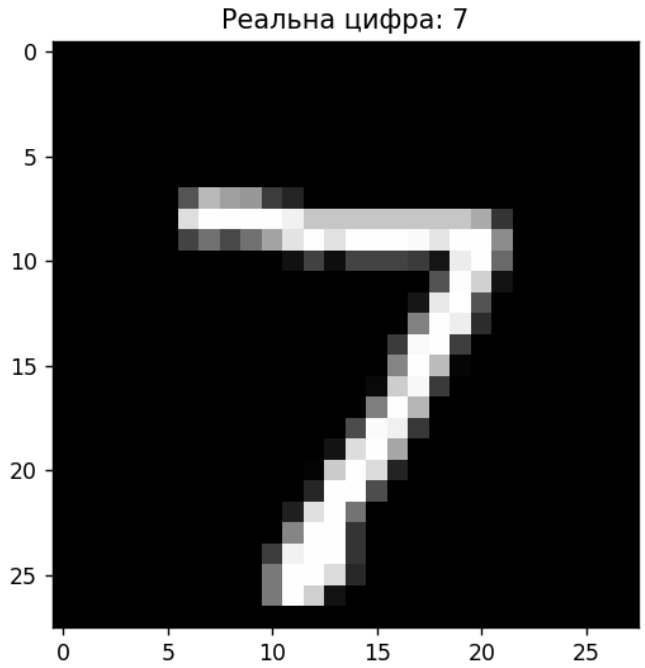
```
# ------------------------------
# 3. Компіляція моделі
# ------------------------------
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# ------------------------------
# 4. Навчання моделі
# ------------------------------
history = model.fit(
    x_train, y_train_cat,
    epochs=5,
    batch_size=128,
    validation_split=0.2,
    verbose=2
)

# ------------------------------
# 5. Оцінка на тестових даних
# ------------------------------
test_loss, test_acc = model.evaluate(x_test, y_test_cat,
verbose=0)
print(f"\nТочність на тестових даних: {test_acc:.4f}")

# ------------------------------
# 6. Графіки точності та втрат
# ------------------------------
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="Навчання")
plt.plot(history.history["val_accuracy"],
label="Валідація")
plt.title("Точність CNN-моделі")
plt.xlabel("Епоха")
plt.ylabel("Точність")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="Навчання")
plt.plot(history.history["val_loss"], label="Валідація")
plt.title("Функція втрат CNN-моделі")
plt.xlabel("Епоха")
plt.ylabel("Втрати")
```

```
plt.legend()

plt.show()

# ------------------------------
# 7. Матриця плутанини
# ------------------------------
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)

cm = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Матриця плутанини (CNN)")
plt.xlabel("Прогнозований клас")
plt.ylabel("Реальний клас")
plt.show()

# ------------------------------
# 8. Приклад розпізнавання
# ------------------------------
index = 0
plt.imshow(x_test[index].reshape(28, 28), cmap="gray")
plt.title(f"Реальна цифра: {y_test[index]}")
plt.show()

pred = model.predict(x_test[index:index+1])
print("Ймовірності класів:", pred)
print("Розпізнана цифра:", np.argmax(pred))
```



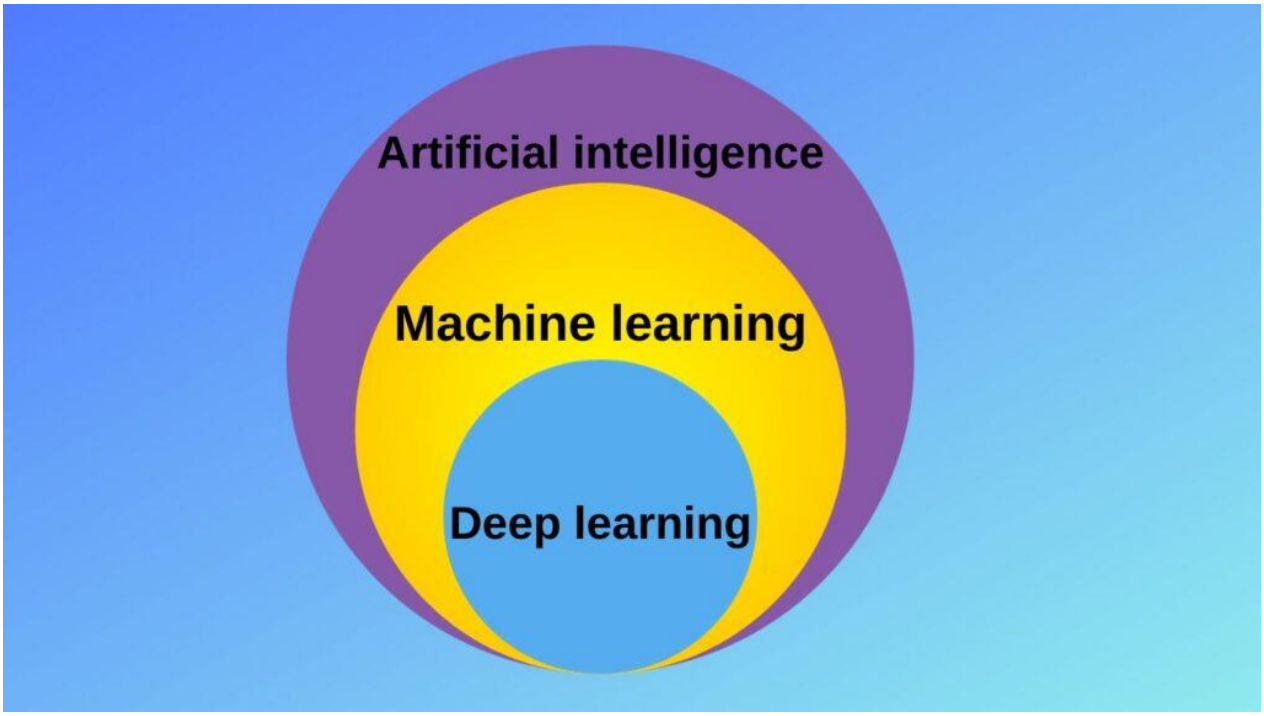






**Висновок:**

Трансформери — це фундамент сучасного AI. Вони дозволили створити системи, які перевершують людину в багатьох завданнях обробки мови й даних.




# *4. АНАЛІЗ АНГЛОМОВНИХ ВИДАНЬ*

## Відомі англомовні журнали з ШІ

**1. Nature Machine Intelligence**

- Щомісячний журнал від Nature Portfolio, охоплює машинне навчання, штучний інтелект і робототехніку.

*artificial intelligence*

*Unmanned Aerial Vehicle (UAV)*

**2. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)**

- Щомісячний журнал IEEE Computer Society, фокусується на комп’ютерному зорі, розпізнаванні образів, машинному навчанні.
- Імпакт-фактор 2023 року — 20.8

- Часто згадується як один із найпрестижніших у своїй галузі Successful StudentPublishingState.comReddit.

**3. Artificial Intelligence (Elsevier)**

- Один із найстаріших і найпрестижніших журналів у ШІ.
- У 2024 році імпакт-фактор — близько 14.05. Журнал охоплює теорію, методи, додатки ШІ (логіка, планування, NLP тощо) jstor.onlineslogix.in.

**4. Journal of Artificial Intelligence Research (JAIR)**

- Відкритий доступ, високий рівень рецензування, широка тематика (робототехніка, знання, багатоагентні системи, NLP) Ecommerce Paradiserestack.iojstor.online.
- Імпакт-фактор (2024) — приблизно 8.71 jstor.online.

**5. Journal of Machine Learning Research (JMLR)**

- Відкритий доступ, охоплює всі аспекти машинного навчання (алгоритми, теорія, практичні впровадження) Ecommerce Paradiserestack.io.

**6. Machine Learning (Springer)**

- Публікує як теоретичні, так і прикладні дослідження з машинного навчання restack.iotheaimatter.com.

**7. Neurocomputing (Elsevier)**

- Фокусується на нейромережах, машинному навчанні й нейрокомп’ютинг.
- Імпакт-фактор (2023) — 5.5

**8. Artificial Intelligence Review**

- Оглядові статті, комплексні аналітичні огляди в ШІ. Імпакт-фактор — близько 12 (за іншими джерелами варіюється) slogix.intheaimatter.comResearch Journals.

**9. Minds and Machines**

- Академічний журнал, що поєднує ШІ, філософію та когнітивні науки.
- Імпакт-фактор (2022) — 7.4

**10. Cognitive Computation**

- Перехресна дисципліна між біонауками, обчислювальними науками, ШІ, нейронауками.
- Імпакт-фактор (2020) — близько 5.418

*Наукові конференції залишаються дуже впливовими у галузі ШІ, однак журнали — важливий шлях для акредитованої наукової публікації (особливо для університетів)*

## *5. ДРОНИ ЗІ ШТУЧНИМ ІНТЕЛЕКТОМ*

Дрони зі штучним інтелектом (AI-powered drones) — це безпілотні літальні апарати (БПЛА), у яких вбудовані алгоритми машинного навчання, комп'ютерного зору, нейронних мереж і систем автономного прийняття рішень. Вони можуть не лише виконувати заздалегідь запрограмовані маршрути, але й **самостійно аналізувати середовище та адаптувати поведінку**.

**Основні можливості дронів з AI**

1. **Комп'ютерний зір**
    - Розпізнавання об'єктів (людей, автомобілів, будівель, техніки).
    - Виявлення перешкод і уникнення зіткнень.
    - Автоматичне відстеження рухомих цілей.
2. **Автономна навігація**
    - Планування маршруту без GPS (на основі камер, LIDAR, сенсорів).

    - Побудова карти місцевості (SLAM — Simultaneous Localization and Mapping).
    - Самостійна адаптація до зміни умов (вітер, перешкоди, нові маршрути).
3. **Аналіз даних у реальному часі**
    - Передача відео та його обробка на борту (edge AI).
    - Класифікація та виявлення аномалій (наприклад, дим, витік газу, пошкоджена інфраструктура).
4. **Взаємодія у групі (swarm intelligence)**
    - Рій дронів може координувати дії, обмінюватися інформацією.
    - Використання колективного алгоритму для пошуку, розвідки чи доставки.

**Приклади застосувань**

- **Військова сфера**: розвідка, спостереження, виявлення та ураження цілей.
- **Цивільні задачі**: доставка товарів (Amazon Prime Air, Wing), інспекція інфраструктури, пошук і рятування.
- **Екологія**: моніторинг лісів, сільськогосподарських полів, викидів.
- **Безпека**: патрулювання територій, контроль масових заходів.

**Використовувані AI-технології**

- **Deep Learning (глибинні нейронні мережі)** — для розпізнавання об’єктів і прийняття рішень.
- **Reinforcement Learning (підкріплене навчання)** — для оптимізації польоту та дій у змінному середовищі.
- **Transformers та LLM** — для аналізу великих потоків даних (наприклад, радіолокаційних сигналів чи повідомлень).
- **Swarm Intelligence** — моделювання поведінки групи за аналогією з мурахами чи птахами.

В англомовній науковій літературі вживають такі терміни:

- **AI drones**
- **intelligent UAVs (Unmanned Aerial Vehicles)**
- **autonomous drones**
- **smart drones**

Дрони зі штучним інтелектом (AI-drones, intelligent UAVs) мають специфічні конструктивні особливості, які відрізняють їх від класичних БПЛА. Вони створюються не лише як літальні апарати, а як **автономні кіберфізичні системи**.

## Особливості конструкції дронів із ШІ

### 1. Апаратна частина

- **Процесорні модулі з підтримкою AI**
  - GPU (NVIDIA Jetson, Intel Movidius, Google Coral TPU) для швидкої обробки даних.
  - ARM-процесори з низьким енергоспоживанням.
- **Багатосенсорна система**
  - Камери RGB, ІЧ (інфрачервоні), стереокамери, тепловізори.
  - Лідар (LiDAR) або радар для 3D-картографії.
  - Ультразвукові датчики для уникнення зіткнень.
  - IMU (інерційний модуль), GPS/GLONASS, барометр для стабілізації.
- **Комунікаційні модулі**
  - 5G/4G, Wi-Fi, LoRa, супутниковий зв’язок для роботи у рої чи з центром керування.

## 2. Програмне забезпечення

- **Операційні системи реального часу (RTOS)** або спеціальні прошивки (PX4, Ardupilot).
- **AI-алгоритми onboard**:
  - комп'ютерний зір (object detection, tracking);
  - reinforcement learning (адаптація траєкторії);
  - swarm intelligence (узгоджені дії групи).
- **Edge AI**: *обробка даних на борту без потреби постійного зв'язку з сервером.*

## 3. Енергосистема

- Використання **високоємних Li-Po/Li-ion батарей**.
- Розробки в напрямку **сонячних панелей** або **гібридних двигунів** для тривалих місій.
- Оптимізація енергоспоживання за рахунок AI (динамічне керування навантаженням).

## 4. Конструктивні рішення

- **Модульність** — можливість швидко замінювати сенсори чи вантажний відсік.
- **Легка та міцна рама** (карбон, композити) для збільшення вантажопідйомності та тривалості польоту.
- **Адаптивна аеродинаміка** — складні лопаті, змінні конфігурації (трансформовані квадрокоптери/літаки).
- **Розумна підвіска (gimbal)** — автоматичне стабілізування камер за допомогою AI.

## 5. Функціональні особливості

- Автономне ухвалення рішень без оператора.
- Навчання на польоті: поступове вдосконалення алгоритмів.
- Можливість **спільної роботи у рої** — розподілена система ШІ між кількома дронами.
- Інтеграція з **хмарними сервісами та Big Data** для аналітики.

В результаті дрони зі ШІ стають **самостійними розумними системами**, які здатні не просто літати, а **аналізувати, приймати рішення, взаємодіяти з іншими апаратами та зменшувати навантаження на людину-оператора**.