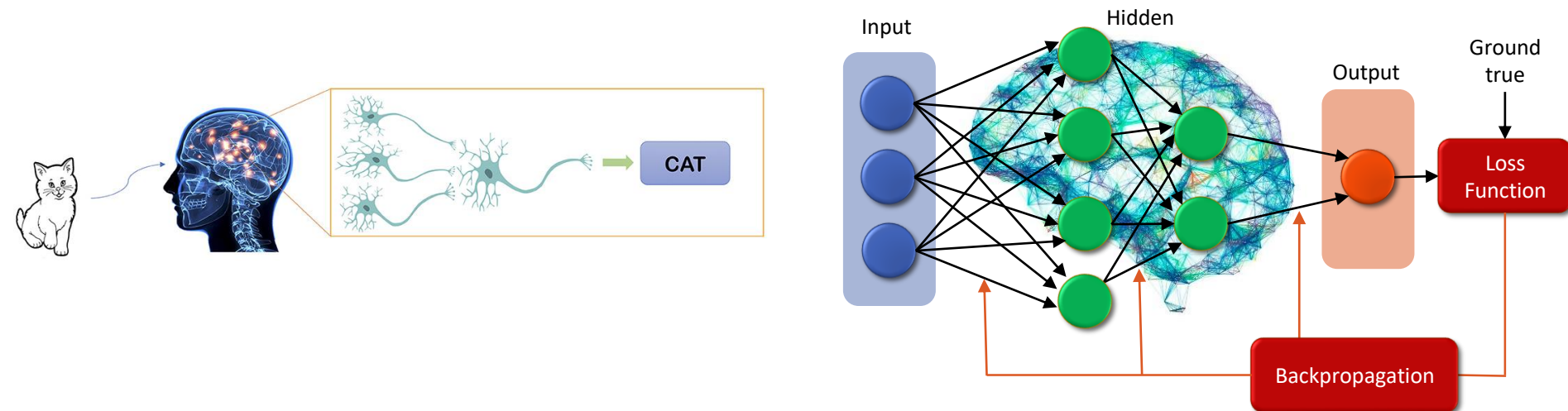


DEEP LEARNING

MULTI LAYER PERCEPTRON



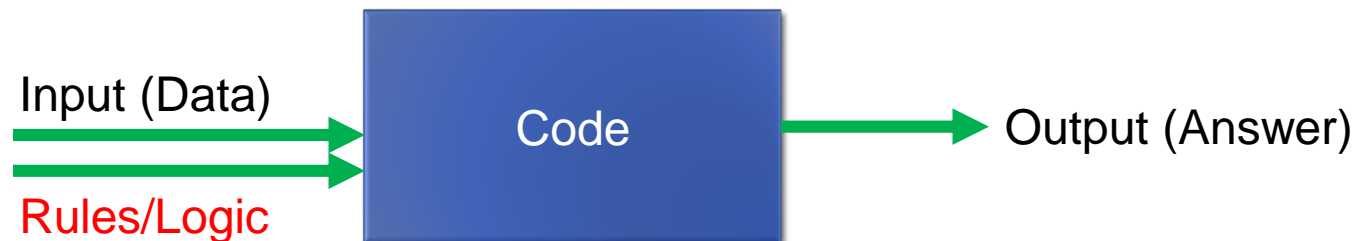
Tôn Quang Toại
Khoa Công nghệ thông tin
Trường đại học Ngoại ngữ - Tin học TP.HCM (HUFLIT)

Nội dung

- Nguyên tắc tiếp cận
- Bài toán phân lớp
- Xây dựng không gian đặc trưng (feature space) từ dữ liệu thô
- Perceptron và Quy tắc học Delta
- Mạng neuron nhiều tầng (MLP)
 - Forward propagation
 - Activation function
 - Loss function
 - Optimization (Backpropagation)
 - Regularization
 - Đánh giá hiệu năng

Nguyên tắc tiếp cận

- **Neural networks:** Neural networks là một trong những mô hình lập trình (programming paradigm) đẹp nhất
- Cách tiếp cận lập trình thông thường



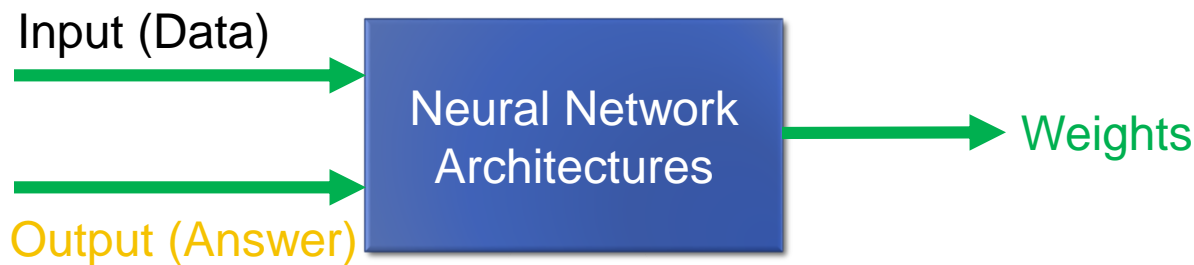
- Ý tưởng
 - Chia bài toán lớn thành các nhiệm vụ nhỏ mà máy có thể làm được
 - Nói cho nó biết những gì phải làm

Nguyên tắc tiếp cận

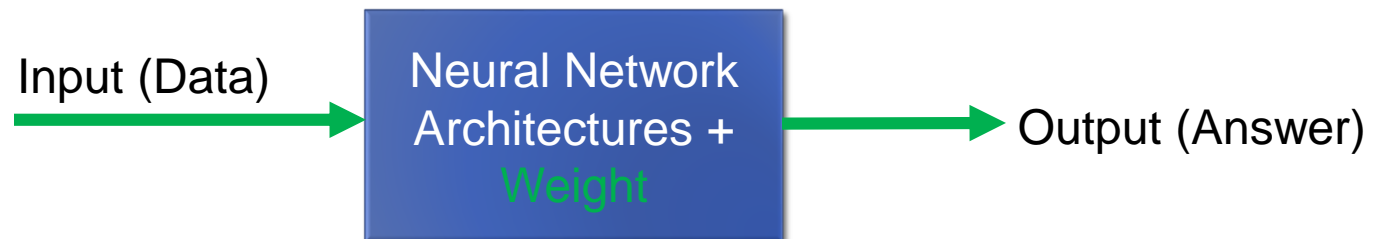
- **Neural networks:** Neural networks là một trong những mô hình lập trình (programming paradigm) đẹp nhất

- Cách tiếp cận neural networks

- Học



- Sử dụng

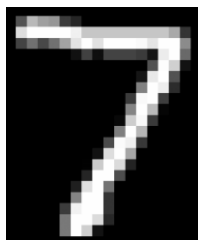


Nguyên tắc tiếp cận

Lập trình thông thường	Lập trình neural networks	Không nên
Nắm vững <ul style="list-style-type: none">- Các cú pháp nền tảng- Cấu trúc dữ liệu quan trọng- Thư viện chính yếu- Thiết kế thuật toán	Nắm vững <ul style="list-style-type: none">- Nguyên tắc cốt lõi của mạng neuron- Nguyên tắc xây dựng kiến trúc	Hiểu mơ hồ Nắm danh sách các ý tưởng
Chỉ cần học phần nhỏ <ul style="list-style-type: none">- Ngôn ngữ- Thư viện không lồ- Thuật toán	Chỉ cần học phần nhỏ <ul style="list-style-type: none">- Các kiến trúc neural network	Chỉ biết sử dụng thư viện cụ thể
Giải quyết từng bài toán cụ thể	Giải quyết từng bài toán cụ thể	Chỉ hiểu ý tưởng
Dễ dàng đọc hiểu cấu trúc dữ liệu mới, thư viện mới	Dễ dàng đọc hiểu các tài liệu mới	Khó đọc tài liệu mới

Bài toán phân lớp

Yêu cầu



Số nào trong các số 0, 1..., 9

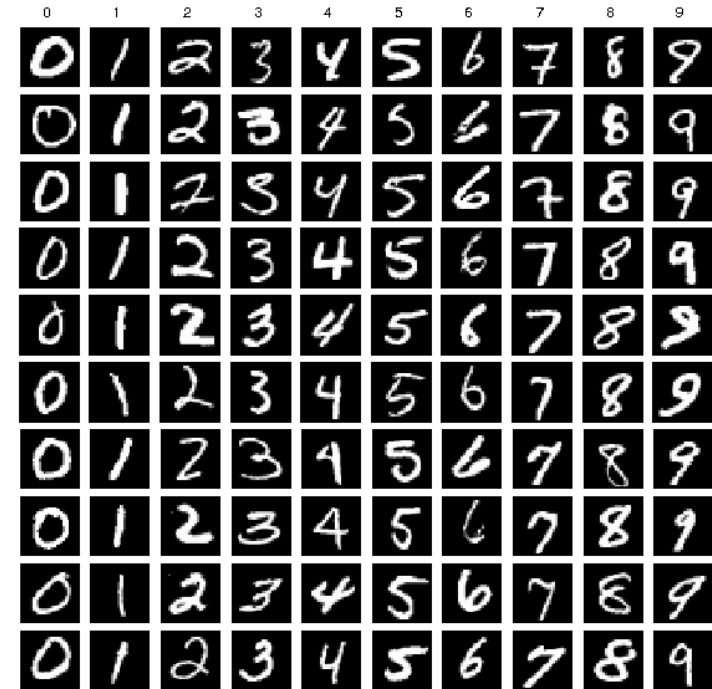
Computer “nhìn thấy”

21	12	45	123
32	43	220	22
4	200	23	233
254	69	88	44

Số nào trong các số 0, 1..., 9

Bài toán phân lớp

- Dataset: MNIST
 - Số lượng: 70.000 images
 - Kích thước ảnh: 28×28 ảnh mức xám



- Mong muốn: Xây dựng hệ thống có thể học từ các dữ liệu thu thập được.

Ký hiệu toán

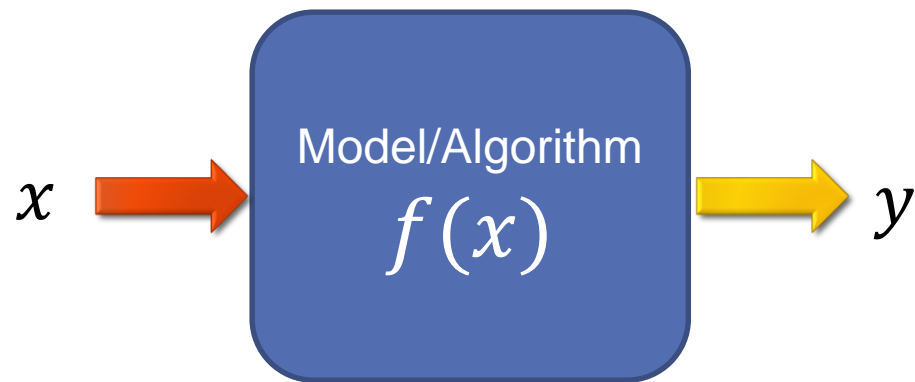
- m là số data point trong data set
- n hay n_x là số feature của vector $x^{(i)}$
- $x^{(i)}$ là vector đặc trưng thứ i trong tập dữ liệu
- $y^{(i)}$ là nhãn của của $x^{(i)}$
- $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^m$ là dataset
- X là ma trận $m \times n$ chứa vector đặc trưng $x^{(i)}$ ở dòng X_i .
- Ta có
 - $x^{(i)} \in \mathbb{R}^n$
 - $X \in \mathbb{R}^{m \times n}$

XÂY DỰNG KHÔNG GIAN ĐẶC TRƯNG

(FEATURE SPACE)

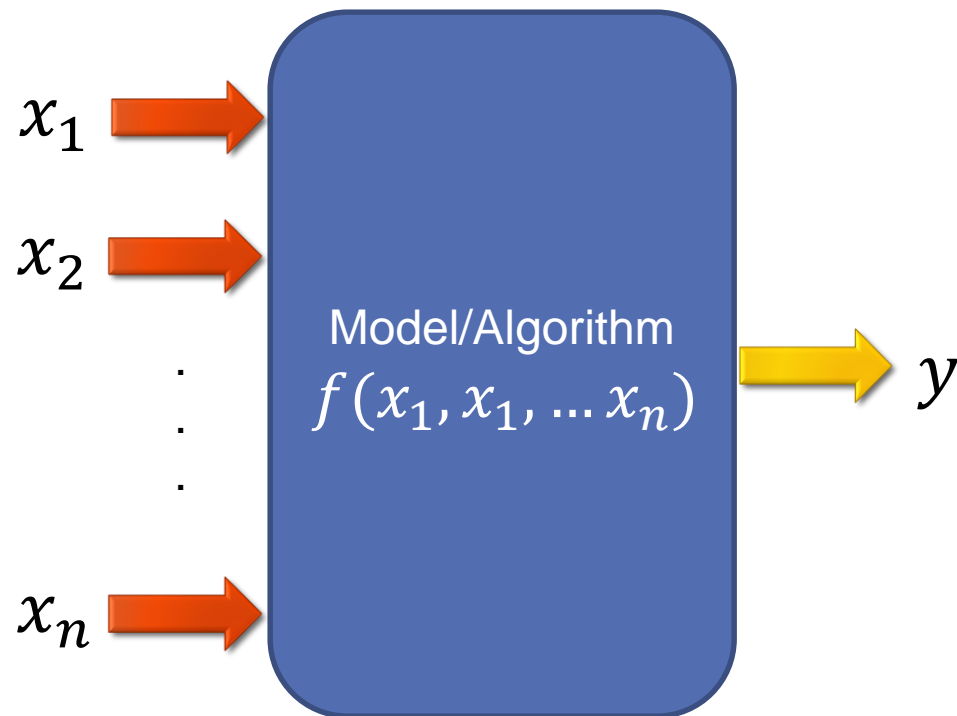
Input data

- Bài toán đơn giản
 - Một input và
 - Một output



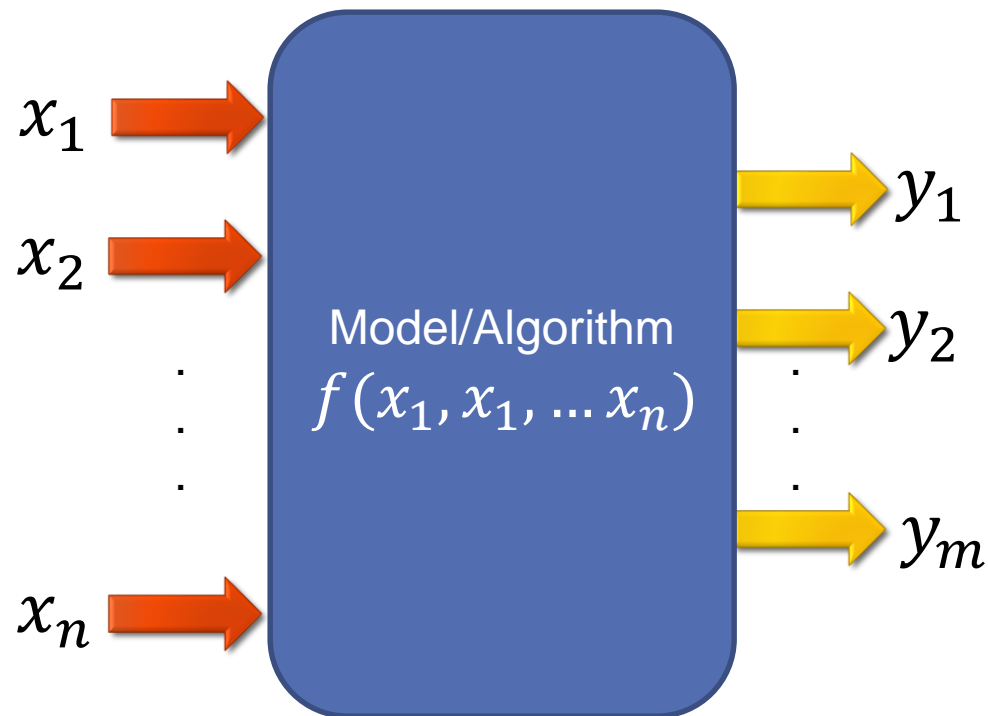
Input data

- Bài toán thực tế
 - Nhiều input và
 - Một output



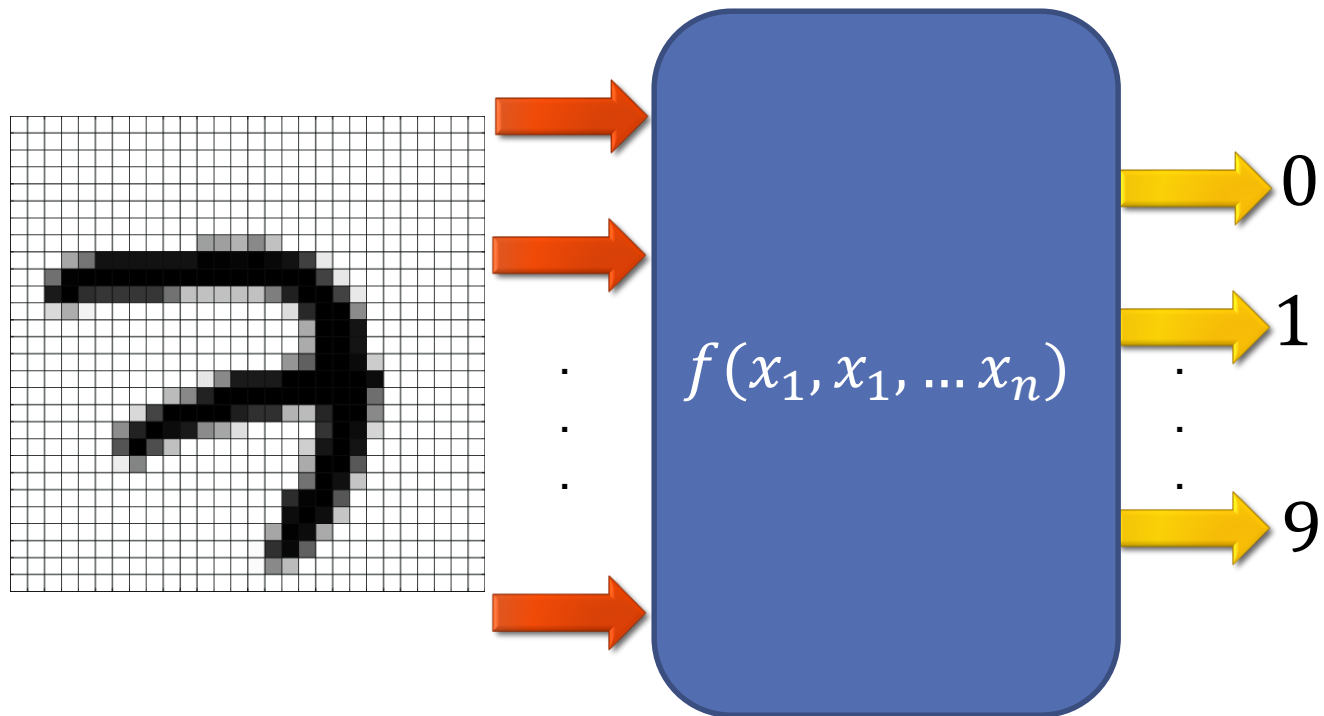
Input data

- Bài toán thực tế
 - Nhiều input và
 - Nhiều output

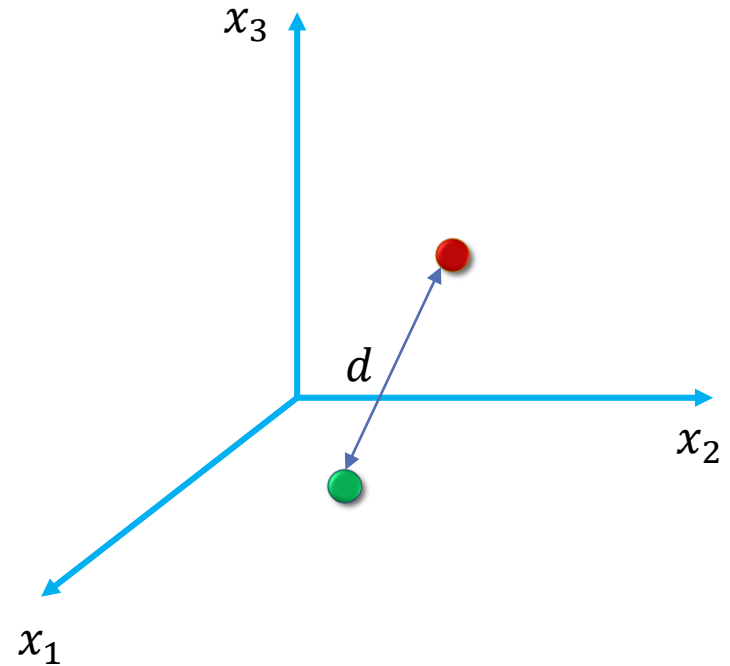
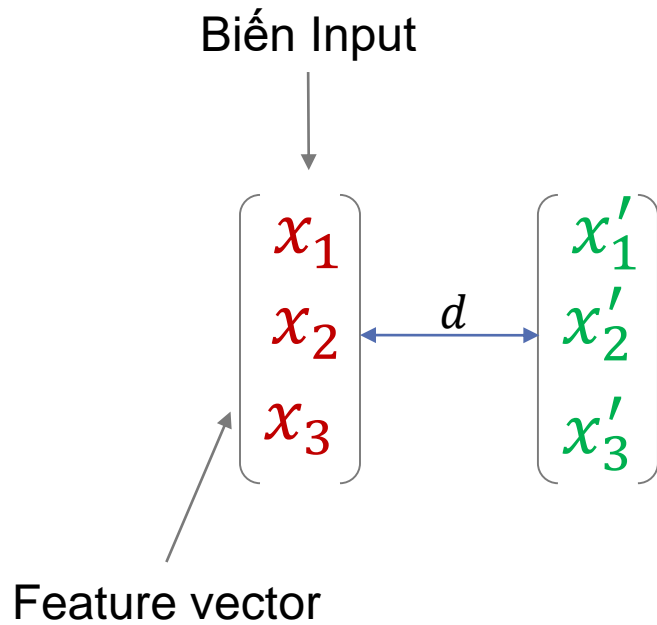


Input data

- Ví dụ



Vector đặc trưng



$$d = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + (x_3 - x'_3)^2}$$

Vector đặc trưng

- **Vấn đề:** Không thể dùng dữ liệu thô để xây dựng vector đặc trưng

- Ví dụ

x_1 (năm)	x_2 (km)	y (giá)
6	1200	10
3	4000	30
8	8000	20

$$\begin{aligned}d &= \sqrt{\left(x_1^{(1)} - x_1^{(2)}\right)^2 + \left(x_2^{(1)} - x_2^{(2)}\right)^2} \\&= \sqrt{(6 - 3)^2 + (1200 - 4000)^2} \\&= 2800\end{aligned}$$

Chuẩn hóa đặc trưng

- Dùng Normalization (Min-Max Scaling)

$$x_i^{(j)scaled} = \frac{x_i^{(j)} - \min(x_i)}{\max(x_i) - \min(x_i)} \in [0,1]$$

$$x_i^{(j)scaled} = \frac{x_{i,j}}{\max(x_i)} \in [0,1] \quad \text{Nếu } \min(x_i) = 0$$

- Ví dụ

x_1	x_1^{scaled}
6	$\frac{6 - 3}{5} = 0.6$
3	$\frac{3 - 3}{5} = 0.0$
8	$\frac{8 - 3}{5} = 1.0$

$$\max(x_1) = 8$$

$$\min(x_1) = 3$$

$$\max(x_1) - \min(x) = 5$$

x_2	x_2^{scaled}
1200	$\frac{1200 - 1200}{6800} = 0.0$
4000	$\frac{4000 - 1200}{6800} = 0.4$
8000	$\frac{8000 - 1200}{6800} = 1.0$

$$\max(x_2) = 8000$$

$$\min(x_2) = 1200$$

$$\max(x_2) - \min(x) = 6800$$

Chuẩn hóa đặc trưng

- Dùng Standardization (Z-Score Normalization)

$$x_i^{(j)scaled} = \frac{x_{i,j} - \mu_i}{\sigma_i}$$

$$\mu_i = \frac{\sum x_{i,j}}{m}$$

$$\sigma_i = \sqrt{\frac{\sum (x_{i,j} - \mu_i)^2}{m}}$$

- Ví dụ

x_1	x_1^{scaled}
6	0.16
3	-1.30
8	1.14

$$\mu_1 = 5.67$$

$$\sigma_1 = 2.05$$

x_2
1200
4000
8000

$$\mu_2 = 4400$$

$$\sigma_2 = 2790.46$$

x_2^{scaled}
-1.15
-0.14
1.29

Chuẩn hóa đặc trưng

STT	Normalization	Standardization
1	Sử dụng giá trị max, min của feature để chuẩn hóa	Sử dụng giá trị trung bình, độ lệch chuẩn hóa
2	Được dùng khi các features có miền giá trị khác nhau	Được dùng khi muốn đặc trưng có mean=0 và độ lệch chuẩn = 1
3	Được dùng khi không biết phân bố của đặc trưng	Được dùng khi phân bố của đặc trưng là phân bố chuẩn
4	Giá trị được scale vào [0,1]	Không có miền xác định
5	Bị tác động bởi outliers	Rất ít bị tác động bởi outliers

Categorical feature

x_1 (năm)	x_2 (km)	x_3 (màu)	y (giá)
6	1200	Xanh	10
3	4000	Đỏ	30
8	8000	Vàng	20

- **Encoding 1:**
 - Xanh = 1
 - Đỏ = 2
 - Vàng = 3
- **Nhận xét:** Vi phạm về khoảng cách
 - Xanh gần với Đỏ hơn là gần với Vàng

Categorical feature

x_1 (năm)	x_2 (km)	x_2 (màu)	y (giá)
6	1200	Xanh	10
3	4000	Đỏ	30
8	8000	Vàng	20

- **Encoding 2: One hot encoding**

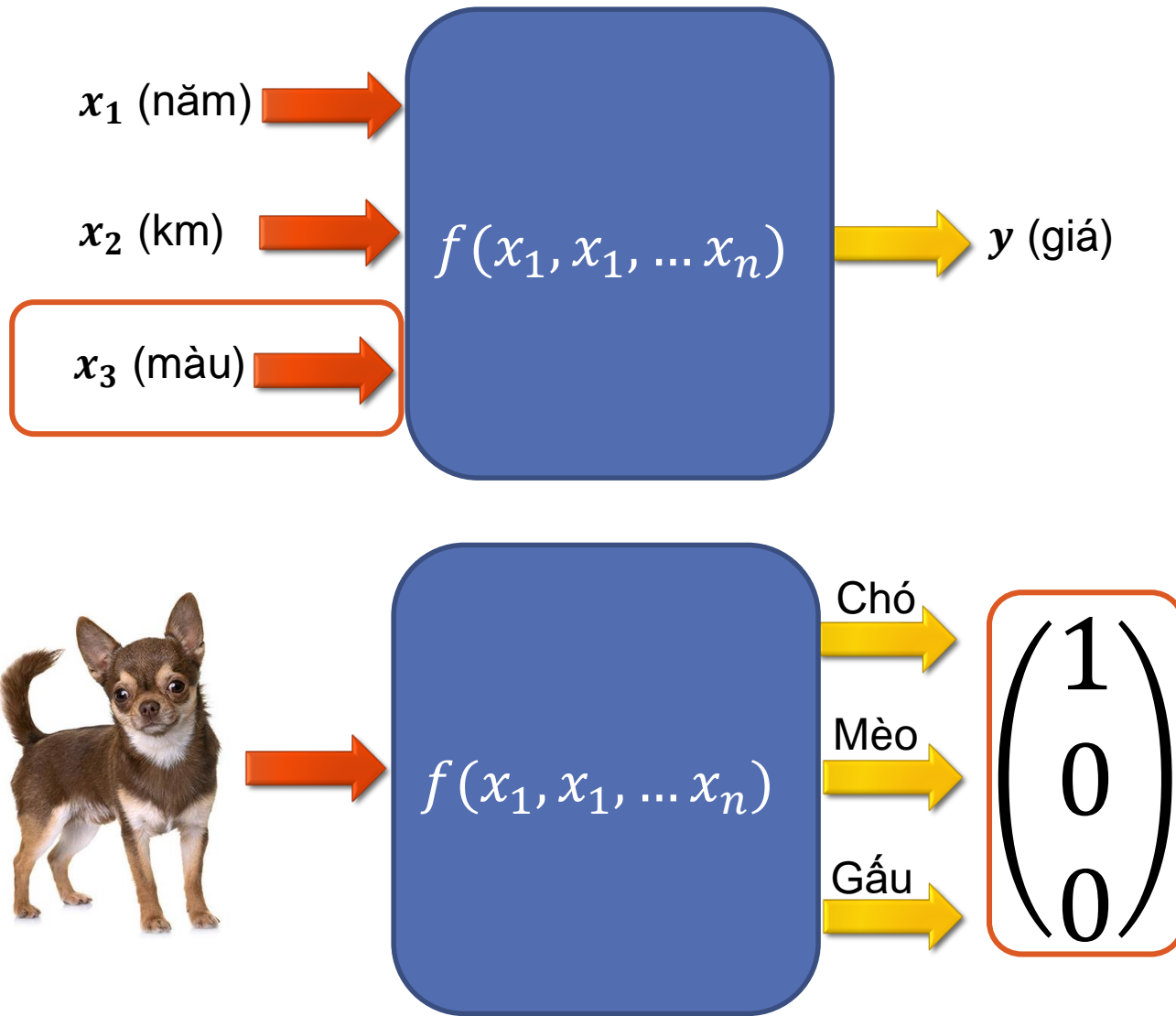
- Tạo vector có k chiều (k là số giá trị khác nhau của categorical feature)
- Mỗi loại sẽ có giá trị **1 (hot)** chỉ trong một chiều
- Mỗi loại sẽ có khoảng cách bằng nhau với các loại khác

$$Xanh = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$Đỏ = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$Vàng = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Categorical feature



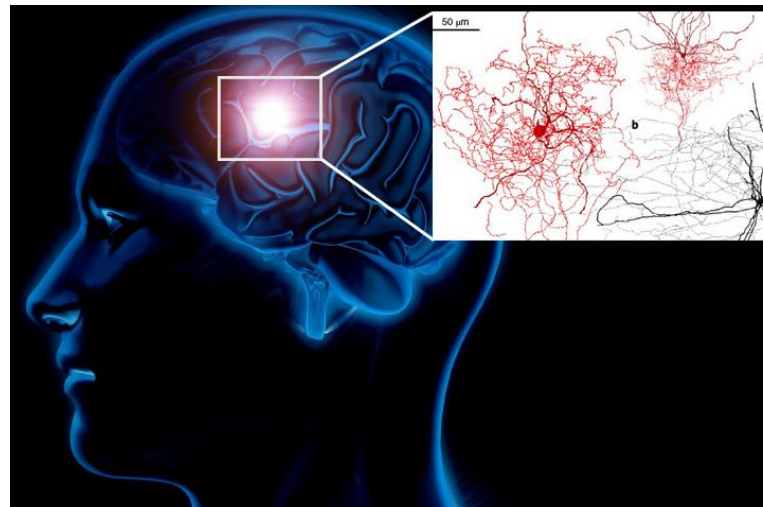
Tóm tắt

- Chuyển **Input data** thành **feature vector**
- Mỗi **feature vector** là một điểm trong **feature space**
- **Feature space** phải có ý nghĩa về khoảng cách
- Nếu feature values là
 - Số: phải được **chuẩn hóa**
 - Chữ: nên dùng **one hot encoding**

NEURON SINH HỌC

Mạng neuron sinh học

- **Dog** có khả năng
 - Nhận ra người quen và người lạ
 - Đọc được biểu cảm trên khuôn mặt
- Mỗi động vật chứa **một mạng lưới neuron sinh học** (biological neural networks)



Mạng neuron sinh học

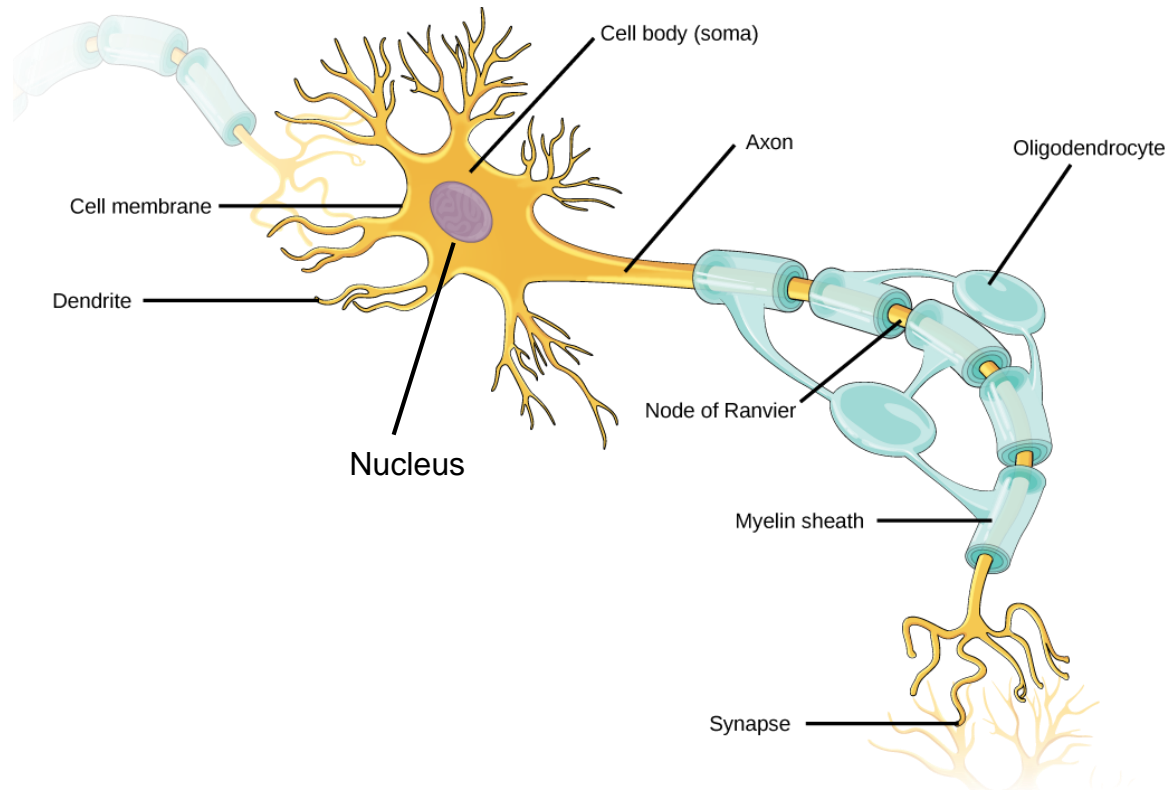
- Biological **neural network**
 - **Neural** là tính từ của neuron
 - **Network** chỉ một cấu trúc Graph
 - Biological **neural network**: gồm một số rất lớn các **neurons**, các neuron kết nối với nhau thành một mạng

Name	Neurons in the brain	Cortex	Synapses
African elephant	257 tỷ	5,6 tỷ	
Human	86 tỷ	16 tỷ /1224 g	1.5×10^{14}
Long-finned pilot whale		37,2 tỷ	
Gorilla	33,4 tỷ	9,1 tỷ	
Dog	2 tỷ 253 triệu	530 triệu	
Cat	760 triệu	250 triệu	1×10^{13}

Mật độ neurons trong não người dày đặc hơn trong não cá voi

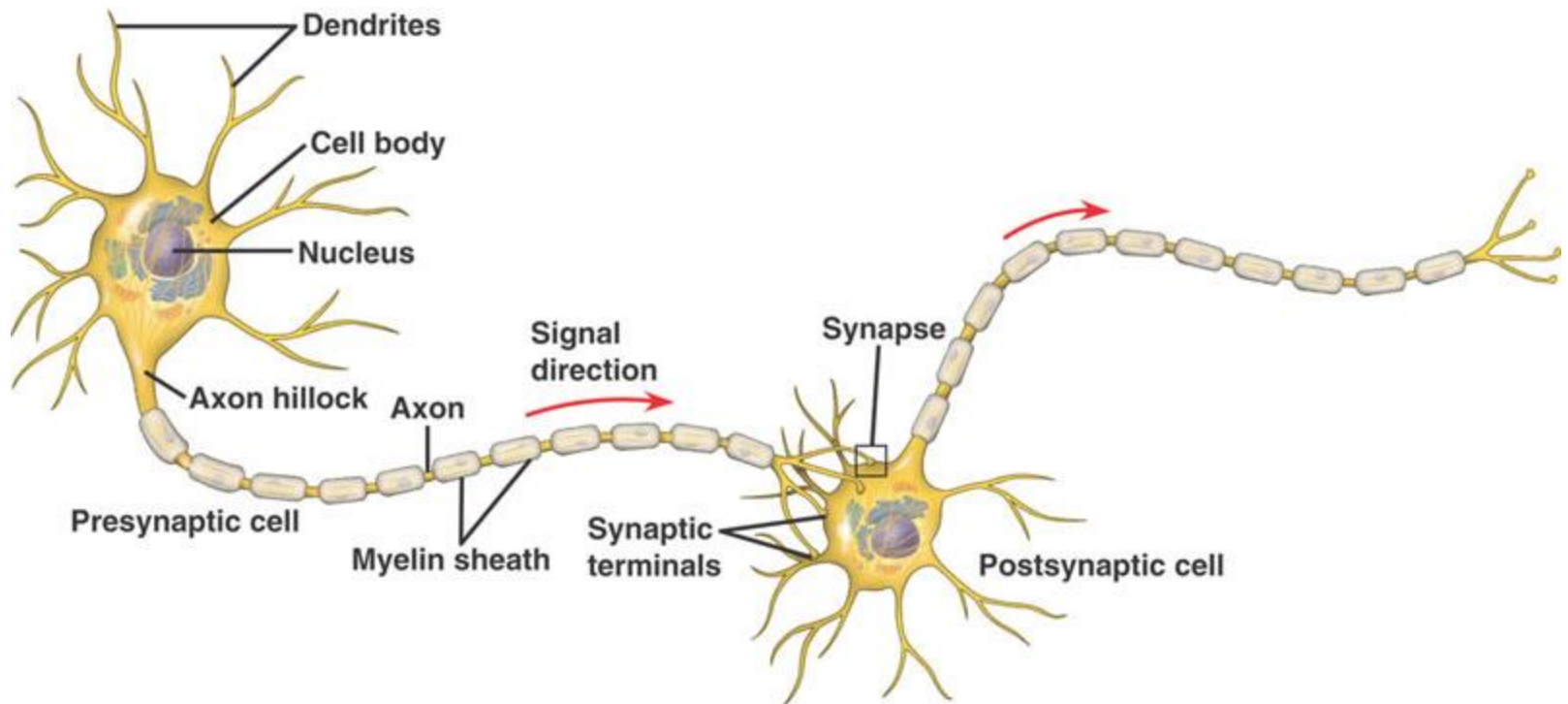
Mạng neuron sinh học

- **Neuron (Brain cell):** Neuron là **tế bào nhận và truyền thông tin** trong hệ thống thần kinh
 - **Nhận tín hiệu** kích thích từ các giác quan, từ các neuron khác
 - **Truyền tín hiệu** đến các neuron khác



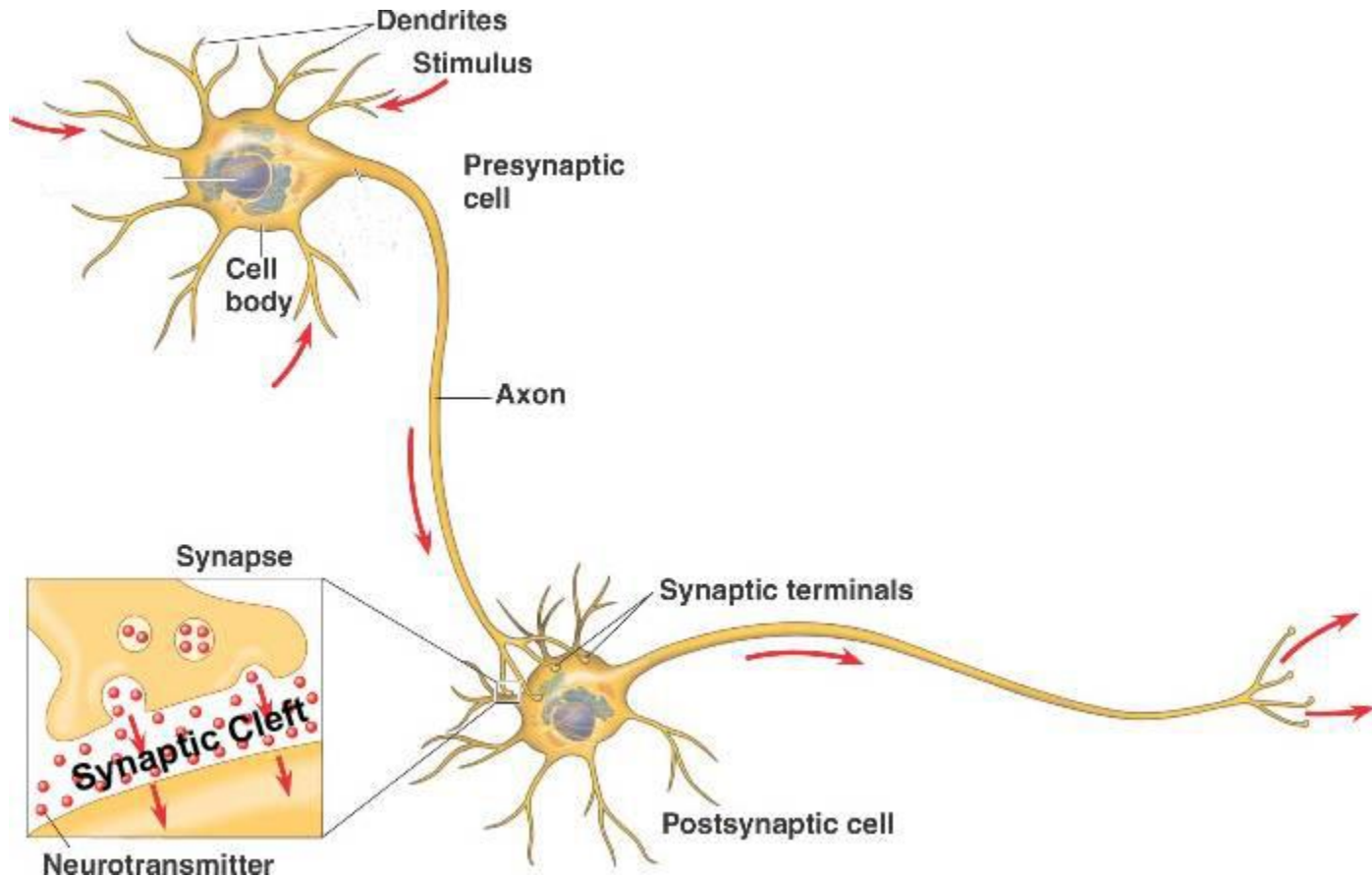
Mạng neuron sinh học

- Kết nối giữa 2 neurons



Mạng neuron sinh học

- **Input:** Nhận tín hiệu điện hóa từ các Dendrites
- **Process:** Nếu các tín hiệu đủ lớn, neuron sẽ **active/fire**
- **Output:** truyền tín hiệu qua axon đến neuron khác



NEURON NHÂN TẠO

(ARTIFICIAL NEURON = PERCEPTRON)

Tác giả

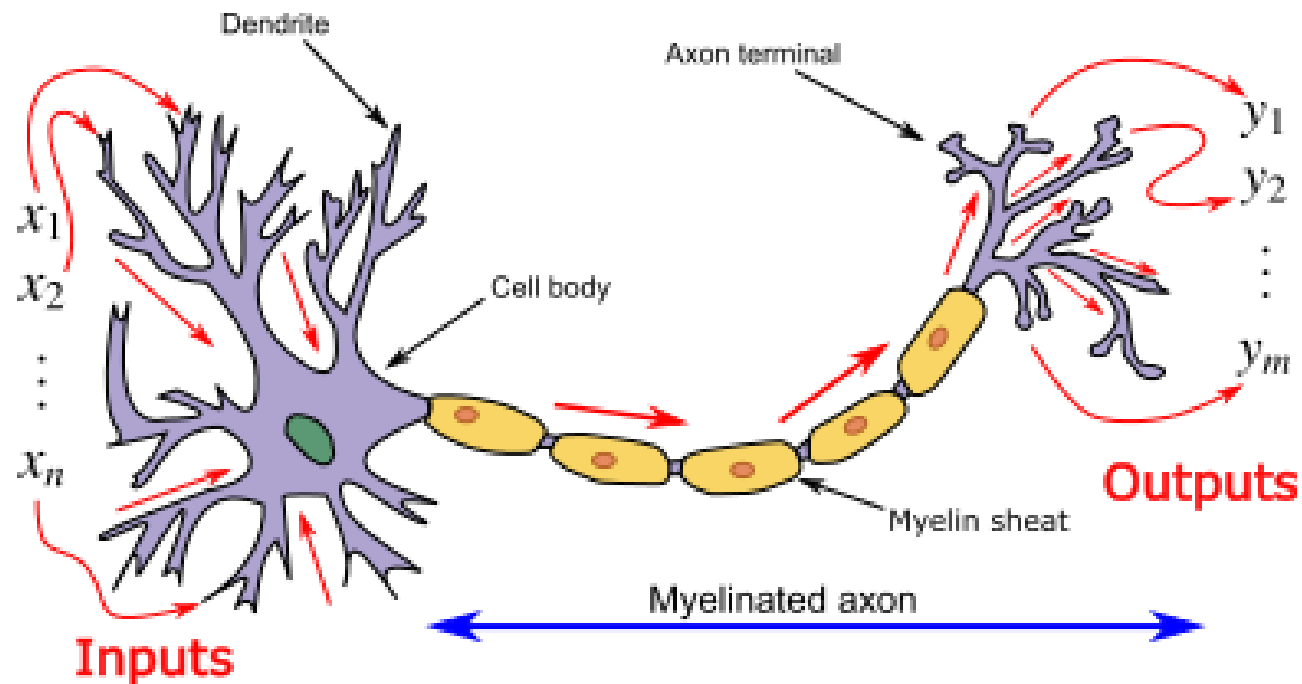
- Tác giả: **Frank Rosenblatt**
- Năm sinh: **1928 – 1971**
- Nhà tâm lý học Mỹ
- **Perceptron** được phát triển trong những năm 1950s và 1960s (cảm hứng từ công trình của Warren McCulloch và Walter Pitts)
- Phát minh **Perceptron** để mô phỏng nguyên tắc của não
 - Có khả năng học
 - Ra quyết định



Perceptron

- Multi Layer Perceptron (Mạng neuron nhân tạo hay Artificial Neural Network)
 - Lấy cảm hứng về cách thức não hoạt động, và áp dụng vào việc giải các bài toán tính toán
 - Deep learning không có mục đích
 - Mô hình/bắt chước cách thức não hoạt động
 - Thay thế não sinh học
- Đơn vị cấu tạo của Multi Layer Perceptron
 - Perceptron (Artificial Neuron)

Neuron

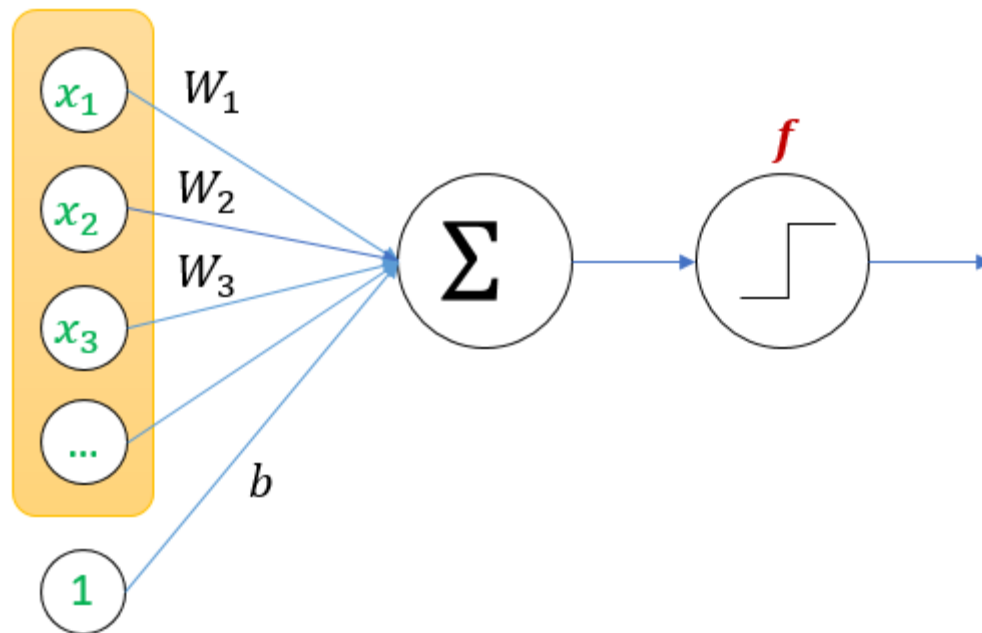


Perceptron

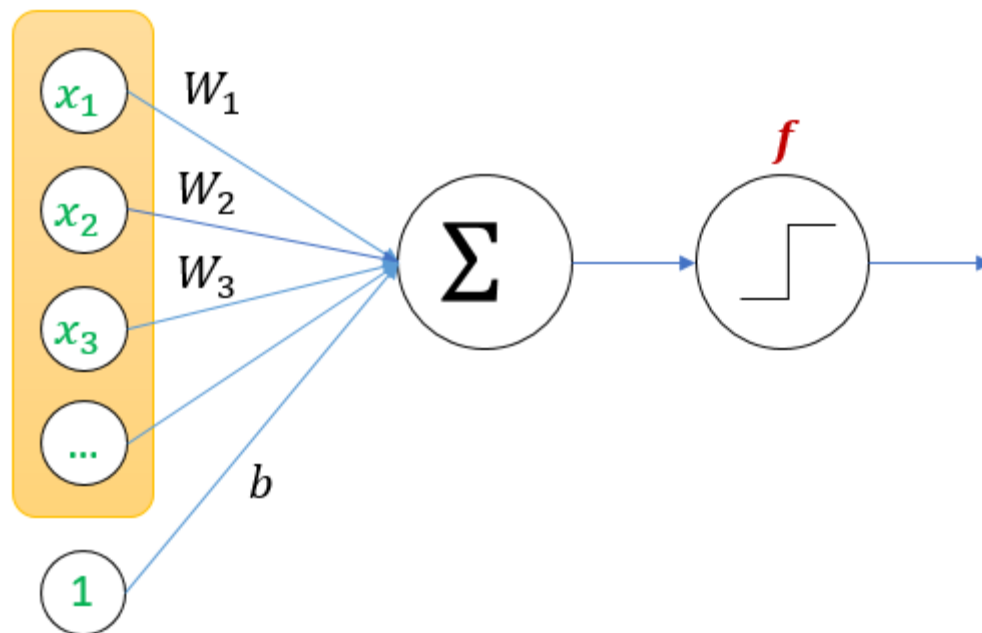
- **Perceptron** (Single Layer Perceptron – **SLP**)

- Nhận các thông tin vào x_1, x_2, \dots
- Sinh ra một giá trị đầu ra y

- Kiến trúc



Perceptron



- **Input**
 - x_1, x_2, \dots tương ứng features của data point
 - b : bias
 - Mỗi x_1, x_2, \dots kết nối với neuron thông qua trọng số (weight) W_1, W_2, \dots
- **Process**
 - Tính tổng trọng số giữa input và các *weight*
 - Sau đó áp dụng hàm f để quyết định neuron có **fire/activate** hay không. f gọi là hàm activation
- **Output**
 - Gửi kết quả tính toán đến các neurons khác

Perceptron

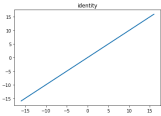
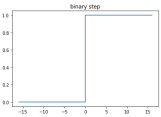
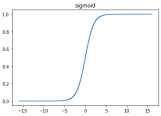
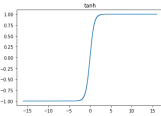
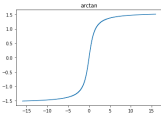
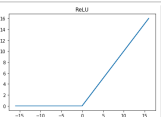
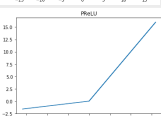
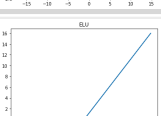
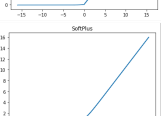
- Biểu diễn toán học

$$f(x_1W_1 + x_2W_2 + \cdots + x_nW_n)$$

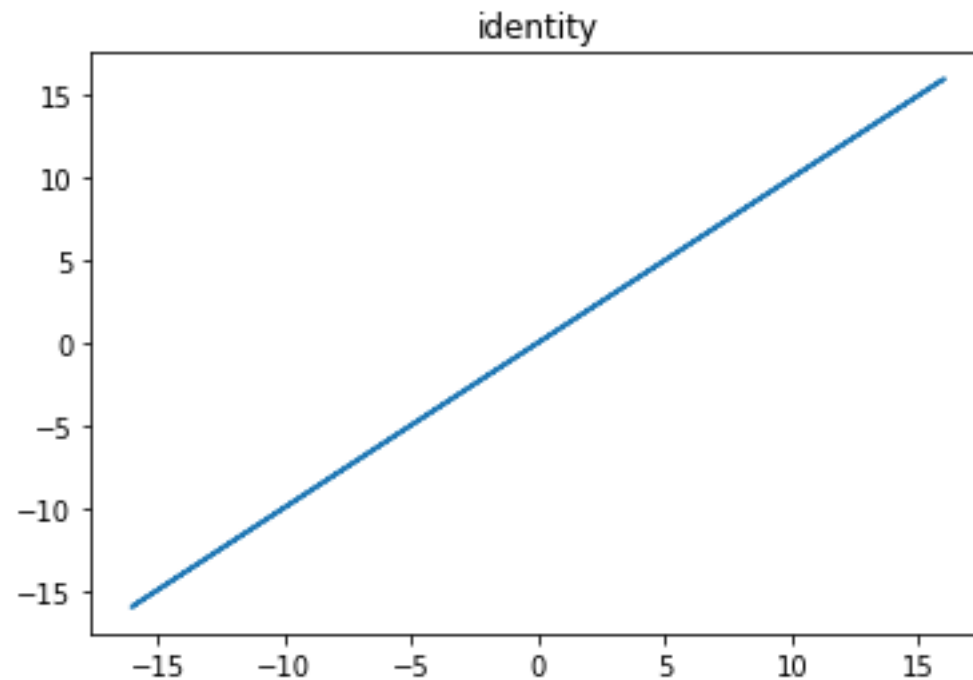
$$f\left(\sum_{i=1}^n x_iW_i\right)$$

$$f(net) \text{ với } net = \sum_{i=1}^n x_iW_i$$

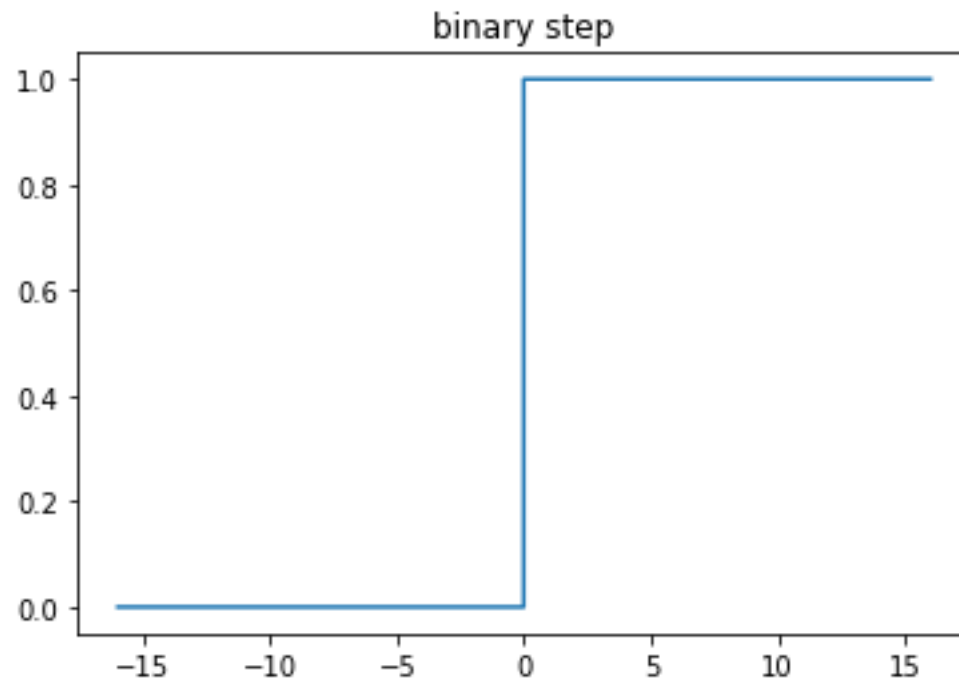
Hàm activation

Name	Plot	Equation	Derivate
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
sigmoid, logistic, soft step		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{1 + x^2}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha \cdot x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha \cdot (e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

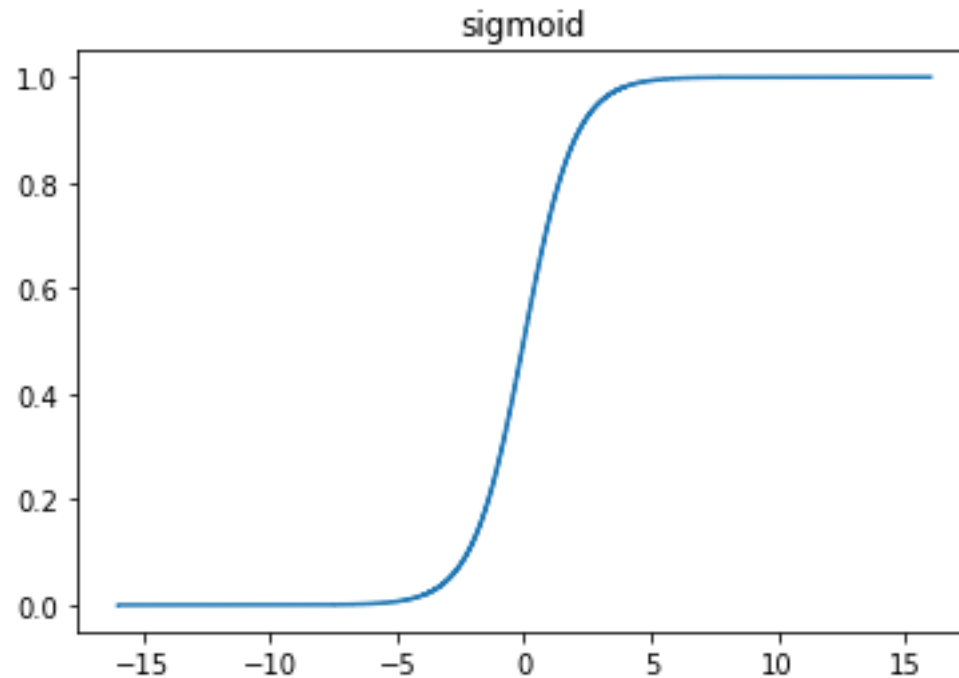
Hàm Identity



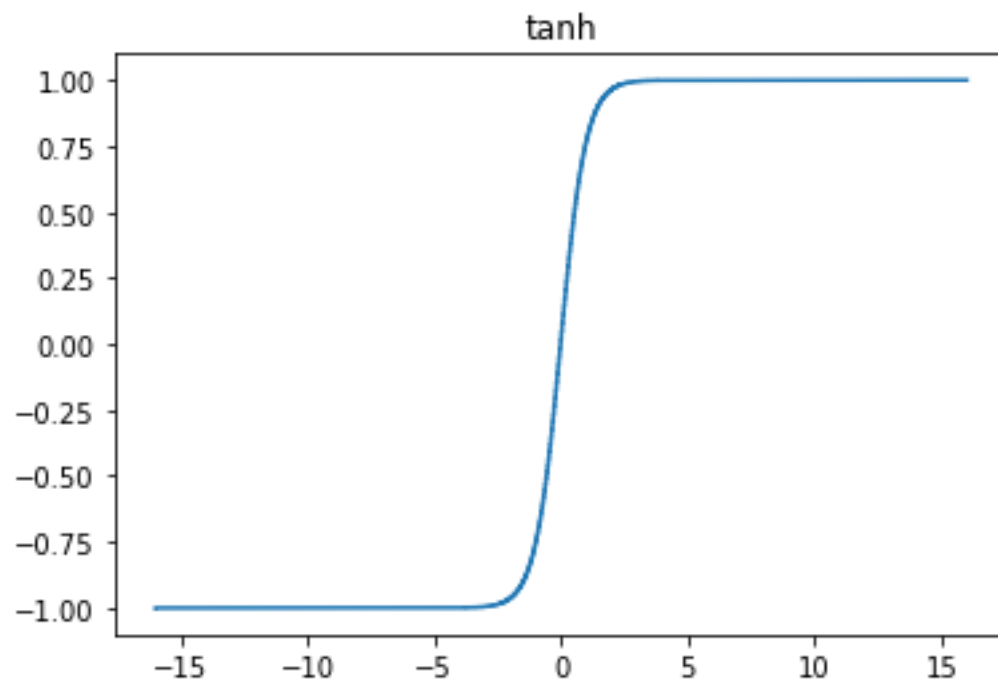
Hàm Binary step



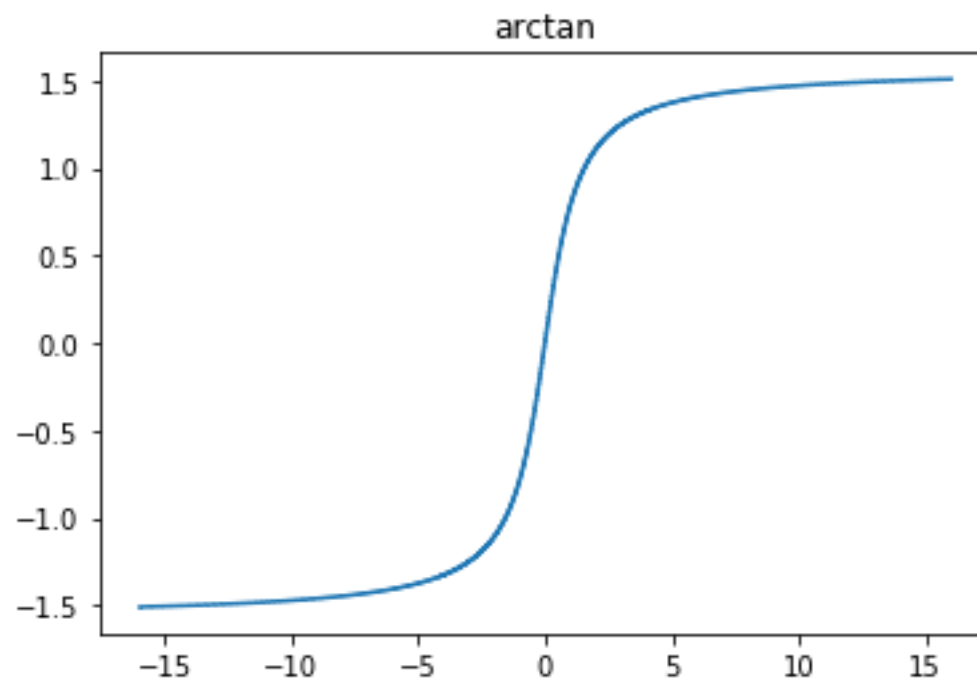
Hàm Sigmoid, logistic, soft step



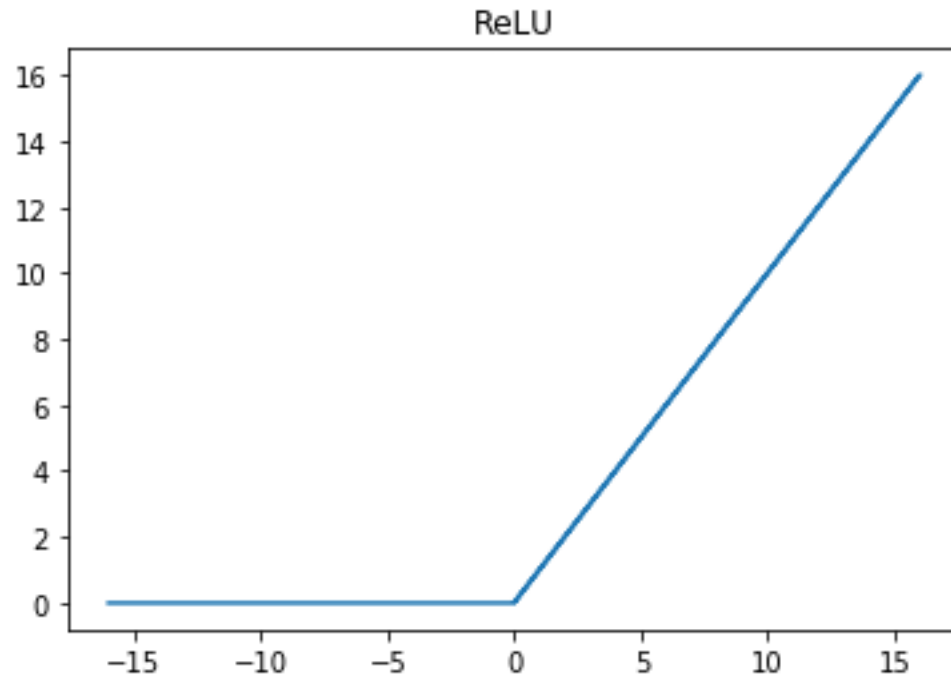
Hàm Tanh



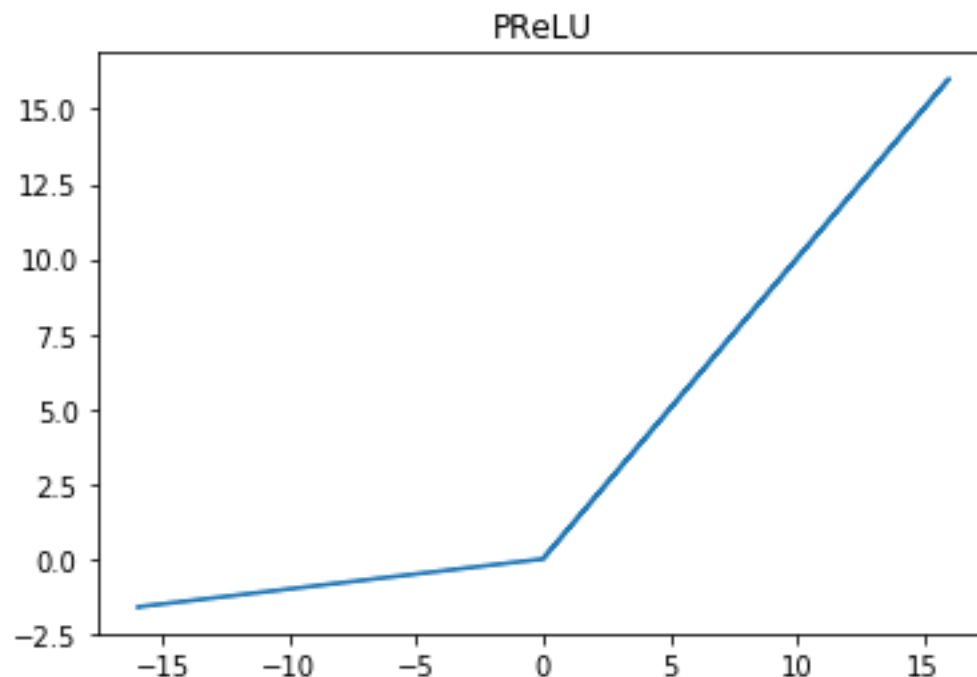
Hàm Arctan



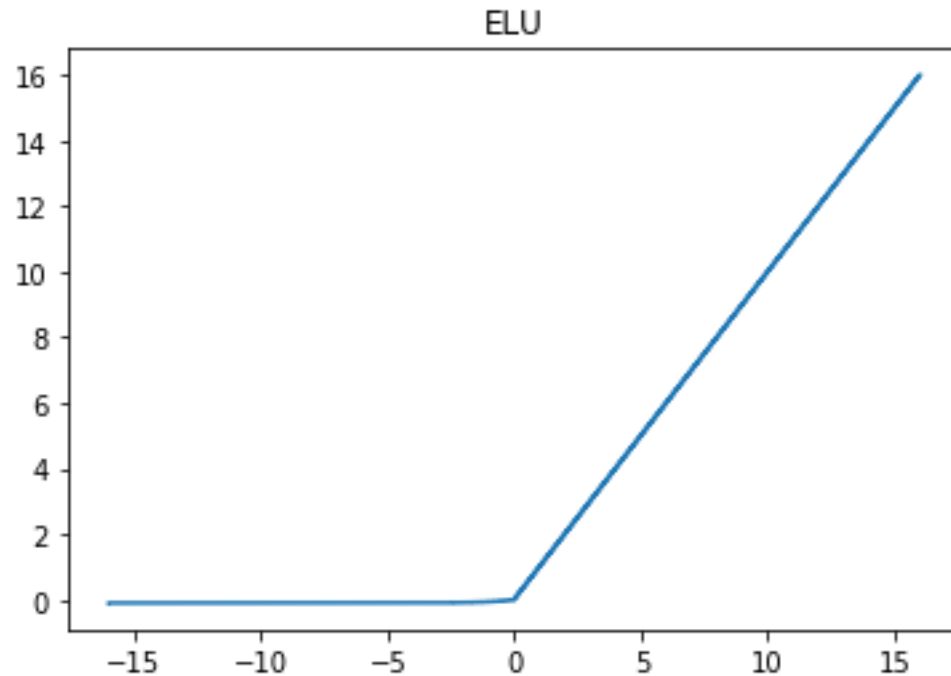
Hàm Rectified Linear Unit (ReLU)



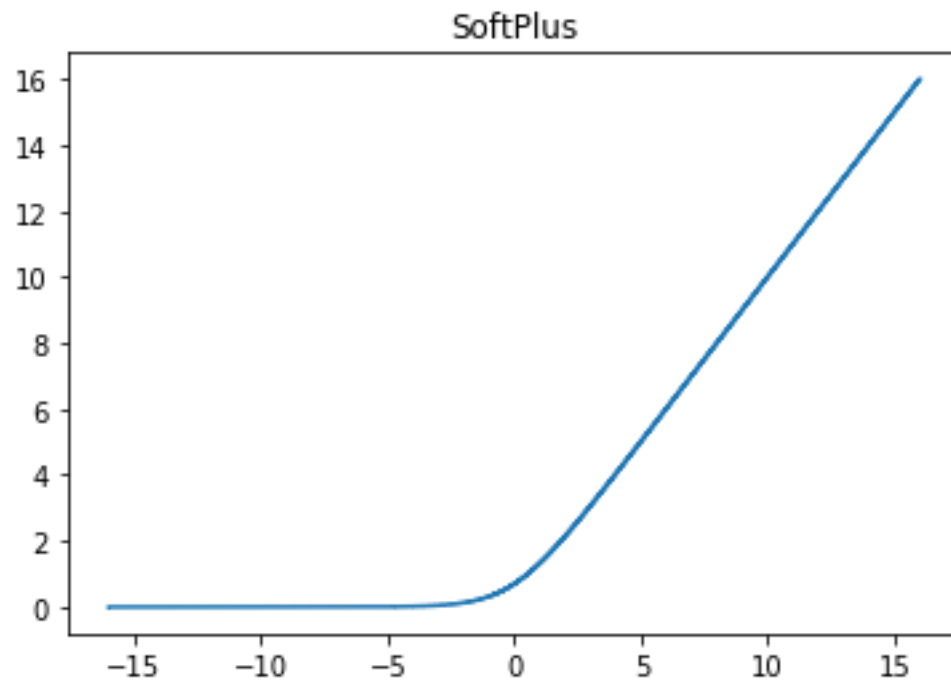
Hàm Parametric Rectified Linear Unit (PReLU)



Hàm Exponential Linear Unit (ELU)



Hàm SoftPlus



Hàm activation

- *Sử dụng hàm activation nào?*
 - **Bước 1.** Lúc đầu nên sử dụng **ReLU**
 - **Bước 2.** Sử dụng **ReLU**, tinh chỉnh các **hyperparameters**:
architecture, learning rate, regularization, ... phù hợp
 - **Bước 3.** Thay **ReLU** bằng PReLU, ELU, ...

Thuật toán học perceptron

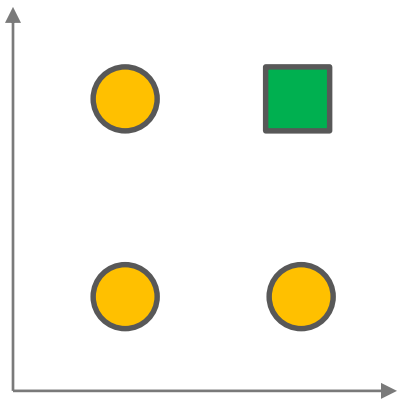
- Bài toán học phép toán bitwise: AND (&), OR (|), XOR (^)

- Dataset

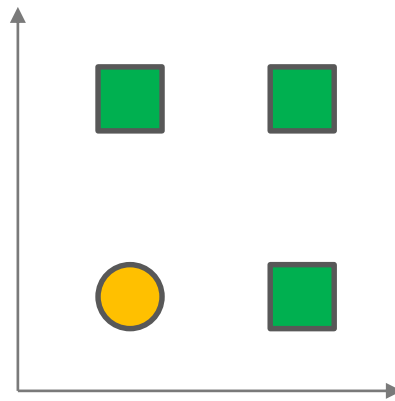
x_1	x_2	&		x_1	x_2			x_1	x_2	^
0	0	0		0	0	0		0	0	0
0	1	0		0	1	1		0	1	1
1	0	0		1	0	1		1	0	1
1	1	1		1	1	1		1	1	0

Thuật toán học perceptron

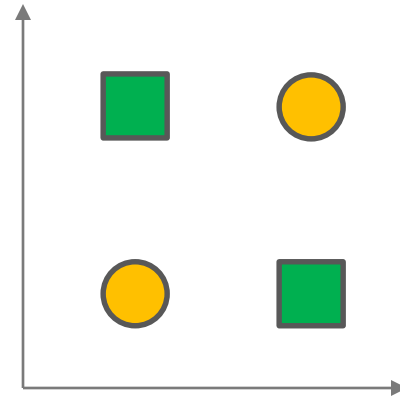
- Biểu diễn dữ liệu



AND



OR



XOR

- Nhận xét

- AND, OR: dữ liệu tuyến tính
- XOR: dữ liệu không tuyến tính

Thuật toán học perceptron

- Học theo *Delta rule*

- Một dạng Gradient descent
- Dựa trên sai lệch giữa (Đạo hàm)
 - Giá trị dự đoán (output, predicted value) và
 - Giá trị thật (growth true value)

- Thuật toán huấn luyện perceptron

Bước 1. Khởi tạo ngẫu nhiên các giá trị nhỏ cho trọng số w

Bước 2. Lặp

- Lặp qua từng data point $(x^{(i)}, y^{(i)})$ trong training set
 - Truyền $x^{(i)}$ qua mạng, tính giá trị output: $\hat{y}^{(i)} = f(w \cdot x^{(i)})$
 - Update các weight w_j : $w_j(t+1) = w_j(t) - \eta(\hat{y}^{(i)} - y^{(i)})x_j$

Cho đến khi hội tụ

Cài đặt perceptron

- Xây dựng lớp Perceptron

```
class Perceptron:
    # Perceptron có 2 biến: W và alpha
    def __init__(self, N, alpha=0.1):
        self.W = np.random.rand(N + 1) / np.sqrt(N)
        self.alpha = alpha

    def step(self, x):
        if x > 0:
            return 1
        return 0
```

Cài đặt perceptron

- Hàm huấn luyện (training)

```
def fit(self, X, y, epochs=10):  
    X = np.c_[X, np.ones((X.shape[0]))]  
  
    for epoch in np.arange(0, epochs):  
        for (x, target) in zip(X, y):  
            y_hat = self.step(np.dot(x, self.W))  
  
            if y_hat != target:  
                error = y_hat - target  
                self.W += -self.alpha * error * x
```

Cài đặt perceptron

- Hàm dự đoán kết quả của perceptron

```
def predict(self, X, addBias=True):  
    X = X.reshape(1,2)  
    X = np.c_[X, np.ones((X.shape[0]))]  
  
    return self.step(np.dot(X, self.W))
```

Cài đặt perceptron

- Sử dụng perceptron cho phép toán AND

```
import numpy as np

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [0], [0], [1]])

print("Training perceptron...")
model = Perceptron(X.shape[1], alpha=0.1)
model.fit(X, y, epochs=20)

print("Testing perceptron...")
for (x, target) in zip(X, y):
    pred = model.predict(x)
    print("data={}, ground-truth={}, pred={}".format(x, target[0], pred))
```

Perceptron

- **THỰC HÀNH**

- Sử dụng Perceptron để huấn luyện mạng học phép toán **OR**
- Sử dụng Perceptron để huấn luyện mạng học phép toán **XOR**

MULTI LAYER PERCEPTRON

Multi Layer Perceptron

1

SLP

Chỉ có thể giải quyết
bài toán tuyến tính

2

FEATURE ENGINEERING

Chuyển dữ liệu phi tuyến về tuyến
tính + SLP

Feature Engineering có hạn chế

3

MLP + LINEAR ACTIVATION

Cần MLP để giải quyết bài toán phi
tuyến

$\text{MLP} + \text{Linear activation} = \text{SLP}$

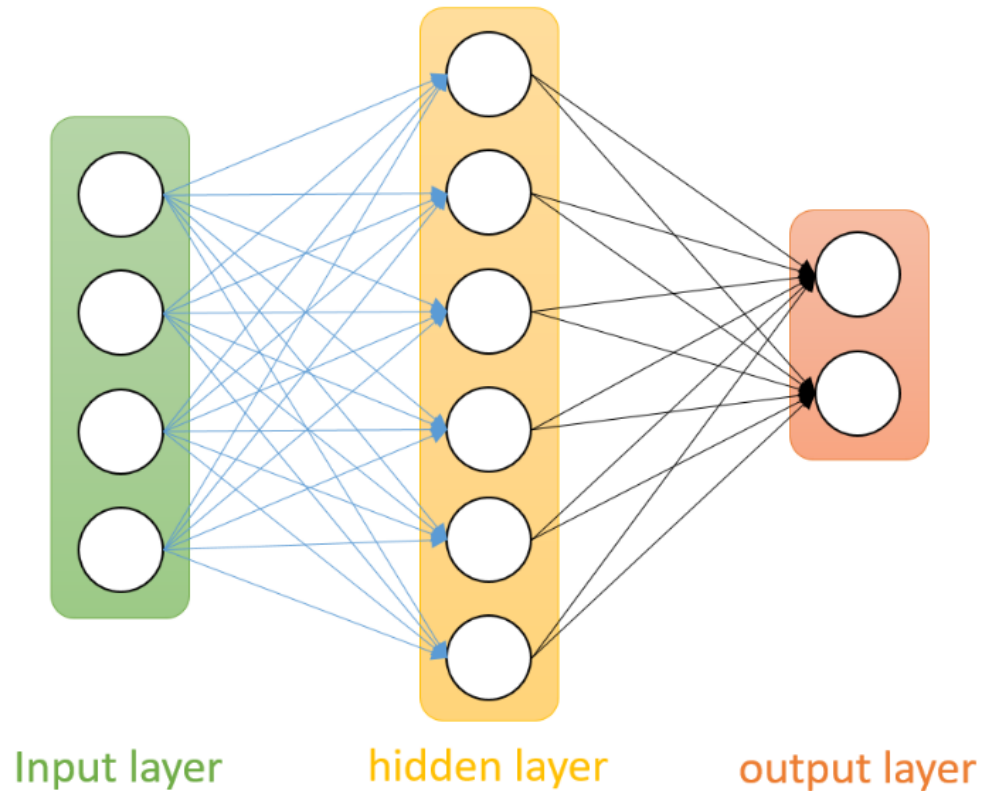
4

MLP + NON-LINEAR ACTIVATION

Giải quyết nhiều bài toán phi tuyến

Multi Layer Perceptron

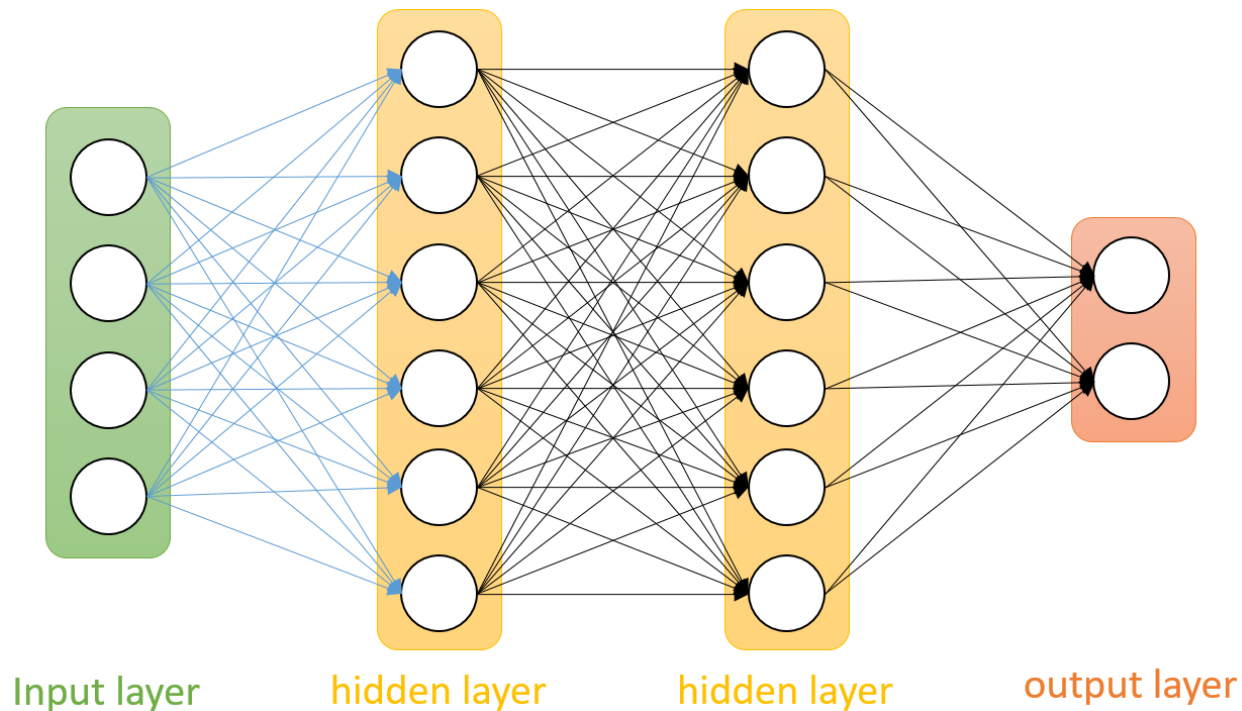
- **Multi Layer Perceptron (MLP)**



Mạng neuron một tầng ẩn

Multi Layer Perceptron

- **Multi Layer Perceptron (MLP)**



Mạng neuron hai tầng ẩn

Multi Layer Perceptron

- **Multi Layer Perceptron (MLP)**

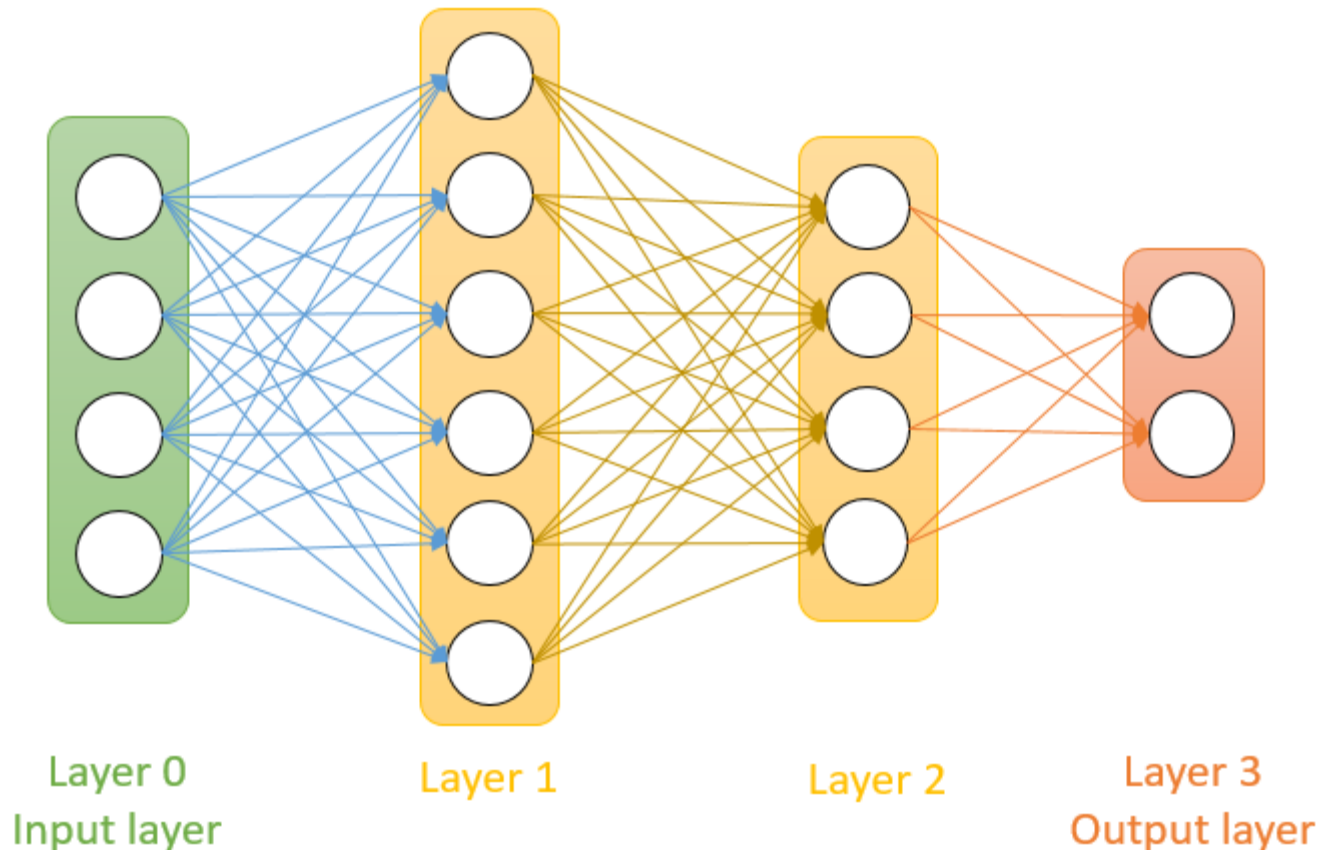
- Một tầng **input**
- Một hay nhiều tầng **hidden**
- Một tầng **output**
- Mọi neuron trong một tầng đều kết nối đầy đủ với các neuron trong tầng kế tiếp (**fully connected**)
- Mọi tầng (trừ tầng output), các neuron đều có giá trị bias
- ANN có hai hay nhiều hidden gọi là **deep neural network** (DNN)

*Multi Layer Perceptrons, Multi Layer Neural Network,
Feedforward Neural Network, Deep feedforward network*

Multi Layer Perceptron

- **Kiến trúc Feedforward network**

- Kiến trúc thông dụng nhất
- Các neurons tầng i nối với các neurons tầng $i + 1$



Multi Layer Perceptron

- **Layer 0**: nhận các giá trị input từ data point
 - Số neurons bằng số features của data point
- **Layer 1, 2**: tầng hidden
 - Số tầng ẩn là **hyperparameter**
 - Số lượng neurons trong từng tầng ẩn là **hyperparameter**
- **Layer 3**: tầng output
 - Số neurons trong output layer bằng số class

Multi Layer Perceptron

- **Neural Learning**

- Phương pháp chỉnh sửa các weights kết nối giữa các neuron

- **Trong sinh học: nguyên tắc Hebb** về quá trình học

“When an axon of cell A is near enough to excite cell B, and repeatedly or persistently takes place in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased” – [Donald Hebb](#)

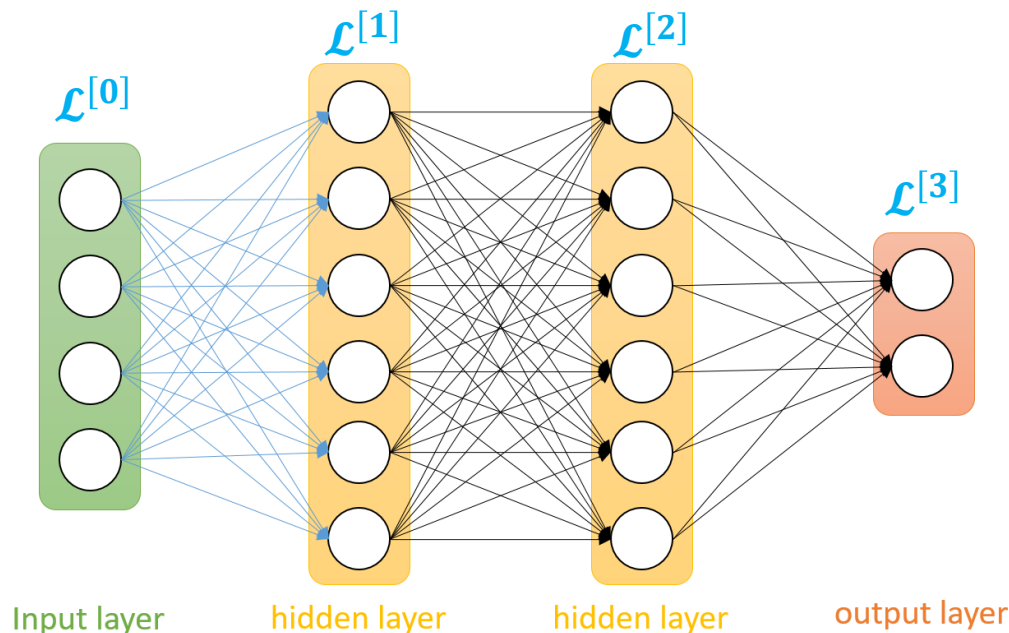
- **Trong MLP: Lấy ý tưởng của Hebb**

- Quá trình học bằng cách tăng cường các kết nối giữa các neurons có output tương tự khi nhận cùng input

Toán học cho MLP

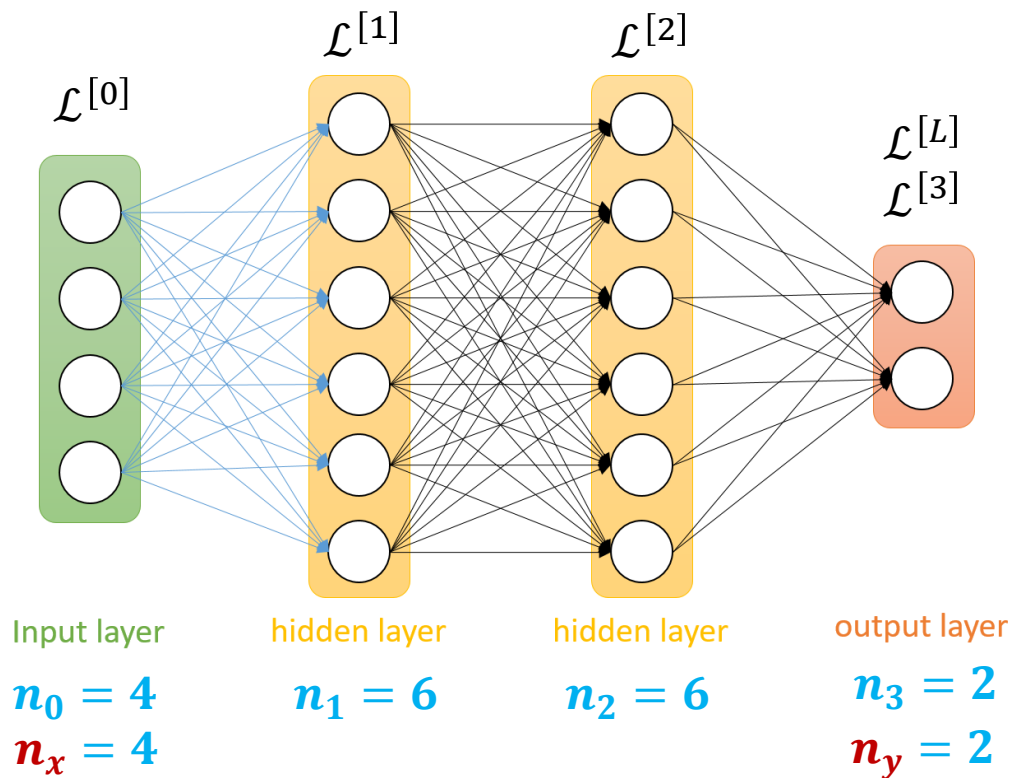
- **Layers**

- Các tầng được đánh số **0, 1, ..., L**
- Gọi $\mathcal{L}^{[0]}$ là tập hợp các neuron ở **tầng input**
- Gọi $\mathcal{L}^{[1]}, \mathcal{L}^{[2]}, \dots$ là tập hợp các neuron ở **tầng hidden**
- Gọi $\mathcal{L}^{[L]}$ là tập hợp các neuron ở **tầng output**



Toán học cho MLP

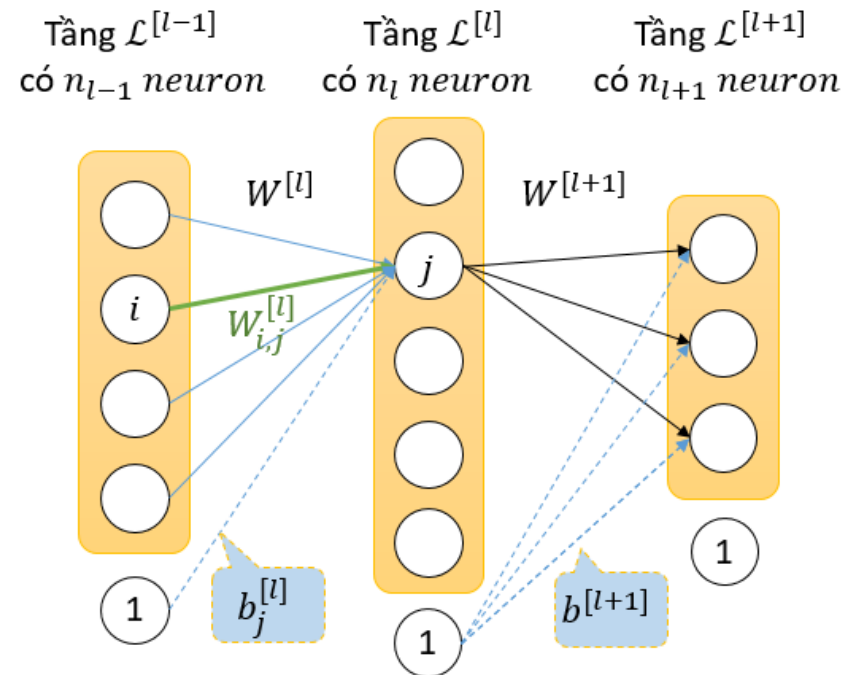
- Số neuron trong tầng l là n_l ($n_x = n_0$ và $n_y = n_L$).



Toán học cho MLP

- **Weights và bias**

- $W = \{W^{[1]}, W^{[2]}, \dots, W^{[L]}\}$
- Giữa 2 tầng kề nhau $\mathcal{L}^{[l-1]}$ và $\mathcal{L}^{[l]}$ có một ma trận trọng số $W^{[l]}$ mô tả mối liên kết giữa hai tầng, trong đó $W_{i,j}^{[l]}$ cho biết mức độ ảnh hưởng của neuron thứ i trong tầng $l - 1$ đến neuron thứ j trong tầng l ($W^{[l]} \in \mathbb{R}^{n_{l-1} \times n_l}$)



- $b = \{b^{[1]}, b^{[2]}, \dots, b^{[L]}\}$ ($b^{[l]} \in \mathbb{R}^{n_l}$)
- Bias của neuron thứ i ở tầng l là $b_i^{[l]}$

Toán học cho MLP

- **Nhiệm vụ của neural network** là xấp xỉ mối quan hệ (x, y) của tập dữ liệu $D = \{x^{(i)}, y^{(i)}\}_{i=1}^m$ bằng một hàm số dạng

$$\hat{y} = g^{[L]}(g^{[L-1]}(\dots(\mathbf{g}^{[2]}(\mathbf{g}^{[1]}(\mathbf{x}))))$$

- Trong đó
 - Tầng 1: $\mathbf{g}^{[1]}(\mathbf{x})$
 - Tầng 2: $\mathbf{g}^{[2]}(\mathbf{g}^{[1]}(\mathbf{x}))$, ...
- Mỗi tầng là hàm có dạng

$$g^{[l]}(a^{[l-1]}) = f(\mathbf{a}^{[l-1]} \cdot \mathbf{W}^{[l]} + \mathbf{b}^{[l]})$$

LAN TRUYỀN TIẾN

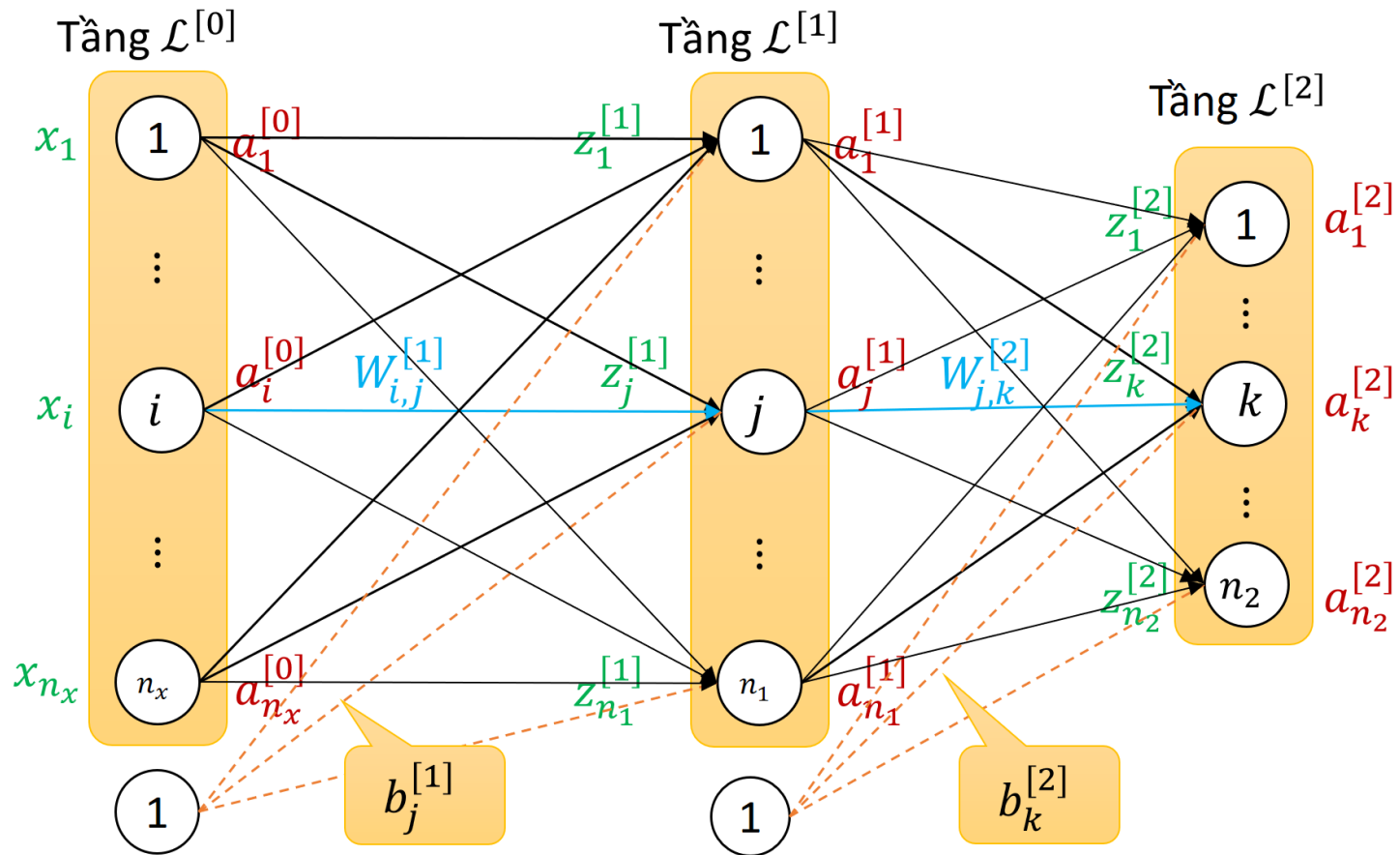
Hoạt động trong mạng neuron

- **Lan truyền tiến (Forward pass):** Dùng để tính giá trị $\hat{y}^{(i)}$ của $x^{(i)}$
 - Dữ liệu $x^{(i)}$ được đưa vào mạng neuron thông qua tầng input.
 - Tính giá trị output của **mỗi neuron** tuần tự các tầng $1, 2, \dots, L$
 - Dữ liệu tại tầng output là $\hat{y}^{(i)}$ của $x^{(i)}$.
- **Lan truyền ngược (Backward pass):** Dùng để cập nhật các $W = \{W^{[1]}, W^{[2]}, \dots, W^{[L]}\}$ và $b = \{b^{[1]}, b^{[2]}, \dots, b^{[L]}\}$
 - Tính giá trị lỗi của network tại tầng output
 - Tính xem mỗi neuron trong tầng hidden cuối cùng đóng góp bao nhiêu vào mỗi lỗi của output neuron, và cứ thế cho đến tầng input

Tiến trình ngược này đo lường đạo hàm lỗi qua tất cả kết nối weights trong mạng bằng **lan truyền (propagate)** đạo hàm lỗi ngược lại trong mạng

Lan truyền tiến

- Các giá trị phải tính trong lan truyền tiến



Lan truyền tiến

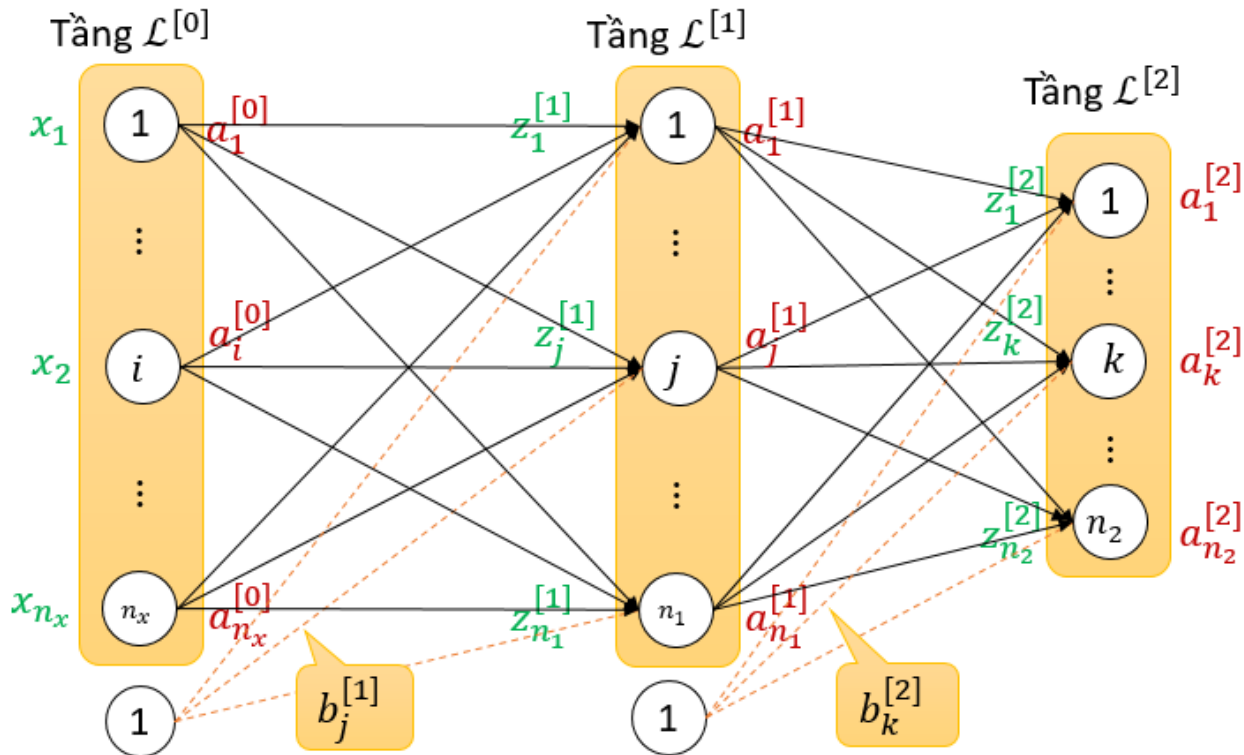
- Gọi dữ liệu vào của bài toán là $x \in \mathbb{R}^{n_x}$ và $a_i^{[l]}$ là giá trị output (hay activation) của neuron i tại tầng l ($a^{[l]} \in \mathbb{R}^{n_l}$)
- Trong Lan truyền tiến, chúng ta tính output của từng tầng: $a^{[1]}, a^{[2]}, \dots, a^{[L]}$

$$\begin{aligned}a^{[0]} &= x \\a^{[1]} &= \sigma(a^{[0]} \cdot W^{[1]} + b^{[1]}) \\a^{[2]} &= \sigma(a^{[1]} \cdot W^{[2]} + b^{[2]}) \\&\dots \\a^{[L]} &= \sigma(a^{[L-1]} \cdot W^{[L]} + b^{[L]})\end{aligned}$$

- $\hat{y} = a^{[L]}$

Lan truyền tiến

- **Nhận xét:** Trong quá trình tính $a^{[l]}$
 - Chúng ta tính toán giá trị trung gian $z^{[l]} \triangleq a^{[l-1]} \cdot W^{[l]} + b^{[l]}$, ($z^{[l]} \in \mathbb{R}^{n_l}$) là tổng trọng số các giá trị input
 - $a^{[l]}$ giá trị hàm activation σ của $z^{[l]}$.



Lan truyền tiến

Thuật toán Lan truyền tiến

Input : Dữ liệu $x \in \mathbb{R}^{n_x}$

Output: Giá trị tại tầng $\mathcal{L}^{[L]}$

$$a^{[0]} \leftarrow x \in \mathbb{R}^{n_x}$$

for $l = 1$ **to** L **do**

$$z^{[l]} \leftarrow a^{[l-1]} \cdot W^{[l]} + b^{[l]}$$

$$a^{[l]} \leftarrow \sigma(z^{[l]})$$

end

return $a^{[L]}$

Tóm lại, mạng neuron dùng để xây dựng mô hình f cho bài toán theo hai tập tham số W, b . Hàm biểu diễn bởi mạng neuron là $\hat{y} = f(x, W, b) = a^{[L]}$.

ĐẠO HÀM

Đạo hàm

- Đạo hàm của một số hàm thông dụng

Hàm số	Đạo hàm
$f(x) = C$	$f'(x) = 0$
$f(x) = x^n$	$f'(x) = n \cdot x^{n-1}$
$f(x) = \frac{1}{x}$	$f'(x) = -\frac{1}{x^2}$
$f(x) = \sqrt{x}$	$f'(x) = \frac{1}{2\sqrt{x}}$
$f(x) = e^x$	$f'(x) = e^x$
$f(x) = a^x$	$f'(x) = a^x \cdot \ln a$
$f(x) = \ln x$	$f'(x) = \frac{1}{x}$
$f(x) = \log_a x$	$f'(x) = \frac{1}{x \cdot \ln a}$

Đạo hàm

- Quy tắc tính đạo hàm

Quy tắc	Hàm số	Đạo hàm
Quy tắc tổng, hiệu	$f(x) = u \pm v$	$f'(x) = u' \pm v'$
Quy tắc mũ	x^n	$n \cdot x^{n-1}$
Quy tắc nhân	$f(x) = u \cdot v$	$f'(x) = u' \cdot v + v' \cdot u$
Quy tắc thương	$f(x) = \frac{u}{v}$	$f'(x) = \frac{u' \cdot v - v' \cdot u}{v^2}$
Quy tắc nghịch đảo	$f(x) = \frac{1}{u}$	$f'(x) = -\frac{u'}{v^2}$
Quy tắc căn	$f(x) = \sqrt{u}$	$f'(x) = \frac{u'}{2\sqrt{u}}$
Quy tắc chuỗi (dùng ')	$f(g(x))$	$f'(g(x)) \cdot g'(x)$
Quy tắc chuỗi (dùng $\frac{d}{dx}$)		$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$

LAN TRUYỀN NGƯỢC

Lan truyền ngược

- Huấn luyện MLP

- Mạng nhiều tầng không thể huấn luyện bằng quy tắc trong perceptron
- Nhiều năm, các nhà nghiên cứu vật lộn tìm cách huấn luyện MLP

- Năm 1986, Rumelhart, Hinton, William

- Giới thiệu thuật toán backpropagation (Gradient Descent dùng reverse-mode autodif)
- Bài báo “*Learning Representations by Back-Propagating Errors*”

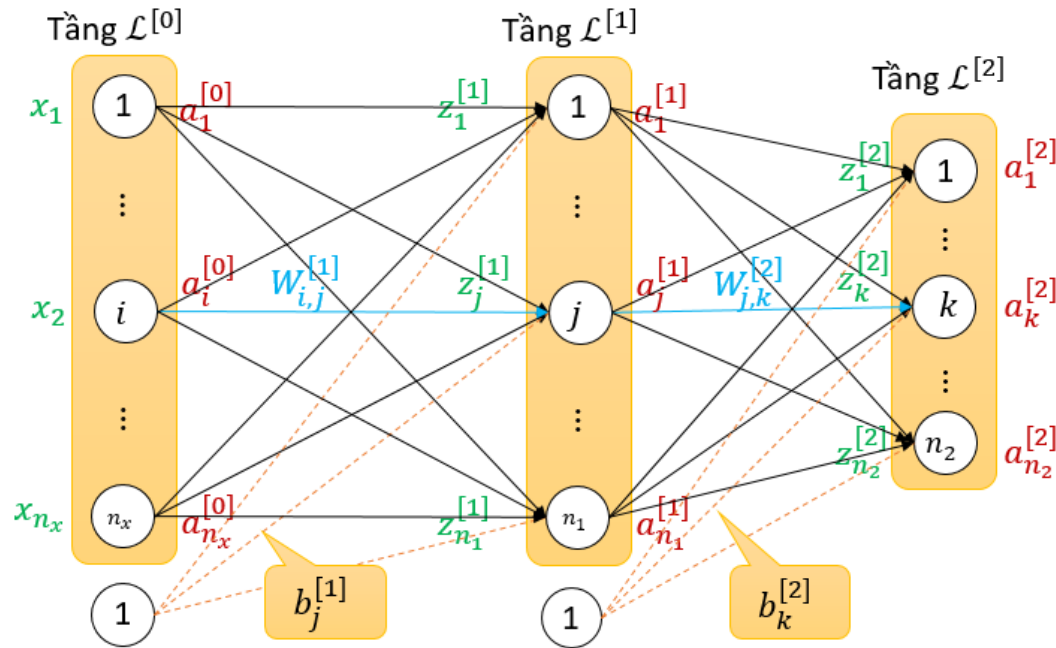
Lan truyền ngược

- Để **cập nhật các weights** trong mạng, ta phải định nghĩa loss function ở tầng output, gọi loss function ở tầng output là $Loss(W, b)$
- Ví dụ: Hàm loss function là Mean Square Error (MSE)

$$Loss(W, b) = \frac{1}{2m} \sum_{train\ set} \|\hat{y} - y\|_2^2 = \frac{1}{2m} \sum_{train\ set} \|a^{[L]} - y\|_2^2$$

- Sau khi đã có loss function, chúng ta phải tính **đạo hàm của loss function trên từng weight**
 - Trực tiếp (rất khó)
 - Gián tiếp, đi từ tầng cuối lên tầng đầu: thuật toán **backpropagation**

Lan truyền ngược



- $W_{j,k}^{[2]}$ nằm trong hàm $z_k^{[2]}$, hàm $z_k^{[2]}$ nằm trong hàm $a_k^{[2]}$

$$W_{j,k}^{[2]} \rightarrow z_k^{[2]} \rightarrow a_k^{[2]}$$

- $W_{i,j}^{[1]}$ nằm trong hàm $z_j^{[1]}$, hàm $z_j^{[1]}$ nằm trong hàm $a_j^{[1]}$, hàm $a_j^{[1]}$ nằm trong các hàm $z_k^{[2]}$, nằm trong các $a_k^{[2]}$

$$W_{i,j}^{[1]} \rightarrow z_j^{[1]} \rightarrow a_j^{[1]} \rightarrow z_k^{[2]} \rightarrow a_k^{[2]}$$

Lan truyền ngược

- **Backward pass (Lan truyền ngược)**

- Thực hiện lan truyền tiến để tính:

- $z^{[1]}, z^{[2]}, \dots, z^{[L]}$

- $a^{[1]}, a^{[2]}, \dots, a^{[L]}$

- Tính đạo hàm của hàm loss cho $W^{[L]}$ ở tầng output
- Áp dụng đạo hàm này một cách đệ quy theo **chain rule** để tính đạo hàm của hàm loss cho $W^{[l]}$ ở tầng l
- Cập nhật các weights

- **Xét 2 phiên bản**

- Phiên bản trên từng biến: $W_{i,j}^l, b_i^l$
- Phiên bản ma trận, vector: W^l, b^l

Lan truyền ngược – Xét từng biến

- Ở tầng cuối

$$z_j^{[L]} \triangleq \sum_{i=1}^{n_{L-1}} a_i^{[L-1]} \cdot W_{i,j}^{[L]} + b_j^{[L]}$$

$$a_j^{[L]} \triangleq \sigma(z_j^{[L]})$$

$$\text{Loss}(W, b) = \|a^{[L]} - y\|_2^2 = \|\sigma(z^{[L]}) - y\|_2^2$$

- Ta muốn tính

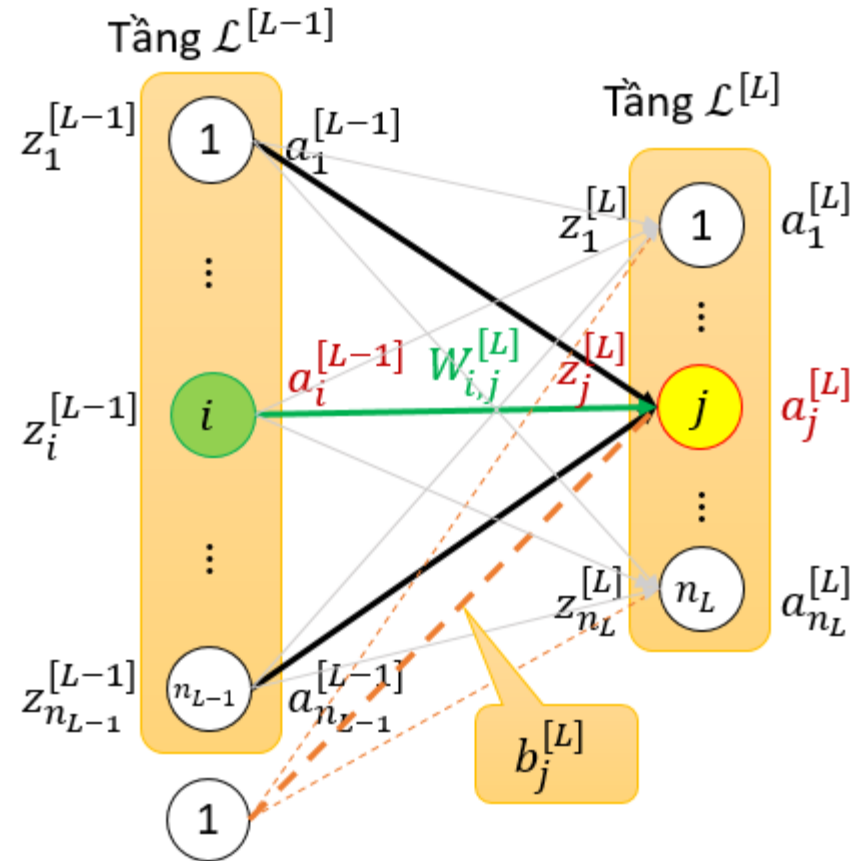
$$\frac{\partial \text{Loss}}{\partial W_{i,j}^{[L]}} = ???$$

$$\frac{\partial \text{Loss}}{\partial b_j^{[L]}} = ???$$

- Nhận xét:

$$W_{i,j}^{[L]} \rightarrow z_j^{[L]} \rightarrow a_j^{[L]}$$

$$b_j^{[L]} \rightarrow z_j^{[L]} \rightarrow a_j^{[L]}$$



Lan truyền ngược – Xét từng biến

• Ở tầng cuối

- Ta có $z_j^{[L]} \triangleq \sum_{i=1}^{n_{L-1}} a_i^{[L-1]} \cdot W_{i,j}^{[L]} + b_j^{[L]}$

$$a_j^{[L]} \triangleq \sigma(z_j^{[L]})$$

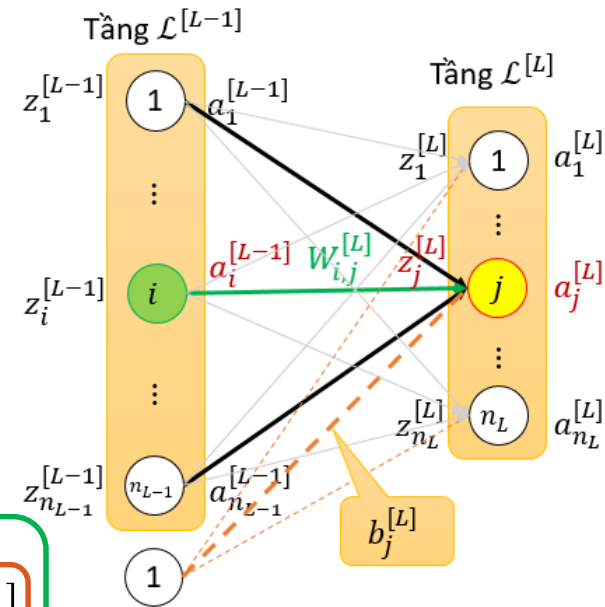
$$\text{Loss}(W, b) = \|a^{[L]} - y\|_2^2 = \|\sigma(z^{[L]}) - y\|_2^2$$

- Suy ra
$$\frac{\partial \text{Loss}}{\partial W_{i,j}^{[L]}} = \frac{\partial \text{Loss}}{\partial z_j^{[L]}} \cdot \frac{\partial z_j^{[L]}}{\partial W_{i,j}^{[L]}} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}} \cdot a_i^{[L-1]}$$

- Trong đó: $\frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$ là đạo hàm của hàm activation $\sigma(x)$ tại giá trị $z_j^{[L]}$

- Tương tự ta có

$$\frac{\partial \text{Loss}}{\partial b_j^{[L]}} = \frac{\partial \text{Loss}}{\partial z_j^{[L]}} \cdot \frac{\partial z_j^{[L]}}{\partial b_j^{[L]}} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}} \cdot 1 = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$$

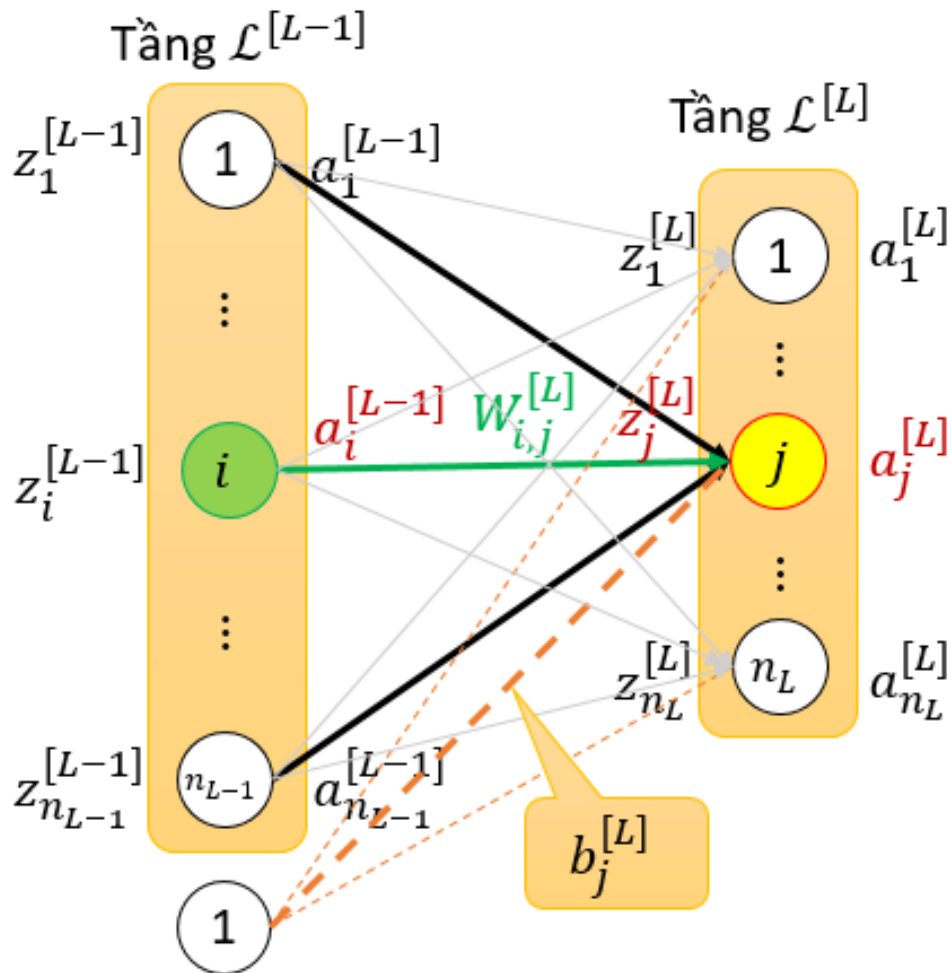


Lan truyền ngược – Xét từng biến

- Ở tầng cuối (tóm tắt)

$$\frac{\partial Loss}{\partial W_{ij}^{[L]}} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}} \cdot a_i^{[L-1]}$$

$$\frac{\partial Loss}{\partial b_j^{[L]}} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$$



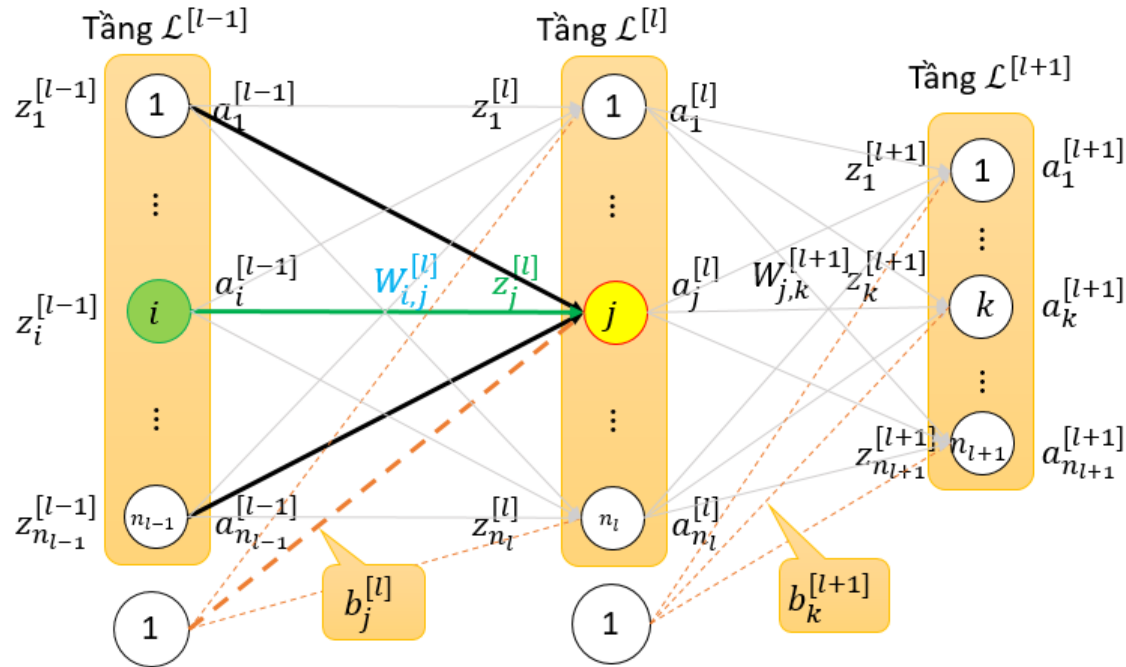
Lan truyền ngược – Xét từng biến

- Ở tầng l ($1 \leq l < L$)

- Muốn tính

$$\frac{\partial Loss}{\partial W_{i,j}^{[l]}} = ???$$

$$\frac{\partial Loss}{\partial b_j^{[l]}} = ???$$



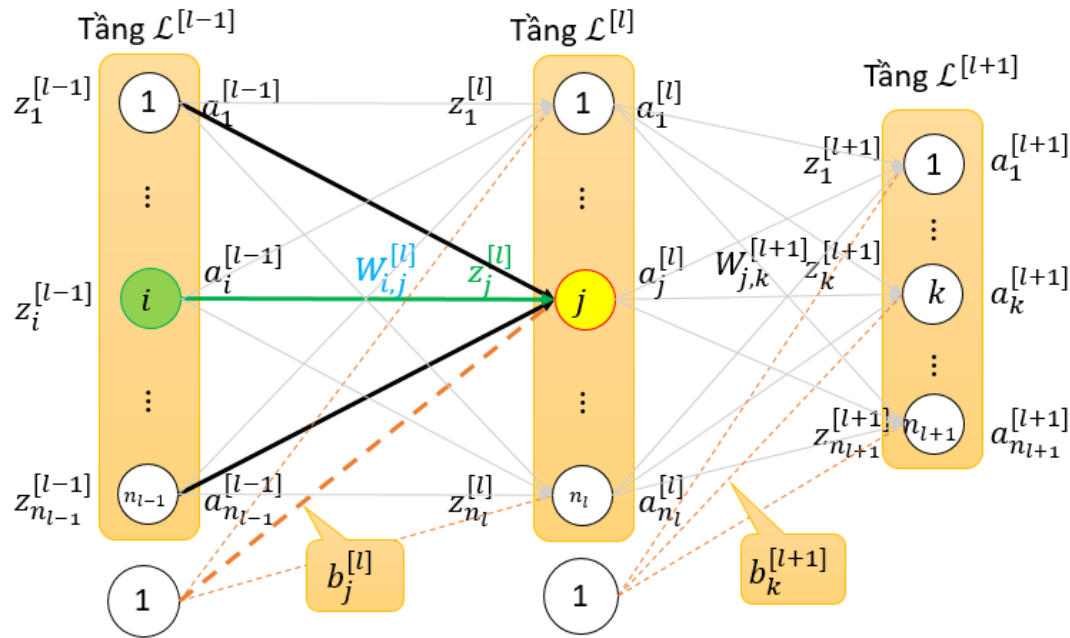
- Nhận xét

$$W_{i,j}^{[l]} \rightarrow z_j^{[l]} \rightarrow a_j^{[l]} \rightarrow \text{các } z^{[l+1]} \rightarrow \text{các } a^{[l+1]}$$

$$b_j^{[l]} \rightarrow z_j^{[l]} \rightarrow a_j^{[l]} \rightarrow \text{các } z^{[l+1]} \rightarrow \text{các } a^{[l+1]}$$

Lan truyền ngược – Xét từng biến

- Ở tầng l ($1 \leq l < L$)



$$z_j^{[l]} \triangleq \sum_{i=1}^{n_{l-1}} a_i^{[l-1]} \cdot W_{i,j}^{[l]} + b_j^{[l]} \quad a_j^{[l]} \triangleq \sigma(z_j^{[l]})$$

$$Loss(W, b) = \|a^{[L]} - y\|_2^2 = \|\sigma(z^{[L]}) - y\|_2^2$$

Lan truyền ngược – Xét từng biến

- Ở tầng l ($1 \leq l < L$)

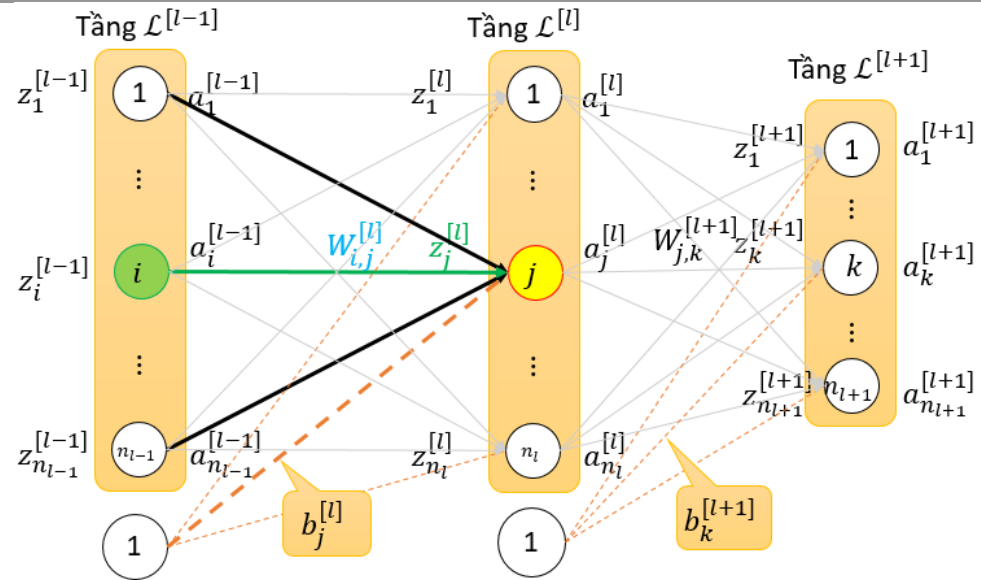
$$Loss(W, b) = \|a^{[L]} - y\|_2^2 = \|\sigma(z^{[L]}) - y\|_2^2$$

- Ta có

$$\frac{\partial Loss}{\partial W_{i,j}^{[l]}} = \frac{\partial Loss}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial W_{i,j}^{[l]}}$$

$$\frac{\partial z_j^{[l]}}{\partial W_{i,j}^{[l]}} = a_i^{[l-1]}$$

$$\frac{\partial Loss}{\partial W_{i,j}^{[l]}} = \frac{\partial Loss}{\partial z_j^{[l]}} \cdot a_i^{[l-1]}$$



$$\text{vì } z_j^{[l]} \triangleq \sum_{i=1}^{n_{l-1}} a_i^{[l-1]} \cdot W_{i,j}^{[l]} + b_j^{[l]}$$

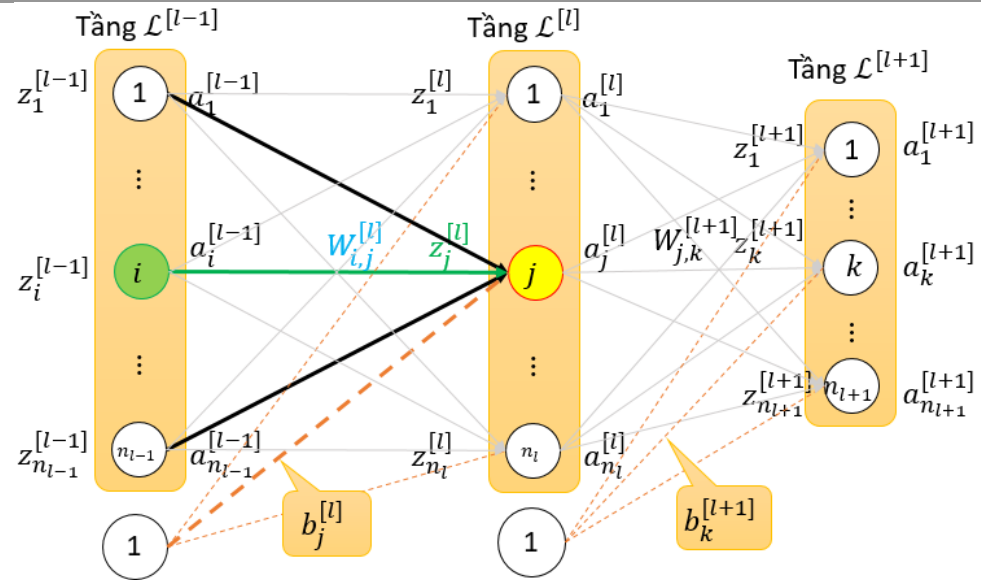
Lan truyền ngược – Xét từng biến

- Ở tầng l ($1 \leq l < L$)

$$\text{Loss}(W, b) = \|a^{[L]} - y\|_2^2 = \|\sigma(z^{[L]}) - y\|_2^2$$

- Ta có

$$\frac{\partial \text{Loss}}{\partial w_{ij}^{[l]}} = \frac{\partial \text{Loss}}{\partial z_j^{[l]}} \cdot a_i^{[l-1]}$$



$$\frac{\partial \text{Loss}}{\partial z_j^{[l]}} = \frac{\partial \text{Loss}}{\partial a_j^{[l]}} \cdot \frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} = \left(\sum_{k=1}^{n_{l+1}} \frac{\partial \text{Loss}}{\partial z_k^{[l+1]}} \cdot \frac{\partial z_k^{[l+1]}}{\partial a_j^{[l]}} \right) \cdot \frac{\partial \sigma(z_j^{[l]})}{\partial z_j^{[l]}} = \left(\sum_{k=1}^{n_{l+1}} \underbrace{\frac{\partial \text{Loss}}{\partial z_k^{[l+1]}} \cdot w_{j,k}^{[l+1]}}_{\text{from diagram}} \right) \cdot \frac{\partial \sigma(z_j^{[l]})}{\partial z_j^{[l]}}$$

Lan truyền ngược – Xét từng biến

- Đặt $delta_j^{[l]} = \frac{\partial Loss}{\partial z_j^{[l]}}$

- Ta có

$$delta_j^{[l]} = \left(\sum_{k=1}^{n_{l+1}} \frac{\partial Loss}{\partial z_k^{[l+1]}} \cdot w_{j,k}^{[l+1]} \right) \cdot \frac{\partial \sigma(z_j^{[l]})}{\partial z_j^{[l]}}$$

$$delta_j^{[l]} = \left(\sum_{k=1}^{n_{l+1}} delta_k^{[l+1]} \cdot w_{j,k}^{[l+1]} \right) \cdot \frac{\partial \sigma(z_j^{[l]})}{\partial z_j^{[l]}}$$

Lan truyền ngược – Xét từng biến

- Ở tầng l ($1 \leq l < L$) (tóm tắt)

$$delta_j^{[L]} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$$

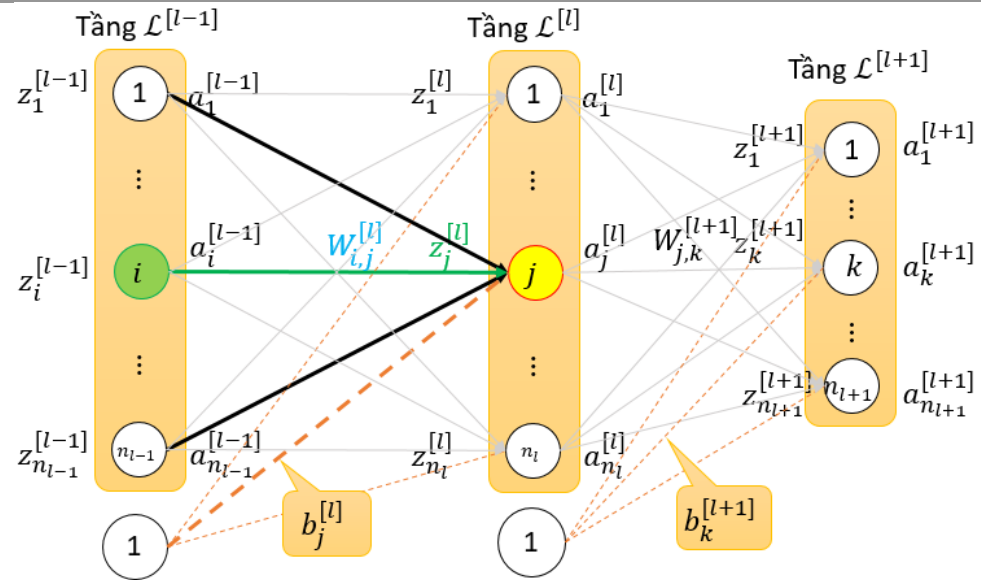
$$delta_j^{[L-1]} = \left(\sum_{k=1}^{n_L} delta_k^{[L]} \cdot W_{j,k}^{[L]} \right) \cdot \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$$

$$delta_j^{[l]} = \left(\sum_{k=1}^{n_{l+1}} delta_k^{[l+1]} \cdot W_{j,k}^{[l+1]} \right) \cdot \frac{\partial \sigma(z_j^{[l]})}{\partial z_j^{[l]}}$$

$$\frac{\partial Loss}{\partial W_{i,j}^{[l]}} = delta_j^{[l]} \cdot a_i^{[l-1]}$$

Lan truyền ngược – Xét từng biến

- Ở tầng l ($1 \leq l < L$)



- Đạo hàm trên bias

$$\frac{\partial Loss}{\partial b_j^{[l]}} = \frac{\partial Loss}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \frac{\partial Loss}{\partial z_j^{[l]}} \cdot 1 = \frac{\partial Loss}{\partial z_j^{[l]}} = \boxed{\text{delta}_j^{[l]}}$$

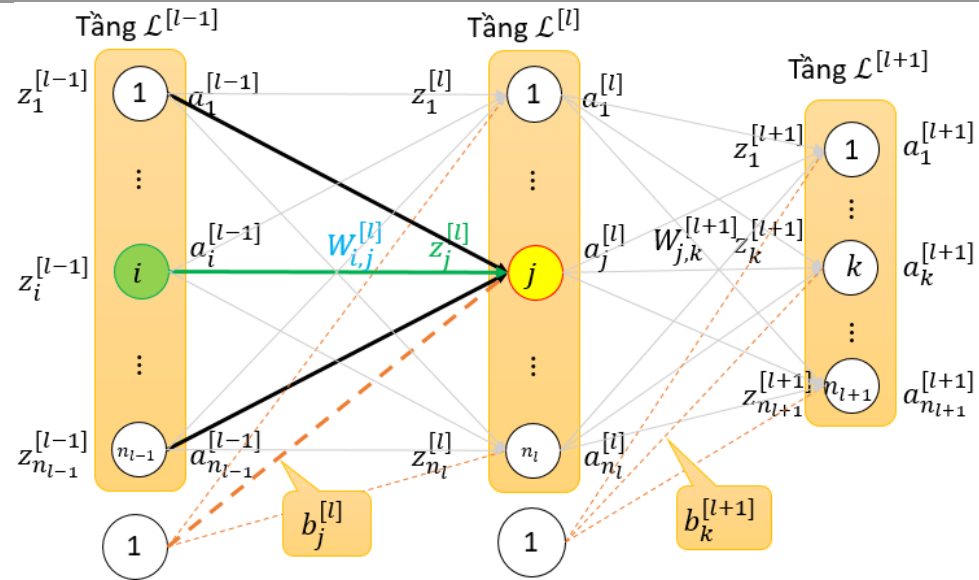
Lan truyền ngược – Xét từng biến

- **Tóm tắt**
 - **Ở tầng output**

$$\text{delta}_j^{[L]} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$$

$$\frac{\partial \text{Loss}}{\partial W_{ij}^{[L]}} = \text{delta}_j^{[L]} \cdot a_i^{[L-1]}$$

$$\frac{\partial \text{Loss}}{\partial b_j^{[L]}} = \text{delta}_j^{[L]}$$



Lan truyền ngược – Xét từng biến

- Tóm tắt**

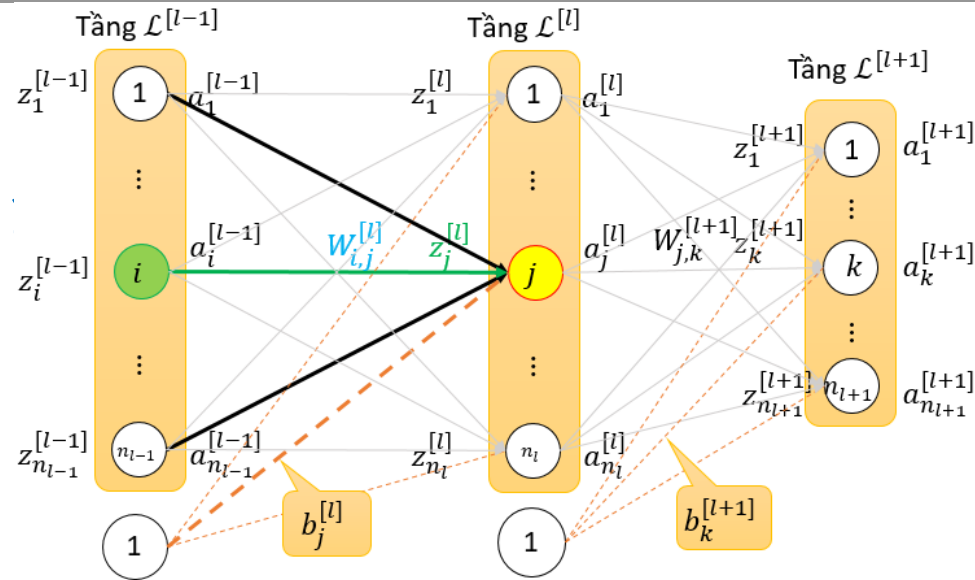
- Ở tầng $l = L - 1, L - 2, \dots$,

$$\text{delta}_j^{[L]} = \frac{\partial \sigma(z_j^{[L]})}{\partial z_j^{[L]}}$$

$$\text{delta}_j^{[l]} = \left(\sum_{k=1}^{n_{l+1}} \text{delta}_k^{[l+1]} \cdot W_{j,k}^{[l+1]} \right) \cdot \frac{\partial \sigma(z_j^{[l]})}{\partial z_j^{[l]}}$$

$$\frac{\partial \text{Loss}}{\partial W_{ij}^{[l]}} = \text{delta}_j^{[l]} \cdot a_i^{[l-1]}$$

$$\frac{\partial \text{Loss}}{\partial b_j^{[l]}} = \text{delta}_j^{[l]}$$



Lan truyền ngược – Dạng vector, matrix

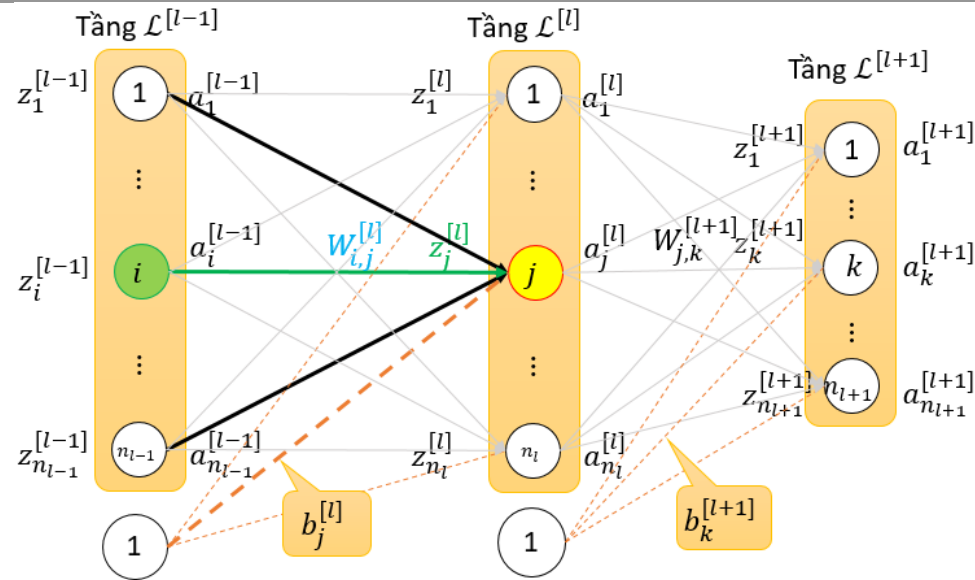
- **Tóm tắt**

- **Ở tầng output**

$$\text{delta}^{[L]} = \frac{\partial \sigma(z^{[L]})}{\partial z^{[L]}}$$

$$\frac{\partial \text{Loss}}{\partial W^{[L]}} = a^{[L-1]} \cdot \text{delta}^{[L]}$$

$$\frac{\partial \text{Loss}}{\partial b^L} = \text{delta}^{[L]}$$



Lan truyền ngược – **Dạng vector, matrix**

- Tóm tắt**

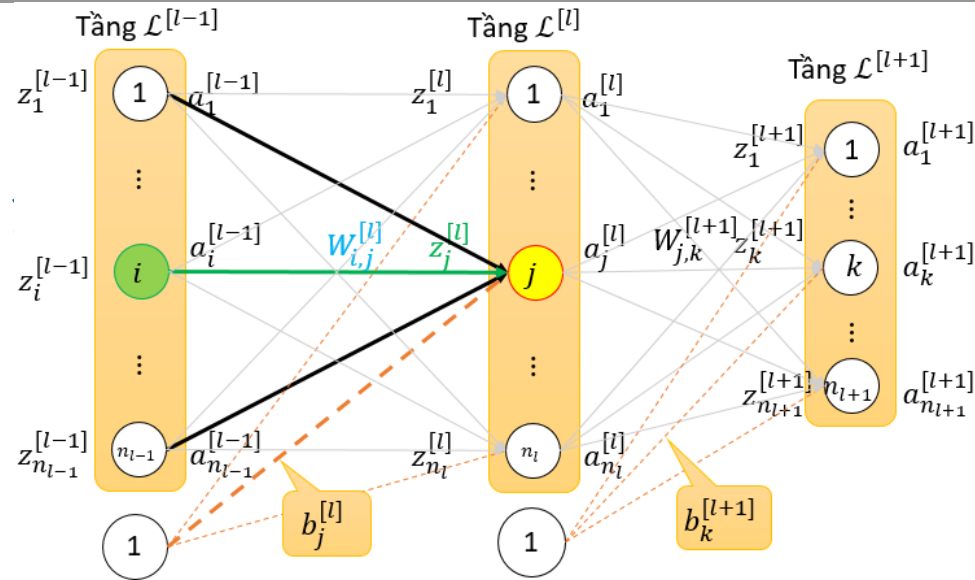
- Ở tầng $l = L - 1, L - 2, \dots$

$$\text{delta}^{[L]} = \frac{\partial \sigma(z^{[L]})}{\partial z^{[L]}}$$

$$\text{delta}^{[l]} = (\text{delta}^{[l+1]} \cdot W^{[l+1]}) \odot \frac{\partial \sigma(z^{[l]})}{\partial z^{[l]}}$$

$$\frac{\partial \text{Loss}}{\partial W^{[l]}} = a^{[l-1]} \cdot \text{delta}^{[l]}$$

$$\frac{\partial \text{Loss}}{\partial b^l} = \text{delta}^{[l]}$$



\odot là phép toán nhân element-wise, nghĩa là nhân từng phần tử của vector với một số

Thuật toán Lan truyền ngược

Thuật toán Lan truyền ngược

Input : Dữ liệu $x \in \mathbb{R}^{n_0}$ và $W = \{W^{[1]}, W^{[2]}, \dots, W^{[L]}\}$, $b = \{b^{[1]}, b^{[2]}, \dots, b^{[L]}\}$

Output: Tập $\left\{\frac{\partial \text{Loss}}{\partial W^{[1]}}, \frac{\partial \text{Loss}}{\partial W^{[2]}}, \dots, \frac{\partial \text{Loss}}{\partial W^{[L]}}\right\}$ và $\left\{\frac{\partial \text{Loss}}{\partial b^{[1]}}, \frac{\partial \text{Loss}}{\partial b^{[2]}}, \dots, \frac{\partial \text{Loss}}{\partial b^{[L]}}\right\}$

$a^{[0]} \leftarrow x \in \mathbb{R}^{n_0}$

for $l = 1$ **to** L **do**

$z^{[l]} \leftarrow a^{[l-1]} \cdot W^{[l]} + b^{[l]}$

$a^{[l]} \leftarrow \sigma(z^{[l]})$

end

$\delta^{[L]} \leftarrow \sigma'(z^{[L]})$

$\frac{\partial \text{Loss}}{\partial W^{[L]}} \leftarrow a^{[L-1]} \cdot \delta^{[L]}$

$\frac{\partial \text{Loss}}{\partial b^{[L]}} \leftarrow \delta^{[L]}$

for $l = L - 1$ **to** 1 **do**

$\delta^{[l]} \leftarrow (\delta^{[l+1]} \cdot W^{[l+1]}) \odot \sigma'(z^{[l]})$

$\frac{\partial \text{Loss}}{\partial W^{[l]}} \leftarrow a^{[l-1]} \cdot \delta^{[l]}$

$\frac{\partial \text{Loss}}{\partial b^{[l]}} \leftarrow \delta^{[l]}$

end

return $\left\{\frac{\partial \text{Loss}}{\partial W^{[1]}}, \frac{\partial \text{Loss}}{\partial W^{[2]}}, \dots, \frac{\partial \text{Loss}}{\partial W^{[L]}}\right\}$ và $\left\{\frac{\partial \text{Loss}}{\partial b^{[1]}}, \frac{\partial \text{Loss}}{\partial b^{[2]}}, \dots, \frac{\partial \text{Loss}}{\partial b^{[L]}}\right\}$

Thuật toán Cập nhật tham số

Thuật toán Cập nhật tham số

Input : Dữ liệu $\left\{ \frac{\partial Loss}{\partial W^{[1]}}, \frac{\partial Loss}{\partial W^{[2]}}, \dots, \frac{\partial Loss}{\partial W^{[L]}} \right\}$ và $\left\{ \frac{\partial Loss}{\partial b^{[1]}}, \frac{\partial Loss}{\partial b^{[2]}}, \dots, \frac{\partial Loss}{\partial b^{[L]}} \right\}$ và hệ số học α

Output: $W = \{W^{[1]}, W^{[2]}, \dots, W^{[L]}\}$ và $b = \{b^{[1]}, b^{[2]}, \dots, b^{[L]}\}$

for $l = L$ **to** 1 **do**

$$W^{[l]} \leftarrow W^{[l]} - \alpha \cdot \frac{\partial Loss}{\partial W^{[l]}}$$

$$b^{[l]} \leftarrow b^{[l]} - \alpha \cdot \frac{\partial Loss}{\partial b^{[l]}}$$

end

return $W = \{W^{[1]}, W^{[2]}, \dots, W^{[L]}\}$ và $b = \{b^{[1]}, b^{[2]}, \dots, b^{[L]}\}$

Cài đặt MLP

- Thực hành: Xây dựng MLP bằng python
- Hàm loss là hàm Mean Square Error (MSE)

$$l^{(i)}(W, b) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

- Trong đó:
 - y : giá trị true ground (hằng số)
 - \hat{y} : hàm số theo W, b

Cài đặt MLP

```
import numpy as np

class MLP:
    def __init__(self, layers, alpha=0.1):
        # Dùng bias trick kết hợp b vào W
        self.W = []
        self.layers = layers
        self.alpha = alpha

        # Tạo các layer có giá trị ngẫu nhiên
        for i in np.arange(0, len(layers) - 2):
            w = np.random.randn(layers[i] + 1, layers[i + 1] + 1)
            self.W.append(w / np.sqrt(layers[i]))

        # Output Layer
        w = np.random.randn(layers[-2] + 1, layers[-1])
        self.W.append(w / np.sqrt(layers[-2]))
```

Cài đặt MLP

Hàm activation: sigmoid

```
def sigmoid(self, x):  
    return 1.0 / (1 + np.exp(-x))
```

Đạo hàm của sigmoid

```
def sigmoid_deriv(self, x):  
    return x * (1 - x)
```

Training model

```
def fit(self, X, y, epochs=1000, verbose=100):  
    X = np.c_[X, np.ones((X.shape[0]))]  
  
    for epoch in np.arange(0, epochs):  
        for (x, target) in zip(X, y):  
            self.fit_partial(x, target)  
  
        if epoch == 0 or (epoch + 1) % verbose == 0:  
            loss = self.calculate_loss(X, y)  
            print("epoch={}, loss={:.7f}".format(epoch + 1, loss))
```

Cài đặt MLP

```
def fit_partial(self, x, y):  
    # Tính giá trị output từng neurons  
    a = [np.atleast_2d(x)]  
    for layer in np.arange(0, len(self.W)):  
        net = a[layer].dot(self.W[layer])  
        out = self.sigmoid(net)  
        a.append(out)  
  
    error = a[-1] - y  
    D = [error * self.sigmoid_deriv(a[-1])]  
  
    for layer in np.arange(len(a) - 2, 0, -1):  
        delta = D[-1].dot(self.W[layer].T)  
        delta = delta * self.sigmoid_deriv(a[layer])  
        D.append(delta)  
  
    D = D[::-1]  
    for layer in np.arange(0, len(self.W)):  
        self.W[layer] += -self.alpha * a[layer].T.dot(D[layer])
```

Cài đặt MLP

Tính \hat{y} của X

```
def predict(self, X, addBias=True):
    a = np.atleast_2d(X)
    if addBias:
        a = np.c_[a, np.ones((a.shape[0]))]

    for layer in np.arange(0, len(self.W)):
        z = np.dot(a, self.W[layer])
        a = self.sigmoid(z)

    return a
```

Tính Lỗi

```
def calculate_loss(self, X, y):
    y = np.atleast_2d(y)
    y_hat = self.predict(X, addBias=False)
    loss = 0.5 * np.sum((y_hat - y) ** 2)

    return loss
```

Cài đặt MLP

- Phân lớp XOR

```
import numpy as np

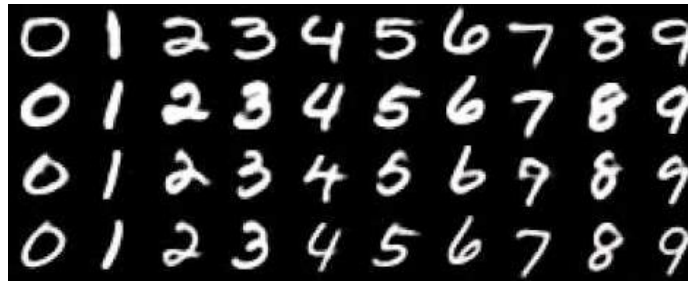
# Dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Training
model = MLP([2, 2, 1], alpha=0.1)
model.fit(X, y, epochs=5000)

# So sánh Kết quả
for (x, target) in zip(X, y):
    pred = model.predict(x)[0][0]
    step = 1 if pred > 0.5 else 0
    print("data={}, ground-truth={}, pred={:.4f}, step={}".format(x,
target[0], pred, step))
```

Cài đặt MLP

- Nhận dạng các chữ số viết tay trên mini **MNIST**
 - **mini MNIST** dataset trong sklearn là tập con của tập MNIST
 - 1797 examples (dữ liệu gốc: 70000 example)
 - 8×8 ảnh mức xám (ảnh gốc: 28×28)



```
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from sklearn import datasets
```

Cài đặt MLP

Load dữ liệu

```
print("Loading MNIST dataset...")  
digits = datasets.load_digits()
```

Chuẩn hóa dữ liệu

```
data = digits.data.astype("float")  
data = (data - data.min()) / (data.max() - data.min())  
print("Số data points: {}, Số features: {}".format(data.shape[0],  
                                                    data.shape[1]))
```

Chia tập dữ liệu

```
(trainX, testX, trainY, testY) = train_test_split(data, digits.target,  
                                                    test_size=0.25)
```

```
trainY = LabelBinarizer().fit_transform(trainY)  
testY = LabelBinarizer().fit_transform(testY)
```


Cài đặt MLP

Training

```
print("Training network...")  
model = MLP([trainX.shape[1], 32, 16, 10])  
model.fit(trainX, trainY, epochs=1000)
```

Test

```
print("Evaluating network...")  
predictions = model.predict(testX)  
predictions = predictions.argmax(axis=1)  
print(classification_report(testY.argmax(axis=1), predictions))
```

Cài đặt MLP - Sử dụng thư viện Keras

- Cơ sở dữ liệu gốc MNIST
 - 70.000 data points (images)
 - 28×28 ảnh mức xám

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

from tensorflow.keras import utils
from tensorflow.keras.datasets import mnist

from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import numpy as np
```

Cài đặt MLP - Sử dụng thư viện Keras

Load dữ liệu

```
print("Loading MNIST dataset...")
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
```

Chuẩn hóa dữ liệu

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

classNames = [0,1,2,3,4,5,6,7,8,9]

print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

Cài đặt MLP - Sử dụng thư viện Keras

```
y_train = utils.to_categorical(y_train, 10)  
y_test  = utils.to_categorical(y_test, 10)
```

```
model = Sequential()  
model.add(Dense(256, input_shape=(784,), activation="sigmoid"))  
model.add(Dense(128, activation="sigmoid"))  
model.add(Dense(10, activation="softmax"))
```

Cài đặt MLP - Sử dụng thư viện Keras

```
print("Training network...")
sgd = SGD(0.01)

model.compile(loss="categorical_crossentropy", optimizer=sgd,
              metrics=["accuracy"])

H = model.fit(X_train, y_train, validation_data=(X_test, y_test),
             epochs=100,
             batch_size=128)
```

```
print("Evaluating network...")
predictions = model.predict(X_test, batch_size=128)

y_test_argmax = y_test.argmax(axis=1)
predict_argmax = predictions.argmax(axis=1)
print(classification_report(y_test_argmax, predict_argmax,
                          target_names=[str(x) for x in classNames]))
```

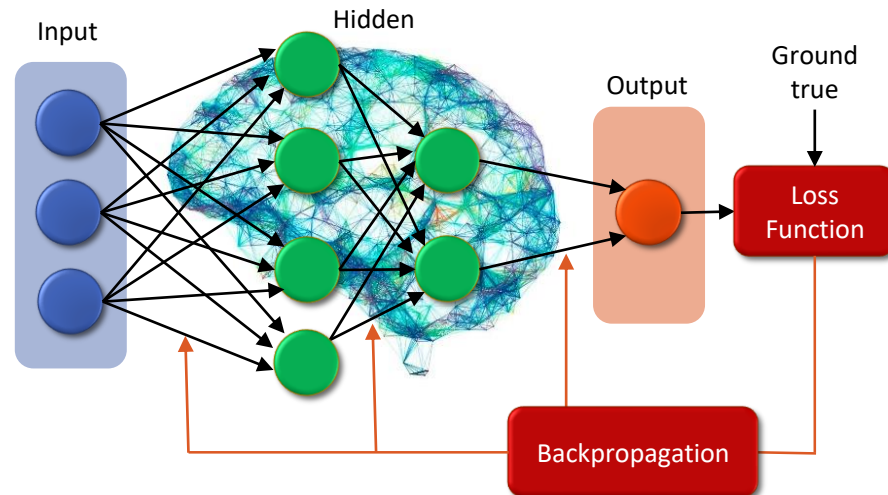
4 thành phần của Neural networks

1
Data

3
Loss function

2
Network architecture

4
Optimizer



Tiếp theo ...

- Tìm hiểu kỹ thuật toán Lan truyền ngược
- Cho mạng có số neuron ở các tầng là [2-3-2], ..., chạy từng bước
 - Cho các số cụ thể vào mạng: input, W (không cần bias)
 - Lan truyền tiến
 - Lan truyền ngược 1 lần để cập nhật các W
- Sử dụng MLP trên dataset
 - AND, OR, XOR
 - Animals
 - MNIST