

## Computer Vision

# THỊ GIÁC MÁY TÍNH

**ThS. Huỳnh Minh Vũ**

**Khoa Kỹ thuật cơ khí**

**Đại học Kỹ thuật – Công nghệ Cần Thơ**

**Email: [hmvu@ctuet.edu.vn](mailto:hmvu@ctuet.edu.vn)**



# Chương 6: Phân lớp ảnh (Image Classification)

---

**6.1 K-Nearest Neighbors (KNN)**

**6.2 Support Vector Machine (SVM)**

**6.3 Artificial Neural Networks (ANN)**

**6.4 Convolution Neural Network (CNN\*)**

# Tài liệu tham khảo

---

Các trang website:

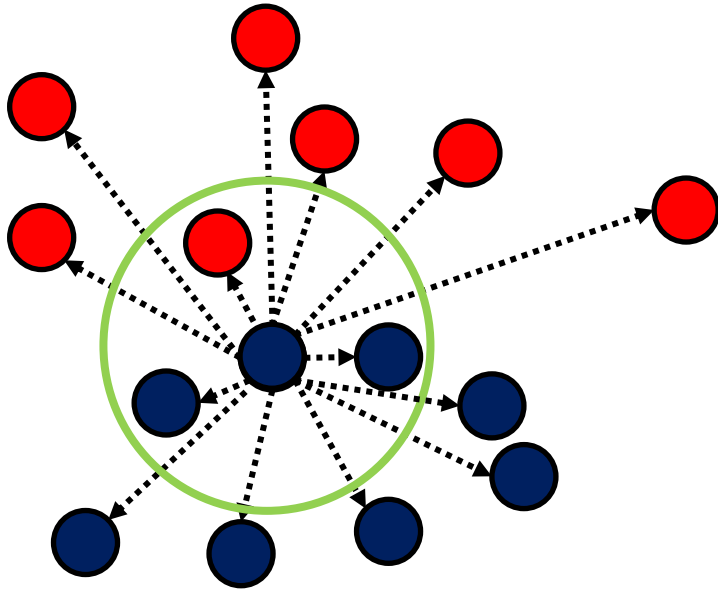
<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

## 6.1 K-Nearest Neighbors (KNN)

“Hãy nói cho tôi biết bạn của anh là ai, tôi sẽ nói cho anh biết anh là người như thế nào”

☞ Đối tượng X sẽ được phân lớp, dựa vào K đối tượng gần X nhất.



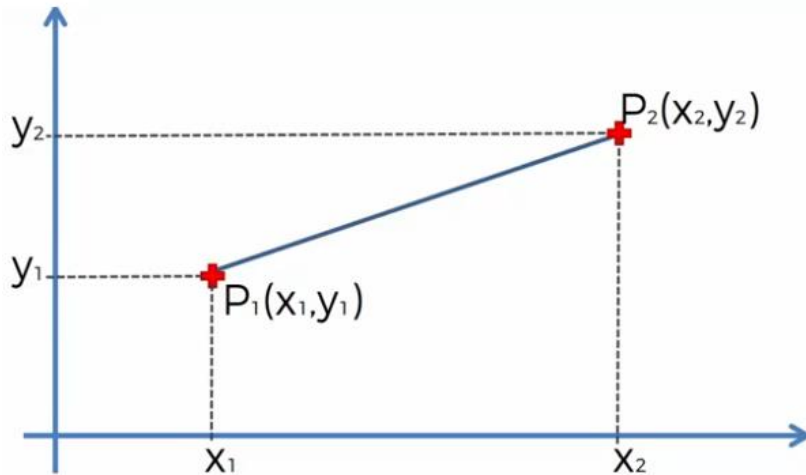
- STEP 1: **Tính khoảng cách** từ X đến tất cả các đối tượng trong dữ liệu huấn luyện.
- STEP 2: **Chọn ra K đối tượng** gần X nhất (khoảng cách nhỏ nhất).
- Phân lớp X vào lớp xuất hiện phổ biến nhất trong K đối tượng.

# 6.1 K-Nearest Neighbors (KNN)

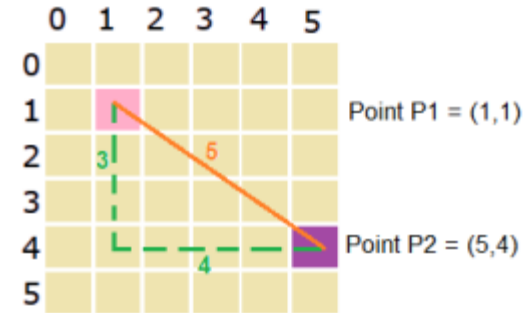
Công thức tính khoảng cách:

Euclidean  $\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$

Manhattan  $\sum_{i=1}^k |x_i - y_i|$

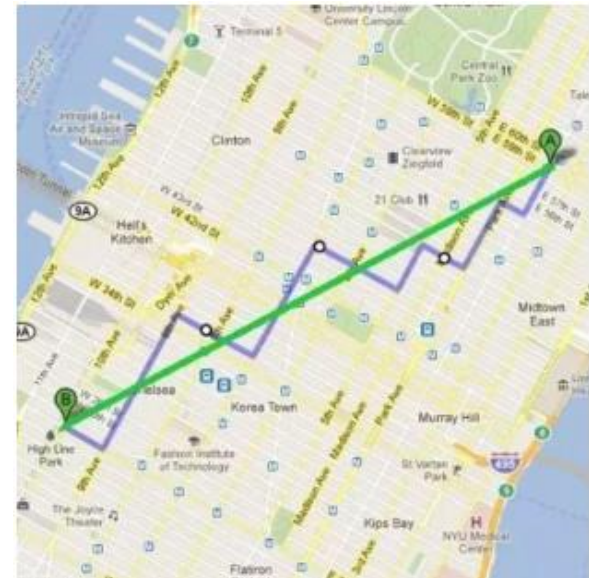


$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



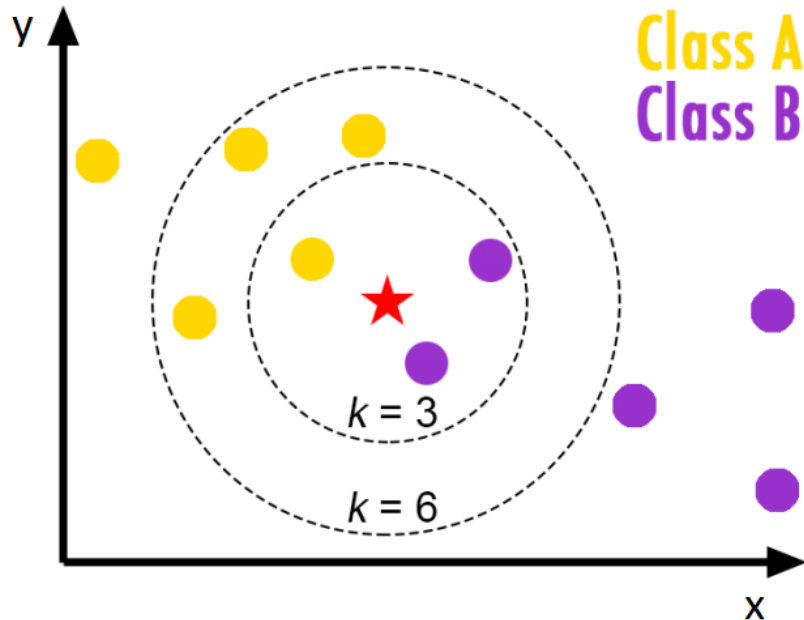
$$\text{Euclidean distance} = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$\text{Manhattan distance} = |5-1| + |4-1| = 7$$



# 6.1 K-Nearest Neighbors (KNN)

Chọn K:



$k = 3$  ★  $\Rightarrow$  ●  
 $k = 6$  ★  $\Rightarrow$  ●

- Nếu K quá nhỏ, nó rất nhạy cảm với các điểm nhiễu.
- Nếu K lớn, thuật toán hoạt động tốt nhưng K quá lớn có thể bao gồm các điểm từ các lớp khác.
- Thông thường chọn  $K < \sqrt{n}$ , với n là tổng số lượng điểm dữ liệu.
- Chọn K một giá trị lẻ.

# 6.1 K-Nearest Neighbors (KNN)

## Ví dụ 1:

X	y
0	0
1	0
2	1
3	1
1.1	?

```
x = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]

from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)

print(neigh.predict([[1.1]]))

print(neigh.predict_proba([[1.1]]))
```

→ [0]  
[[0.66666667 0.33333333]]

<code>fit(X, y)</code>	Fit the model using X as training data and y as target values
<code>kneighbors([X, n_neighbors, return_distance])</code>	Finds the K-neighbors of a point.
<code>predict(X)</code>	Predict the class labels for the provided data.
<code>predict_proba(X)</code>	Return probability estimates for the test data X.

# 6.1 K-Nearest Neighbors (KNN)

## Ví dụ 2:

X	y
[0, 0, 1]	0
[1, 0, 1]	1
[1, 1, 0]	1
[0, 1, 1]	0
[1, 1, 1]	?

```
import numpy as np
X = np.array([[0, 0, 1], [1, 0, 1], [1, 1, 0], [0, 1, 1]])
y = np.array([[0, 1, 1, 0]]).T
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
print(X)
print(y)
print(neigh.predict(np.array([[1, 1, 1]])))
print(neigh.predict_proba(np.array([[1, 1, 1]])))
```

```
[[0 0 1]
 [1 0 1]
 [1 1 0]
 [0 1 1]]
[[0]
 [1]
 [1]
 [0]]
[1]
[[0.33333333 0.66666667]]
```



## 6.1 K-Nearest Neighbors (KNN)

### Ví dụ 3:

*Iris versicolor*



*Iris setosa*



*Iris virginica*



<https://archive.ics.uci.edu/ml/datasets/iris>

# 6.1 K-Nearest Neighbors (KNN)

## Ví dụ 3:

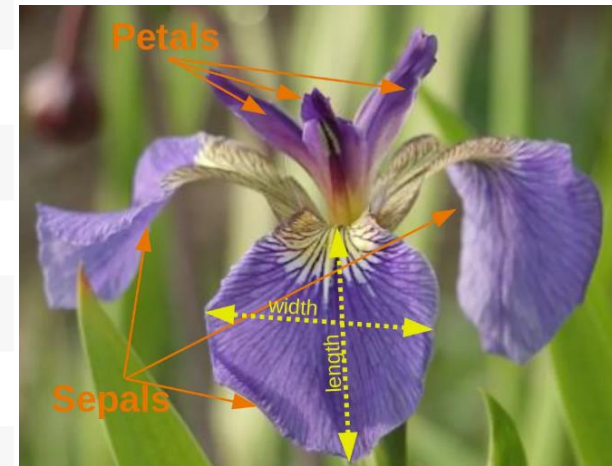
```
from sklearn import datasets  
iris = datasets.load_iris()
```



Iris.csv

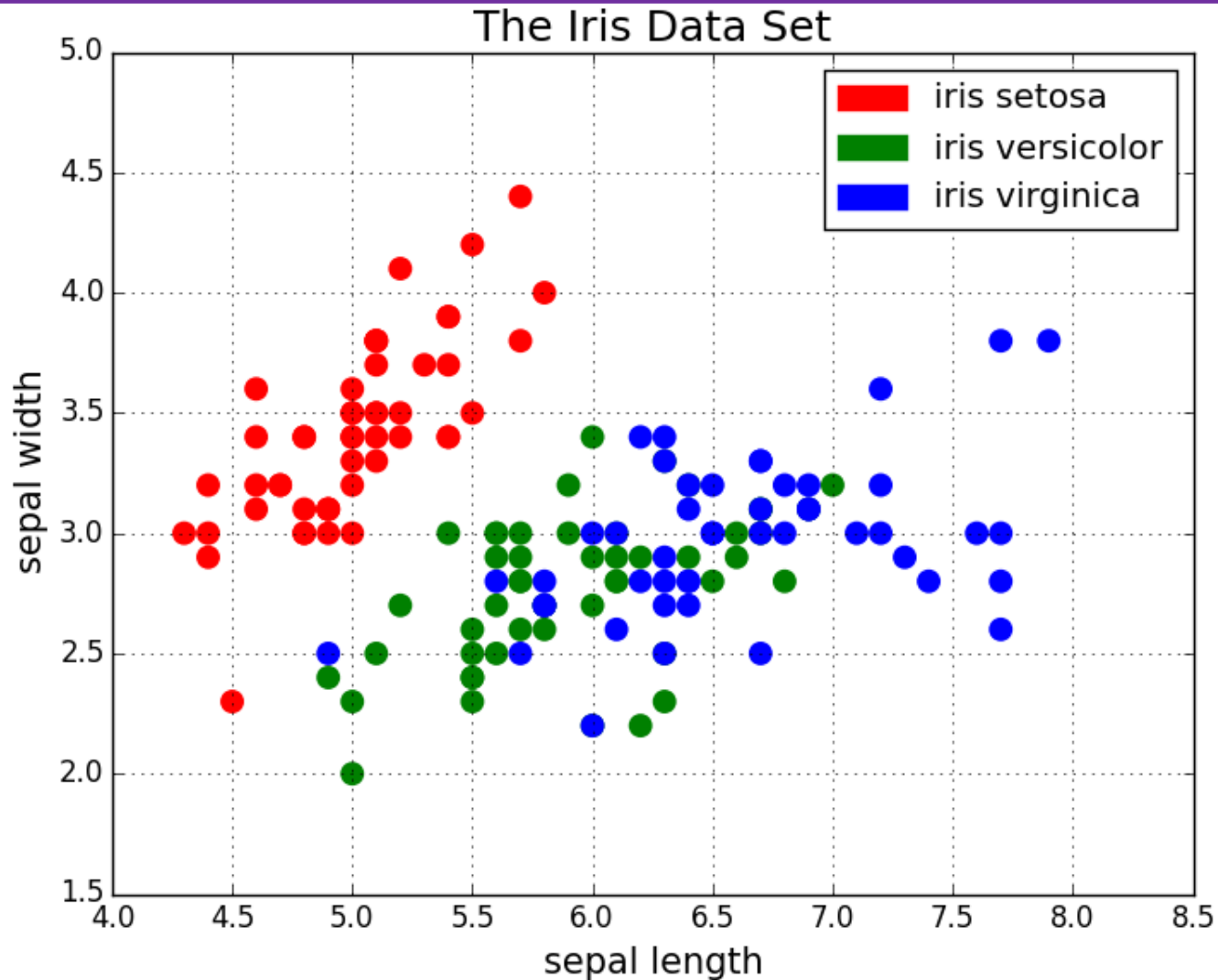
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns



# 6.1 K-Nearest Neighbors (KNN)

Ví dụ 3:



<http://www.pybloggers.com/2015/09/my-first-time-using-matplotlib/>

# 6.1 K-Nearest Neighbors (KNN)

## Ví dụ 3:

```
#Đọc dữ liệu
from sklearn import datasets
import pandas as pd
#iris = datasets.load_iris()
#print(iris.data)
#print(iris.target)
iris = pd.read_csv("/content/gdrive/MyDrive/Google Colab Code/KNN/iris.csv")
X = iris.iloc[:, 0:4]
y = iris.iloc[:, -1]
print(X)
print(y)
```

```
#Xây dựng mô hình KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
```

```
↳ KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                        metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                        weights='uniform')
```

## 6.1 K-Nearest Neighbors (KNN)

### Ví dụ 3:

```
#Dự đoán kết quả  
knn.predict([[5, 3, 1, 2]])
```

```
array(['setosa'], dtype=object)
```

```
#Kiểm tra dự đoán  
import numpy as np  
test = np.array([[5, 3, 1, 2]])  
print(test)  
knn.kneighbors(test, 3)
```

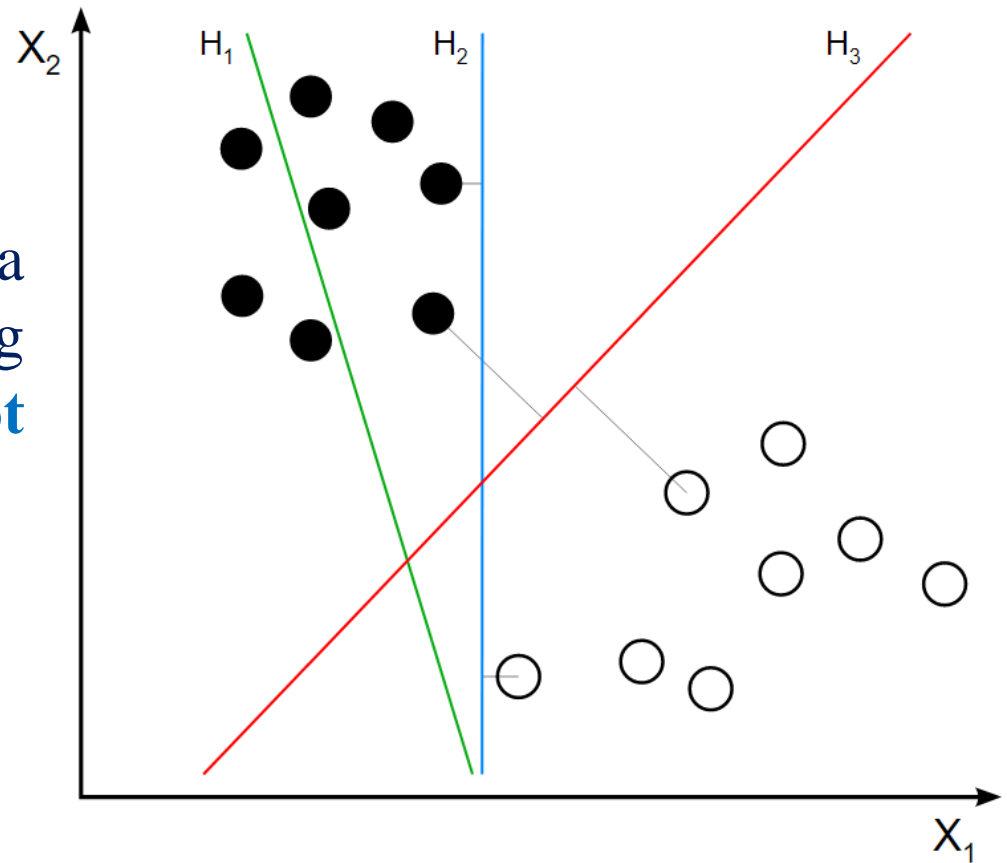
```
[[5 3 1 2]]  
(array([[1.60312195, 1.68522995, 1.75499288]]), array([[43, 23, 26]]))
```

```
iris.iloc[[ 43, 23, 26]]
```

```
sepal_length  sepal_width  petal_length  petal_width  species  
43           5.0         3.5           1.6           0.6    setosa  
23           5.1         3.3           1.7           0.5    setosa  
26           5.0         3.4           1.6           0.4    setosa
```

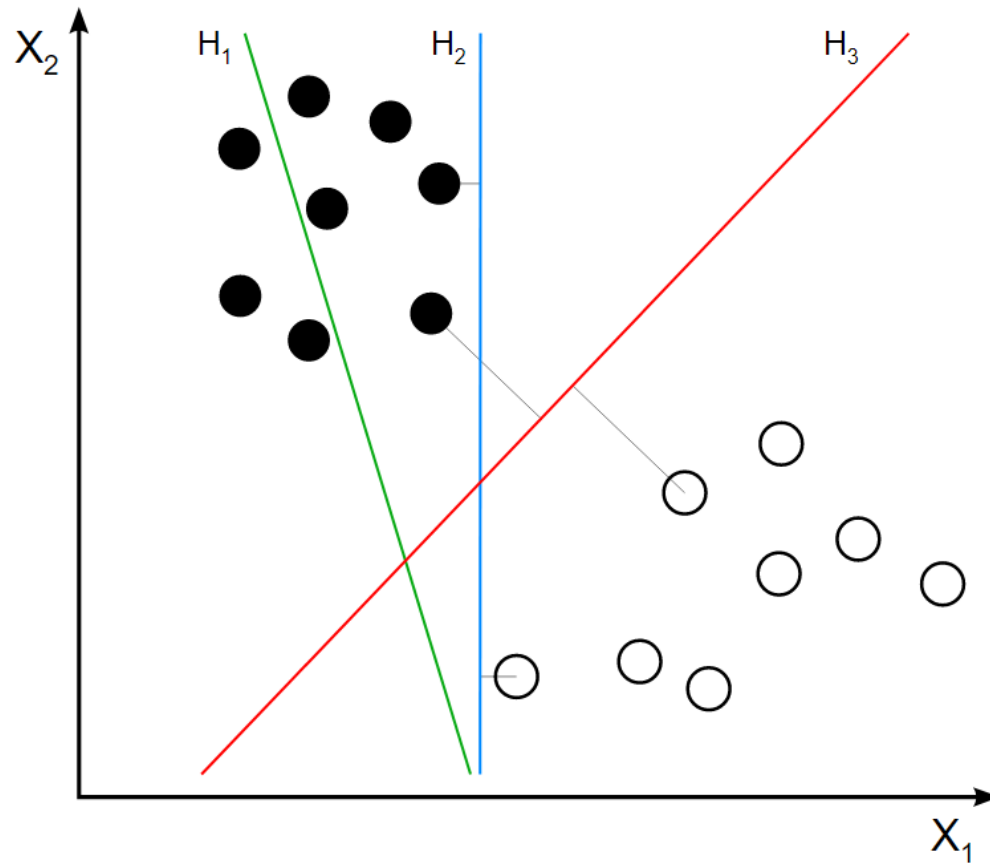
## 6.2 Support Vector Machine (SVM)

Mục tiêu của SVM là tìm ra một **siêu phẳng** trong không gian  $n$  chiều, **phân chia tốt nhất** tập dữ liệu.



## 6.2 Support Vector Machine (SVM)

- **Siêu phẳng:** trong không gian 2 chiều là đường thẳng, trong không gian 3 chiều là mặt phẳng, trong không gian  $n$  chiều là  $\langle a, x \rangle + b = 1$ , hoặc  $\langle \mathbf{w}, \mathbf{x} \rangle + \mathbf{b} = 1$ , với  $a$  (hoặc  $w$ ) là không gian  $n$  chiều,  $b$  thuộc  $\mathbb{R}$ .
- **Phân chia tốt nhất:** khoảng cách từ các đối tượng gần nhất đến các siêu phẳng là cực đại (còn gọi là khoảng cách lề cực đại)



## 6.2 Support Vector Machine (SVM)

**Bài toán SVM:** Tìm  $w \in R^n$  và  $b \in R$  sao cho lề cực đại

$$\begin{cases} \mathbf{x}_2 = \mathbf{x}_1 + \lambda \mathbf{w} \\ \mathbf{w}^T \mathbf{x}_2 - b = 1 \end{cases}$$

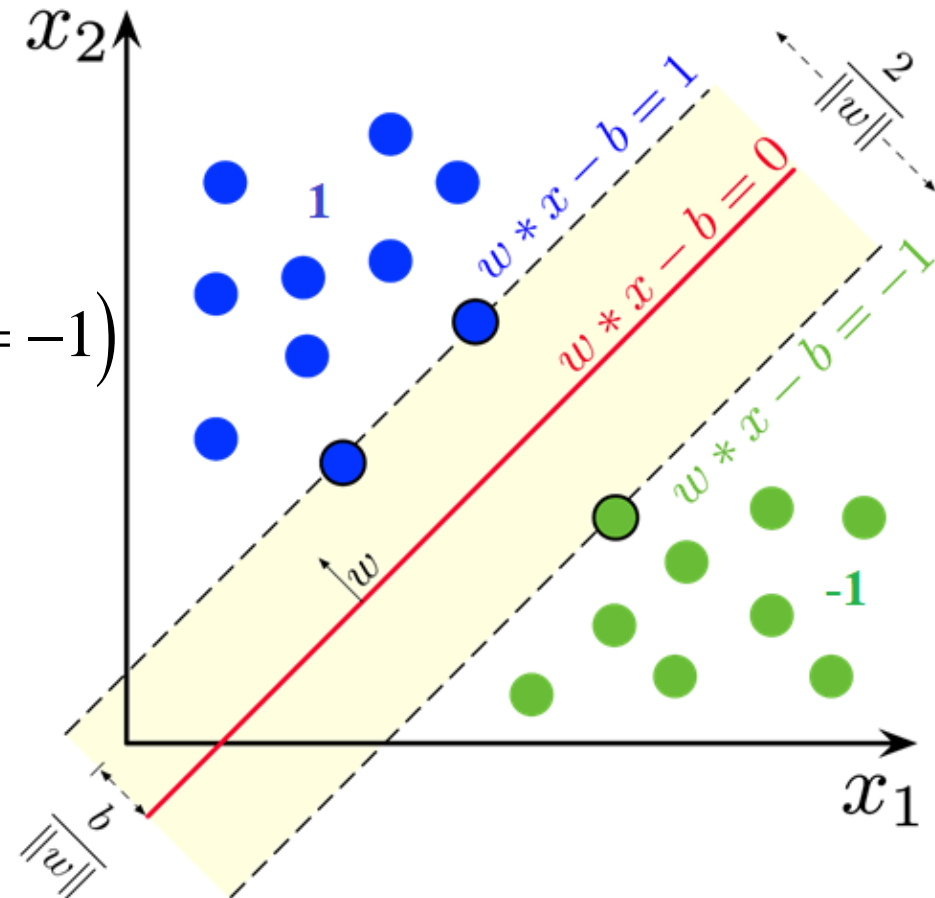
$$\rightarrow \mathbf{w}^T (\mathbf{x}_1 + \lambda \mathbf{w}) - b = 1$$

$$\rightarrow \mathbf{w}^T \mathbf{x}_1 - b + \lambda \mathbf{w}^T \mathbf{w} = 1, (\mathbf{w}^T \mathbf{x}_1 - b = -1)$$

$$\rightarrow -1 + \lambda \mathbf{w}^T \mathbf{w} = 1$$

$$\rightarrow \lambda \mathbf{w}^T \mathbf{w} = 2$$

$$\rightarrow \lambda = \frac{2}{\mathbf{w}^T \mathbf{w}} = \frac{2}{\|\mathbf{w}\|^2}$$





## 6.2 Support Vector Machine (SVM)

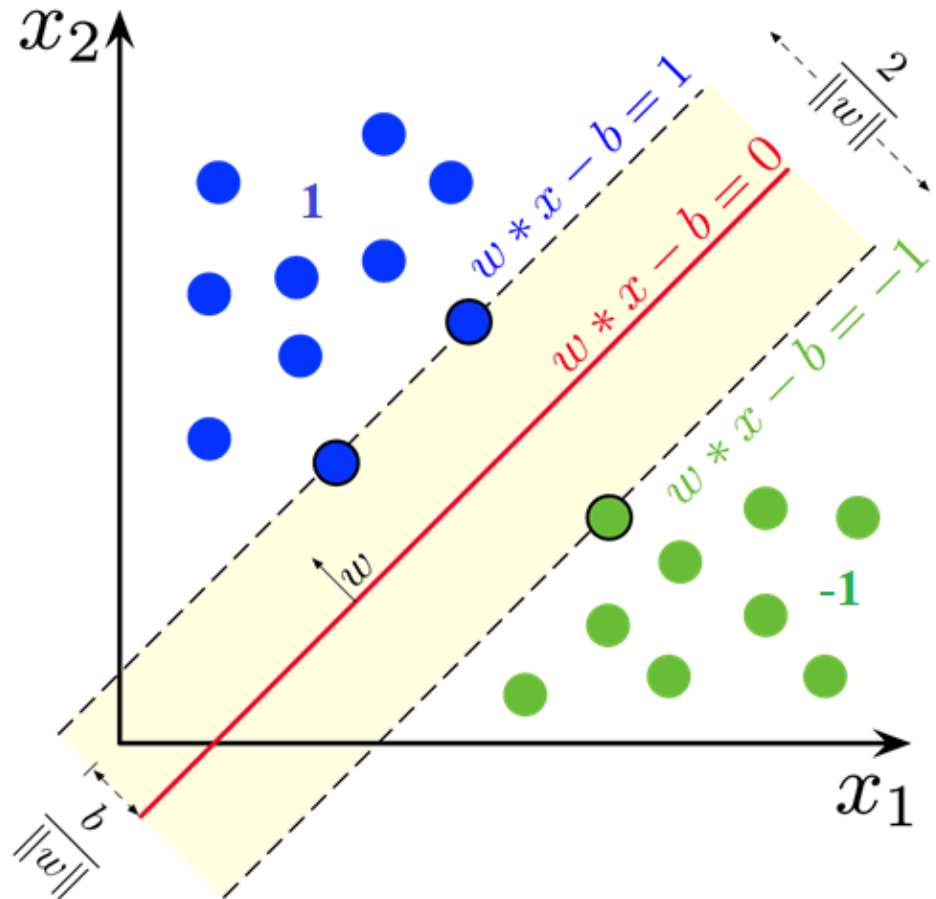
**Bài toán SVM:** Tìm  $w \in R^n$  và  $b \in R$  sao cho lề cực đại

$$\max_{(w,b)} \left\{ \frac{1}{\|w\|^2} \right\}$$

$$\min_{(w,b)} \left\{ \frac{1}{2} \|w\|^2 \right\}$$

$$y_i = (w x_i + b) \geq 1$$

$$\|w\| = \sqrt{\sum_{i=1}^d w_i^2}$$



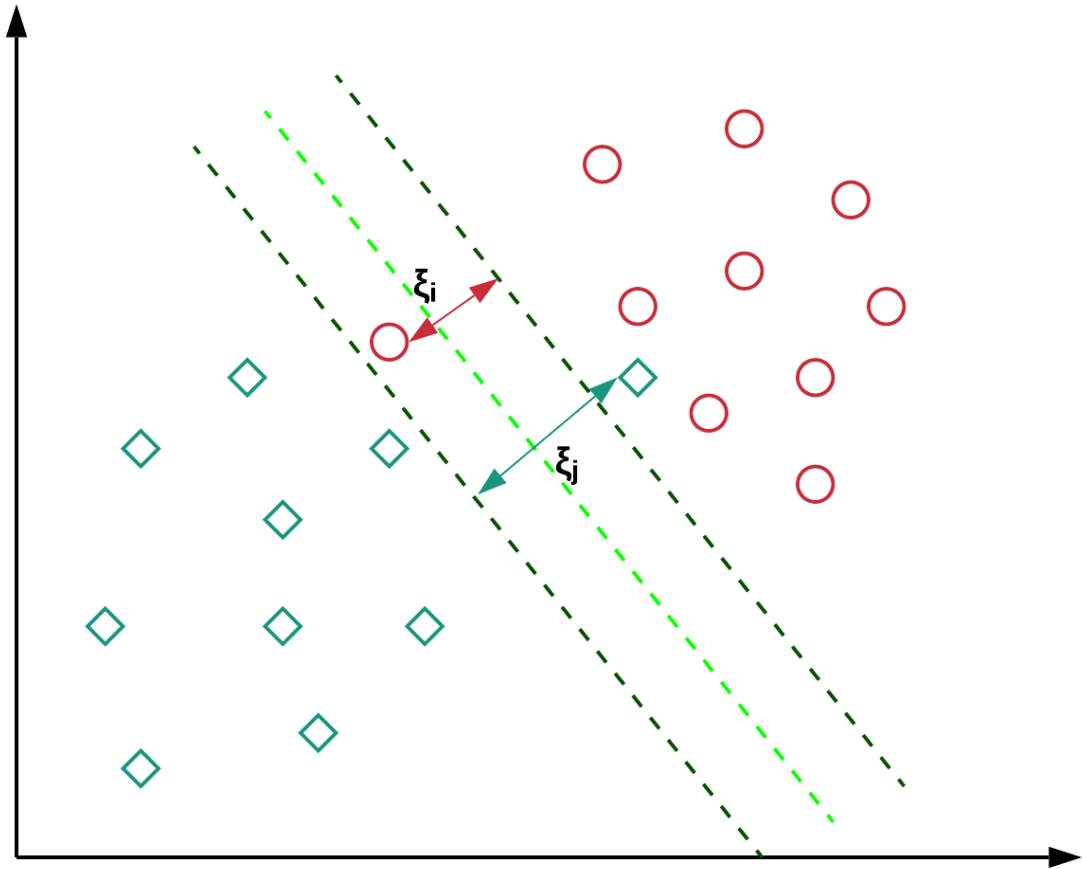
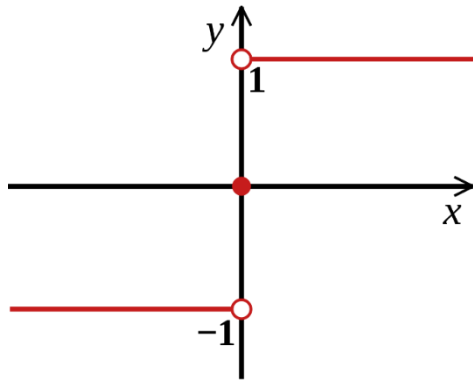
## 6.2 Support Vector Machine (SVM)

### Bài toán SVM lề mềm

$$\min_{(\mathbf{w}, b, \xi)} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \right\}$$

$$\begin{cases} y_i = (\mathbf{w} \mathbf{x}_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases}$$

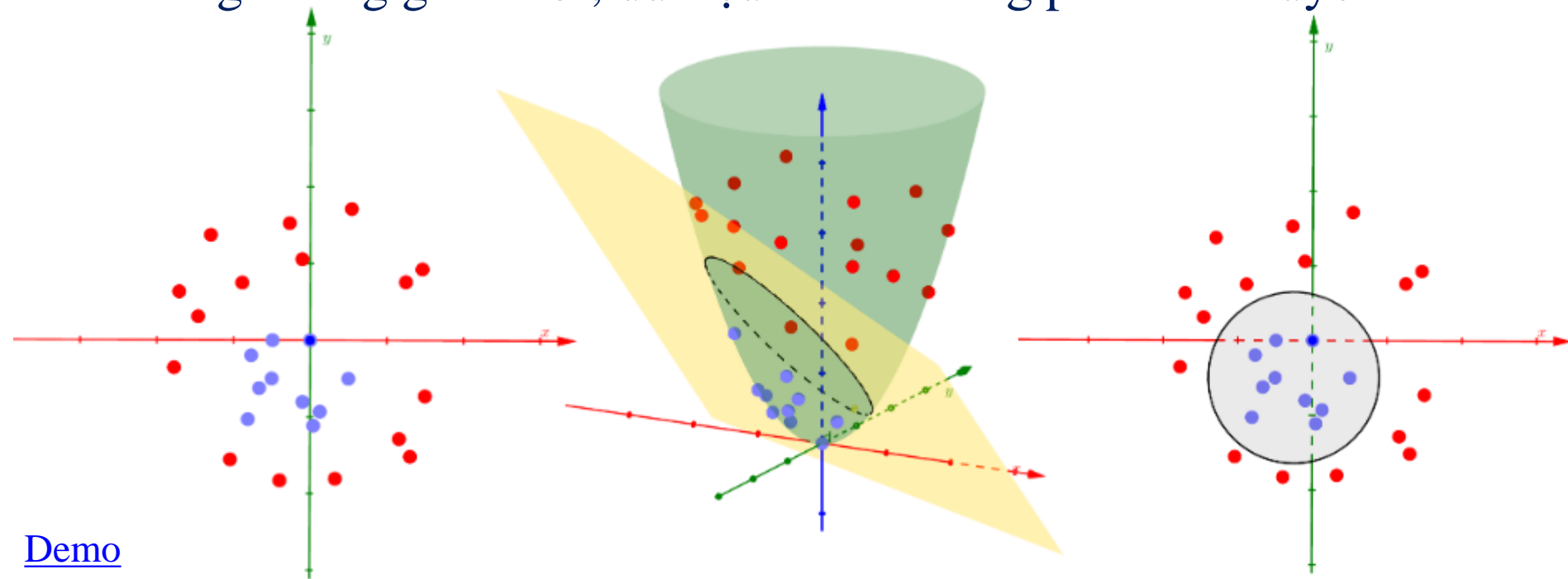
$$G(X) = \text{sign}(\langle w, X \rangle + b)$$



## 6.2 Support Vector Machine (SVM)

### Kernel Function

- Dùng để phân tách dữ liệu phi tuyến.
- Dùng một ánh xạ  $\varphi$  để ánh xạ dữ liệu sang không gian mới, có chiều lớn hơn.
- Trong không gian mới, dữ liệu có khả năng phân tách tuyến tính.



[Demo](#)

## 6.2 Support Vector Machine (SVM)

### Kernel Function

$$\begin{aligned} \min_{(\mathbf{w}, b, \xi)} & \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \right\} \\ & \begin{cases} y_i = (\mathbf{w} \mathbf{x}_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \\ G(X) &= \text{sign}(\langle w, X \rangle + b) \end{aligned} \quad \begin{aligned} \max_{\alpha} & \left\{ \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^m \alpha_i \right\} \\ & \begin{cases} \sum_{i=1}^m \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \end{cases} \\ G(X) &= \text{sign} \left( \sum_{i=1}^m \alpha_i y_i K(x_i \cdot x) - b \right) \end{aligned}$$

## 6.2 Support Vector Machine (SVM)

Ví dụ 1:

X	y
[0, 0]	0
[1, 1]	1
[2, 2]	?

```
from sklearn import svm  
X = [[0, 0], [1, 1]]  
y = [0, 1]  
clf = svm.SVC()  
clf.fit(X, y)
```

```
➤ SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,  
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',  
    max_iter=-1, probability=False, random_state=None, shrinking=True,  
    tol=0.001, verbose=False)
```

```
➤ clf.predict([[2., 2.]])
```

```
➤ array([1])
```

## 6.2 Support Vector Machine (SVM)

### Ví dụ 2: Phân loại số viết tay, dựa trên MNIST dataset

<https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>



```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.svm import SVC
import cv2
from google.colab.patches import cv2_imshow
```

## 6.2 Support Vector Machine (SVM)

### Ví dụ 2:

```
▶ digits = datasets.load_digits()  
n = len(digits.images)  
print(n)  
print(digits.images.shape)  
  
X = digits.images.reshape(n, -1)  
y = digits.target  
print(X.shape)
```

```
↳ 1797  
(1797, 8, 8)  
(1797, 64)
```


```
▶ model = SVC(degree=3, gamma=0.001, kernel='linear')  
model.fit(X, y)
```

```
↳ SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,  
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='linear',  
    max_iter=-1, probability=False, random_state=None, shrinking=True,  
    tol=0.001, verbose=False)
```

## 6.2 Support Vector Machine (SVM)

### Ví dụ 2:

```
img = cv2.imread("/content/gdrive/MyDrive/Google Colab Code/SVM/MNIST/1.jpg", 0)  
cv2.imshow(img)  
cv2.waitKey()
```

↳   
-1

```
img_new = cv2.resize(img, (8, 8))  
print(img_new.shape)  
img_test = img_new.reshape(1, img_new.shape[0]*img_new.shape[1])  
print(img_test.shape)
```

↳ (8, 8)  
(1, 64)

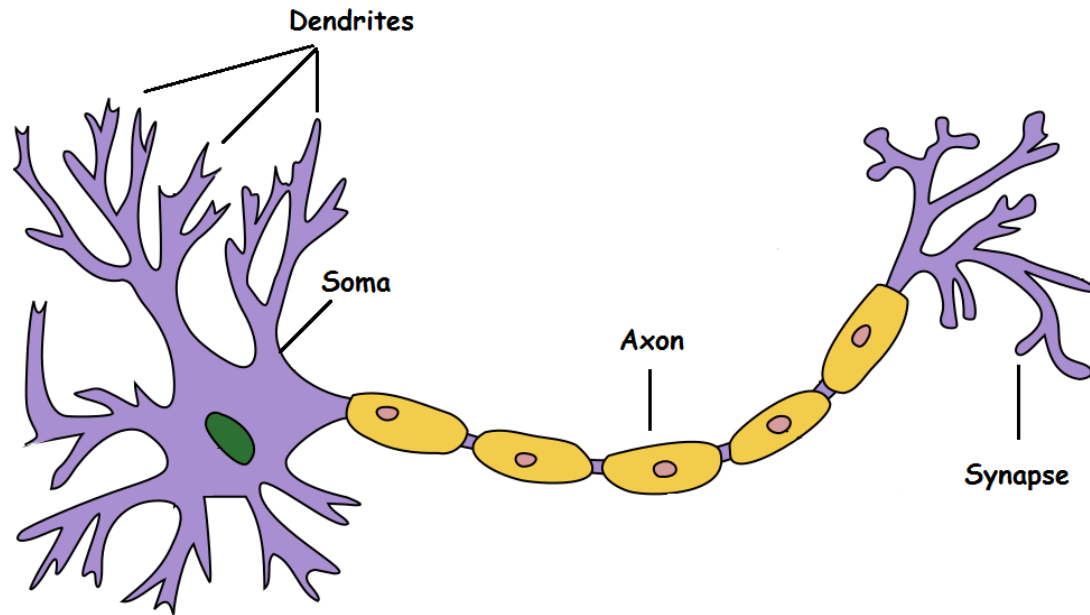
```
y_pred = model.predict(img_test)  
print("Prediction:", y_pred)
```

↳ Prediction: [1]



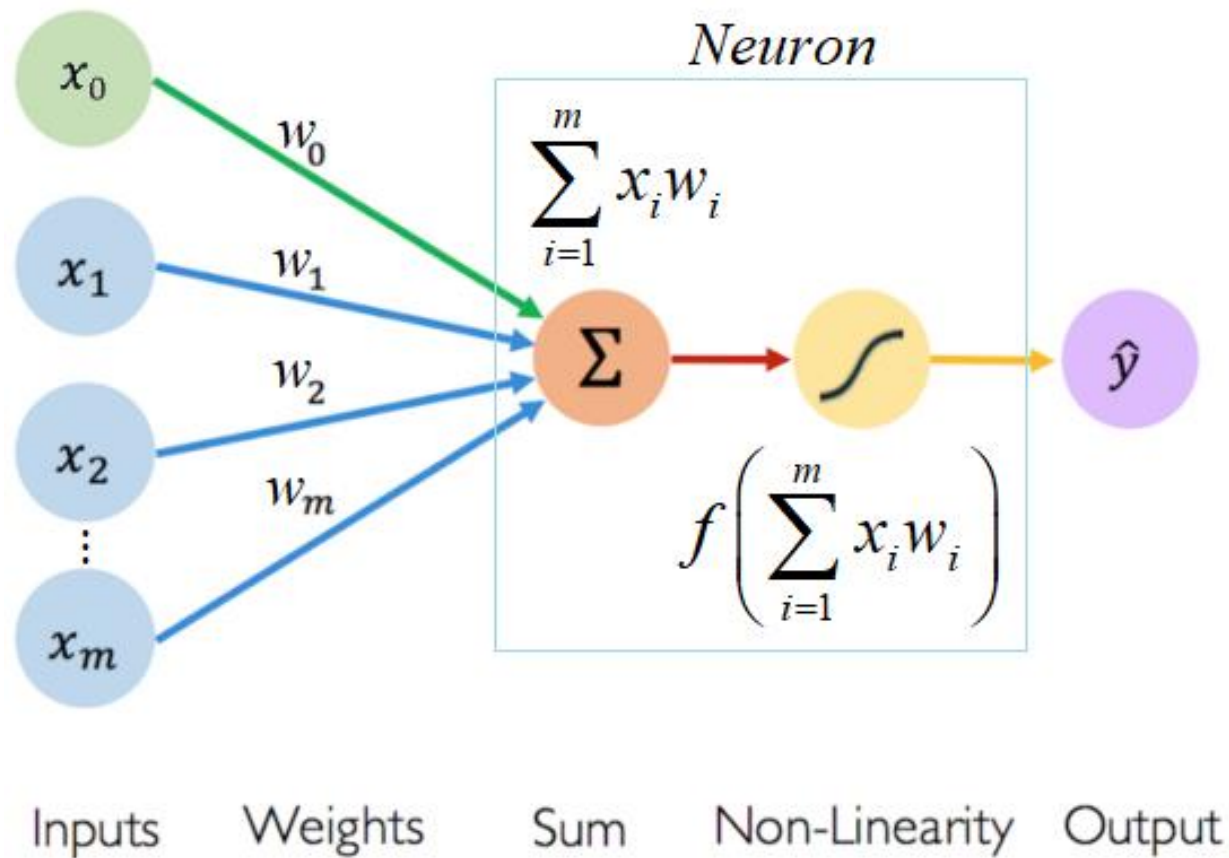
## 6.3 Artificial Neural Networks (ANN)

- Bộ não con người chứa khoảng  $10^{11}$  nơ-ron, với hơn  $10^{14}$  liên kết, tạo thành một mạng tế bào thần kinh khổng lồ.
- Mỗi nơ-ron: phần thân (soma), một trục thần kinh ra (axon) và một hệ thống dạng cây chứa các dây thần kinh vào (dendrites).
- Hoạt động: Khi điện thế ở dây thần kinh vào vượt quá 1 ngưỡng nào đó, nơ-ron bắt đầu giật (firing), tạo nên một xung điện truyền trên trục thần kinh ra và giải phóng năng lượng cho dây thần kinh vào của các nơ-ron khác qua các khớp nối (synapse).



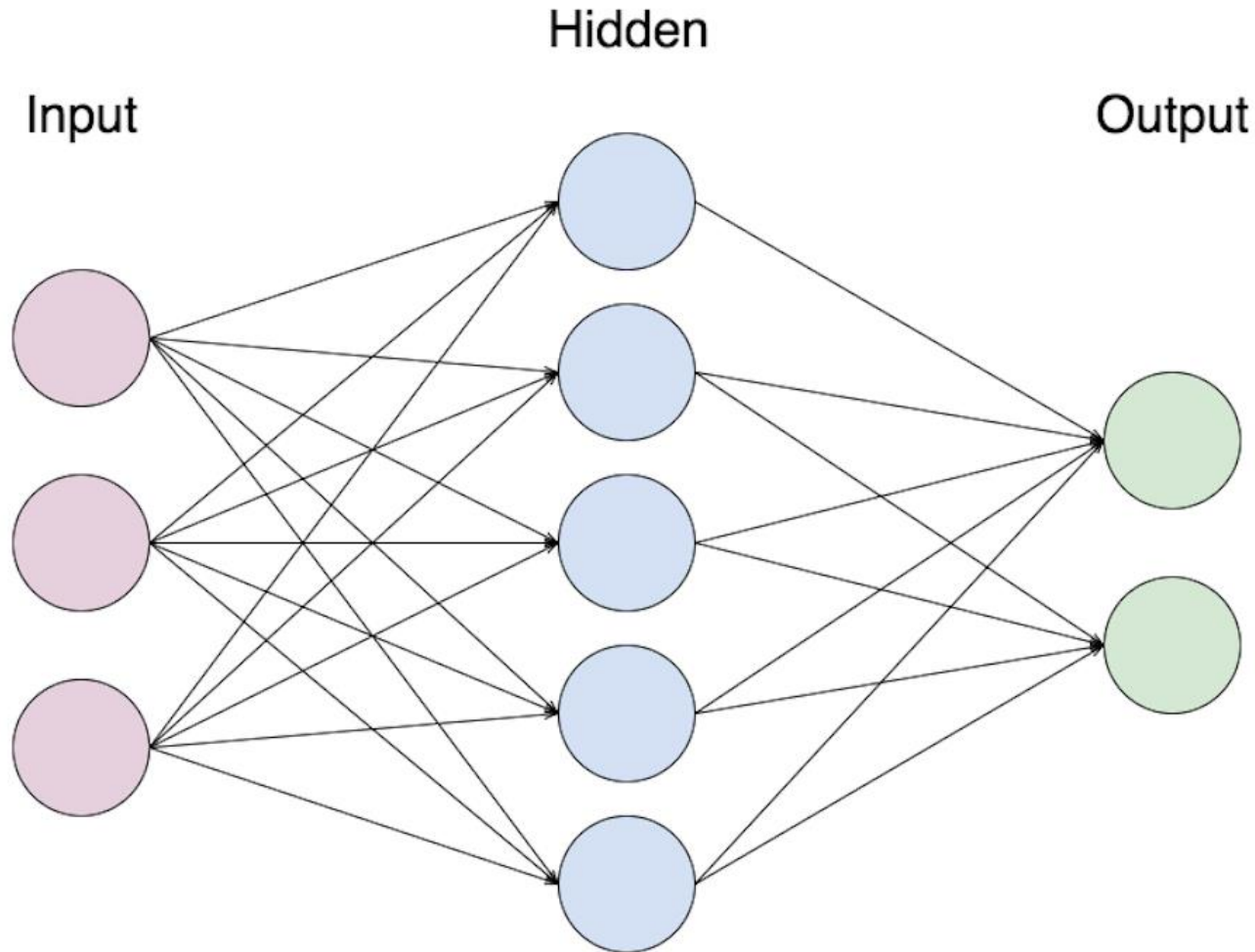
## 6.3 Artificial Neural Networks (ANN)

### Cấu tạo 1 neuron nhân tạo



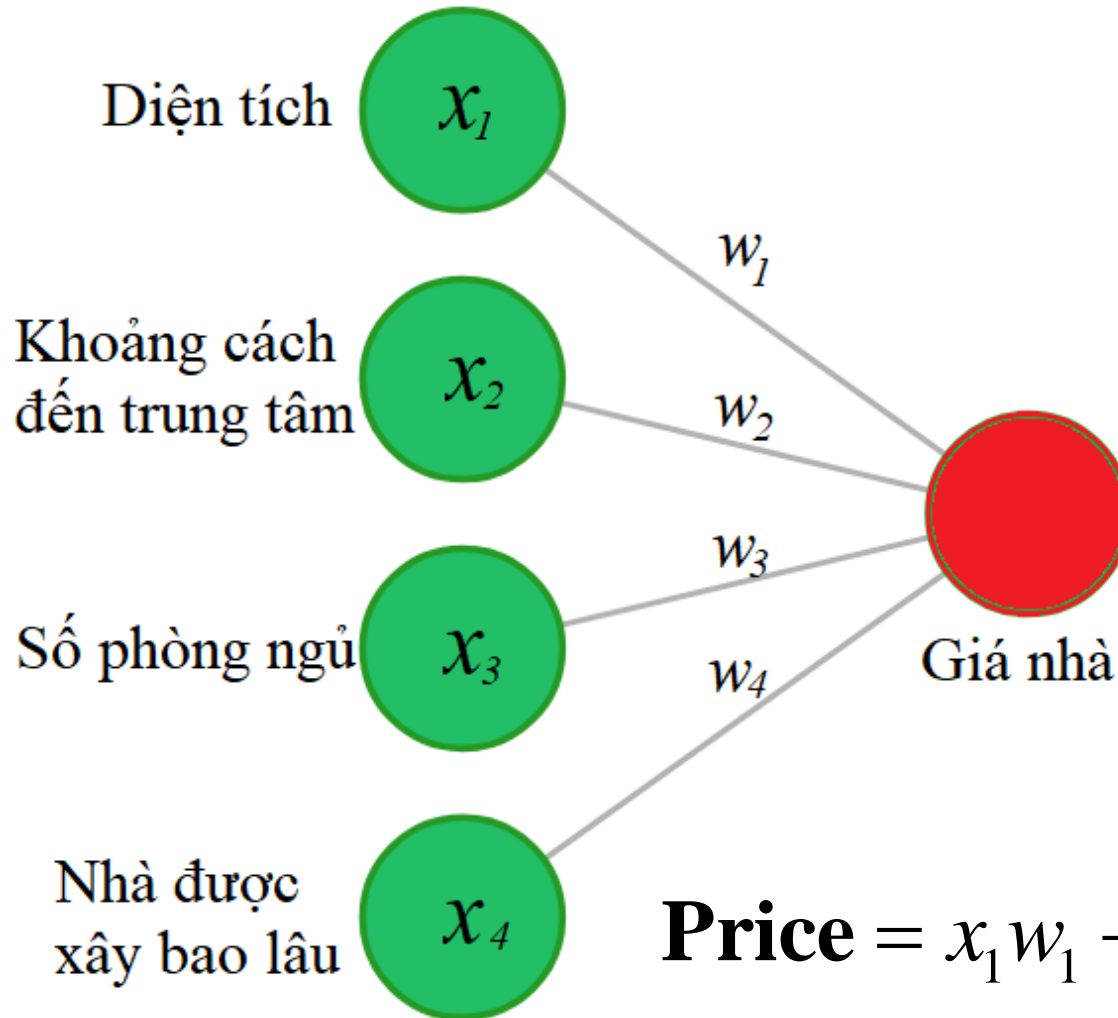
## 6.3 Artificial Neural Networks (ANN)

### Mạng neuron nhân tạo



## 6.3 Artificial Neural Networks (ANN)

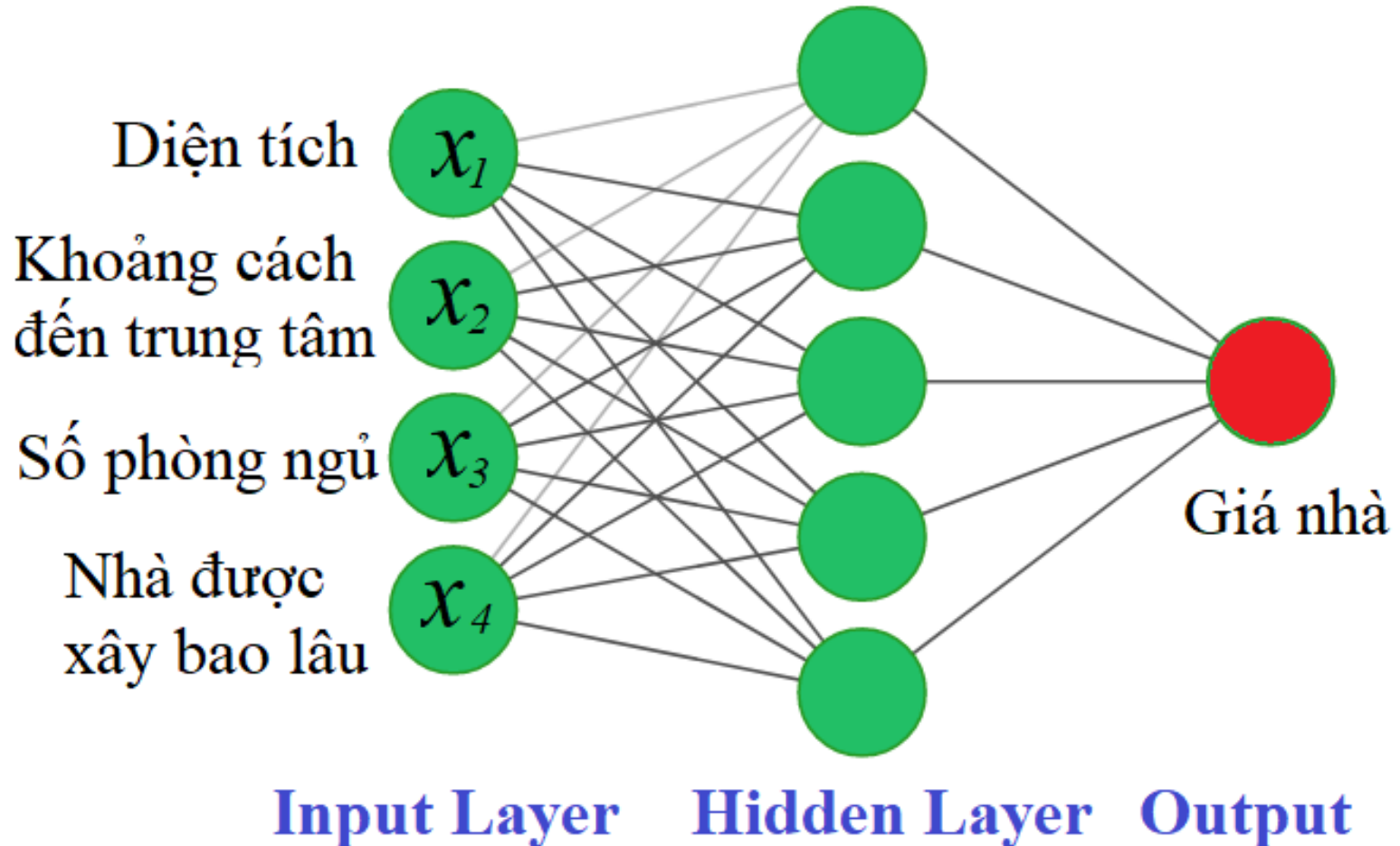
### Hoạt động của neuron networks



$$\text{Price} = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

## 6.3 Artificial Neural Networks (ANN)

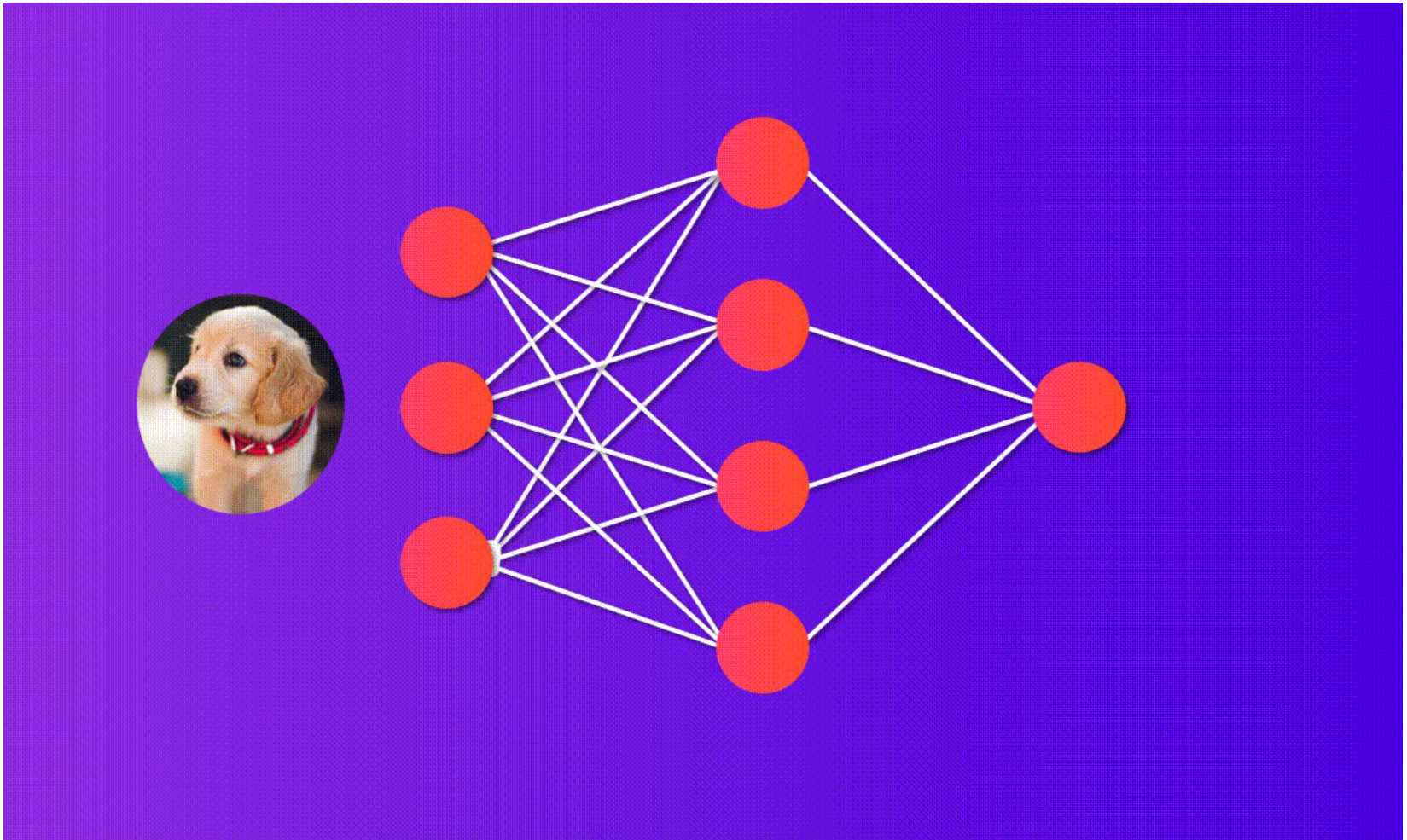
### Hoạt động của neuron networks





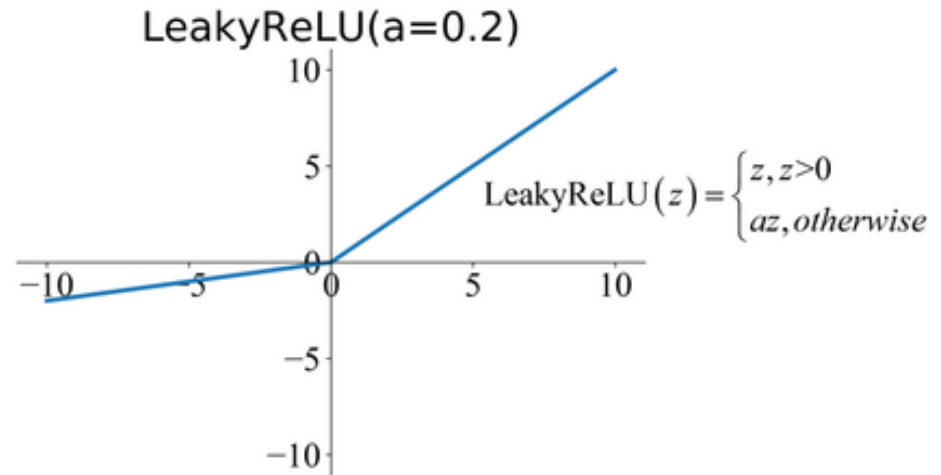
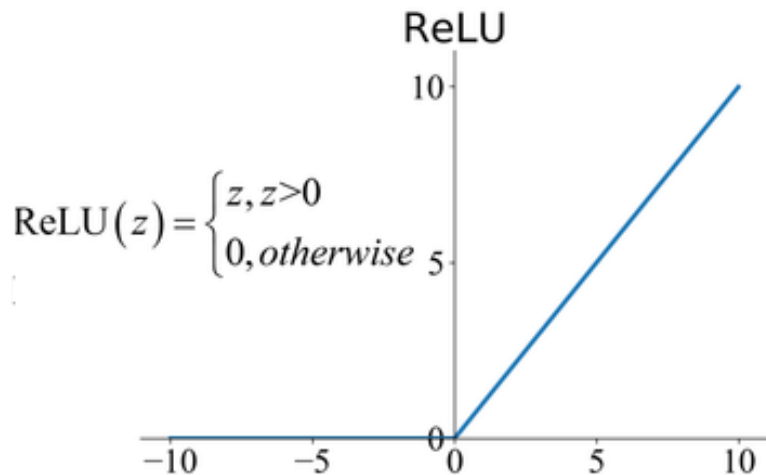
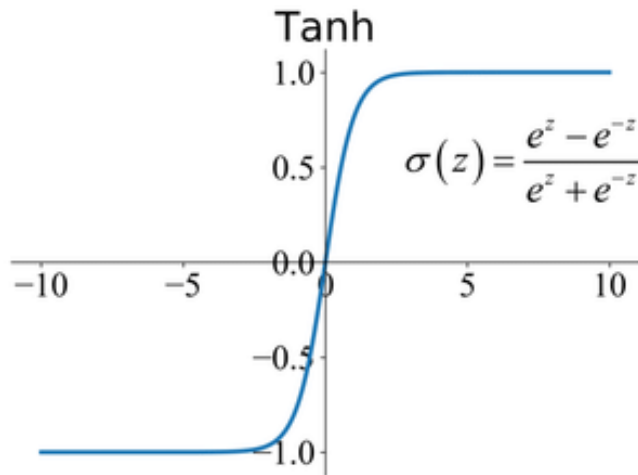
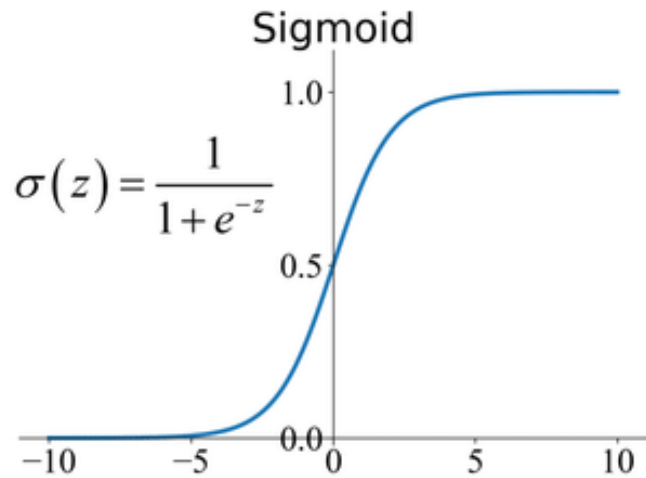
## 6.3 Artificial Neural Networks (ANN)

### Hoạt động của neuron networks



# 6.3 Artificial Neural Networks (ANN)

## Hàm kích hoạt (Activation Function)

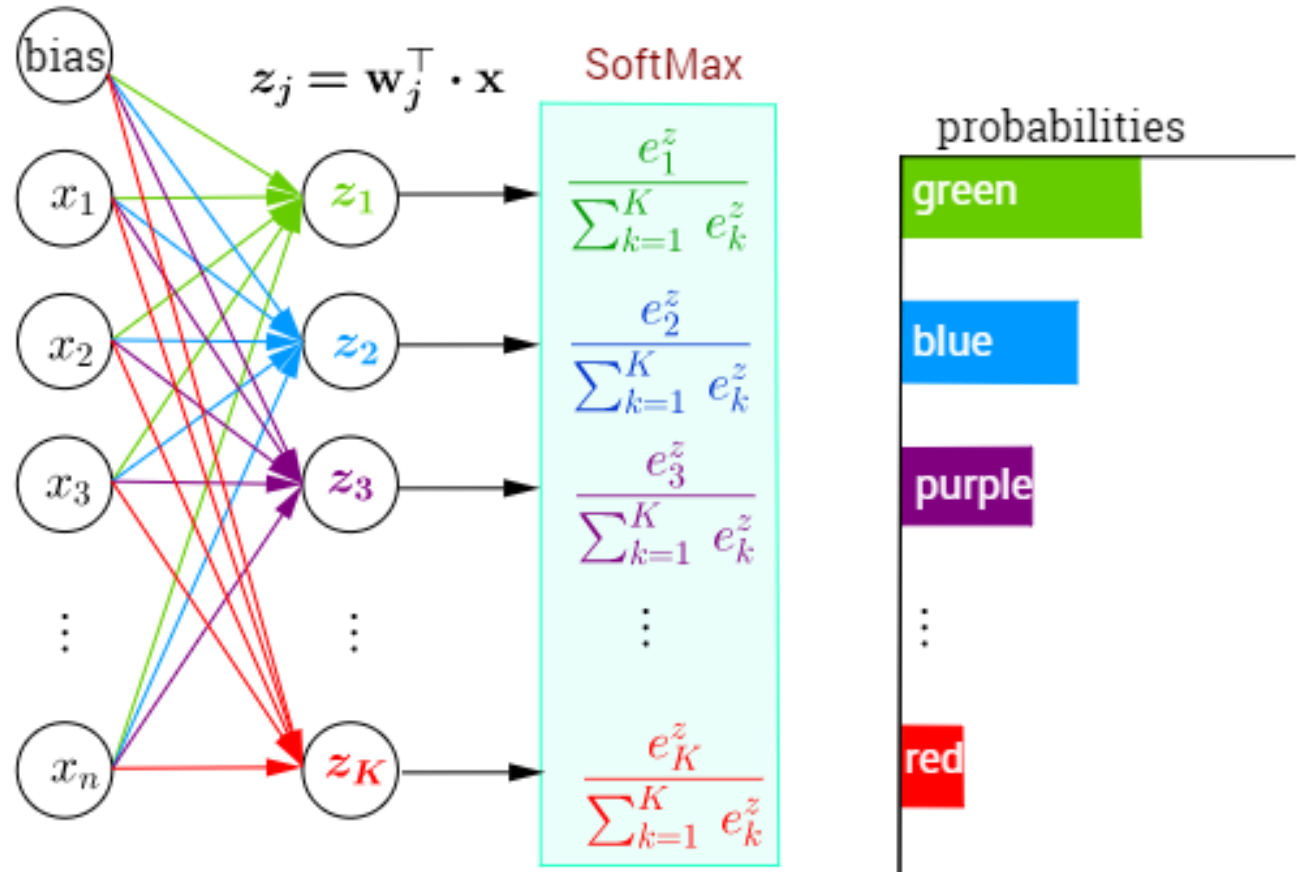


# 6.3 Artificial Neural Networks (ANN)

## Hàm kích hoạt (Activation Function)

### Softmax function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

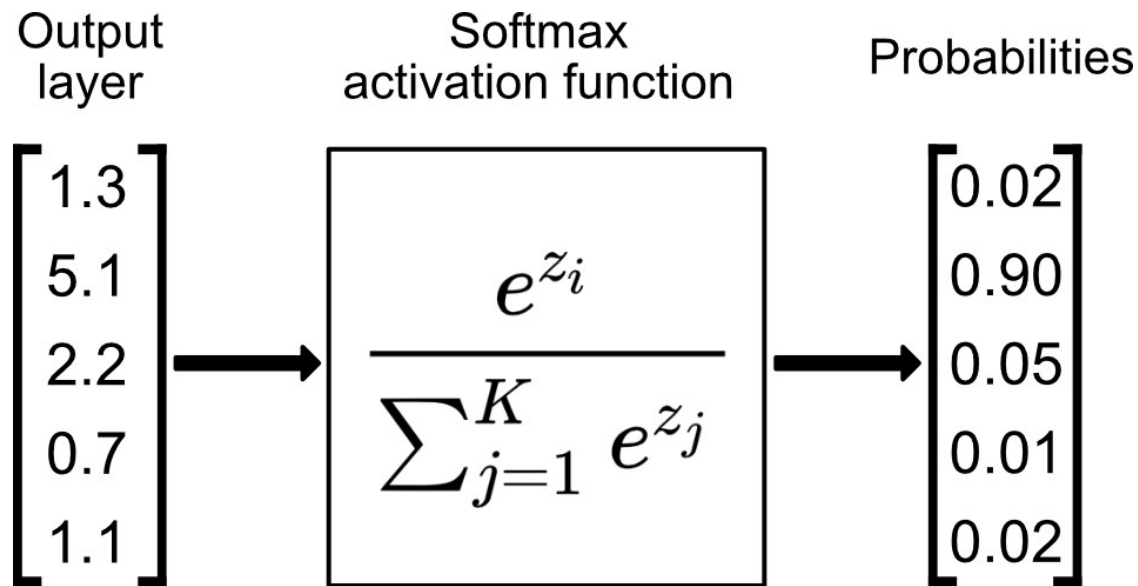




## 6.3 Artificial Neural Networks (ANN)

### Hàm kích hoạt (Activation Function)

#### Softmax function

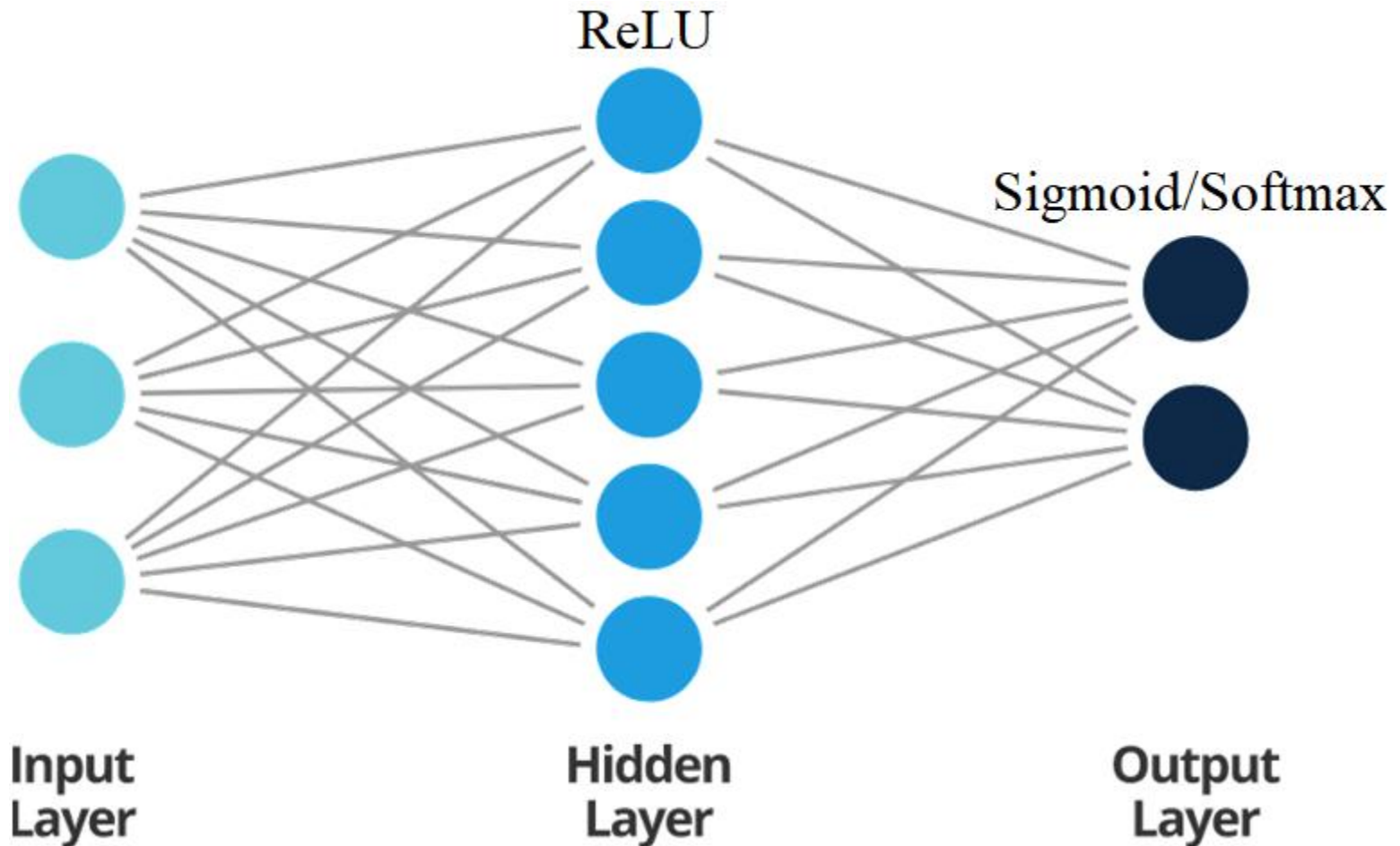


Why Non-linear Activation Functions:

[https://www.youtube.com/watch?v=NkOv\\_k7r6no](https://www.youtube.com/watch?v=NkOv_k7r6no)

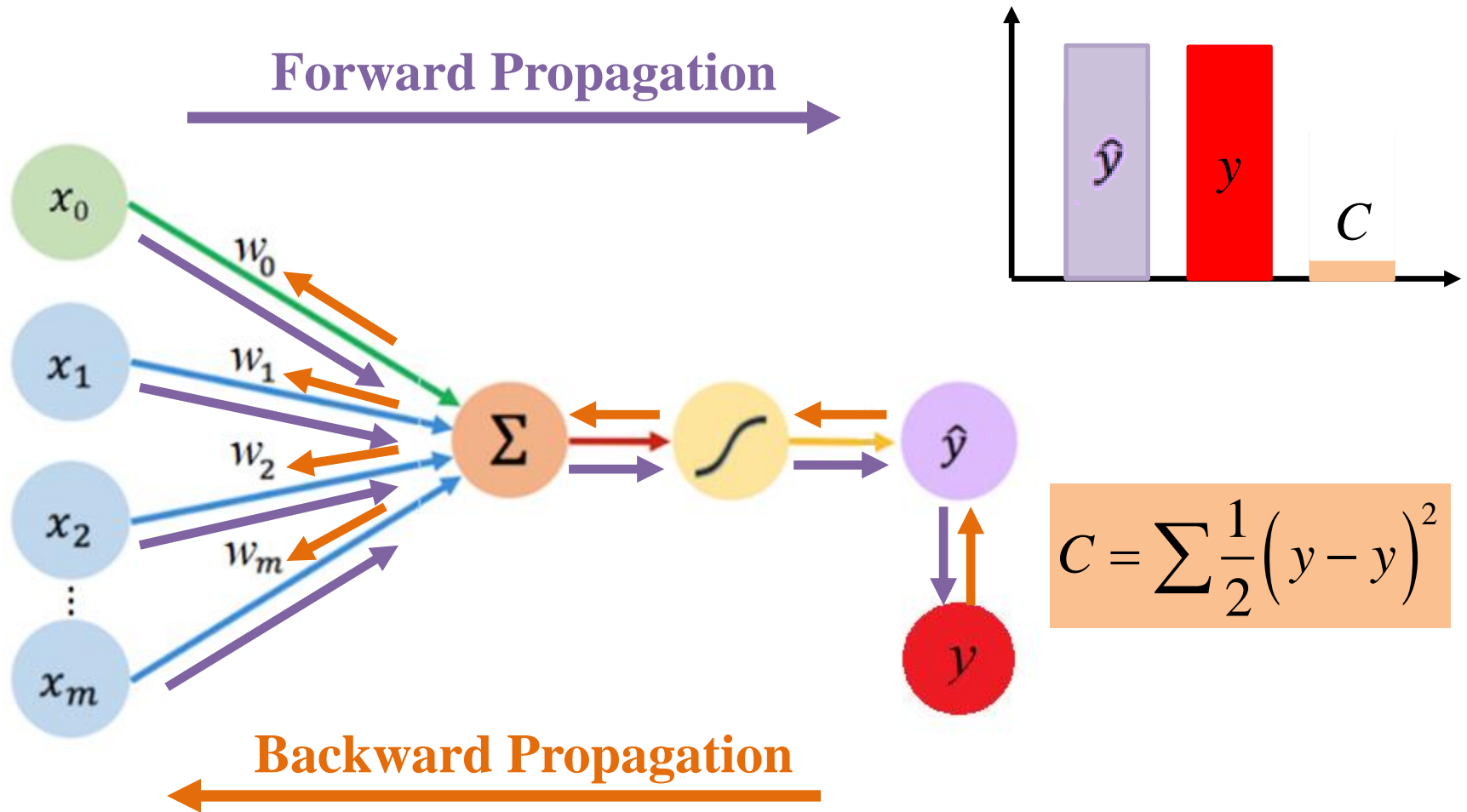
## 6.3 Artificial Neural Networks (ANN)

### Hàm kích hoạt (Activation Function)



## 6.3 Artificial Neural Networks (ANN)

### Quá trình học neuron networks



## 6.3 Artificial Neural Networks (ANN)

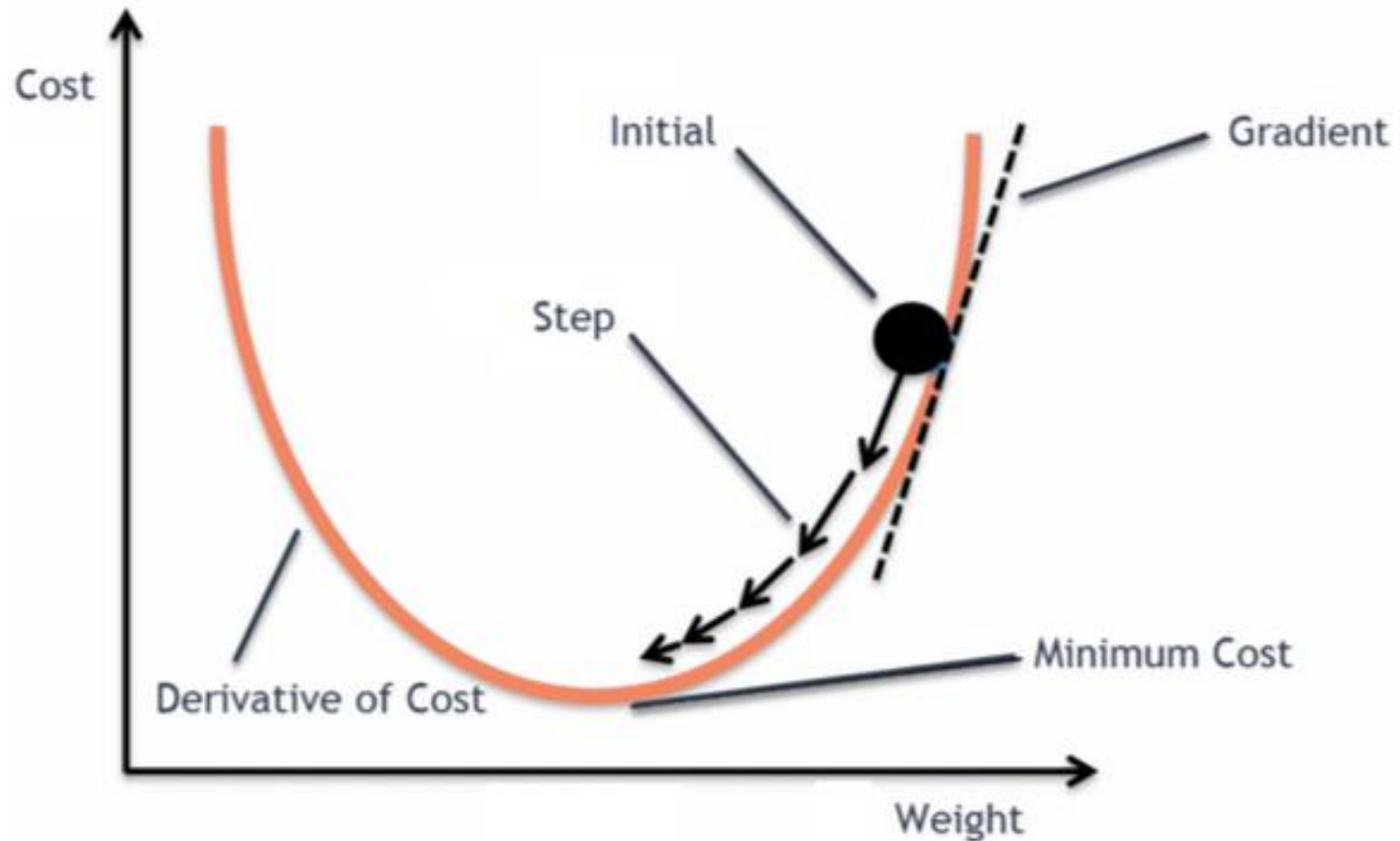
### Gradient Descent

Gọi Cost function là:  $C(\theta; x, y)$ , với  $\theta$  là tham số cần tìm để  $C$  đạt cực tiểu.

- STEP 1: Tính đạo hàm riêng  $\frac{\delta C(\theta)}{\delta \theta}$
- STEP 2: Khởi tạo  $\theta_0$ , hệ số học  $\eta$  và điều kiện dừng  $\varepsilon$
- STEP 3: Vòng lặp
  - Cập nhật:  $\theta_{i+1} = \theta_i - \eta \frac{\delta C(\theta)}{\delta \theta}$
  - Dừng vòng lặp khi:  $\frac{\delta C(\theta)}{\delta \theta} < \varepsilon$

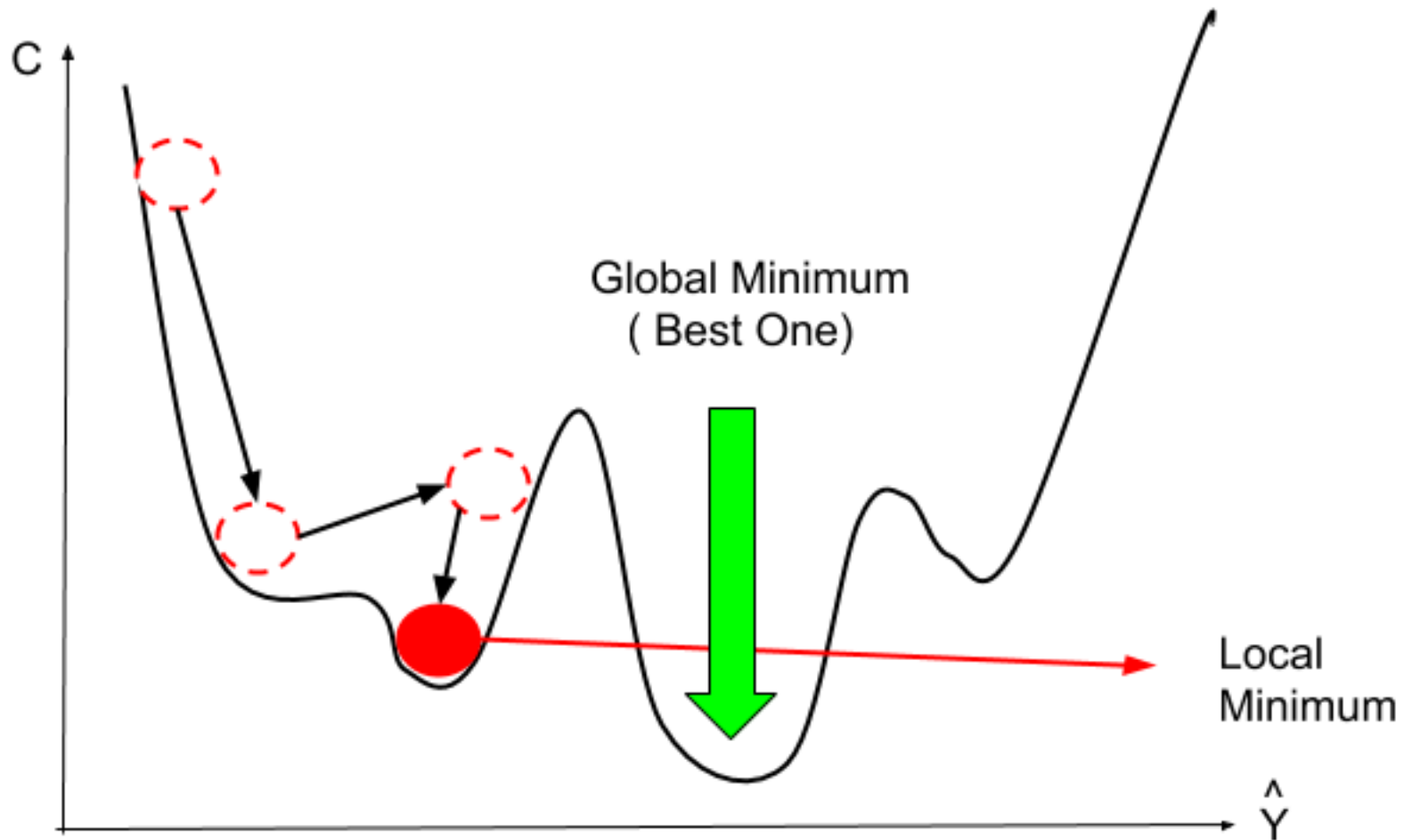
## 6.3 Artificial Neural Networks (ANN)

### Gradient Descent



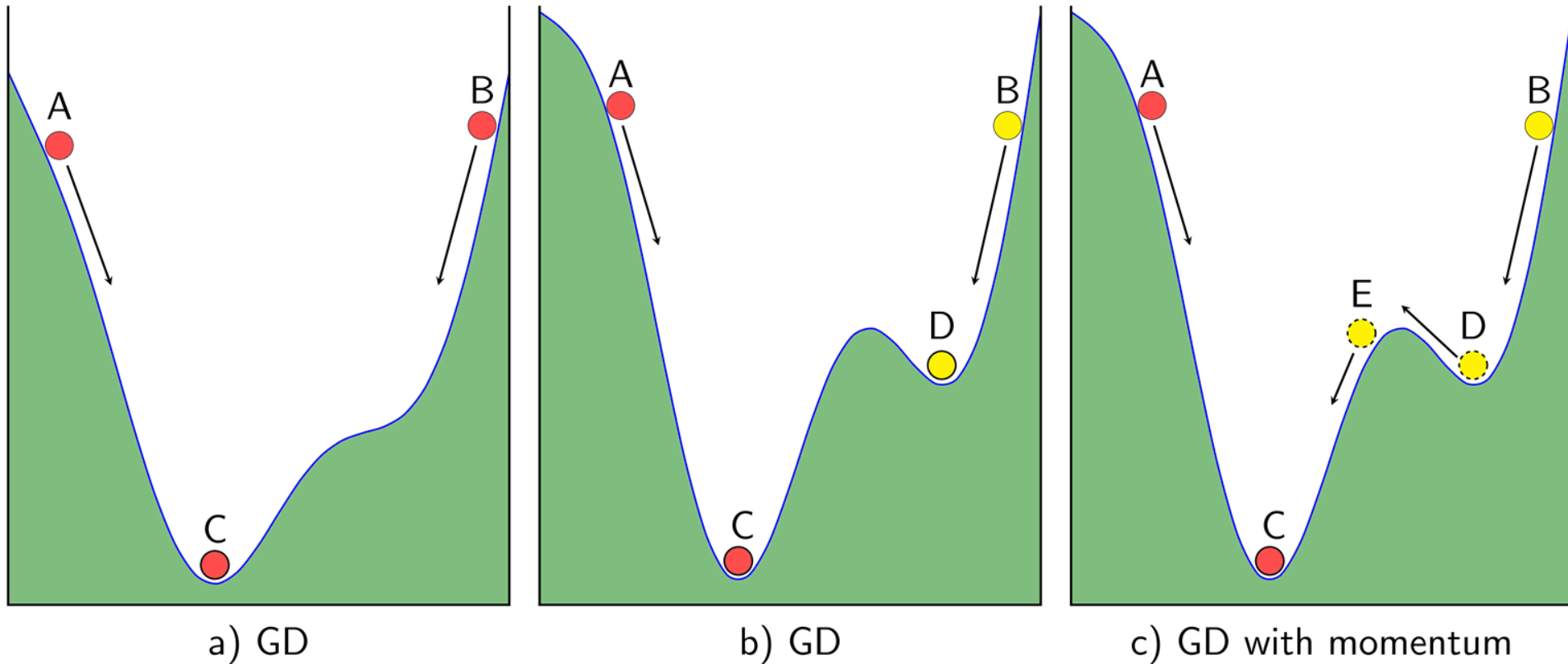
## 6.3 Artificial Neural Networks (ANN)

### Gradient Descent



## 6.3 Artificial Neural Networks (ANN)

### Gradient Descent with Momentum



$$\theta_{i+1} = \theta_i - v_t$$

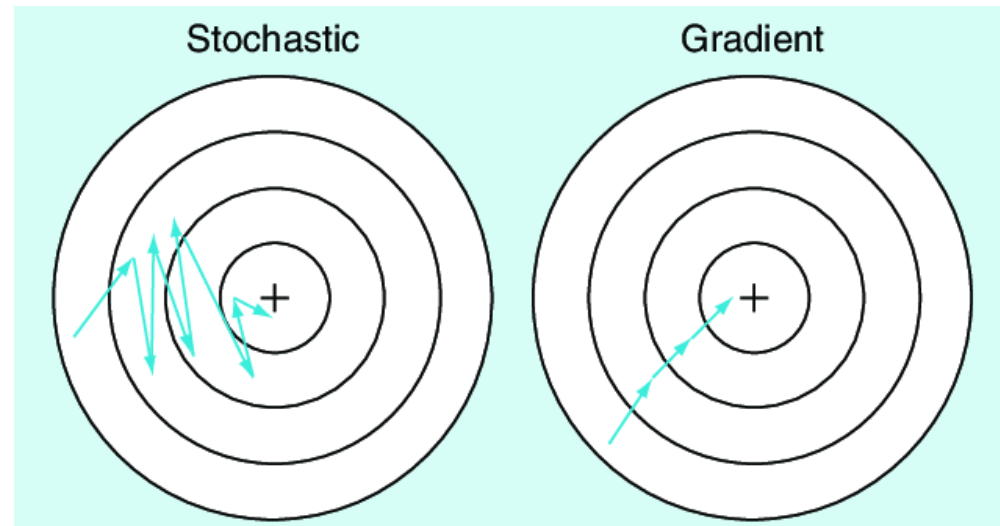
$$v_t = \gamma v_{t-1} + \eta \frac{\delta C(\theta)}{\delta \theta}, \gamma \approx 0.9$$

## 6.3 Artificial Neural Networks (ANN)

### Các biến thể của Gradient Descent

- *Stochastic Gradient Descent (SGD)*: Trong thuật toán này, tại 1 thời điểm, chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu  $(x_i, y_i)$  rồi cập nhật  $\theta$  dựa trên đạo hàm này. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật  $\theta$ , với SGD thì mỗi epoch ứng với  $N$  lần cập nhật  $\theta$ , với  $N$  là số điểm dữ liệu. Sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên.

$$\theta_{i+1} = \theta_i - \eta \frac{\delta C(\theta; x_i, y_i)}{\delta \theta}$$



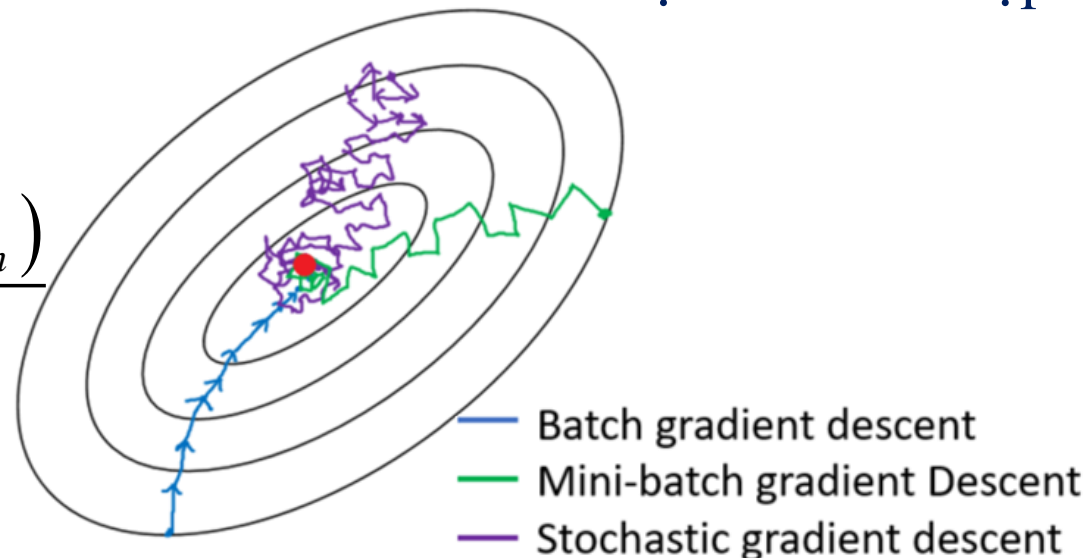


## 6.3 Artificial Neural Networks (ANN)

### Các biến thể của Gradient Descent

- *Mini-batch Gradient Descent*: Sử dụng cụm  $n$  dữ liệu ( $1 < n < N$ ). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các mini-batch, mỗi mini-batch có  $n$  điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu  $N$  không chia hết cho  $n$ ). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật.

$$\theta_{i+1} = \theta_i - \eta \frac{\delta C(\theta; x_{i:i+n}, y_{i:i+n})}{\delta \theta}$$



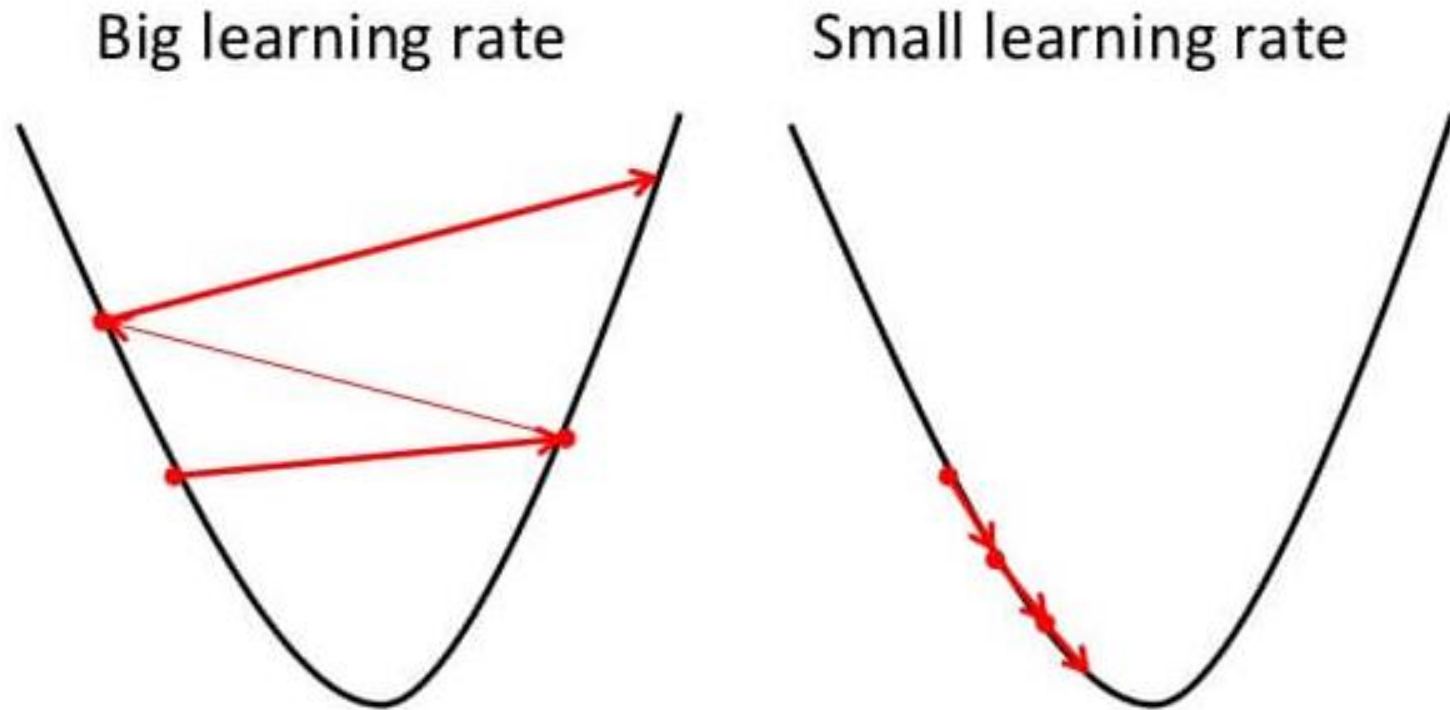
## 6.3 Artificial Neural Networks (ANN)

### Gradient Descent (Điều kiện dừng)

- *Giới hạn số lần cập nhật:* Ta sẽ giới hạn số lần cập nhật tham số để chương trình dừng lại. Tuy nhiên cách này có nhược điểm là chưa biết hàm mất mát có hội tụ hay chưa.
- *Kiểm tra giá trị hàm lỗi:* So sánh giá trị của hàm mất mát sau 2 lần cập nhật liên tiếp, nếu giá trị không khác nhau nhiều thì ta có thể coi là đã phần nào hội tụ được thì dừng lại. Phương pháp này có nhược điểm là rất dễ bị dừng lại tại điểm mà đồ thị của hàm lỗi bằng phẳng.
- *Kiểm tra giá trị đạo hàm:* So sánh giá trị của gradient sau 2 lần cập nhật liên tiếp hoặc sau một số lần cập nhật không quá lớn, nếu không khác biệt nhiều thì ta có thể dừng chương trình lại.
- Trong SGD và mini-batch GD, cách thường dùng là so sánh giá trị của hàm mất mát sau một vài lần cập nhật.

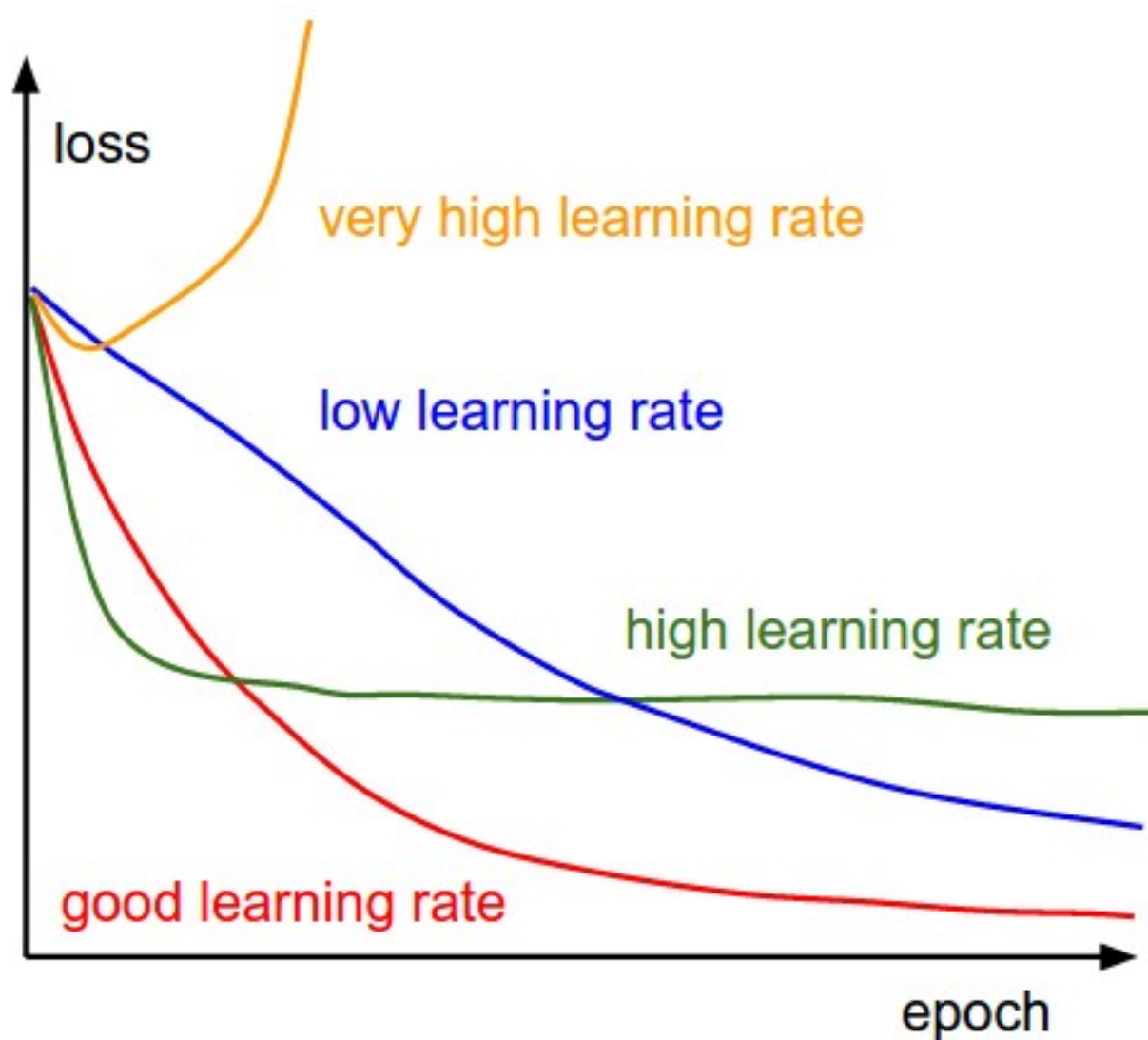
## 6.3 Artificial Neural Networks (ANN)

### Learning Rate



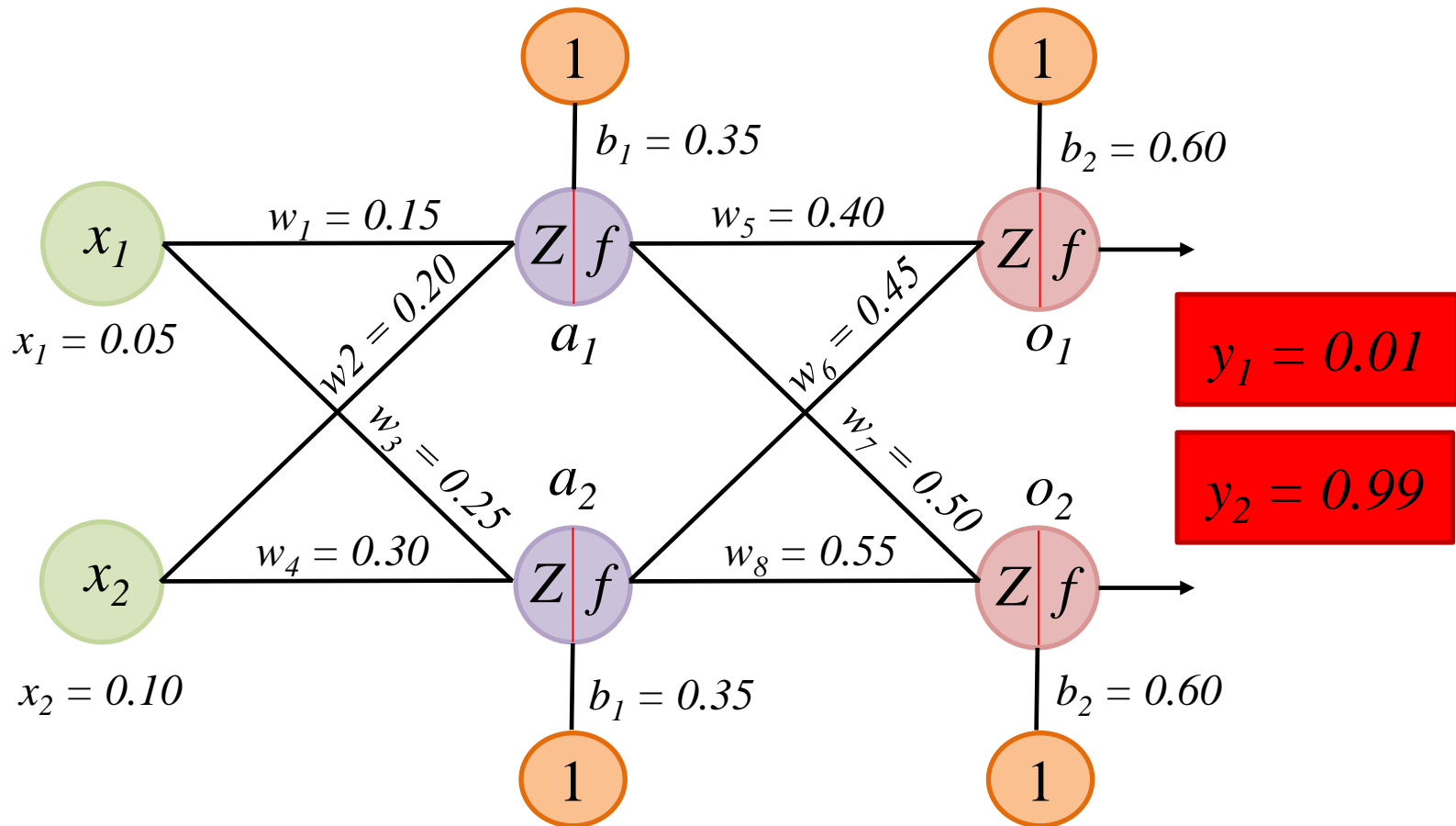
## 6.3 Artificial Neural Networks (ANN)

Learning Rate



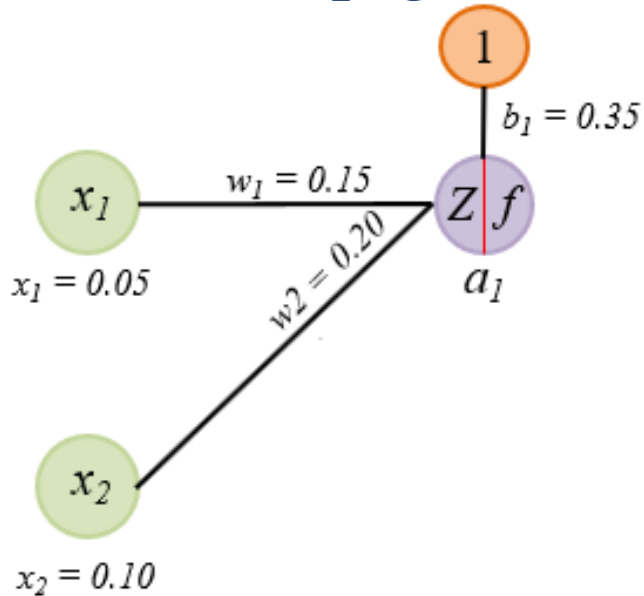
## 6.3 Artificial Neural Networks (ANN)

### Huấn luyện Neural Networks



## 6.3 Artificial Neural Networks (ANN)

### Forward Propagation

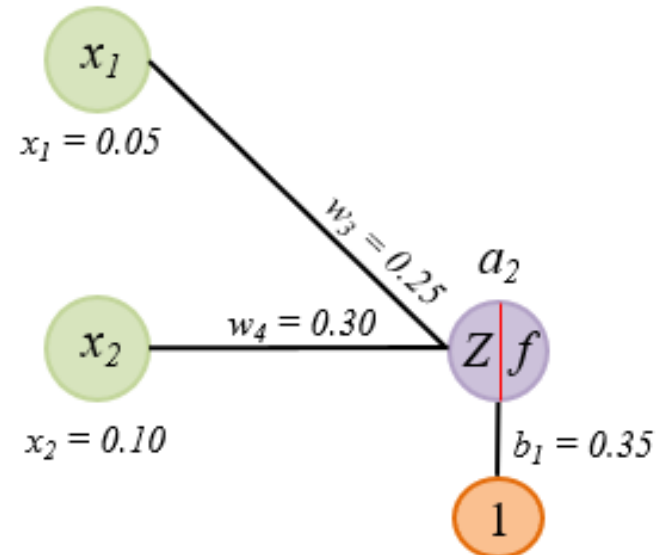


$$Z_{a_1} = x_1 w_1 + x_2 w_2 + 1 * b_1$$
$$= 0.05 * 0.15 + 0.1 * 0.2 + 1 * 0.35 = 0.3775$$

$$f_{a_1} = \frac{1}{1 + e^{-Z}} = \frac{1}{1 + e^{-0.3775}} = 0.5932699921$$

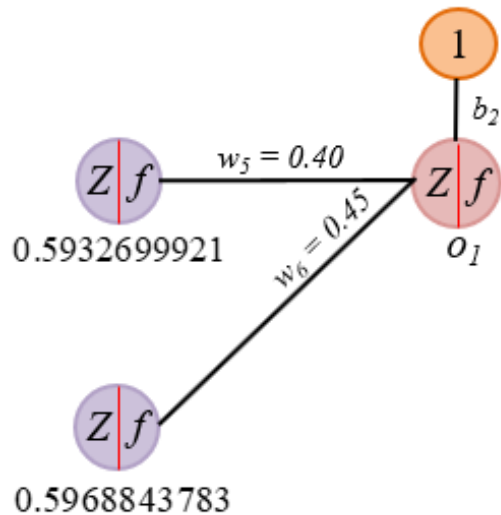
$$Z_{a_2} = x_1 w_3 + x_2 w_4 + 1 * b_1$$
$$= 0.05 * 0.25 + 0.1 * 0.3 + 1 * 0.35 = 0.3925$$

$$f_{a_2} = \frac{1}{1 + e^{-Z}} = \frac{1}{1 + e^{-0.3925}} = 0.5968843783$$



# 6.3 Artificial Neural Networks (ANN)

## Forward Propagation



$$Z_{o_1} = f_{a_1} w_5 + f_{a_2} w_6 + 1 * b_2$$

$$= 0.5932699921 * 0.4 + 0.5968843783 * 0.45 + 1 * 0.6$$

$$= 1.105905967$$

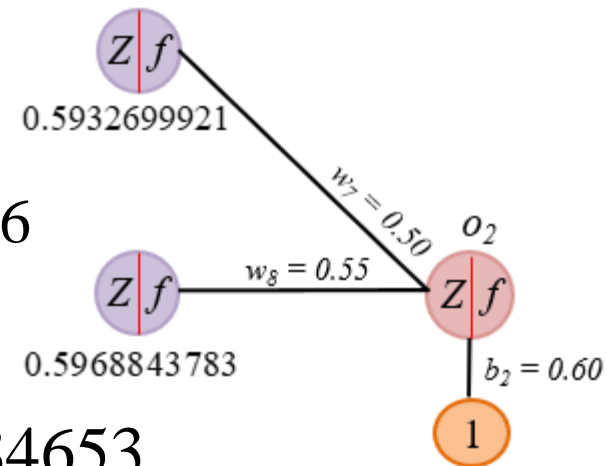
$$f_{o_1} = y_{o_1} = \frac{1}{1 + e^{-Z}} = \frac{1}{1 + e^{-1.105905967}} = 0.7513650696$$

$$Z_{o_2} = f_{a_1} w_7 + f_{a_2} w_8 + 1 * b_2$$

$$= 0.5932699921 * 0.5 + 0.5968843783 * 0.55 + 1 * 0.6$$

$$= 1.2249214404$$

$$f_{o_2} = y_{o_2} = \frac{1}{1 + e^{-Z}} = \frac{1}{1 + e^{-1.2249214404}} = 0.7729284653$$



## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation

$$C = \sum \frac{1}{2} (y - \hat{y})^2 = C_{o_1} + C_{o_2}$$

$$C_{o_1} = \frac{1}{2} (y_{o_1} - \hat{y}_{o_1})^2$$

$$= \frac{1}{2} (0.7513650696 - 0.01)^2$$

$$= 0.2748110832$$

$$C_{o_2} = \frac{1}{2} (y_{o_2} - \hat{y}_{o_2})^2$$

$$= \frac{1}{2} (0.7729284653 - 0.99)^2$$

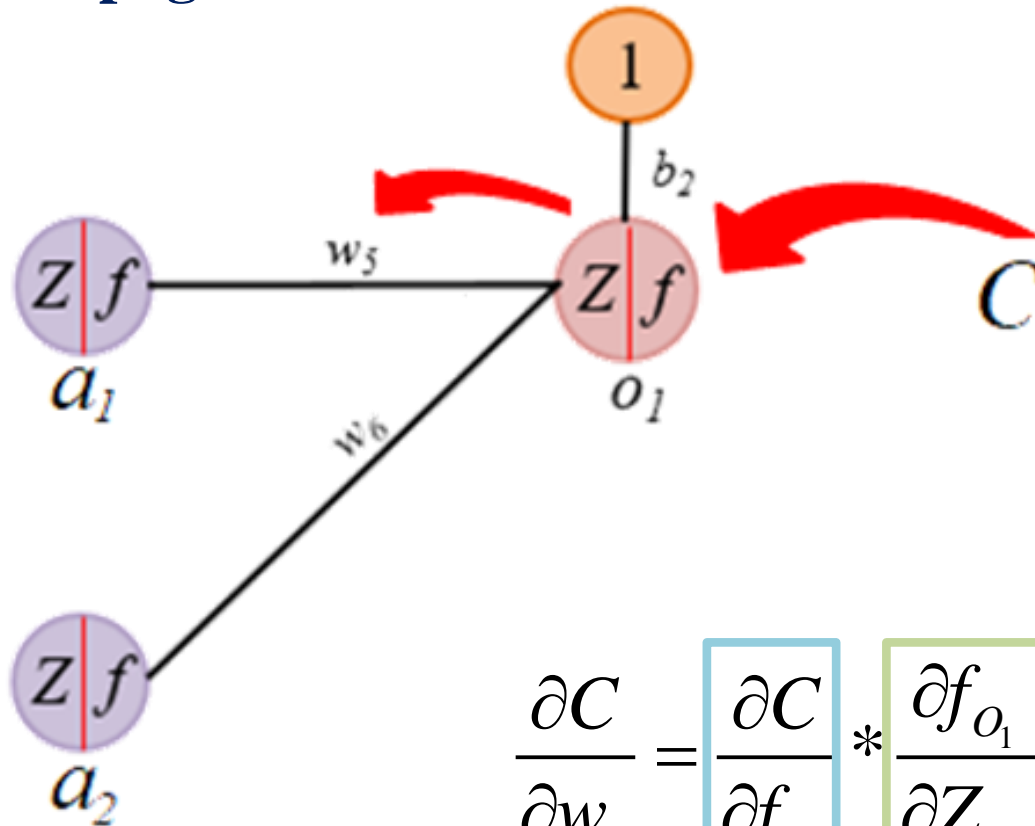
$$= 0.02356002559$$

$$C = 0.2748110832 + 0.02356002559 = 0.2983711088$$



## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation



$$\frac{\partial C}{\partial w_5} = \boxed{\frac{\partial C}{\partial f_{o_1}}} * \boxed{\frac{\partial f_{o_1}}{\partial Z_{o_1}}} * \boxed{\frac{\partial Z_{o_1}}{\partial w_5}}$$

## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation

$$C = \frac{1}{2} \left( y_{o_1} - y_{o_1} \right)^2 + \frac{1}{2} \left( y_{o_2} - y_{o_2} \right)^2$$

$$\begin{aligned} \frac{\partial C}{\partial f_{o_1}} &= 2 * \frac{1}{2} \left( y_{o_1} - y_{o_1} \right)^{2-1} * 1 + 0 = \left( y_{o_1} - y_{o_1} \right) = 0.7513650696 - 0.01 \\ &= 0.7413650696 \end{aligned}$$

$$f_{o_1} = \frac{1}{1 + e^{-z_{o_1}}}$$

$$\begin{aligned} \frac{\partial f_{o_1}}{\partial z_{o_1}} &= f_{o_1} (1 - f_{o_1}) = 0.7513650696 * (1 - 0.7513650696) \\ &= 0.1868156018 \end{aligned}$$

## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation

$$Z_{o_1} = f_{a_1} w_5 + f_{a_2} w_6 + 1 * b_2$$

$$\frac{\partial Z_{o_1}}{\partial w_5} = a_1 + 0 + 0 = 0.5932699921$$

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial f_{o_1}} * \frac{\partial f_{o_1}}{\partial Z_{o_1}} * \frac{\partial Z_{o_1}}{\partial w_5}$$

$$= 0.7413650696 * 0.1868156018 * 0.5932699921$$

$$= 0.08216704056$$

$$w_5^* = w_5 - \eta \frac{\delta C}{\delta w_5} = 0.4 - 0.5 * 0.08216704056 \approx 0.36$$

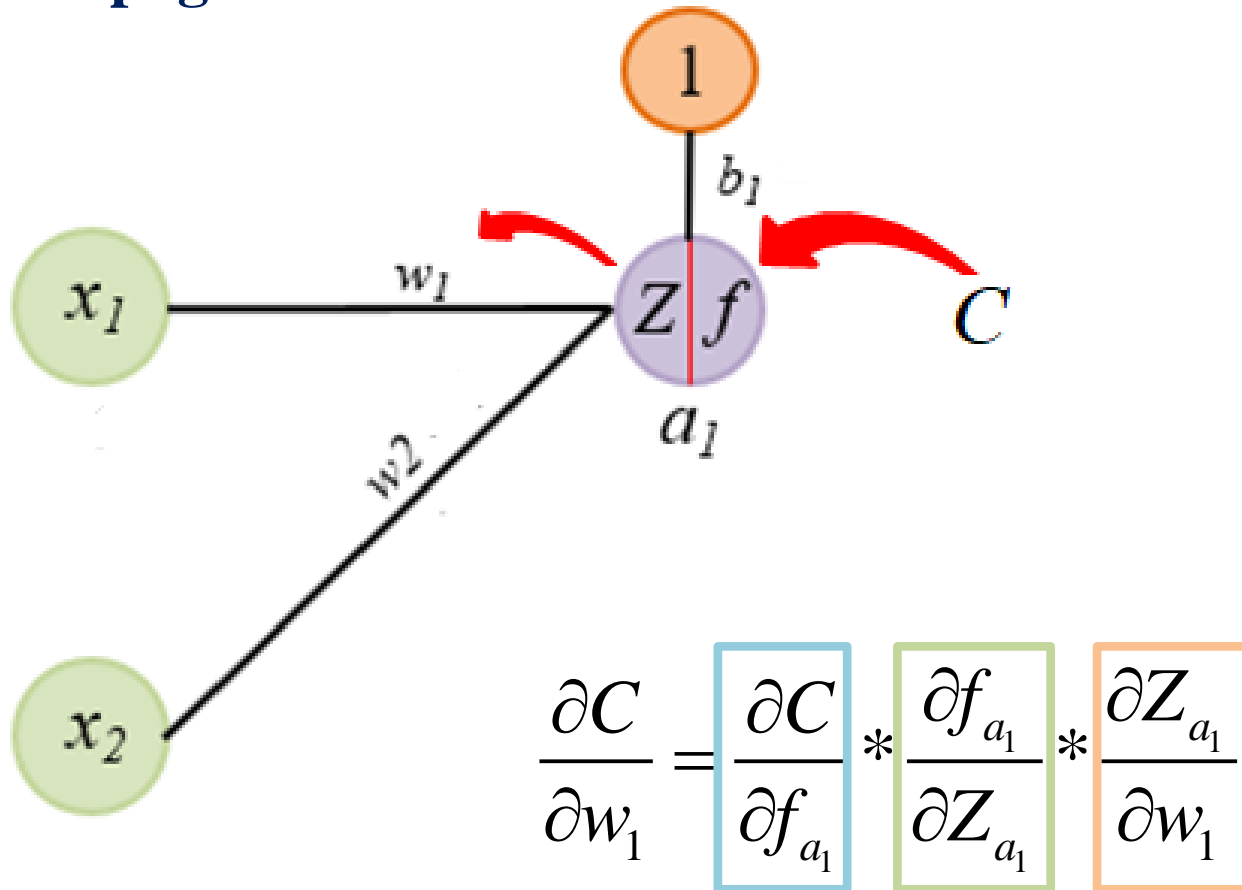
$$w_6^* \approx 0.41$$

$$w_7^* \approx 0.52$$

$$w_8^* \approx 0.56$$

## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation



## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation

$$\frac{\partial C}{\partial f_{a_1}} = \frac{\partial C_{o_1}}{\partial f_{a_1}} + \frac{\partial C_{o_2}}{\partial f_{a_1}} \quad \frac{\partial C_{o_1}}{\partial f_{a_1}} = \frac{\partial C_{o_1}}{\partial Z_{o_1}} + \frac{\partial Z_{o_1}}{\partial f_{a_1}} \quad \frac{\partial C_{o_2}}{\partial f_{a_1}} = -0.019049119$$

$$\frac{\partial C_{o_1}}{\partial Z_{o_1}} = \frac{\partial C_{o_1}}{\partial f_{o_1}} * \frac{\partial f_{o_1}}{\partial Z_{o_1}} = 0.7413650696 * 0.1868156018 \\ = 0.1384985616$$

$$Z_{o_1} = f_{a_1} w_5 + f_{a_2} w_6 + 1 * b_2 \\ \frac{\partial Z_{o_1}}{\partial f_{a_1}} = w_5 = 0.40$$

$$\frac{\partial C_{o_1}}{\partial f_{a_1}} = 0.1384985616 * 0.4 \\ = 0.05539942464$$

## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation

$$\begin{aligned}\frac{\partial C}{\partial f_{a_1}} &= \frac{\partial C_{o_1}}{\partial f_{a_1}} + \frac{\partial C_{o_2}}{\partial f_{a_1}} = 0.05539942464 - 0.019049119 \\ &= 0.0363503074\end{aligned}$$

$$\begin{aligned}f_{a_1} &= \frac{1}{1 + e^{-Z_{a_1}}} \\ \frac{\partial f_{a_1}}{\partial Z_{a_1}} &= f_{a_1} (1 - f_{a_1}) = 0.5932699921 * (1 - 0.5932699921) \\ &= 0.2413007086\end{aligned}$$

## 6.3 Artificial Neural Networks (ANN)

### Backward Propagation

$$Z_{a_1} = x_1 w_1 + x_2 w_2 + 1 * b_1$$

$$\frac{\partial Z_{a_1}}{\partial w_1} = x_1 = 0.05$$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial f_{a_1}} * \frac{\partial f_{a_1}}{\partial Z_{a_1}} * \frac{\partial Z_{a_1}}{\partial w_1}$$

$$= 0.0363503074 * 0.2413007086 * 0.05$$

$$= 0.000438568$$

$$w_1^* = w_1 - \eta \frac{\delta C}{\delta w_1} = 0.4 - 0.15 * 0.000438568 \approx 0.149780716$$

$$w_2^* \approx 0.19916$$

$$w_3^* \approx 0.24975$$

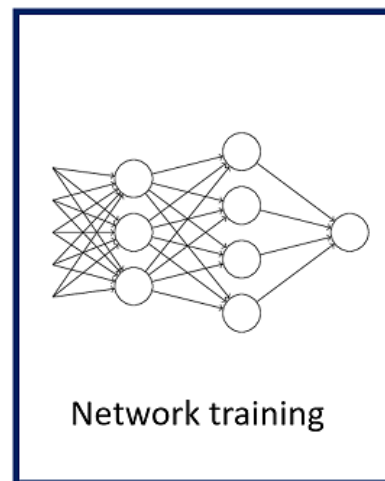
$$w_4^* \approx 0.29950$$

## 6.3 Artificial Neural Networks (ANN)

### Xây dựng mạng ANN dự đoán chữ viết tay

0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9

Data & Labels



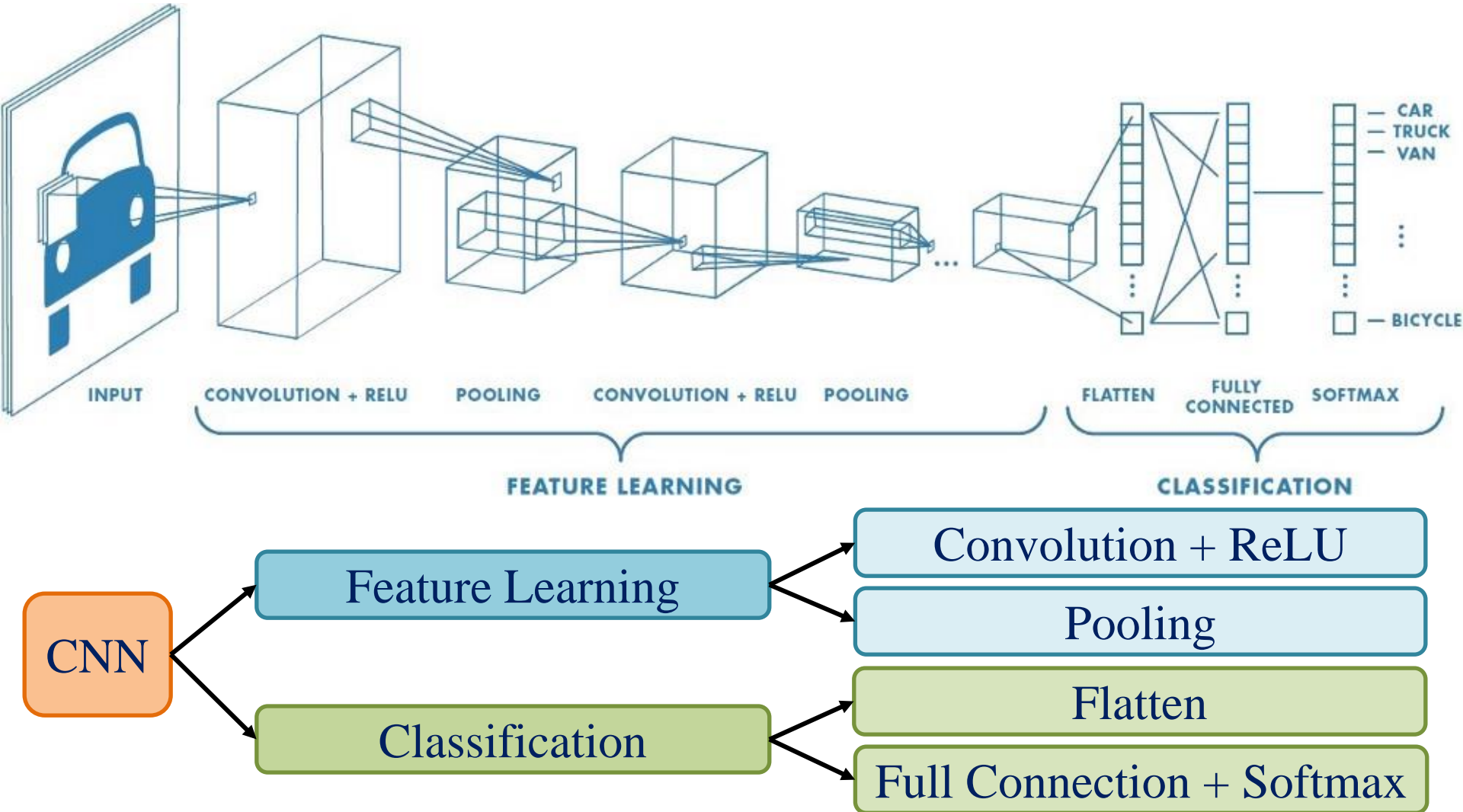
0  
1  
2  
3  
4  
5  
6  
7  
8  
9

[https://colab.research.google.com/drive/1A\\_MzhOWTh9lczB9WuJebDLKILn3u6Tgt?usp=sharing](https://colab.research.google.com/drive/1A_MzhOWTh9lczB9WuJebDLKILn3u6Tgt?usp=sharing)



# 6.4 Convolution Neural Networks (CNN)

## CNN Overview



## 6.4 Convolution Neural Networks (CNN)

### Basic Convolution Operation

- Kí hiệu phép convolution (tích chập) là  $*$ , kí hiệu  $Y = X * W$

1	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1

$X$

$*$

1	0	0
0	1	1
1	0	1

$W$

$=$

5		

$Y$

- Với mỗi phần tử  $x_{ij}$  trong ma trận  $X$  lấy ra một ma trận có kích thước bằng kích thước của kernel  $W$  có phần tử  $x_{ij}$  làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận  $A$  ( $A \in X$ ). Sau đó tính tổng các phần tử của phép tính nhân từng phần tử của ma trận  $A$  và ma trận  $W$ , rồi viết vào ma trận kết quả  $Y$ .

## 6.4 Convolution Neural Networks (CNN)

- Ví dụ về phép convolution với Kernel  $W$

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

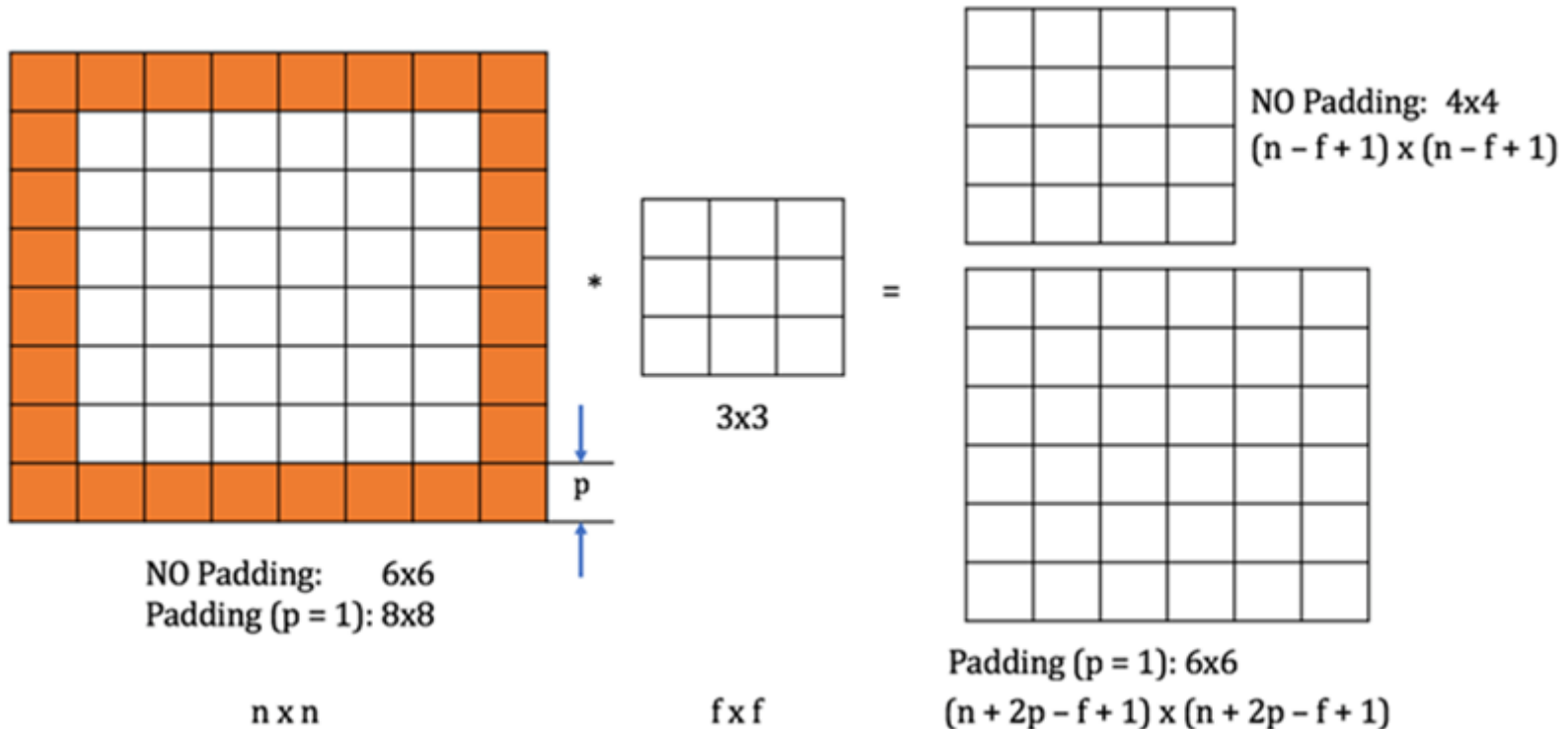
Image

4		

Convolved  
Feature

## 6.4 Convolution Neural Networks (CNN)

- Thông thường khi tính thì sẽ bỏ qua các phần tử ở *viền ngoài*, do không tìm được ma trận A ở trong X. Vì thế, sau phép tính convolution kích thước ma trận Y thường nhỏ hơn ma trận X.
- Một giải pháp được đưa ra để tăng kích thước ma trận Y là phép **Padding**, sẽ thêm giá trị 0 ở viền ngoài ma trận X.



## 6.4 Convolution Neural Networks (CNN)

### Stride

*Input*

4	9	2	5	8	3
→	6	2	4	0	3
	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

$$n_H \times n_W = 6 \times 6$$

*Filter*

1	0	-1
1	0	-1
1	0	-1

\*

**Parameters:**

Size:  $f = 3$

Stride:  $s = 1$

Padding:  $p = 0$

*Result*

2	6		

=

**6** =  $9 \times 1 + 2 \times 0 + 5 \times (-1) +$   
 $6 \times 1 + 2 \times 0 + 4 \times (-1) +$   
 $4 \times 1 + 5 \times 0 + 4 \times (-1)$

<https://indoml.com>

# 6.4 Convolution Neural Networks (CNN)

## Stride

*Input*

4	9	2	5	8	3
2	4	5	4	5	2
5	6	5	4	7	8
5	7	7	9	2	1
5	8	5	3	8	4

Dimension: 6 x 6

*Filter*

1	0	-1
1	0	-1
1	0	-1

**Parameters:**

Size:  $f = 3$

Stride:  $s = 2$

Padding:  $p = 0$

*Result*

2	1

**1** =  $2 \cdot 1 + 5 \cdot 0 + 3 \cdot (-1) +$   
 $2 \cdot 1 + 4 \cdot 0 + 3 \cdot (-1) +$   
 $5 \cdot 1 + 4 \cdot 0 + 2 \cdot (-1)$

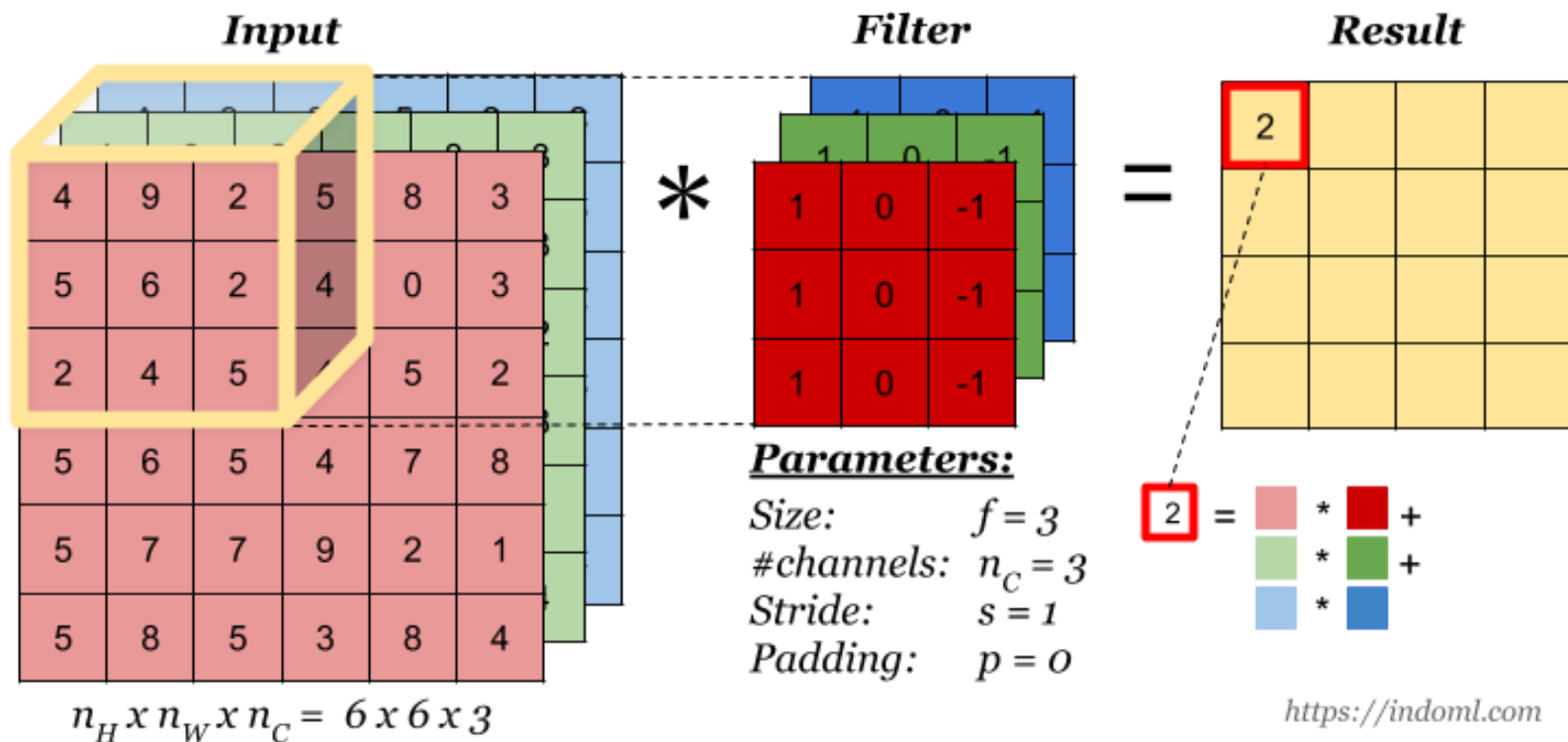
<https://indoml.com>

$$\left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right)$$

## 6.4 Convolution Neural Networks (CNN)

### Convolution Operation on Volume

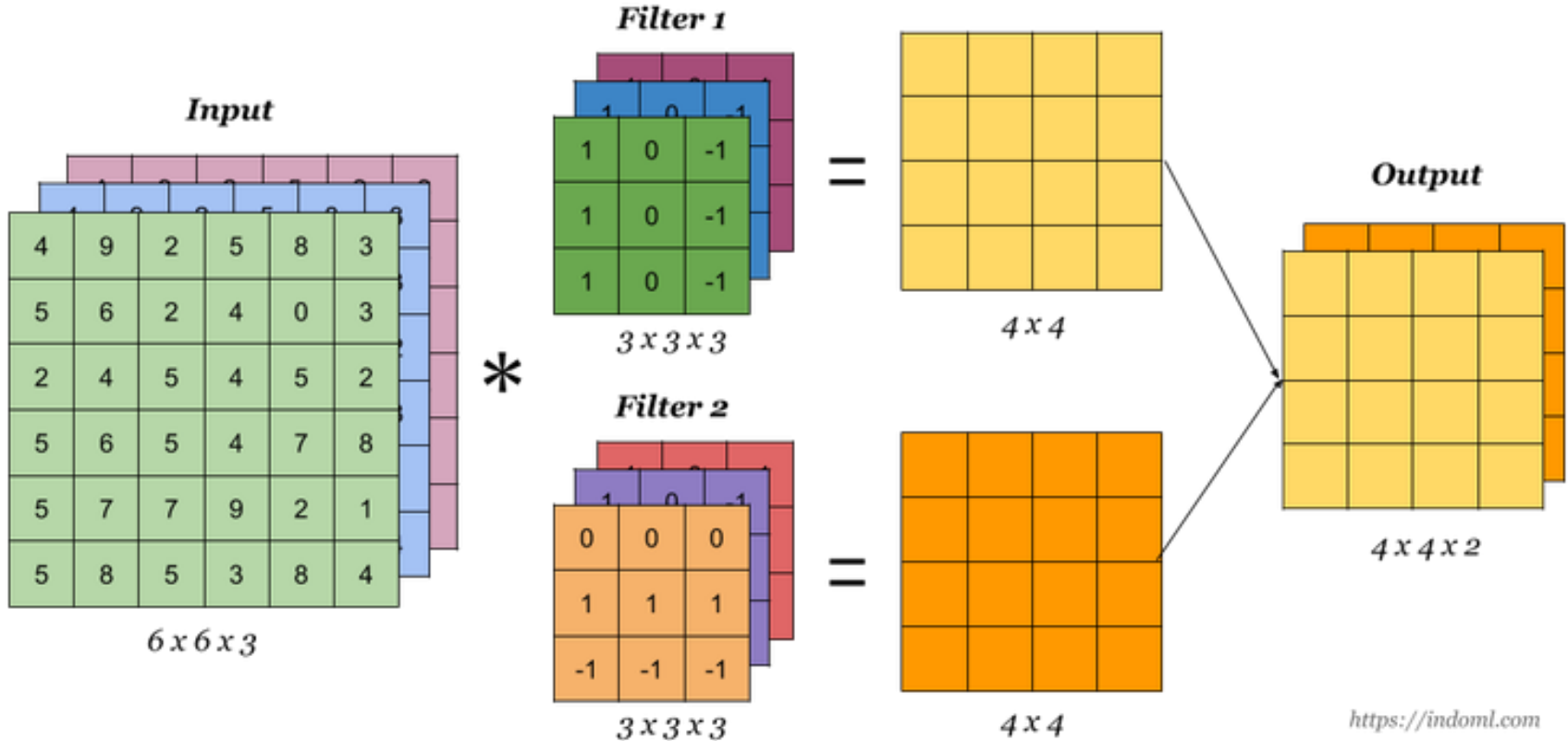
- Khi đầu vào có nhiều kênh (ví dụ: hình ảnh RGB), bộ lọc phải có số kênh phù hợp.
- Để tính toán một ô đầu ra, hãy thực hiện tích chập trên mỗi kênh phù hợp, sau đó cộng kết quả với nhau.



## 6.4 Convolution Neural Networks (CNN)

### Convolution Operation on Volume

- Nhiều bộ lọc có thể được sử dụng trong một lớp tích chập để phát hiện nhiều tính năng.
- Ngõ ra sẽ có số lớp bằng số bộ lọc.

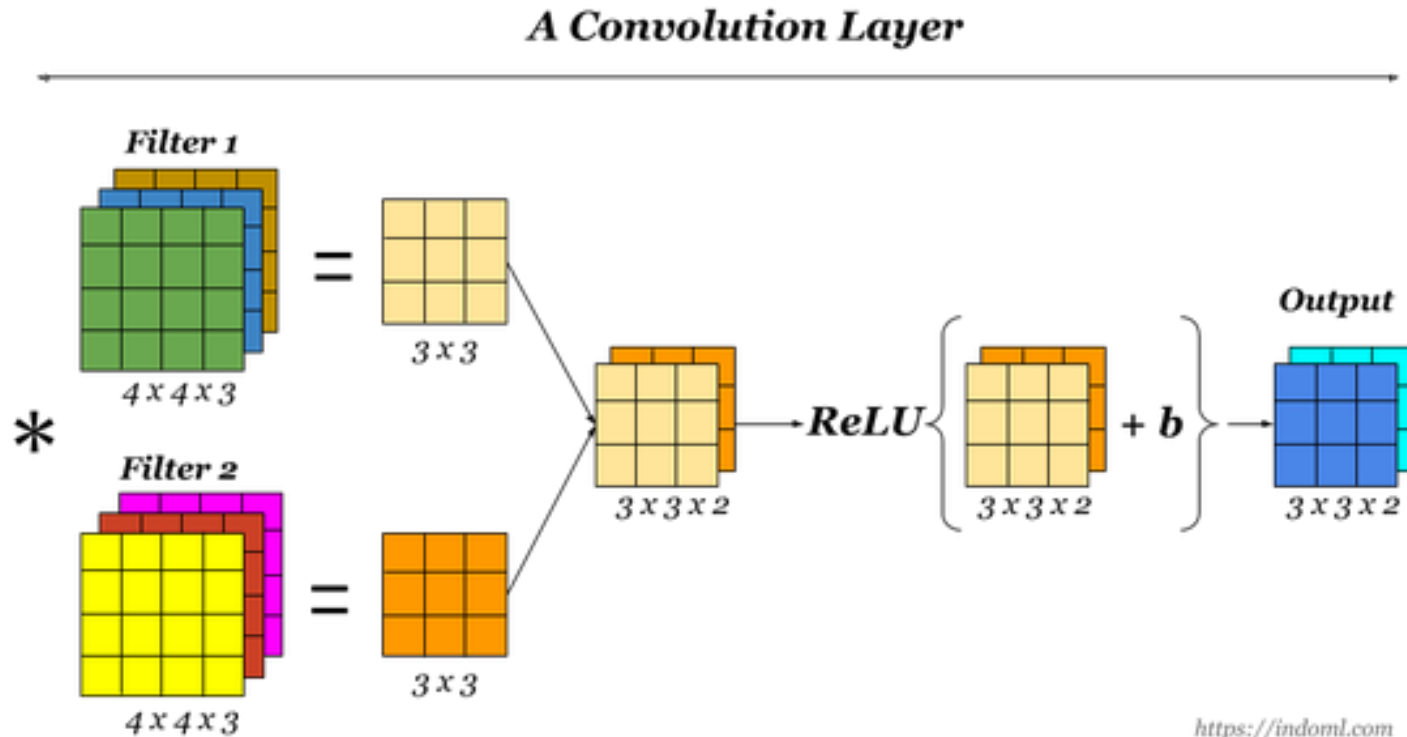




## 6.4 Convolution Neural Networks (CNN)

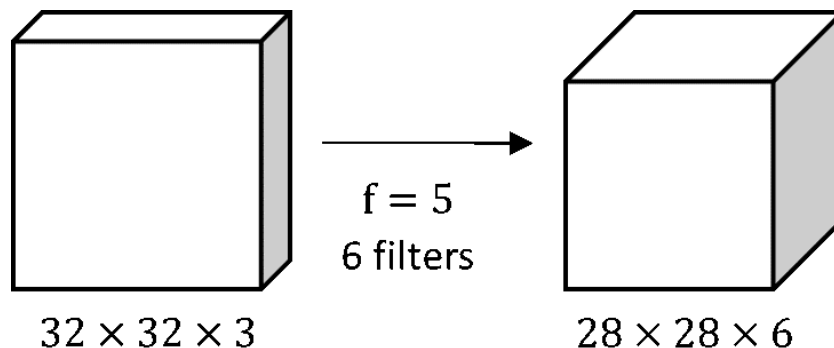
### Convolution Layer

- Để tạo nên một convolution layer cần thêm vào một ngưỡng kích hoạt bias  $b$ .
- Và cần có một activation function như ReLu hoặc Tanh.



## 6.4 Convolution Neural Networks (CNN)

### Why Convolutions?



$$5 \times 5 + 1 = 26$$

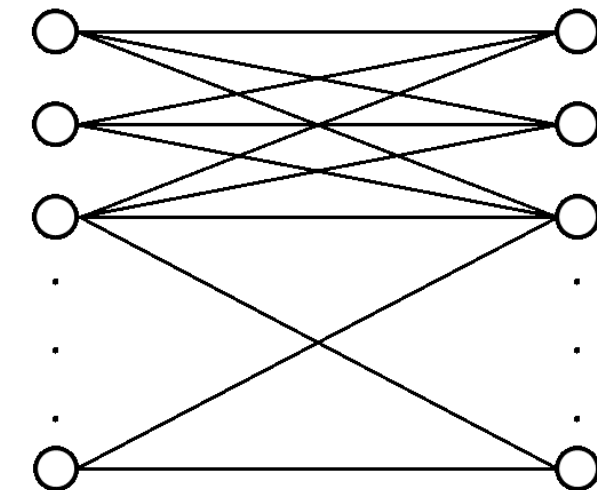
$$6 \times 26 = 156 \text{ parameters}$$



$32 \times 32 \times 3$

3,072

4,704



3,072

4,704

$$3072 \times 4704 \approx 14M$$

## 6.4 Convolution Neural Networks (CNN)

### Parameter Learning

- Mục tiêu của Convolution Layer này là tìm ra bộ tham số của Kernel dựa vào quá trình huấn luyện.

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

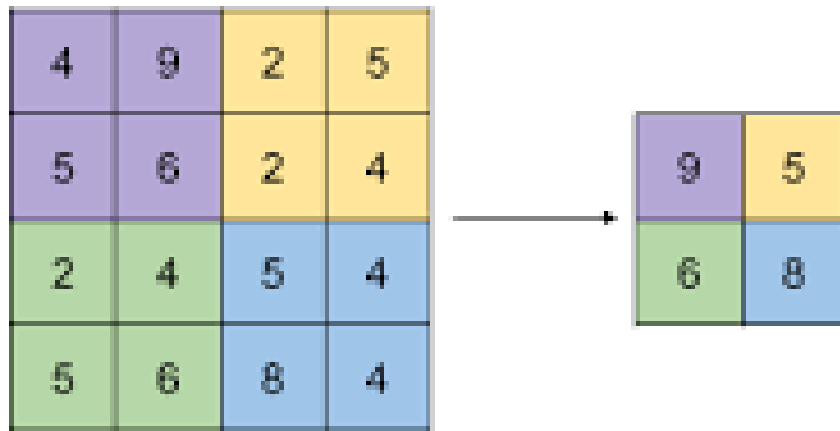
=


## 6.4 Convolution Neural Networks (CNN)

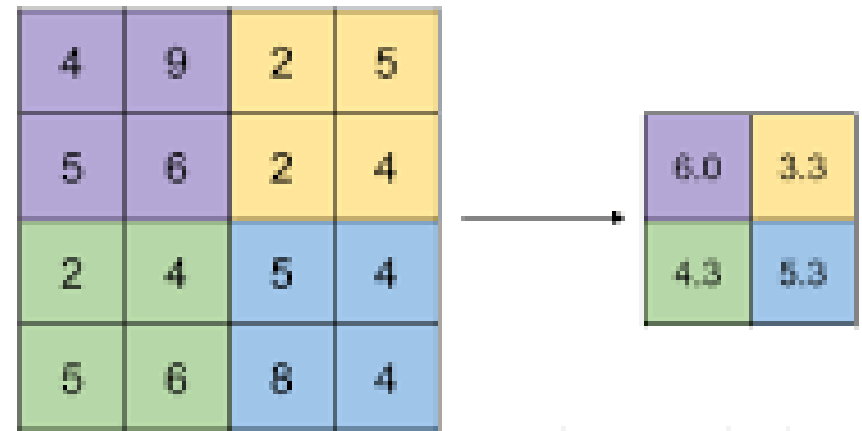
### Pooling Layer

- Pooling layer dùng để giảm kích thước ngõ vào, nhằm tăng tốc độ tính toán.
- Các kiểu pooling cơ bản là Max Pooling và Average Pooling.

*Max Pooling*



*Avg Pooling*

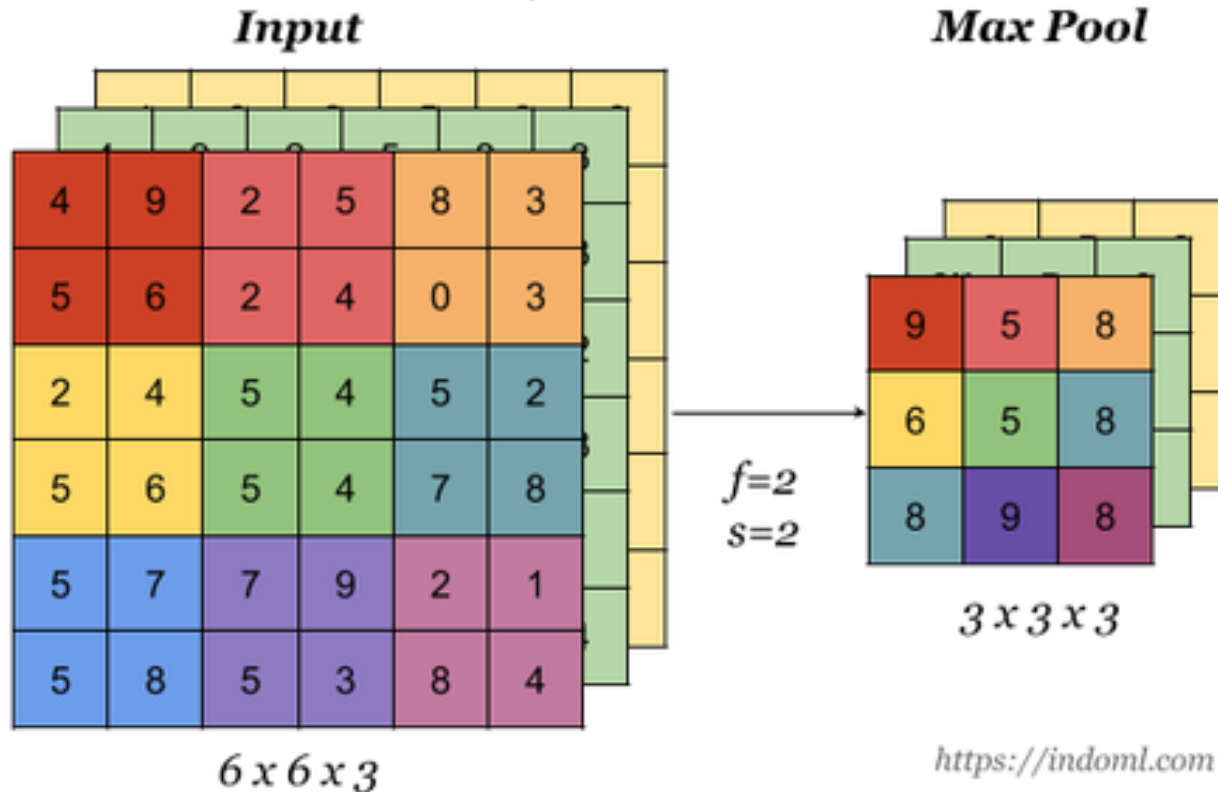


<https://indoml.com>

## 6.4 Convolution Neural Networks (CNN)

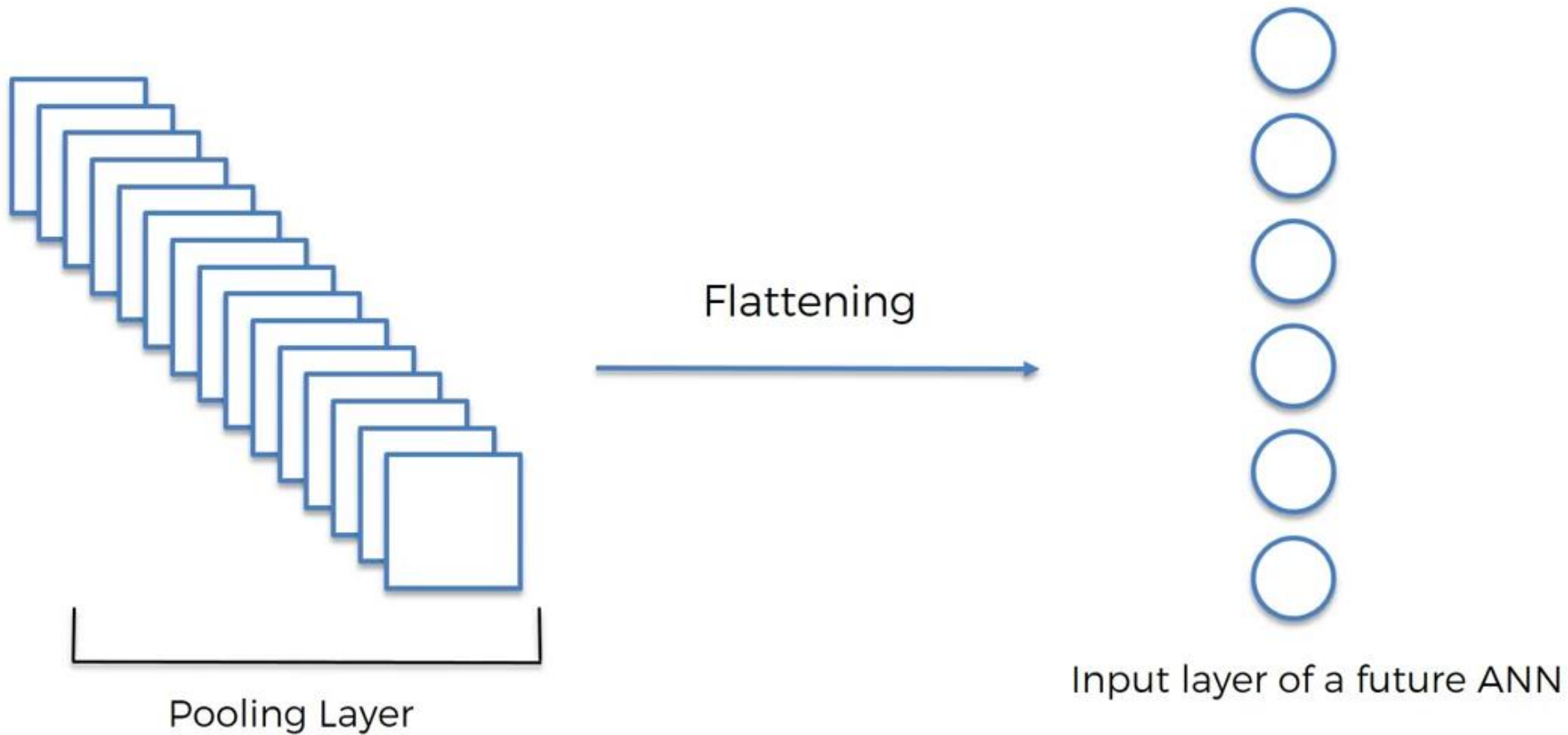
### Pooling Layer

- Khi đầu vào có nhiều kênh pooling thực hiện trên  $n_H$  và  $n_W$ ,  $n_C$  được giữ nguyên.
- Có các tham số là  $f$  và  $s$  nhưng *không có tham số để huấn luyện*.



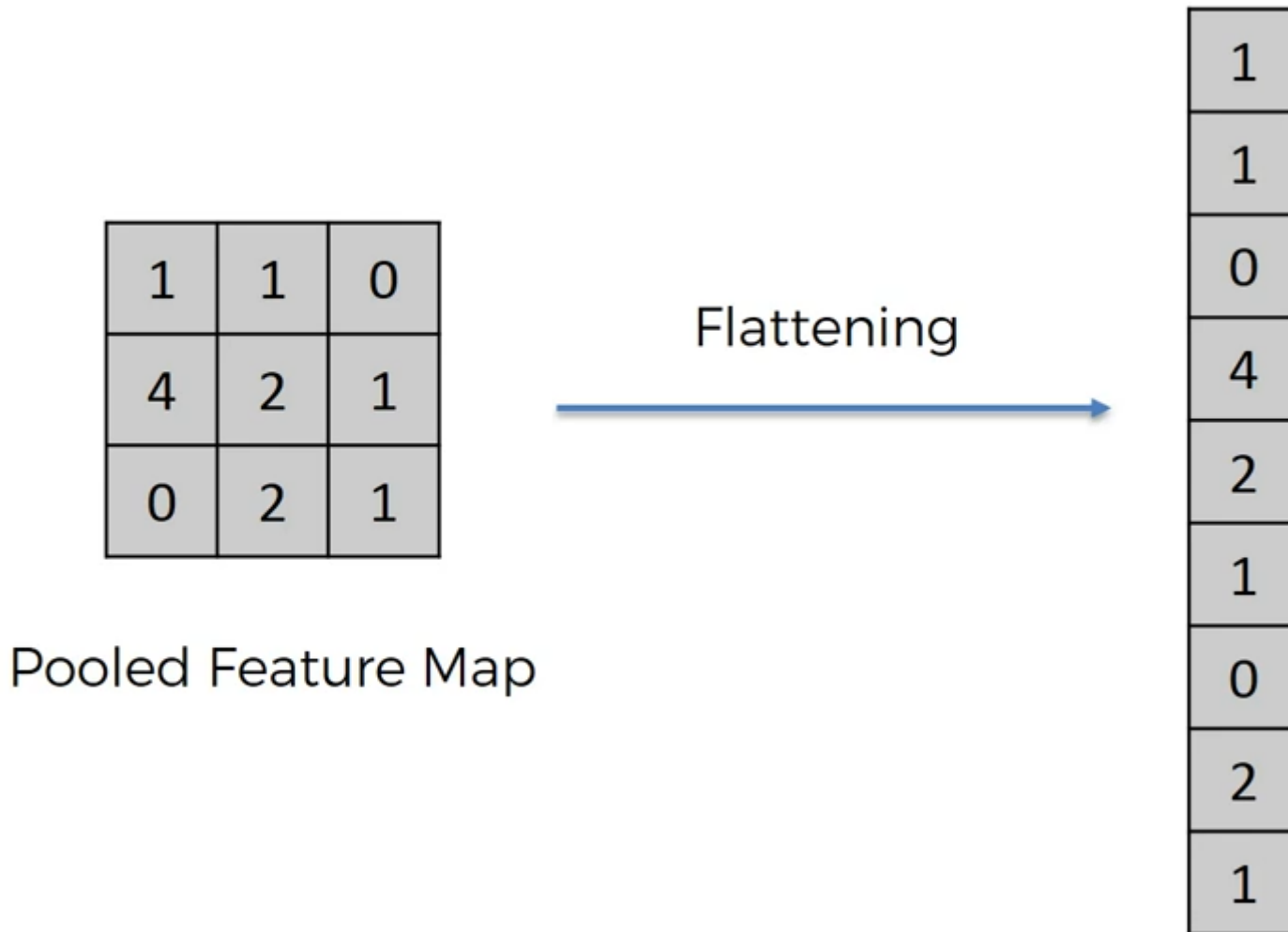
## 6.4 Convolution Neural Networks (CNN)

### Flattening



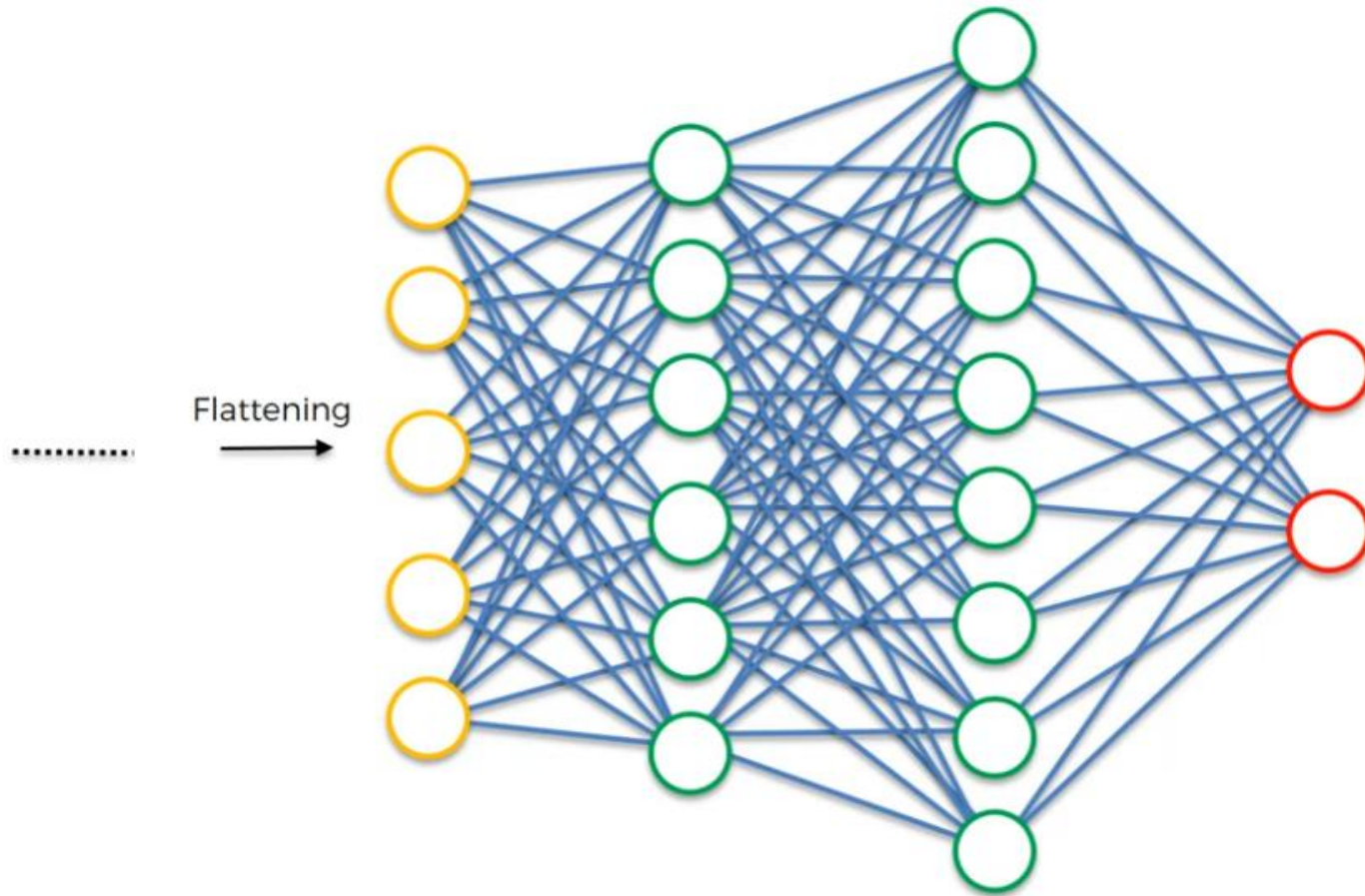
## 6.4 Convolution Neural Networks (CNN)

### Flattening



## 6.4 Convolution Neural Networks (CNN)

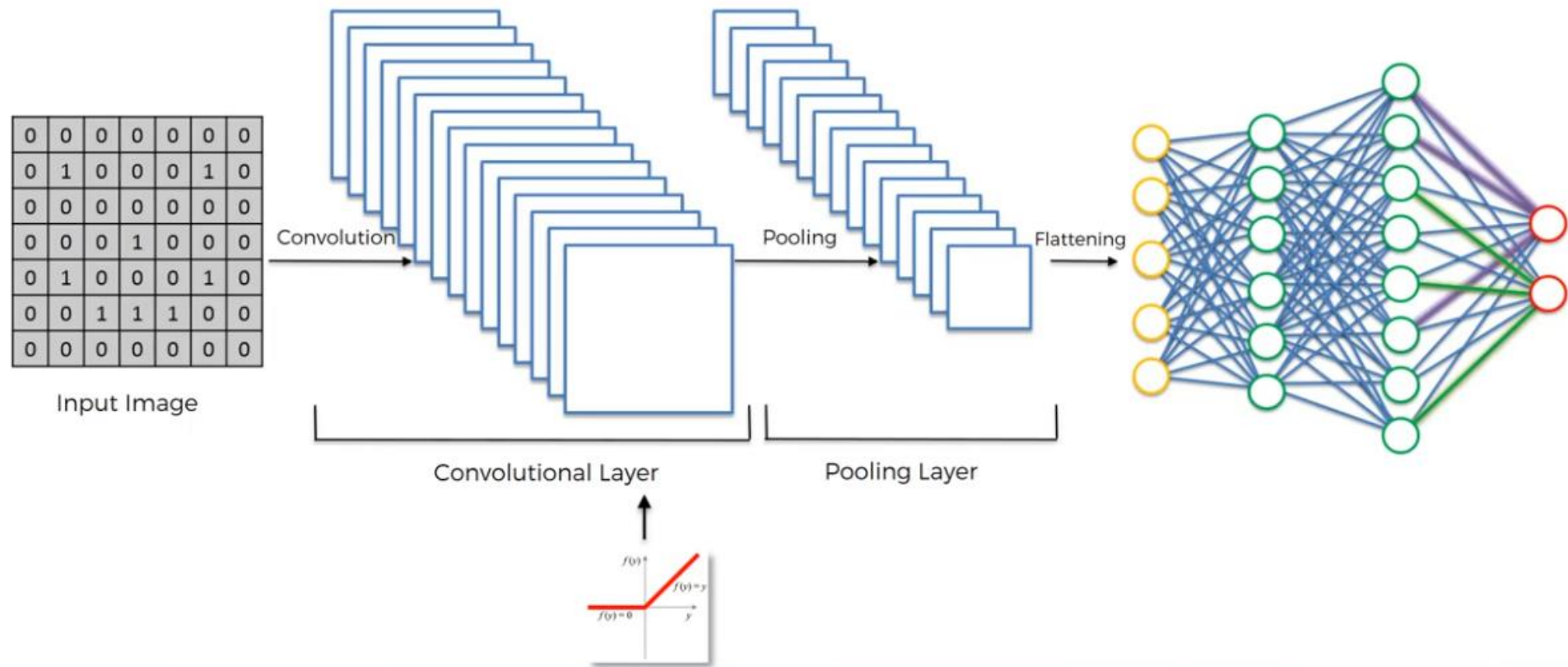
### Full Connection





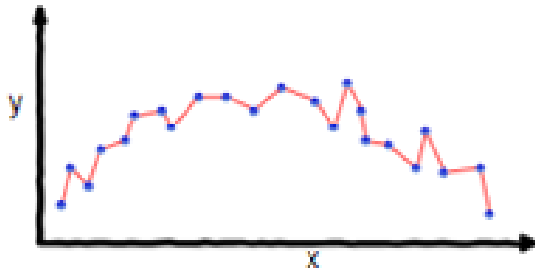
# 6.4 Convolution Neural Networks (CNN)

## Summary



## 6.3 Artificial Neural Networks (ANN)

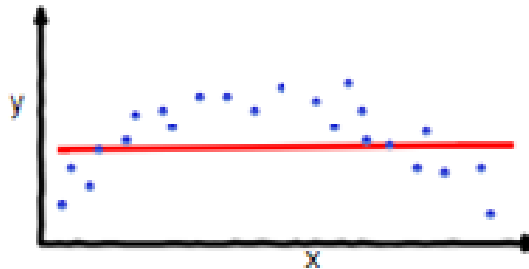
### Overfitting và Underfitting



overfitting

#### Overfitting

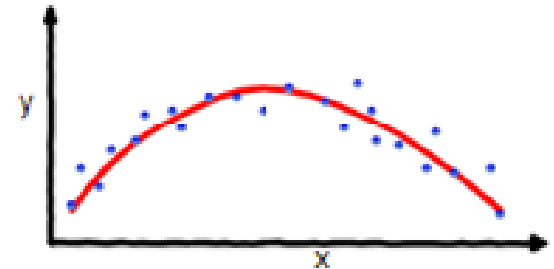
- Model học theo tất cả dữ liệu mẫu bao gồm cả nhiễu.
- Nguyên nhân:
  - Model được huấn luyện quá kỹ dẫn tới học theo tất cả dữ liệu.
  - Xây dựng mô hình quá phức tạp đối với bài toán đơn giản.



underfitting

#### Underfitting

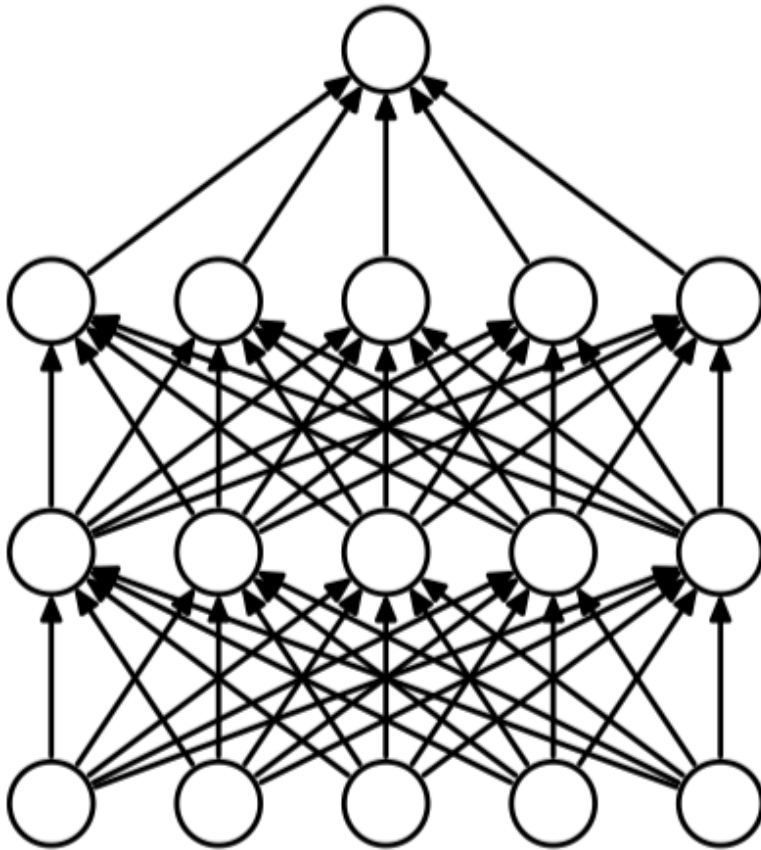
- Model không thể học được theo dữ liệu mẫu.
- Nguyên nhân:
  - Do data quá ít.
  - Xây dựng mô hình tuyến tính cho hệ phi tuyến.
  - Mô hình quá đơn giản cho bài toán phức tạp.



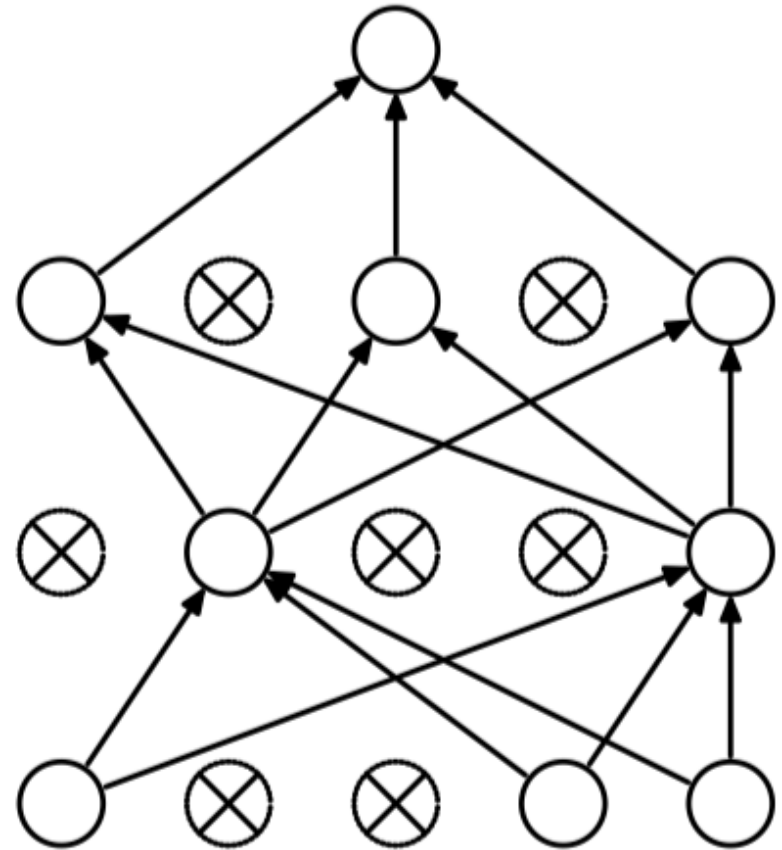
Good balance

## 6.3 Artificial Neural Networks (ANN)

### Drop Out



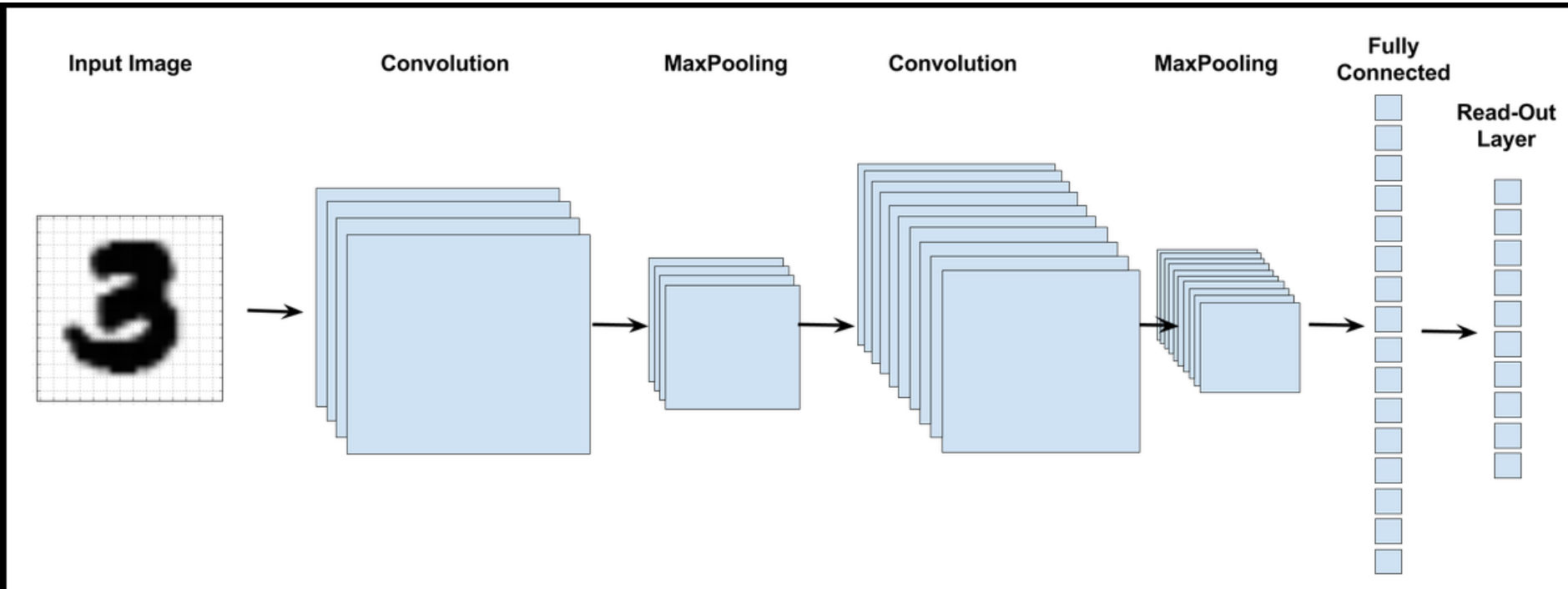
(a) Standard Neural Net



(b) After applying dropout.

## 6.4 Convolution Neural Networks (CNN)

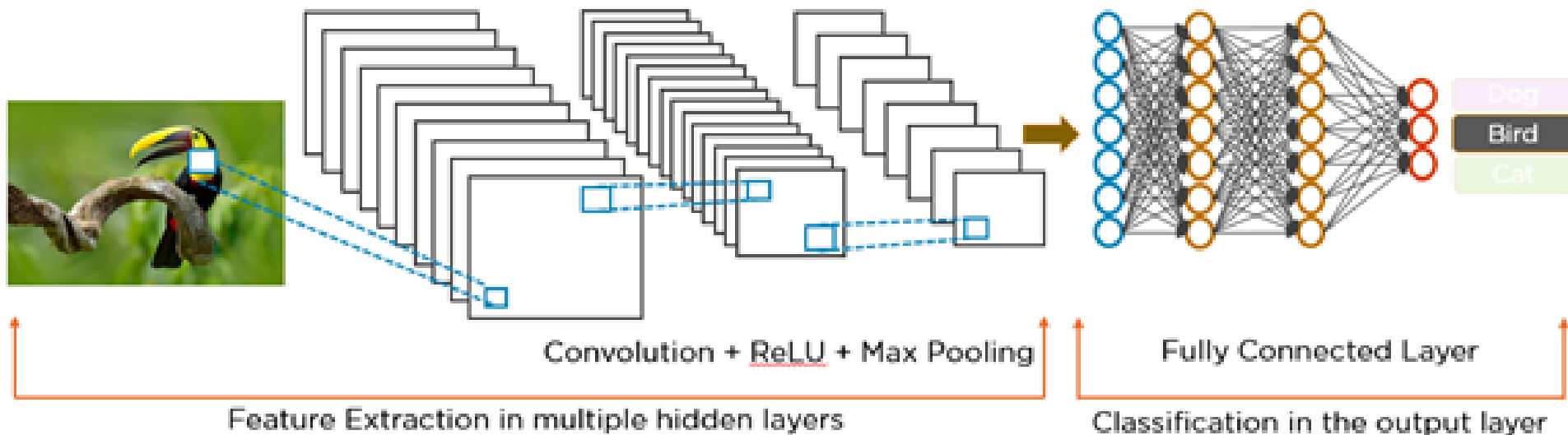
### Xây dựng mạng CNN dự đoán trên tập MNIST



<https://colab.research.google.com/drive/1Fe6fFamUufkggK1jdojpMRGSpzgOpvYk?usp=sharing>

## 6.4 Convolution Neural Networks (CNN)

### Xây dựng mạng CNN dự đoán trên tập CIFAR10



<https://colab.research.google.com/drive/1RKOEDDKR2tAjRpIXpBNEZXc4jWQm8mNx?usp=sharing>

# CASE STUDY 6: CLASSIFICATION VỚI CUSTOM DATA

**Mô tả:** Xây dựng mạng CNN dự đoán đối tượng thực tế trên Google Colab với dataset tự xây dựng và dự đoán mô hình trên máy tính local (Pycharm).

## **Yêu cầu:**

- Xây dựng tập dữ liệu huấn luyện từ hình ảnh của đối tượng thực tế.
- Xây dựng mô hình CNN để huấn luyện.
- Huấn luyện mô hình và lưu file trọng số h5.
- Tiến hành dự đoán trên máy local.

[https://colab.research.google.com/drive/1nVOTqNASHhW1BDzoYRIbSY3TsB6OS5e ?usp=sharing](https://colab.research.google.com/drive/1nVOTqNASHhW1BDzoYRIbSY3TsB6OS5e?usp=sharing)

**Thank you !!!**