



Нейронні мережі

Лекція 2: Багатошаровий перцептрон

Кочура Юрій Петрович
iuriy.kochura@gmail.com
[@y_kochura](https://twitter.com/y_kochura)



Смартфони



Роботи



Медицина

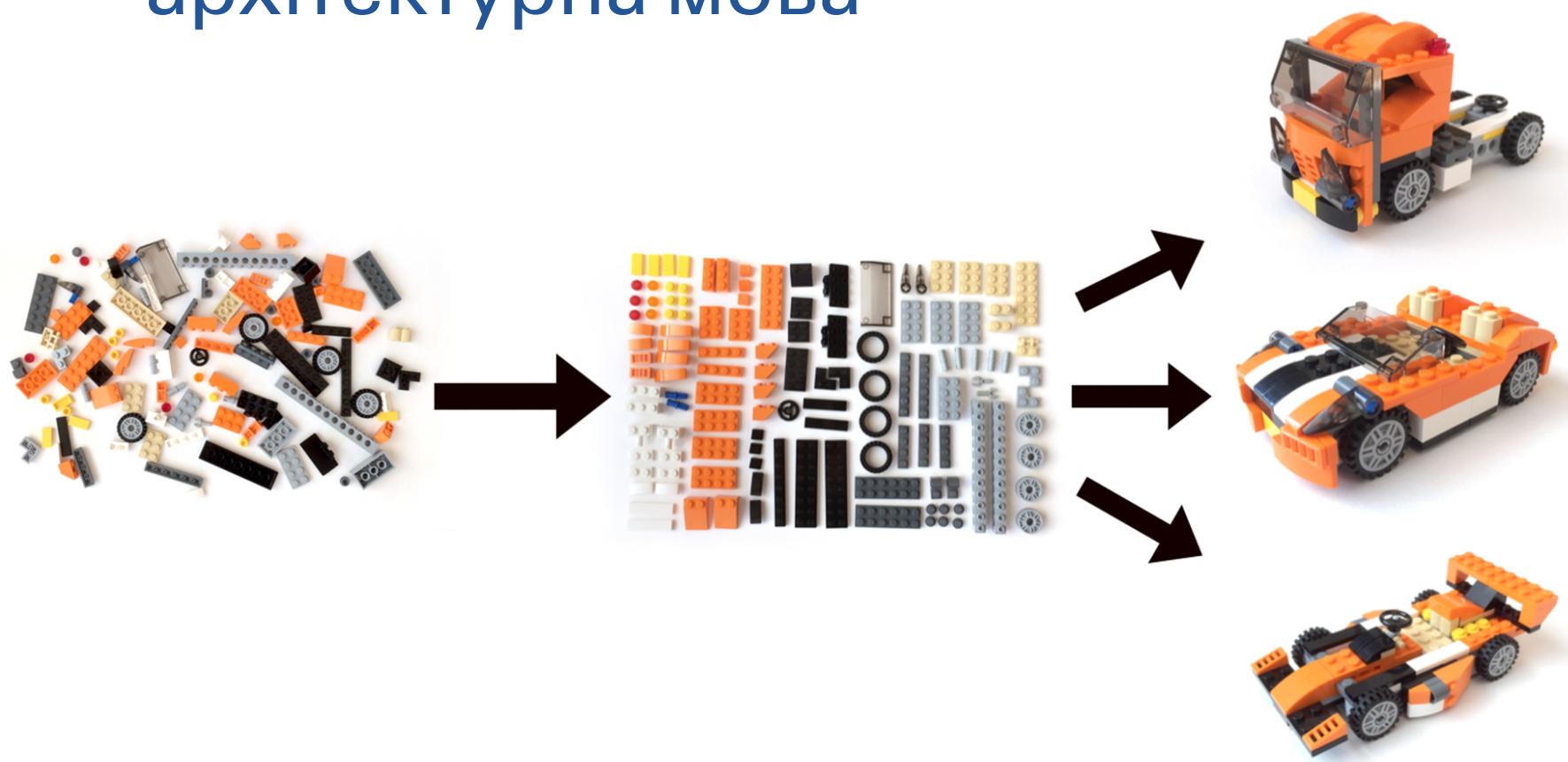


Дрони



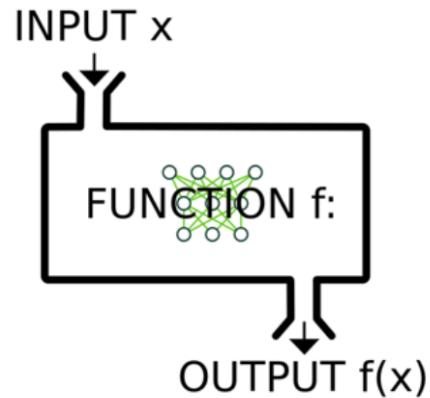
Автономні автомобілі

Глибоке навчання як архітектурна мова

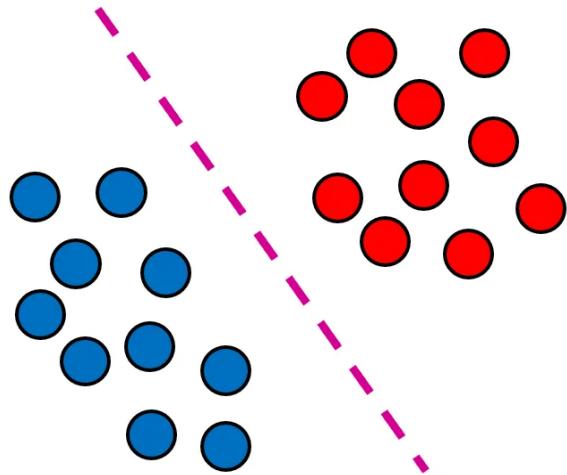


Що таке модель?

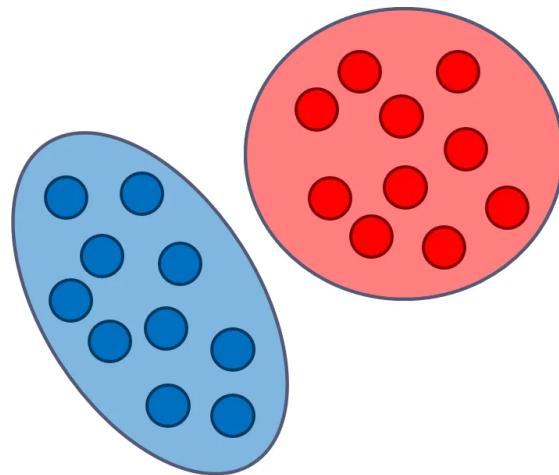
Хоча те, що знаходиться всередині глибинної нейронної мережі, може бути складним, за своєю суттю це просто функції. Вони беруть певні вхідні дані: **INPUT x** і генерують деякі вихідні дані: **OUTPUT f(x)**



Дискримінаційні vs генеративні моделі



Дискримінаційна модель



Генеративна модель

Сьогодні

- Microphone icon: Одношарова нейронна мережа: пряме поширення
- Microphone icon: Одновимірний градієнтний спуск
- Microphone icon: Одношарова нейронна мережа: зворотне поширення
- Microphone icon: Перцептрон з кількома виходами
- Microphone icon: Багатошаровий перцептрон

Перцептрон

Одношарова нейронна мережа: пряме поширення

Модель одного штучного нейрона

Перцептрон vs Логістична регресія

Перцептрон

Перцептрон (Френк Розенблат, 1957)

$$g(z) = \begin{cases} 1 & \text{if } z = \sum_i w_i x_i + b \geq 0 \\ 0 & \text{else} \end{cases}$$

Ця модель спочатку була мотивована біологією, де w_i – це синаптичні ваги для вхідних сигналів x_i та g активація.

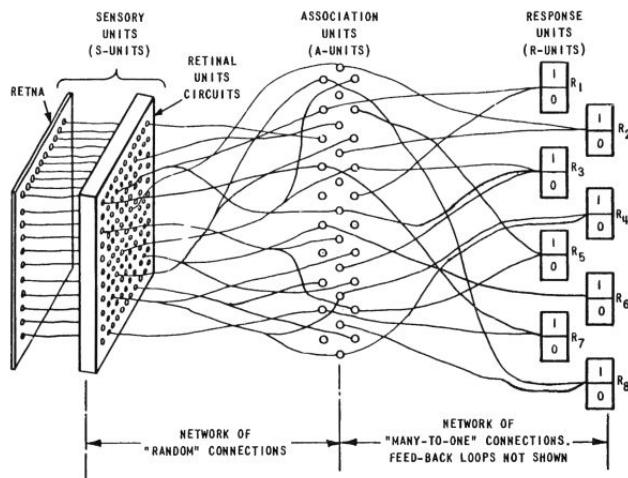
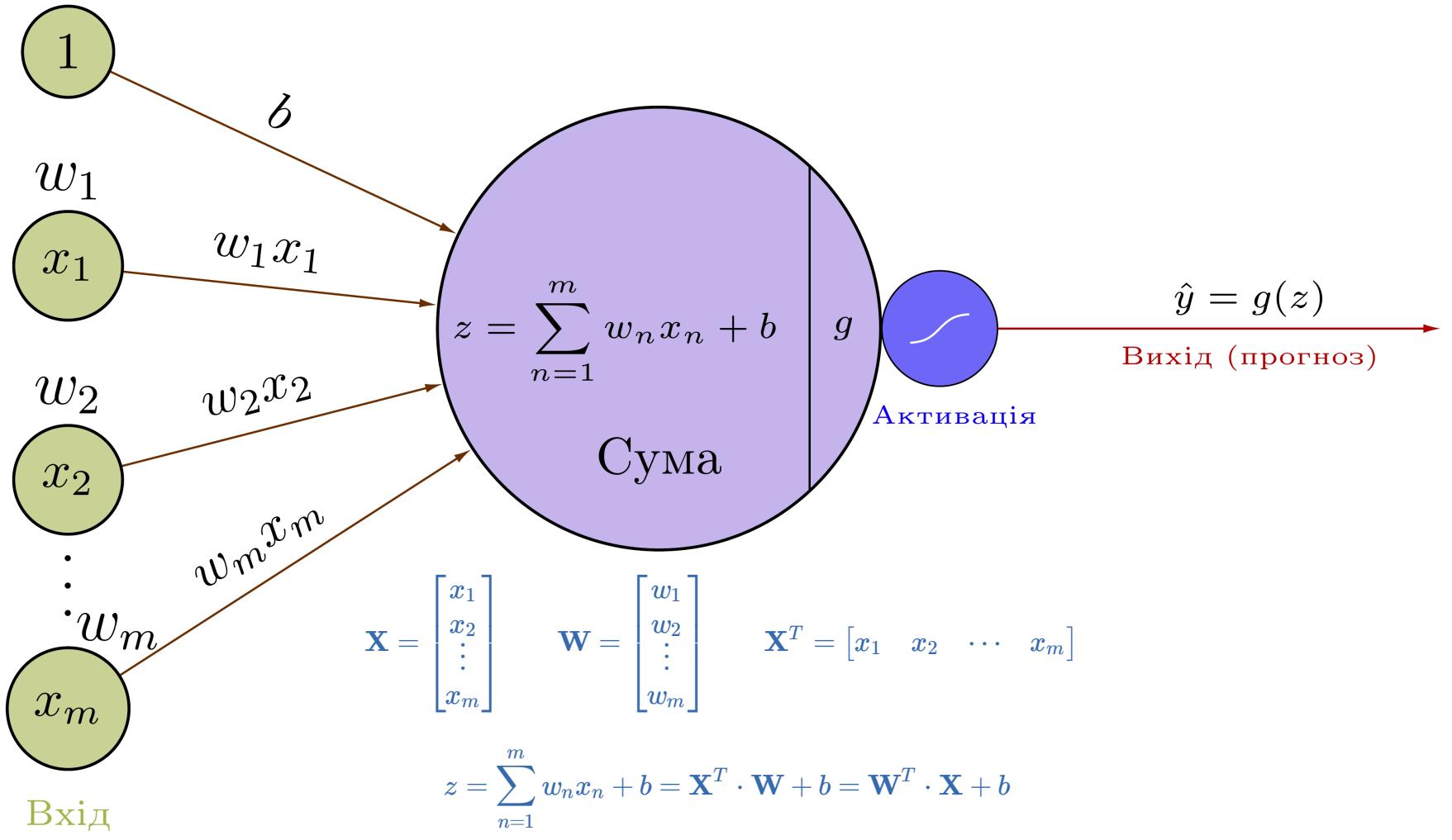


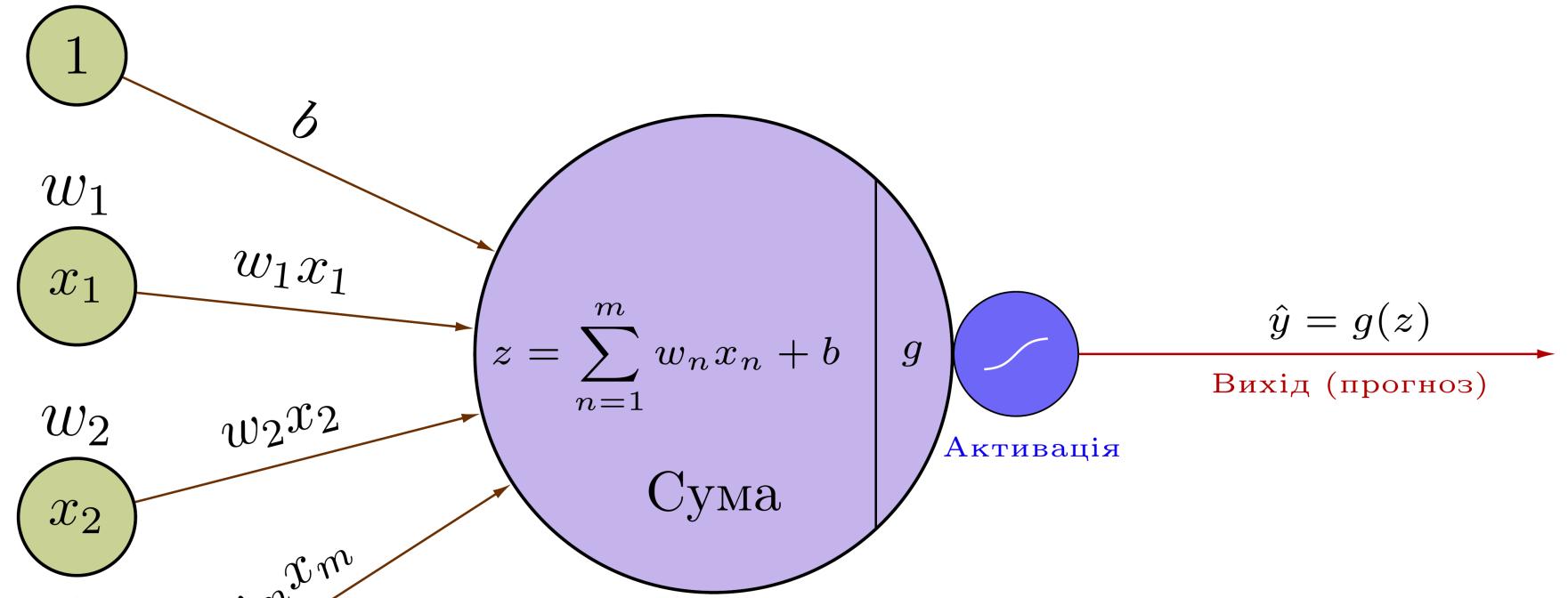
Figure I ORGANIZATION OF THE MARK I PERCEPTRON



Ф. Розенблатт



$$J(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^n (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$



Пряме поширення

$$z = \sum_{n=1}^m w_n x_n + b = \mathbf{X}^T \cdot \mathbf{W} + b = \mathbf{W}^T \cdot \mathbf{X} + b$$

$$\hat{y} = g(z)$$

$$J(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^n (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

Вихід

Перцептрон: приклад обчислень

Припустимо $m = 3$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.7 \\ 0.5 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad b = 0.8$$

$$z = \sum_{n=1}^3 w_n x_n + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b = \\ = 1 \cdot -0.1 + -2 \cdot 0.7 + 2 \cdot 0.5 + 0.8 = 0.3$$

Перцептрон: приклад обчислень

Припустимо $m = 3$

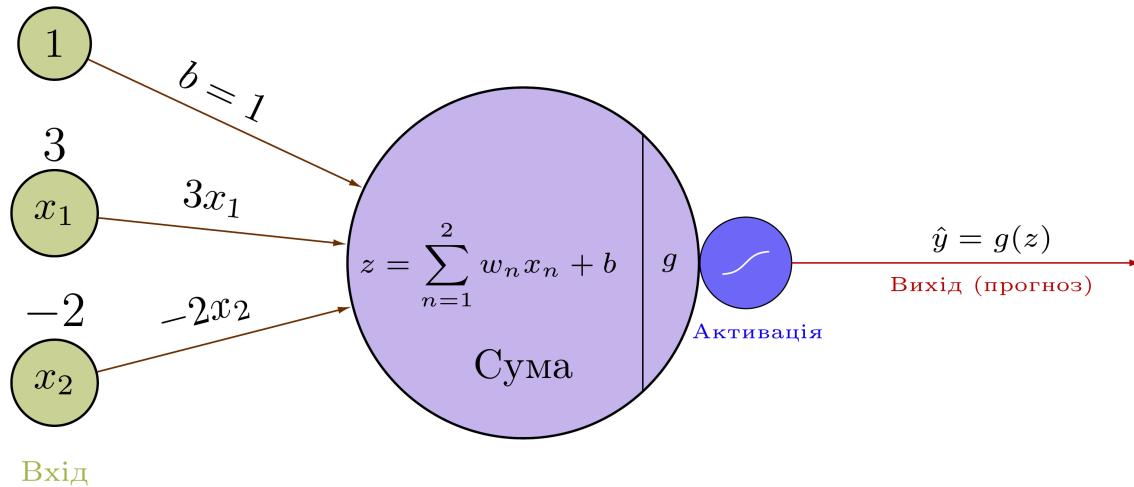
$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.7 \\ 0.5 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \quad b = 0.8$$

$$z = \sum_{n=1}^3 w_n x_n + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b = \\ = 1 \cdot -0.1 + -2 \cdot 0.7 + 2 \cdot 0.5 + 0.8 = 0.3$$

$$z = \mathbf{X}^T \cdot \mathbf{W} + b = [x_1 \quad x_2 \quad x_3] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + b = \\ = w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0.3$$

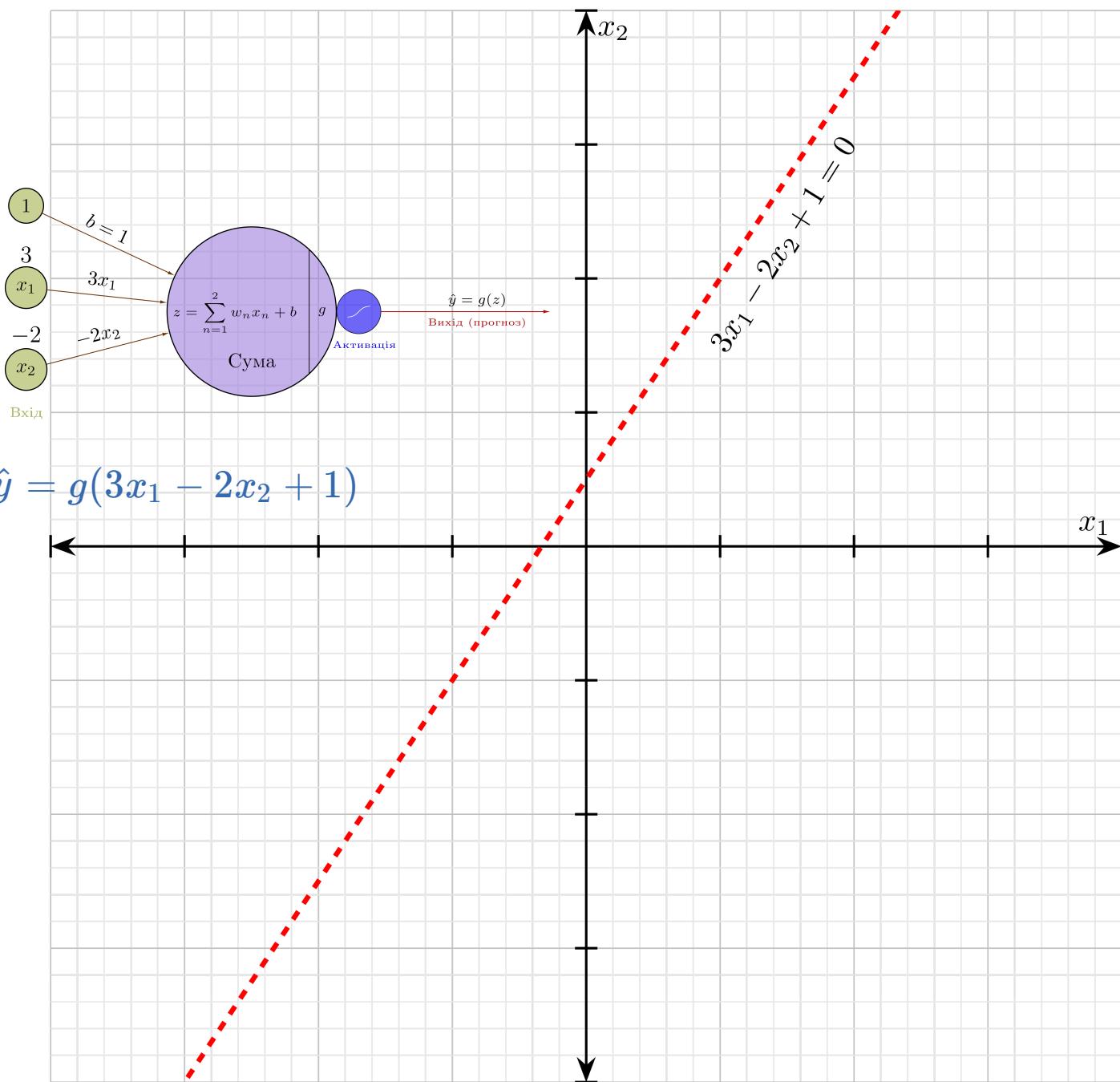
$$\hat{y} = g(z) = g(\mathbf{X}^T \cdot \mathbf{W} + b) = \frac{1}{1 + \exp(-z)} = \frac{1}{1 + \exp(-0.3)} \approx 0.57$$

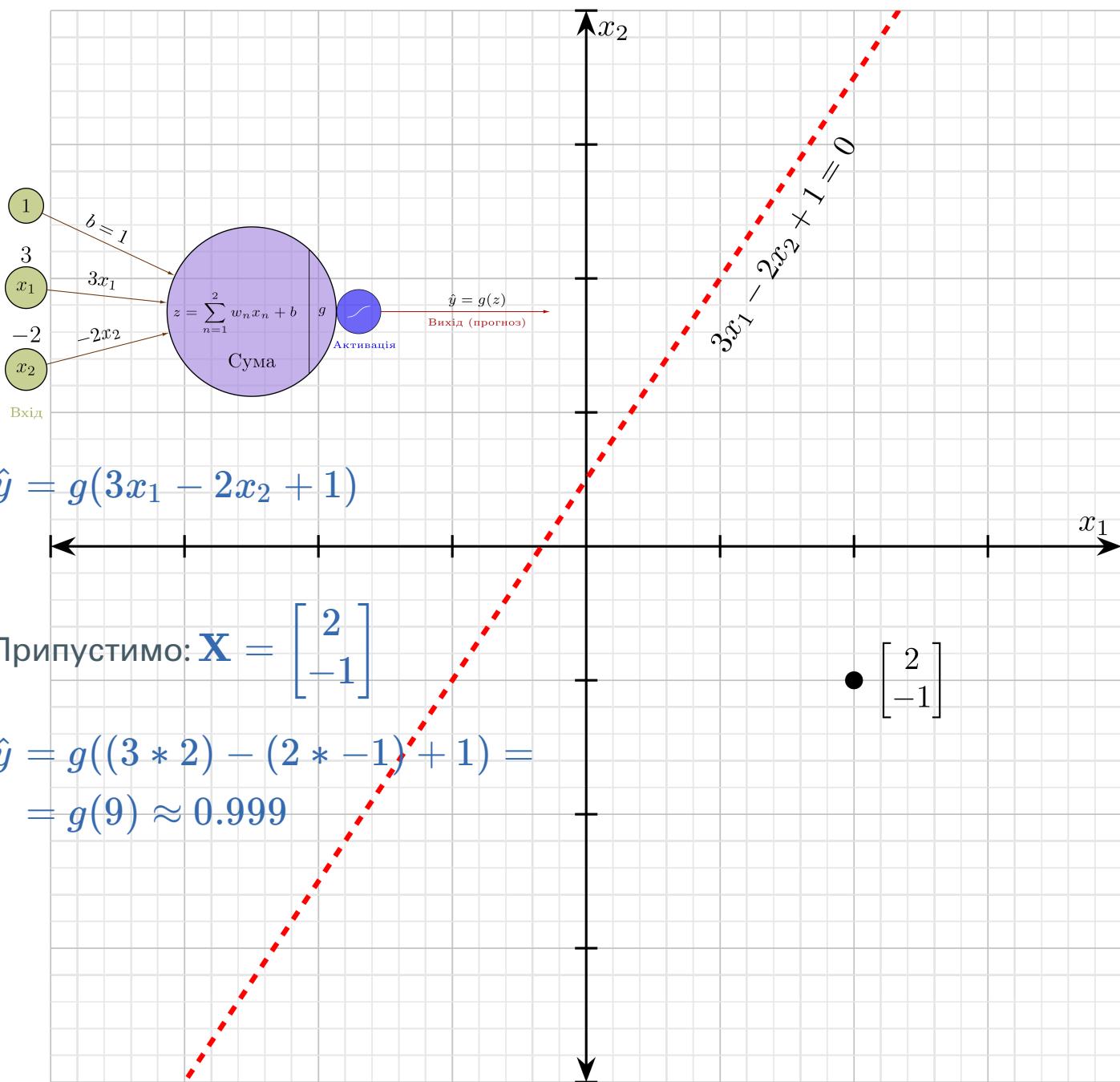
Перцептрон: приклад 2

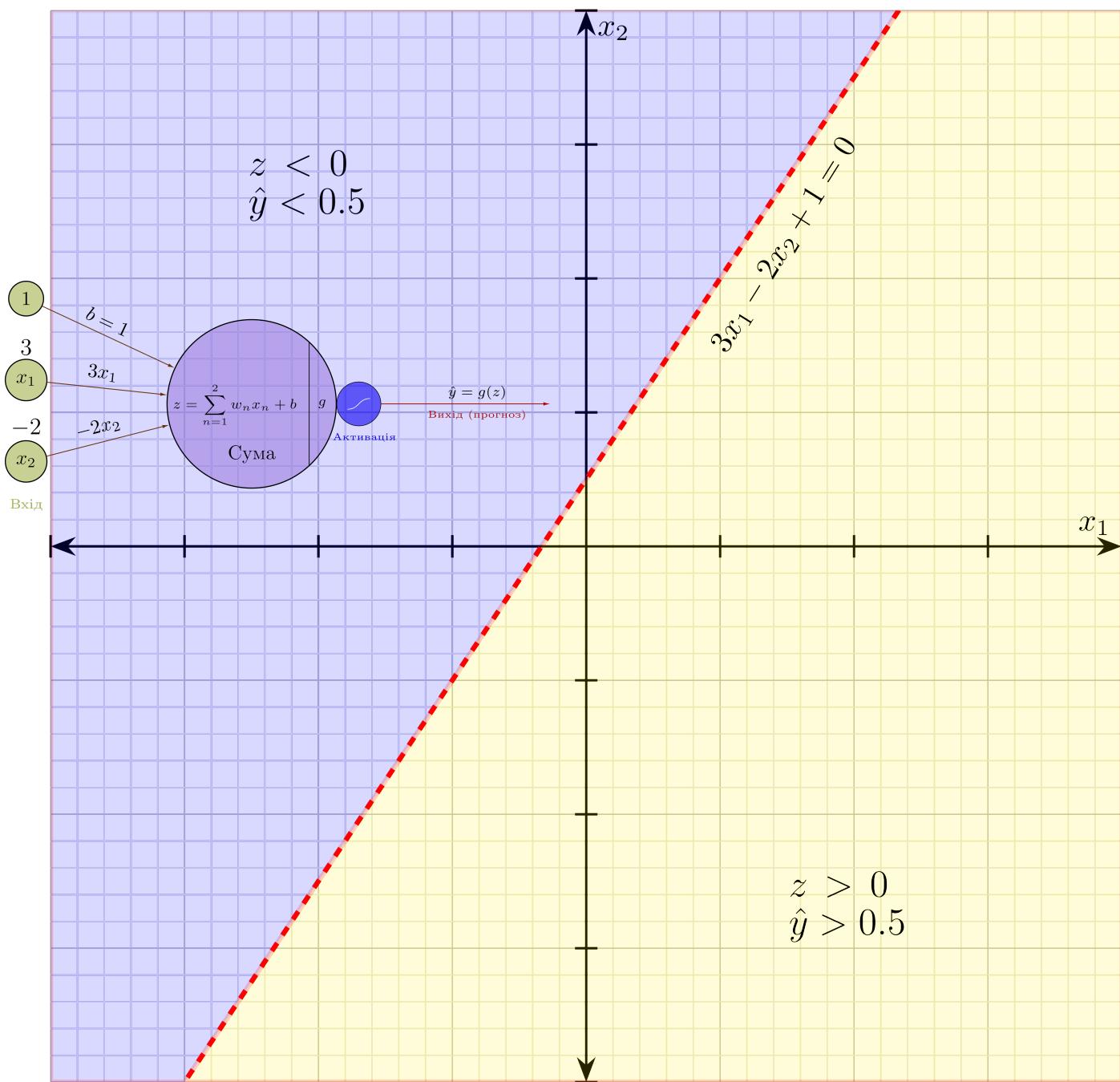


Нехай маємо: $b = 1$ та $\mathbf{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(\mathbf{X}^T \cdot \mathbf{W} + b) = \\ &= g\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \cdot \begin{bmatrix} 3 \\ -2 \end{bmatrix} + 1\right) = \\ &= g(\underbrace{3x_1 - 2x_2 + 1}_{\text{line in 2D}})\end{aligned}$$





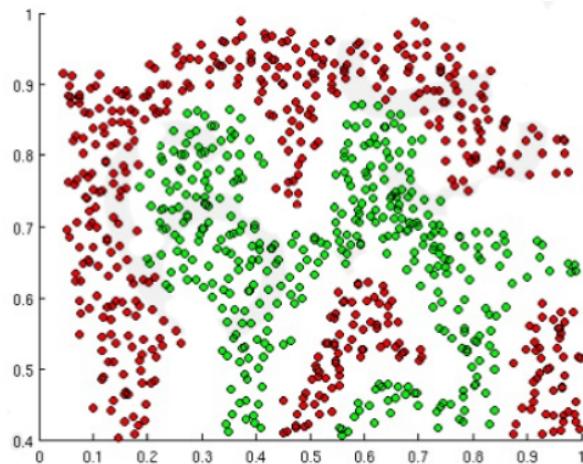


Деякі функції активації

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ $\epsilon \ll 1$

Важливість функцій активації

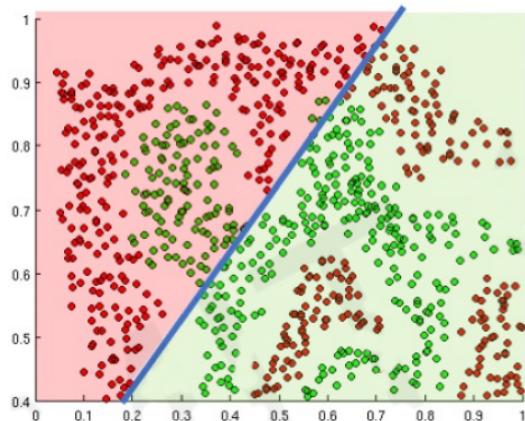
Призначення функцій активації – введення нелінійності



Припустимо, ми хочемо створити таку нейронну мережу, яка зможе розрізняти червоні від зелених прикладів

Важливість функцій активації

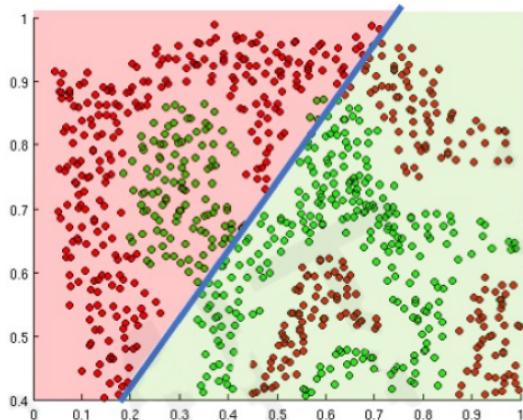
Призначення функцій активації – введення нелінійності



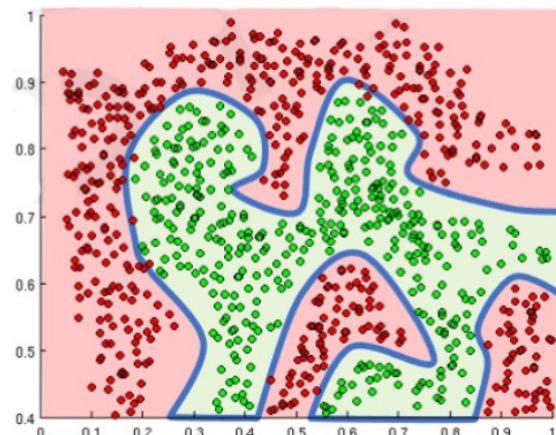
Нейрони з лінійними функціями активації породжують лінійну криву рішень, не має значення розмір мережі (глибина, ширина)

Важливість функцій активації

Призначення функцій активації – введення нелінійності



Нейрони з лінійними функціями активації породжують лінійну криву рішень, не має значення розмір мережі (глибина, ширина)



Нейрони з нелінійними функціями активації дозволяють нам апроксимувати криву рішень будь-якої складності

Одновимірний градієнтний спуск

Одновимірний градієнтний спуск

Розглянемо деяку монотонну неперервну диференційовану функцію $f : \mathbb{R} \rightarrow \mathbb{R}$. Розкладаючи у ряд Тейлора, ми отримуємо:

$$f(x + \varepsilon) = f(x) + \varepsilon f'(x) + \mathcal{O}(\varepsilon^2)$$

Зфіксуємо розмір кроку $\alpha > 0$ та оберемо $\varepsilon = -\alpha f'(x)$. Підставляючи це у ряд Тейлора, отримаємо:

$$f(x - \alpha f'(x)) = f(x) - \alpha f'^2(x) + \mathcal{O}(\alpha^2 f'^2(x))$$

Якщо похідна $f'(x) \neq 0$ не зникає ми робимо прогрес так як $\alpha f'^2(x) > 0$. Крім того, ми завжди можемо вибрати α досить малим, щоб вирази вищих порядків стали нерелевантними. Тому ми приходимо до

$$f(x - \alpha f'(x)) \lesssim f(x)$$

Це означає, якщо ми використовуємо

$$x \leftarrow x - \alpha f'(x)$$

для ітерації по x , значення функції $f(x)$ може зменшитись.

```
import numpy as np

def f(x):  # Objective function
    return x**2

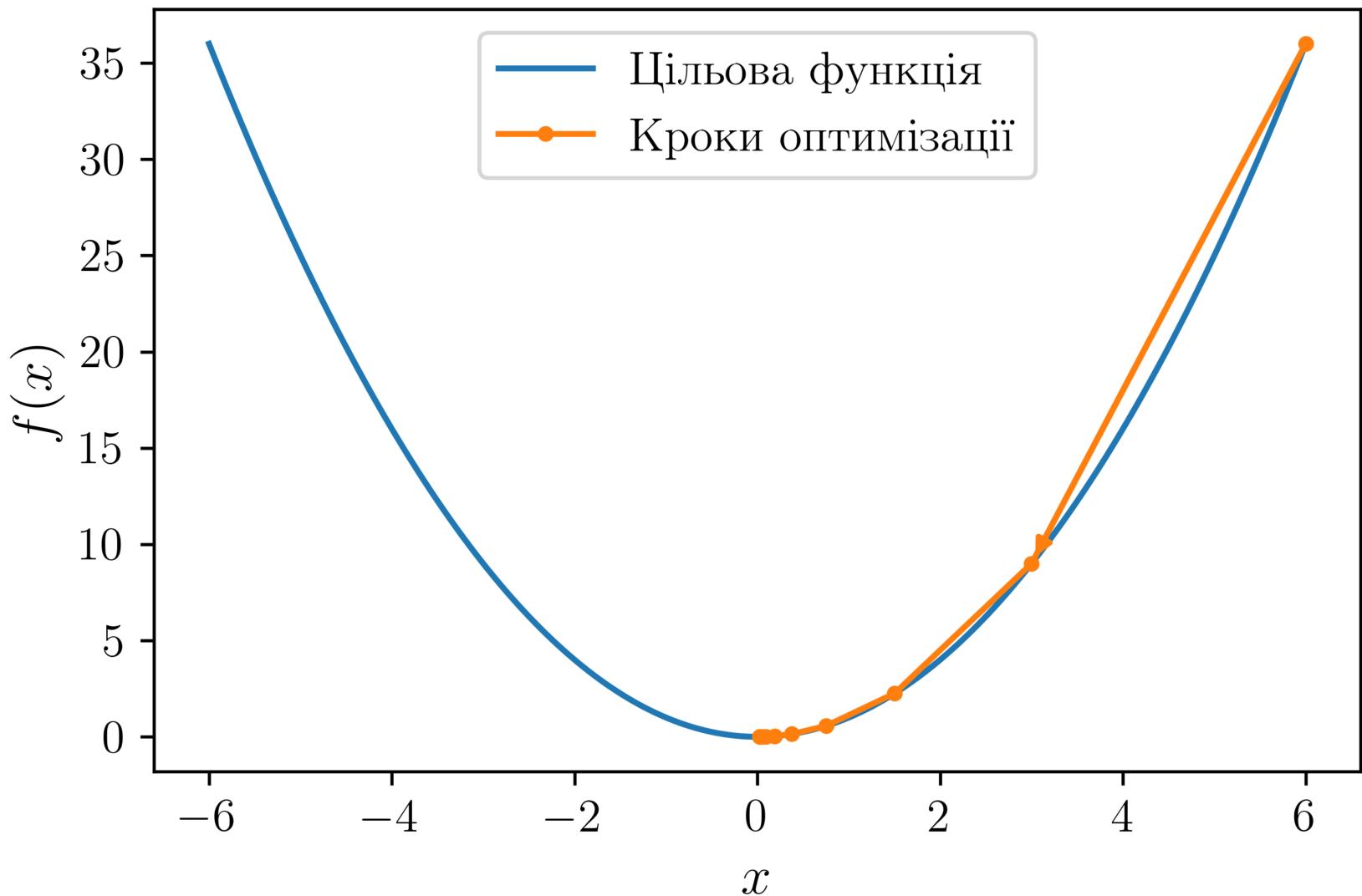
def f_grad(x):  # Gradient (derivative) of the objective function
    return 2 * x

def bgd(alpha, f_grad):
    x = 6.0          # Initial value of x
    results = [x]
    epoch = 8         # Number of iterations
    for i in range(epoch):
        x -= alpha * f_grad(x)
        results.append(float("%.6f" % x))
    print(f'epoch {epoch}, x: {x:.6f}')
    return results

results = bgd(0.25, f_grad)
print(results)
```

```
epoch 8, x: 0.023438
```

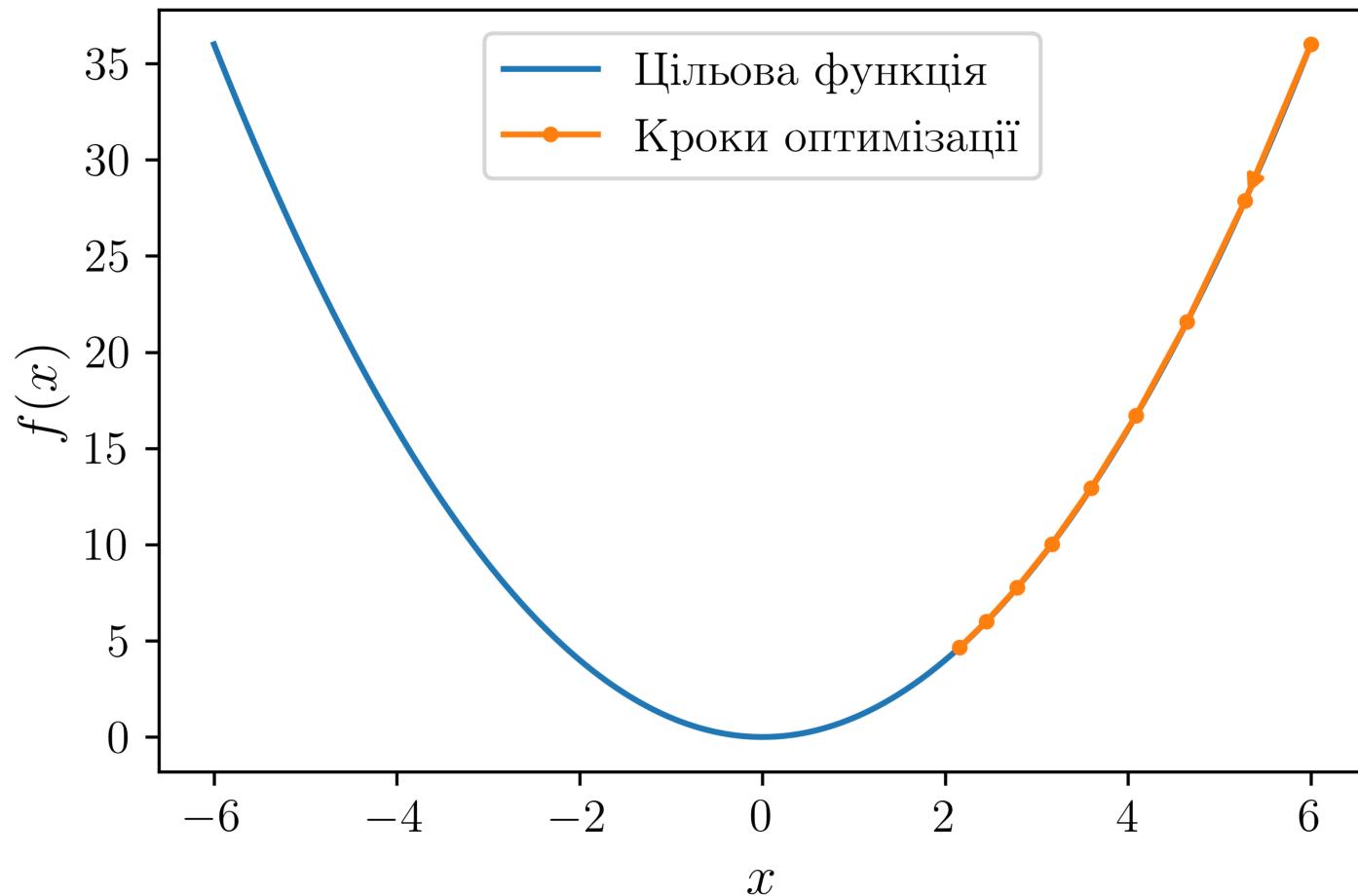
```
[6.0, 3.0, 1.5, 0.75, 0.375, 0.1875, 0.09375, 0.046875, 0.023438] 9/34
```



Одновимірний градієнтний спуск ($\alpha = 0.25$)

```
results = bgd(0.06, f_grad)
```

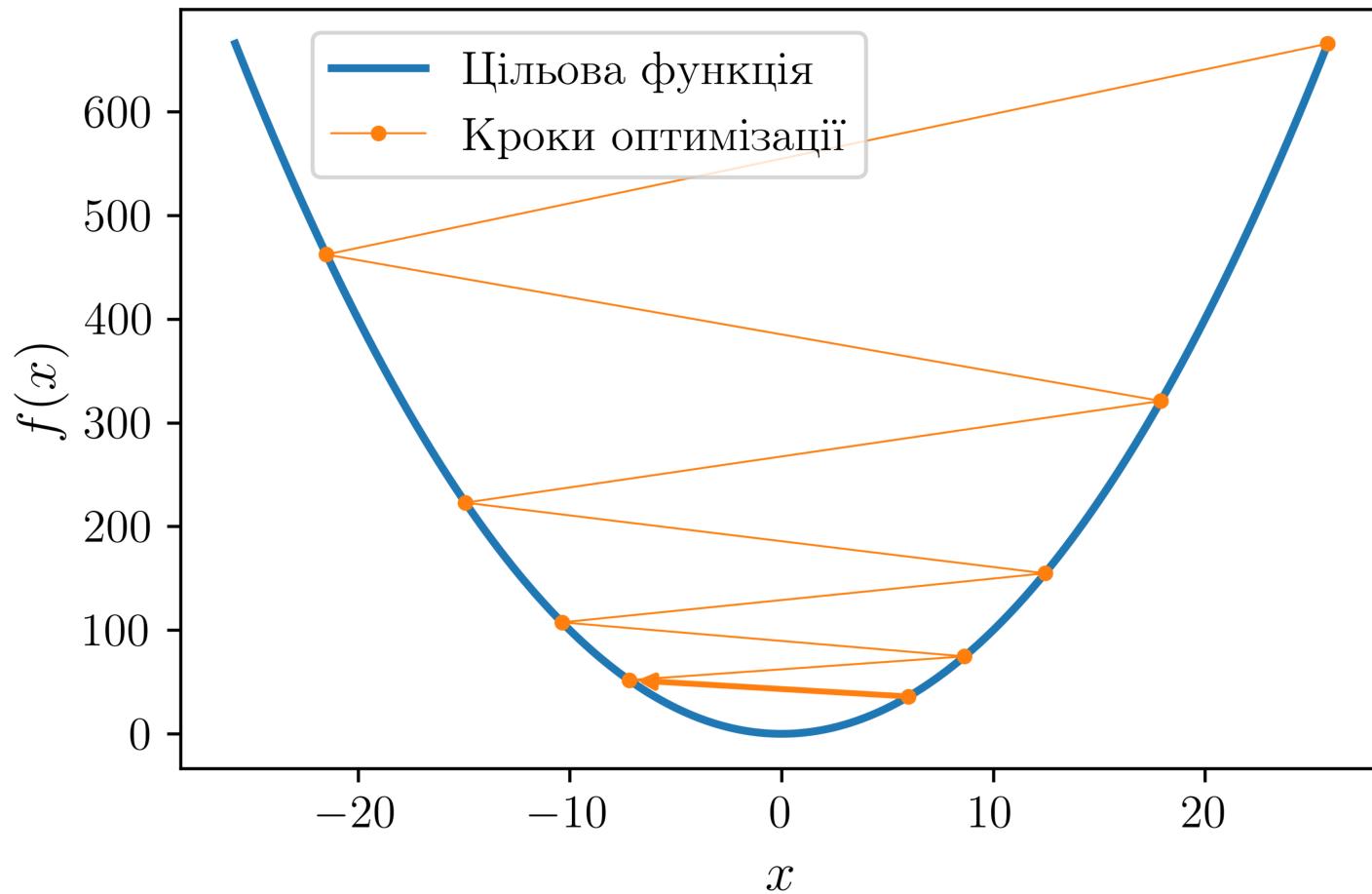
```
epoch 8, x: 2.157807
```



Одновимірний градієнтний спуск ($\alpha = 0.06$)

```
results = bgd(1.1, f_grad)
```

```
epoch 8, x: 25.798902
```



Одновимірний градієнтний спуск ($\alpha = 1.1$)

Перцептрон

Одношарова нейронна мережа: зворотне поширення

Зворотне поширення

Правило ланцюжка стверджує, якщо у нас є складена функція $\ell(g(w))$, тоді похідна цієї функції по w визначається як:

$$\frac{d}{dw} \ell(g(w)) = \frac{d\ell}{dg} \frac{dg}{dw}$$

У контексті глибокого навчання:

- ℓ – цільова функція втрат нейронної мережі,
- g – проміжні активації на кожному шарі
- w – навчальні параметри.

Зворотне поширення

- Оскільки нейронна мережа є **композицією диференційованих функцій**, загальні похідні втрат можна оцінити зворотно, застосовуючи рекурсивно правило ланцюжка до обчислювального графу нейронної мережі.
- Реалізація цієї процедури називається зворотним **автоматичним диференціюванням** або **зворотним поширенням**.

Пряме поширення

$$z = \sum_{n=1}^m w_n x_n + b = \mathbf{X}^T \cdot \mathbf{W} + b = \mathbf{W}^T \cdot \mathbf{X} + b$$

$$\hat{y} = g(z) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$J(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^n \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

Зворотне поширення

$$\frac{\partial J(\hat{y}, y)}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial J(\hat{y}, y)}{\partial z} = \frac{\partial J(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \hat{y} - y$$

$$\frac{\partial J(\hat{y}, y)}{\partial \mathbf{W}} = \frac{\partial J(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \mathbf{W}} = \mathbf{X}^T \cdot (\hat{y} - y)$$

$$\frac{\partial J(\hat{y}, y)}{\partial b} = \frac{\partial J(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} = \hat{y} - y$$

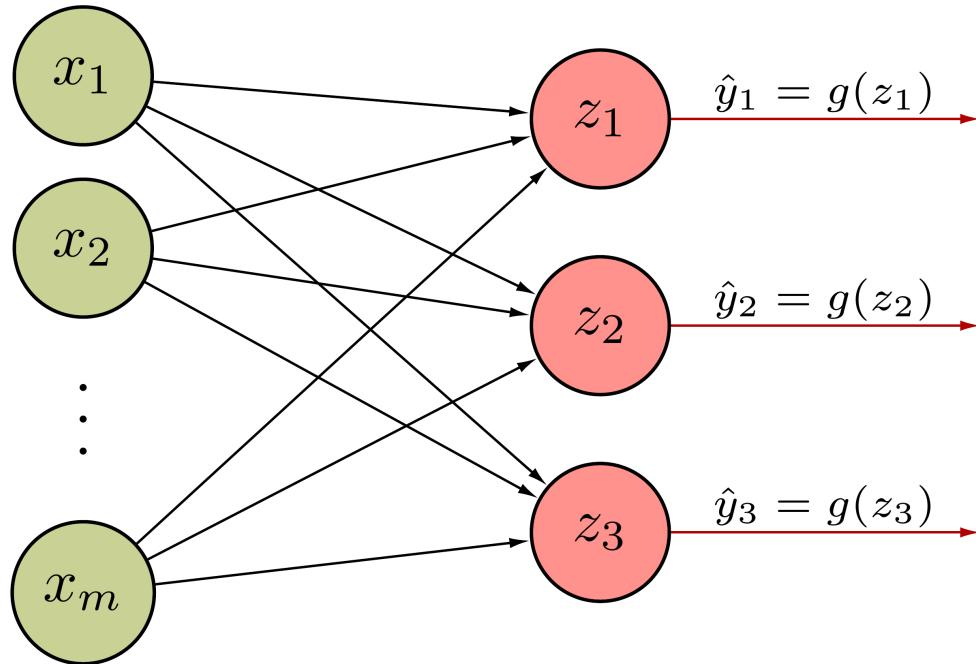
Оновлення параметрів

$$\mathbf{W} = \mathbf{W} - \alpha \frac{\partial J(\hat{y}, y)}{\partial \mathbf{W}}$$

$$b = b - \alpha \frac{\partial J(\hat{y}, y)}{\partial b}$$

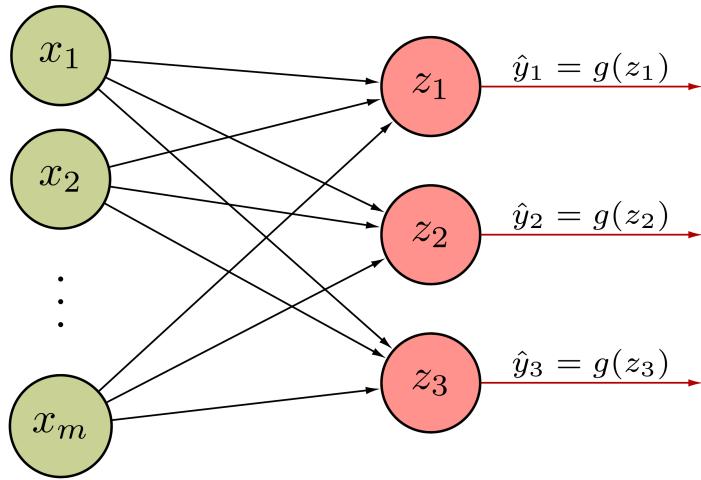
Перцептрон з кількома виходами

Оскільки всі входи щільно з'єднані з усіма виходами, ці шари називаються щільними (Dense) або повнозв'язними



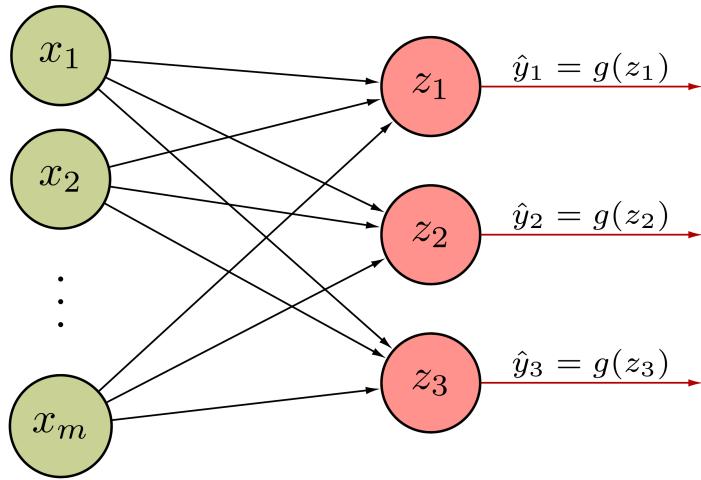
$$z_j = \sum_{n=1}^m w_{j,n} x_n + b_j$$

Приклад обчислень



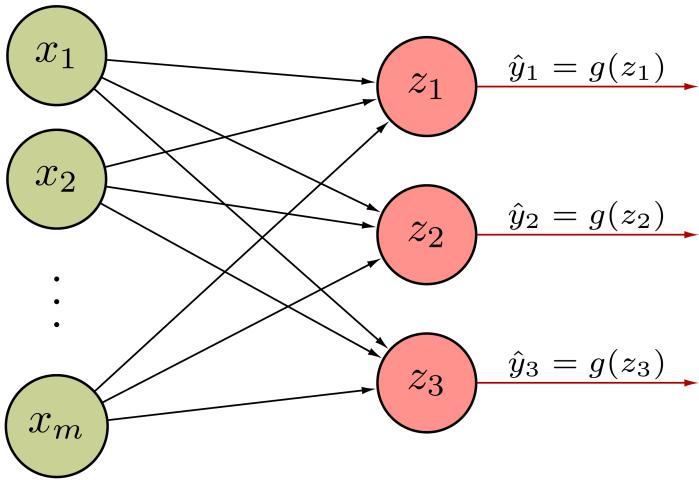
$$\mathbf{X}^{m \times 1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Приклад обчислень



$$\mathbf{X}^{m \times 1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{W}^{3 \times m} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix}$$

Приклад обчислень

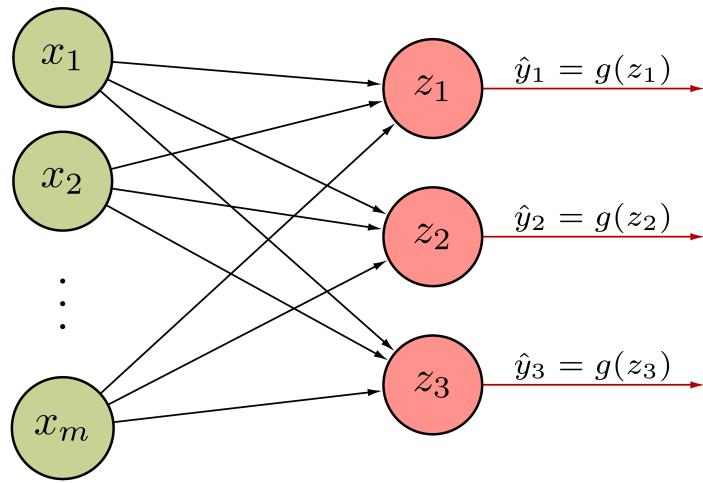


$$\mathbf{X}^{m \times 1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

$$\mathbf{W}^{3 \times m} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix}$$

$$\mathbf{b}^{3 \times 1} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Приклад обчислень



$$\mathbf{X}^{m \times 1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{W}^{3 \times m} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix} \quad \mathbf{b}^{3 \times 1} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\boxed{\begin{aligned} \mathbf{z} = \mathbf{W} \cdot \mathbf{X} + \mathbf{b} &= \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \\ &= \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + \cdots + w_{1m}x_m + b_1 \\ w_{21}x_1 + w_{22}x_2 + \cdots + w_{2m}x_m + b_2 \\ w_{31}x_1 + w_{32}x_2 + \cdots + w_{3m}x_m + b_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \end{aligned}}$$



Dense layer from scratch

```
class MyDenseLayer(tf.keras.layers.Layer):
    def __init__(self, input_dim, output_dim):
        super(MyDenseLayer, self).__init__()

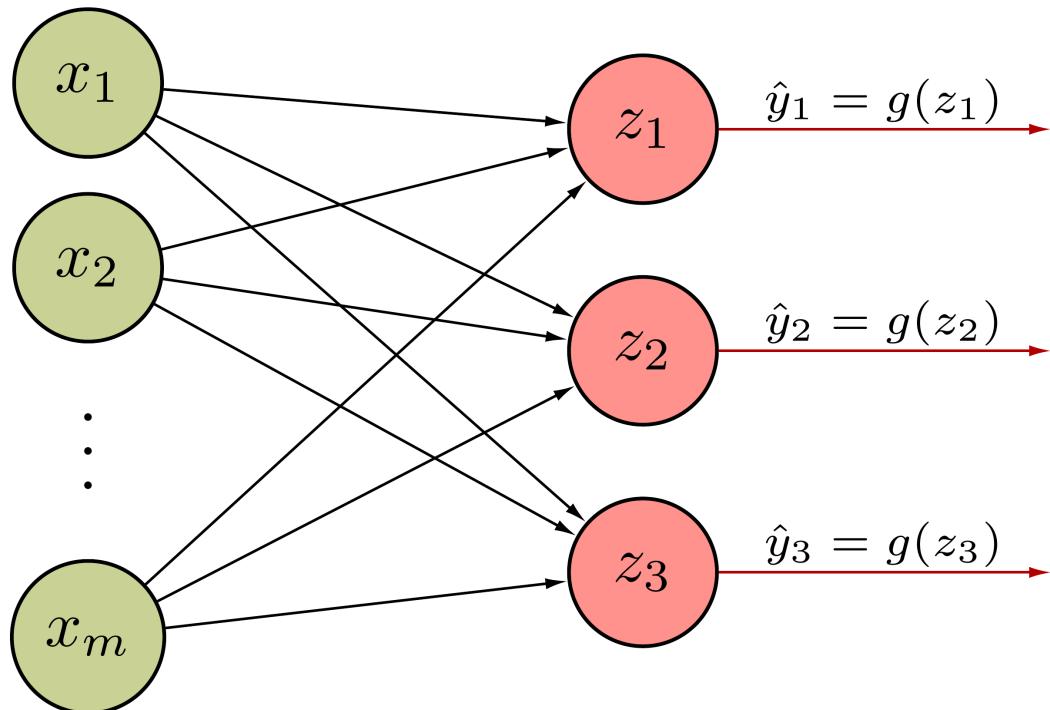
        # Initialize weights and bias
        self.W = self.add_weight([input_dim, output_dim])
        self.b = self.add_weight([1, output_dim])

    def call(self, inputs):
        # Forward propagate the inputs
        z = tf.matmul(inputs, self.W) + self.b

        # Feed through a non-linear activation
        output = tf.math.sigmoid(z)

    return output
```

Приклад реалізації в TF



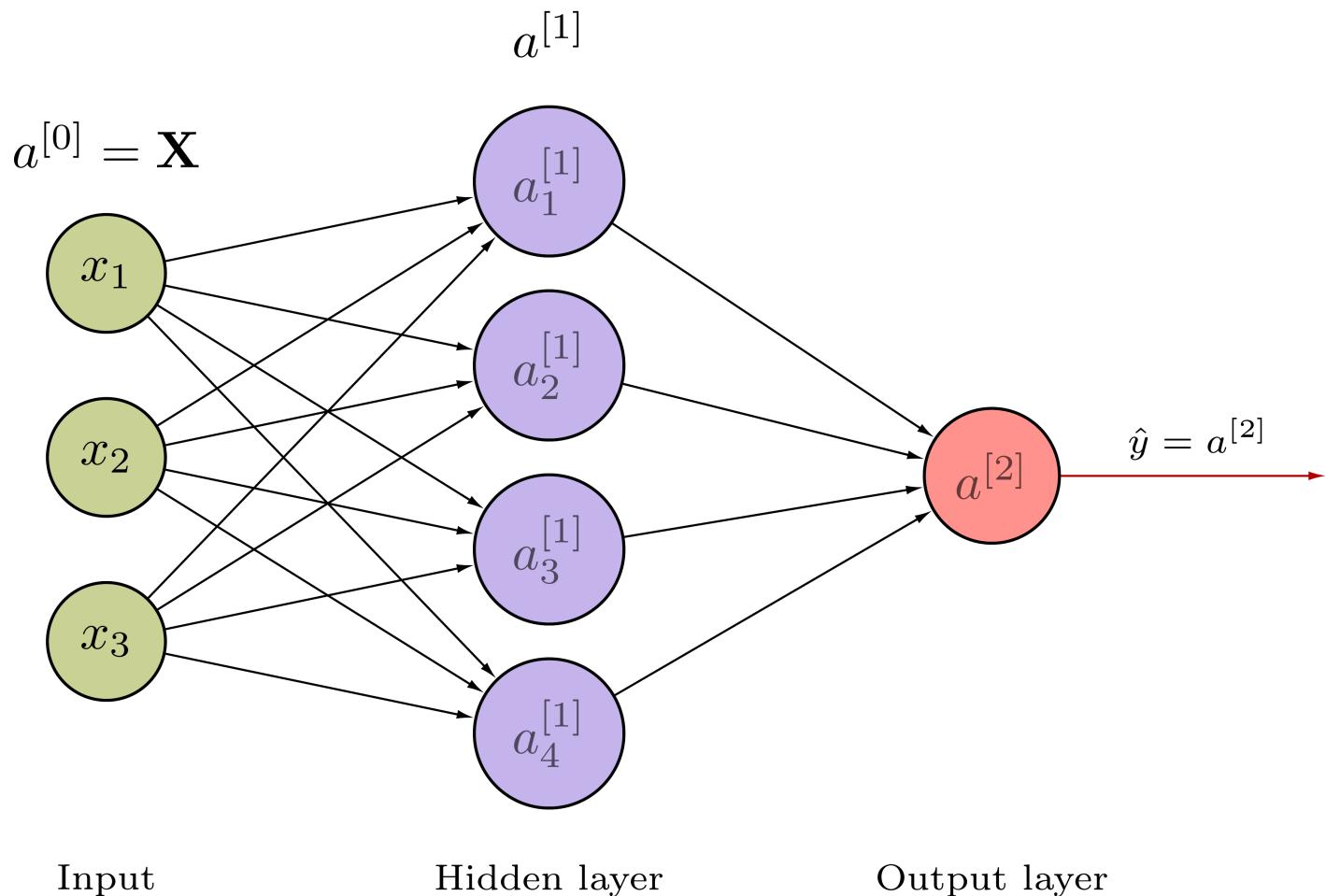
$$z_j = \sum_{n=1}^m w_{j,n} x_n + b_j$$



```
import tensorflow as tf  
  
layer = tf.keras.layers.Dense(  
    units=3)
```

Багатошаровий перцептрон

Один прихований шар

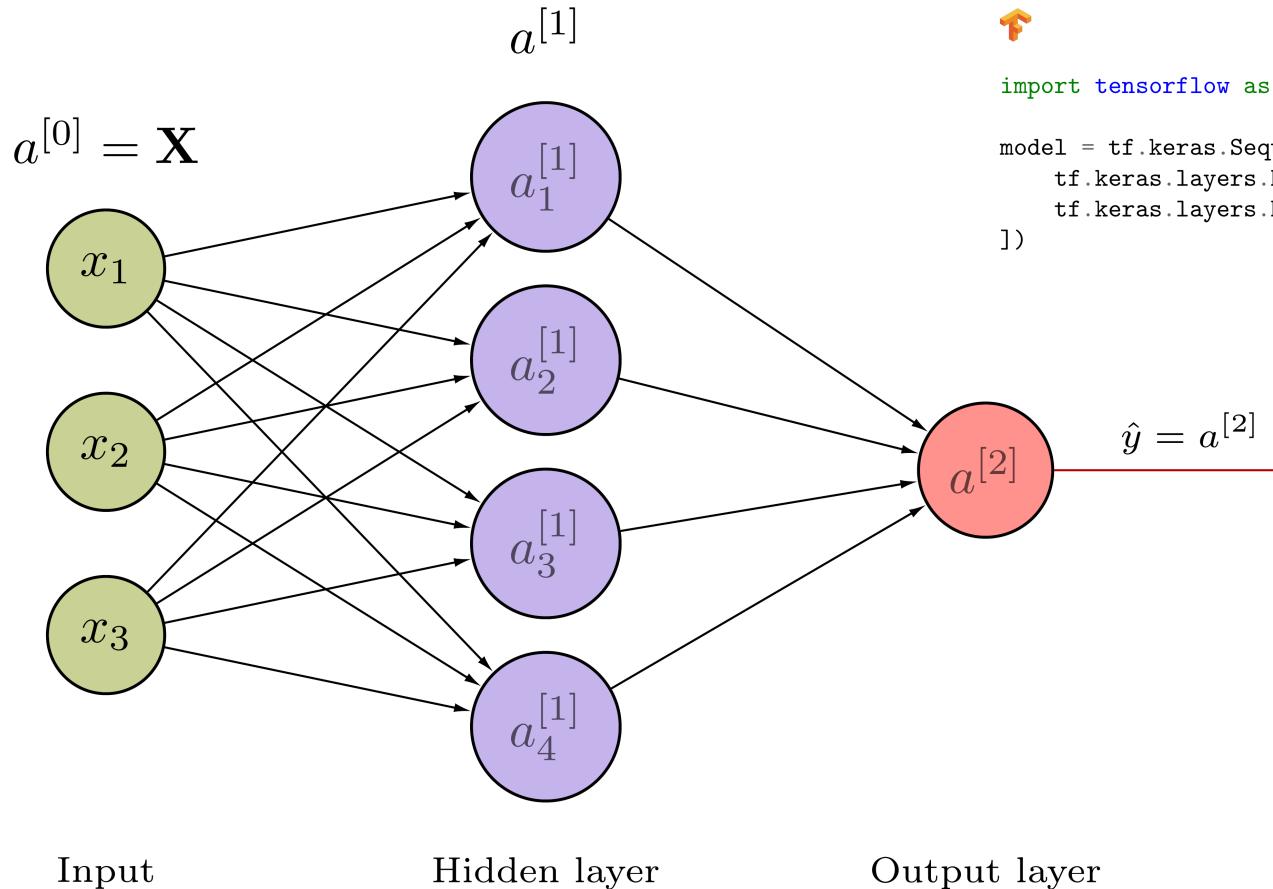


Input

Hidden layer

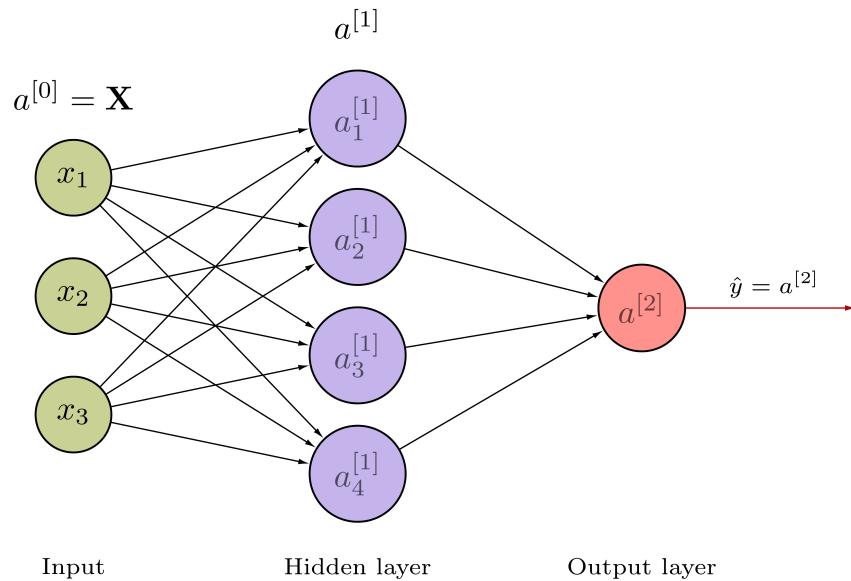
Output layer

Один прихований шар



```
import tensorflow as tf  
  
model = tf.keras.Sequential([  
    tf.keras.layers.Dense(4),  
    tf.keras.layers.Dense(1)  
])
```

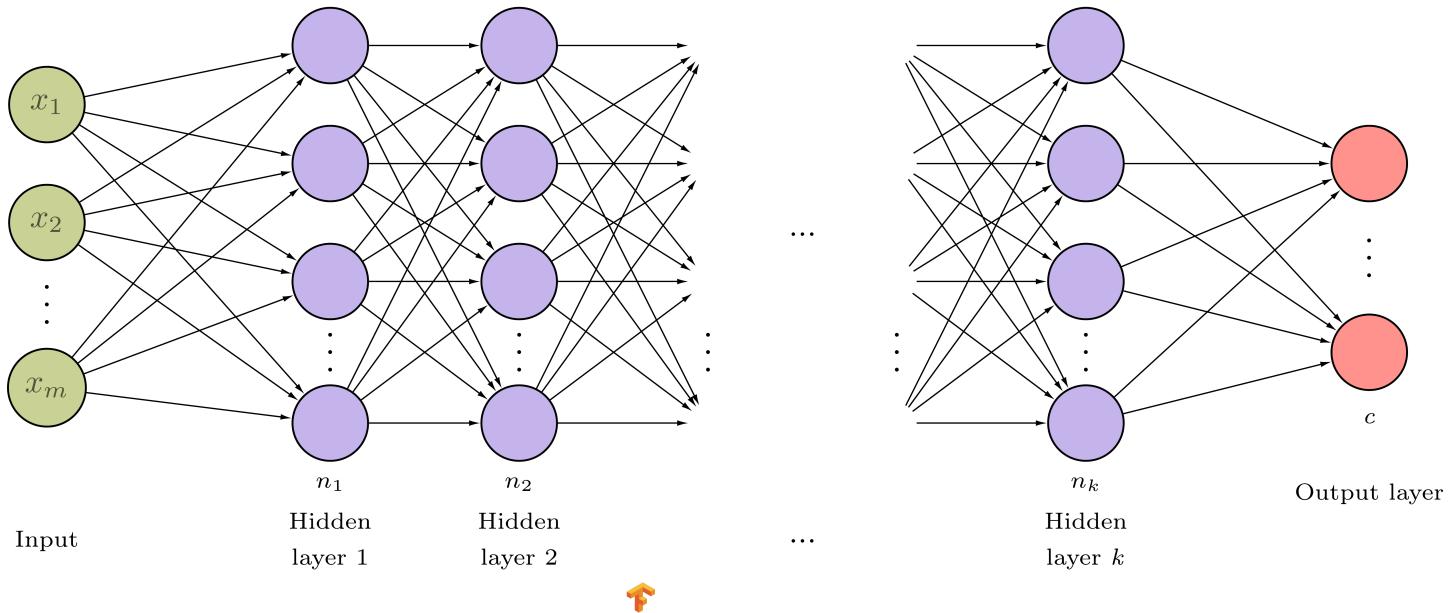
Один прихованій шар



$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{W}^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad \mathbf{W}^{[2]} = [w_1 \quad w_2 \quad w_3 \quad w_4] \quad b^{[2]} = b$$

$$\boxed{\begin{aligned} \mathbf{z}^{[1]} &= \mathbf{W}^{[1]} \cdot \mathbf{X} + \mathbf{b}^{[1]} \\ \mathbf{a}^{[1]} &= g^{[1]}(\mathbf{z}^{[1]}) \\ z^{[2]} &= \mathbf{W}^{[2]} \cdot \mathbf{a}^{[1]} + b^{[2]} \\ \hat{y} &= a^{[2]} = g^{[2]}(z^{[2]}) \end{aligned}}$$

Глибока нейронна мережа



```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n1),
    tf.keras.layers.Dense(n2),
    .
    .
    .
    tf.keras.layers.Dense(nk),
    tf.keras.layers.Dense(c)
])
```

Як навчити свою нейронну мережу?



Френк
Розенблат



Олексій
Івахненко



Сеппо
Ліnnайнмаа



Пол
Вербос



Девід
Румельхарт

1957

1966

1970

1982

1986

Правило
навчання
перцептрона

Метод
групового
урахування
аргументів

Зворотне поширення помилки

Батько глибокого навчання



Олексій Григорович Івахненко

