

AI - FOUNDATION AND APPLICATION

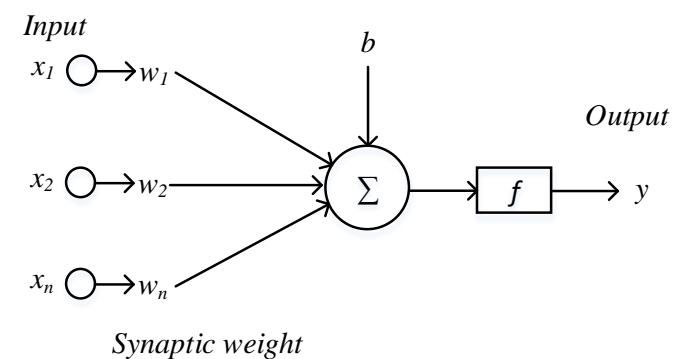
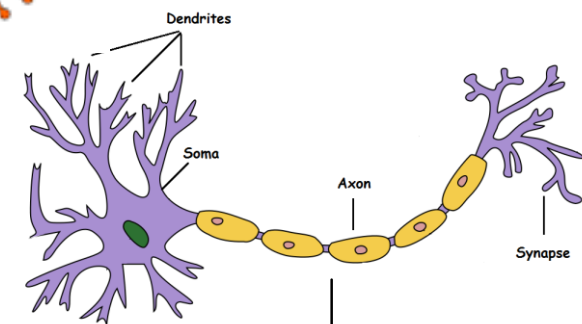
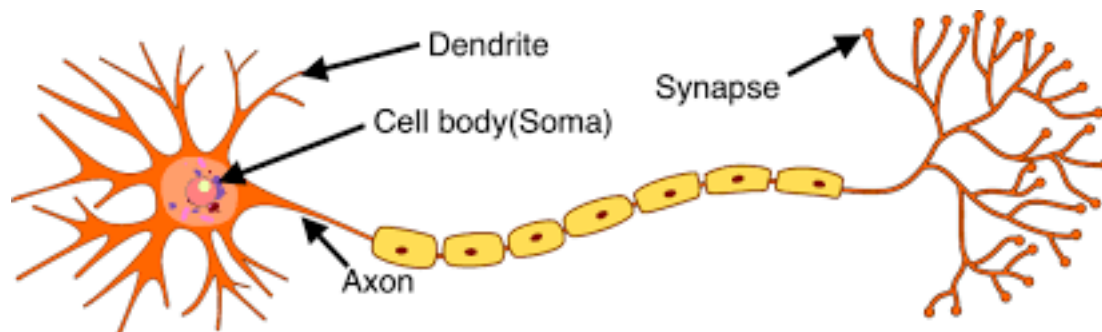
Instructor:

Assoc. Prof. Dr. Truong Ngoc Son

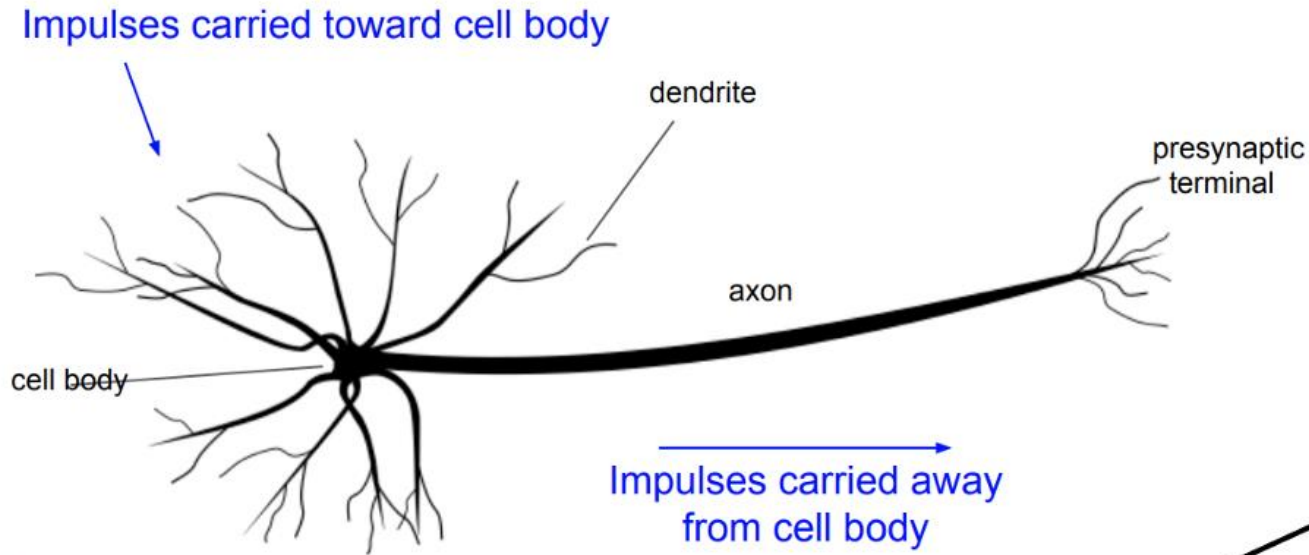
Chapter 1

Introduction of Neural network

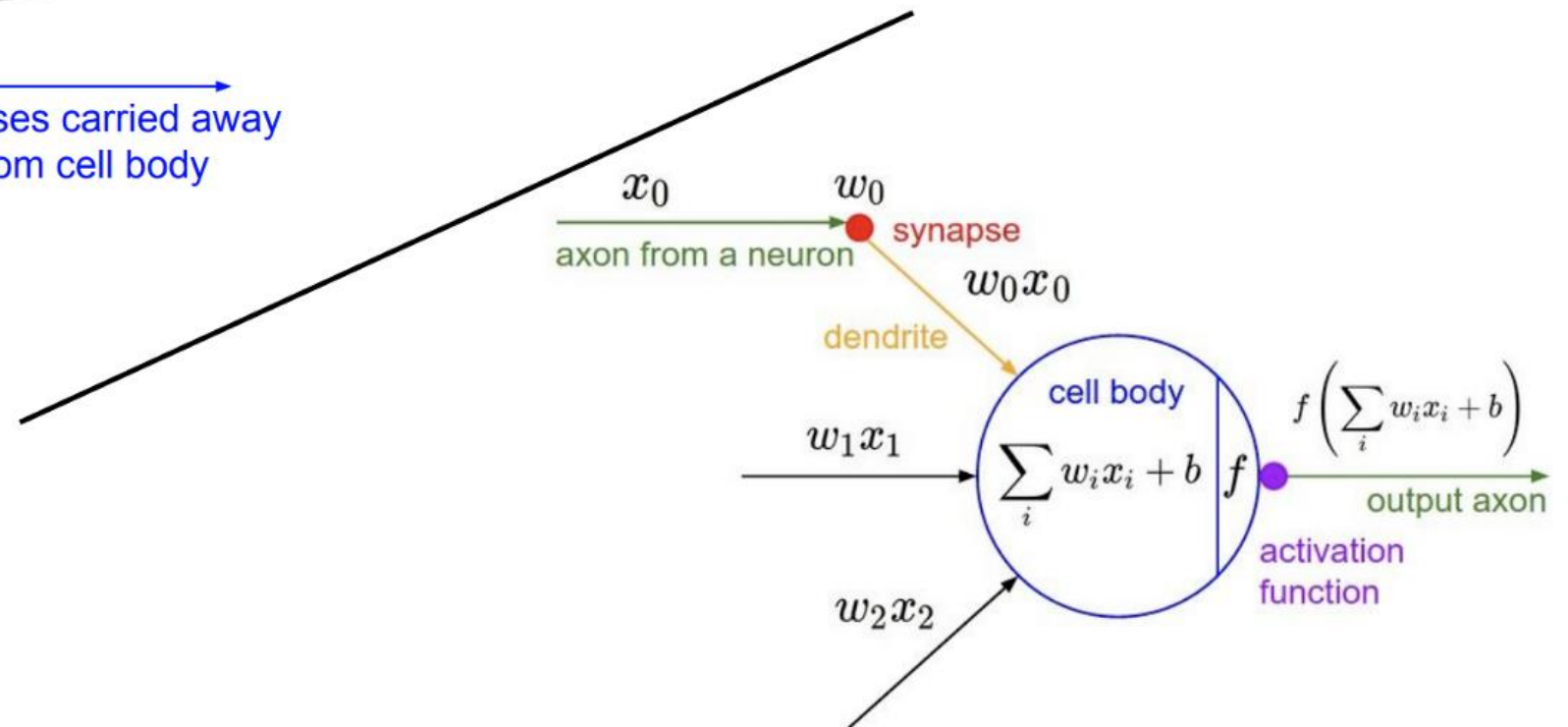
How a neuron is modelled?



How a neuron is modelled ?



This image by Felipe Perucho is licensed under [CC-BY 3.0](#)

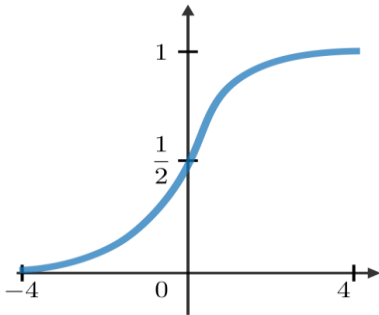


Training a network – Optimization method

Activation functions

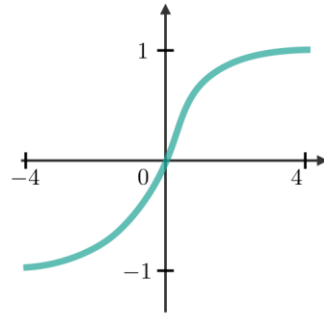
Sigmoid function

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



Tanh function

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

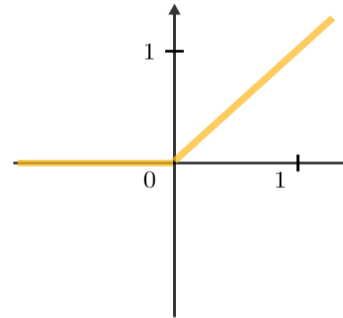


Rectified Linear Unit

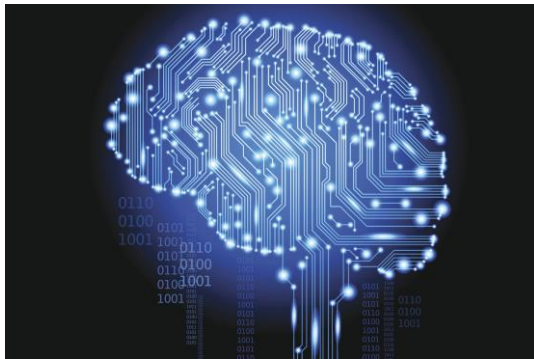
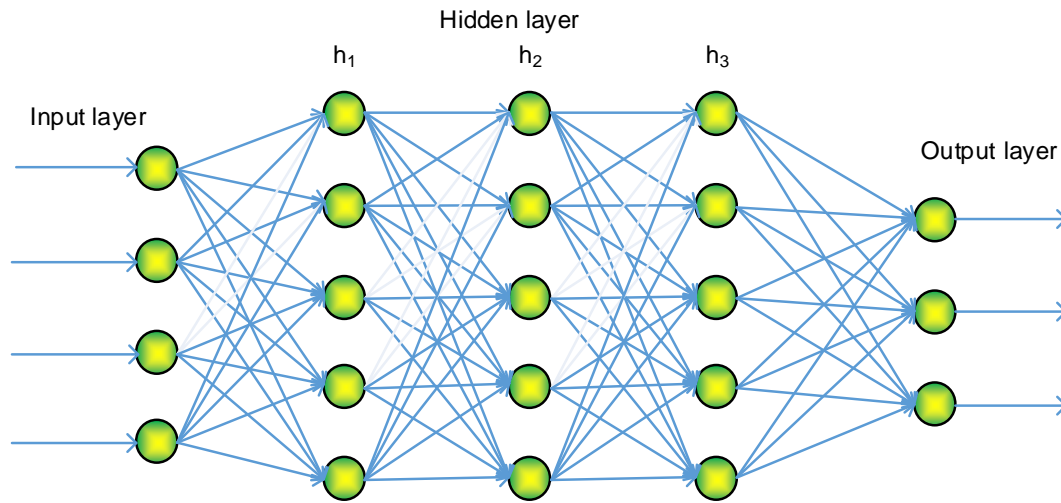
ReLU function

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

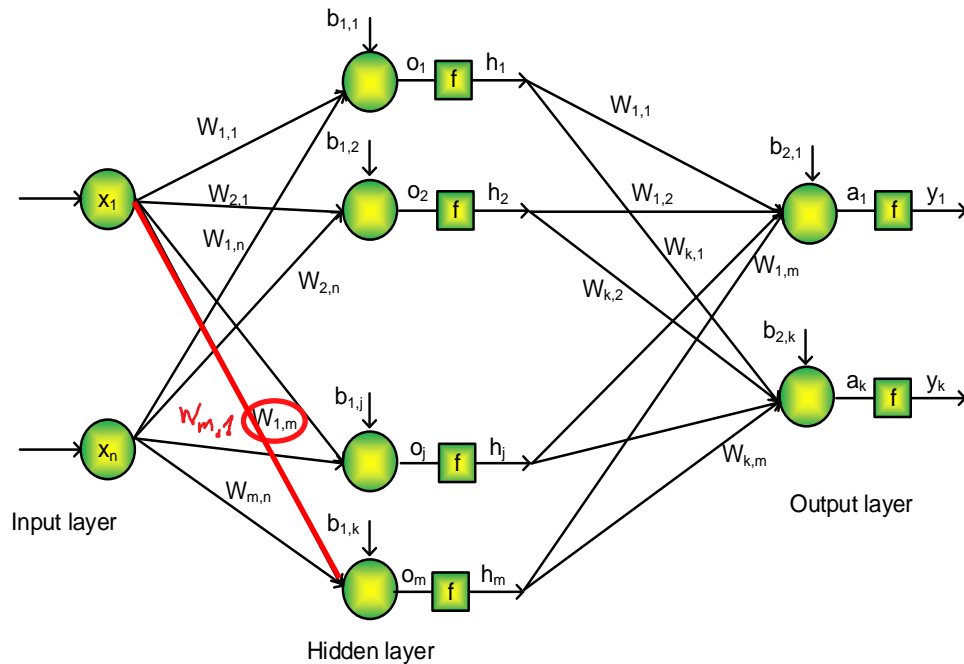
giá trị lớn nhất giữa x và 0



Neural network



Artificial neuron network



$$o_1 = x_1 W_{1,1} + \dots + x_n W_{1,n} + b_{1,1}$$

$$h_1 = f(o_1)$$

$$o_2 = x_1 W_{2,1} + \dots + x_n W_{2,n} + b_{1,2}$$

$$h_2 = f(o_2)$$

$$a_1 = h_1 W_{1,1} + \dots + h_m W_{1,m} + b_{2,1}$$

$$y_1 = f(a_1)$$

$$a_k = h_1 W_{k,1} + \dots + h_m W_{k,m} + b_{2,k}$$

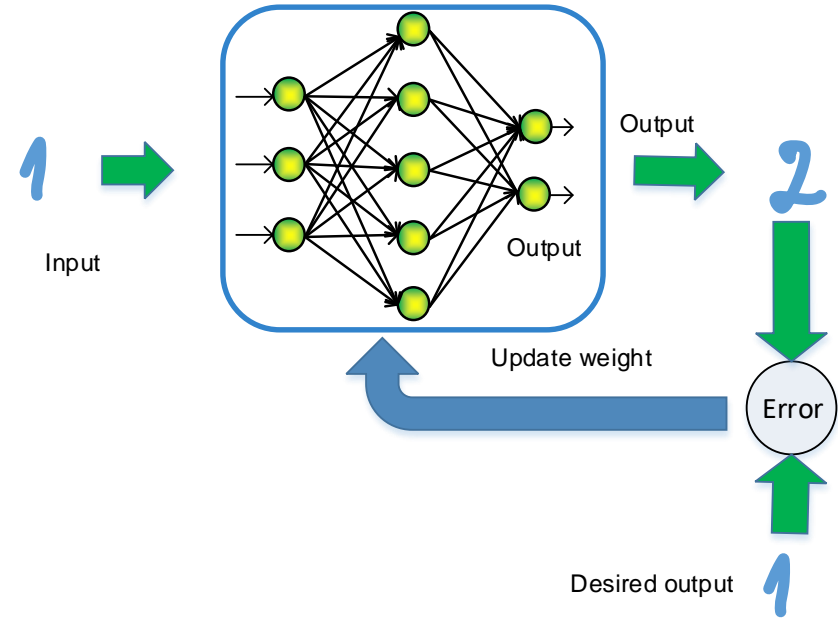
$$y_k = f(a_k)$$

TRAINING NEURAL NETWORK

Supervised learning vs. unsupervised learning

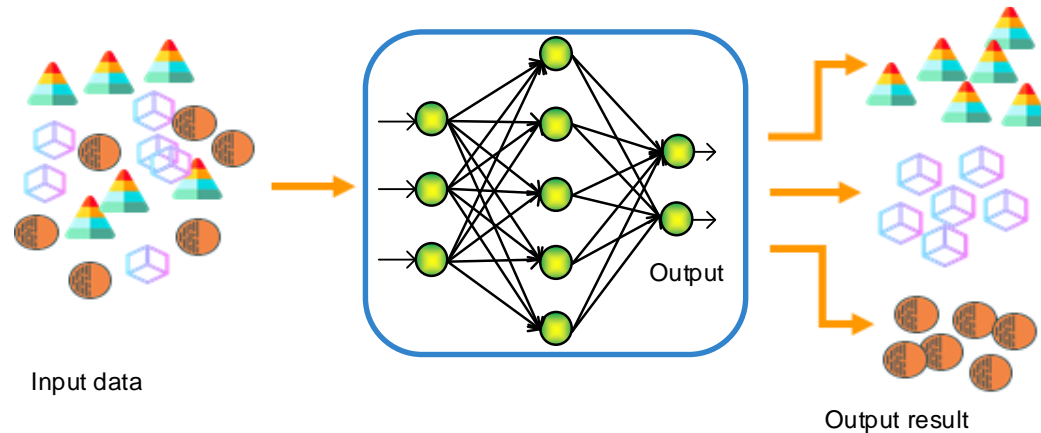
Training an artificial neural network

Supervised learning

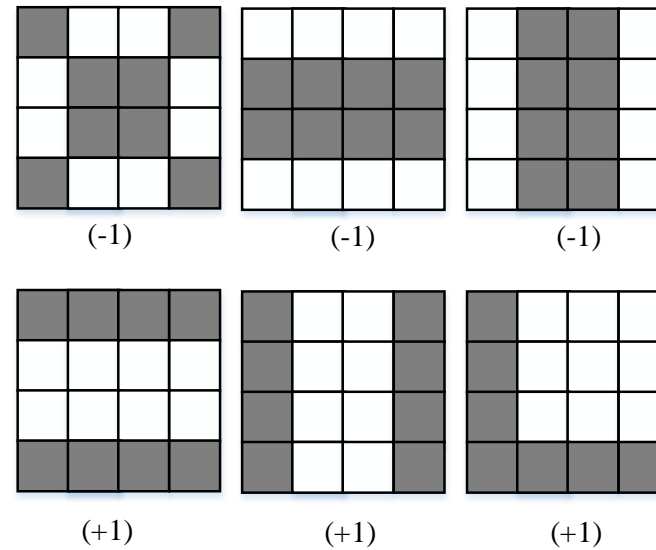
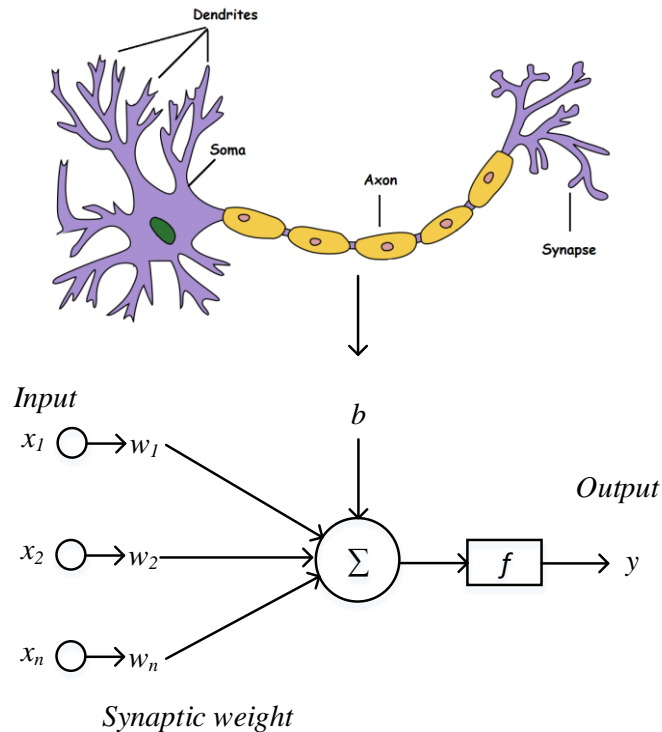


Training an artificial neural network

Unsupervised learning

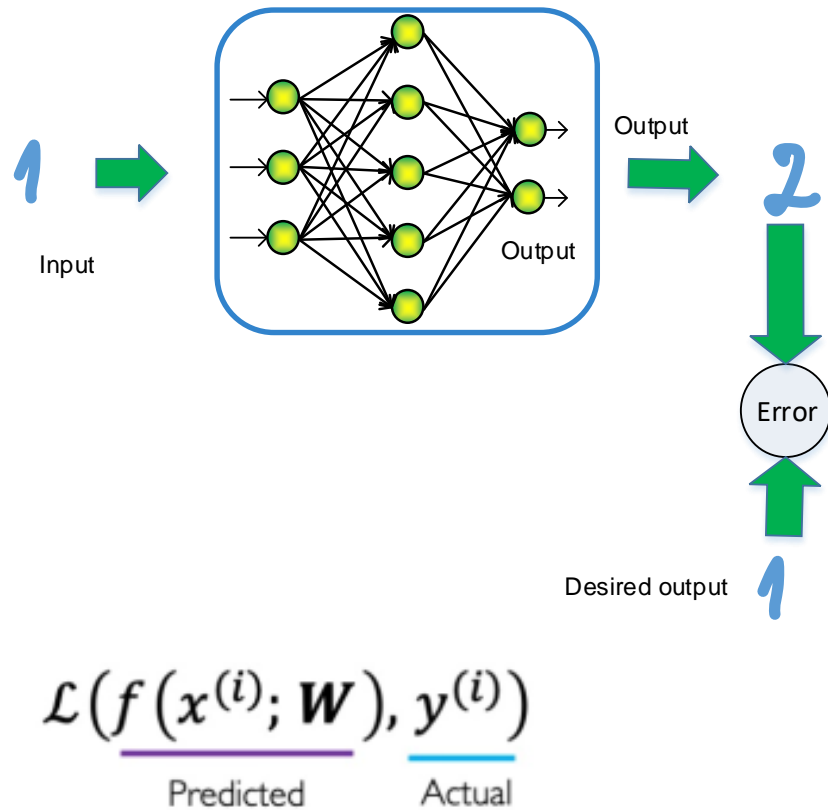


Simple neural network: Understanding of neuron network learning



Quantifying the loss

The loss of a network measure the cost incurred from incorrect prediction



MSE: mean squared error

$$MSE = \frac{1}{n} \sum \underbrace{\left(y - \hat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

Cross-entropy loss

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Training a network

Training a neural network is a process of **using an optimization algorithm** to find a set of weights to best map inputs to outputs.

In other word, this is the way to minimize the loss

$$W^* = \operatorname{argmin}_w \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^i, W), y^i)$$

So hard? Don't worry, we will dive into the detail later

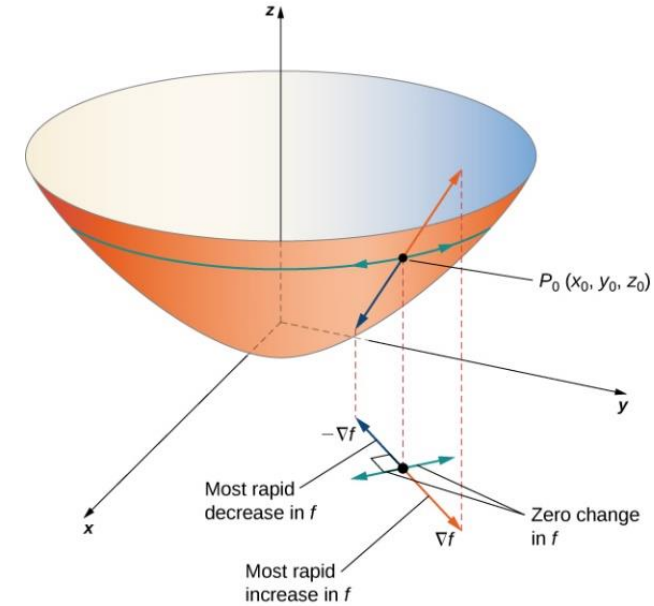
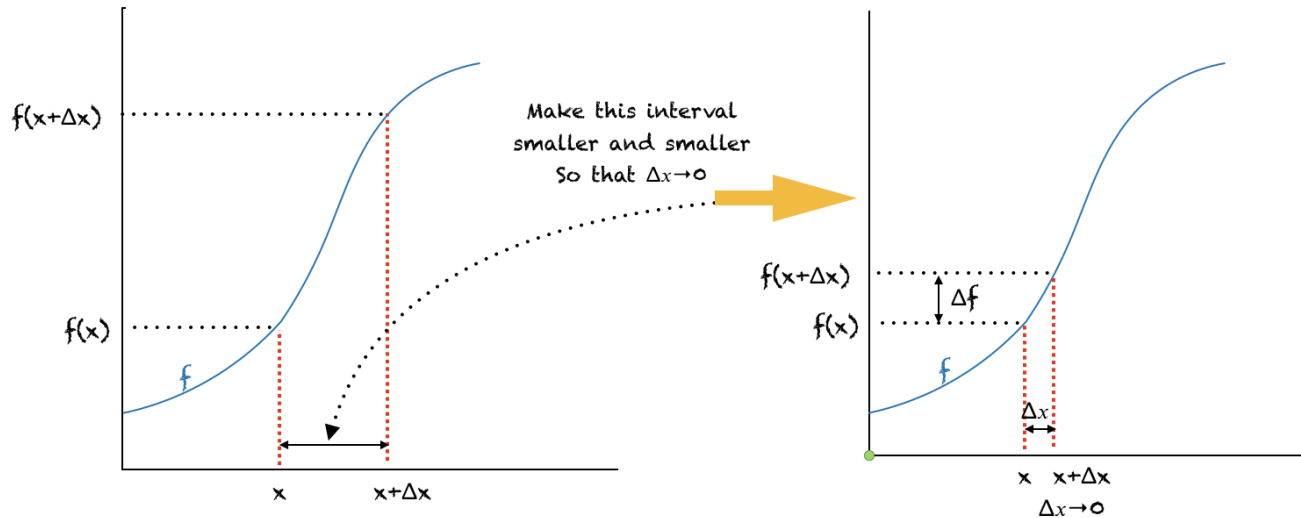
GRADIENT DESCENT

Training Neural Networks– Optimization of the loss

What is Gradient ?

Derivative of f is the rate of change of f

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{\Delta f}{\Delta x}$$



$f(x, y, z)$

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right]$$

Gradient of f

$$\nabla f = \left[\frac{\partial f}{\partial x} i + \frac{\partial f}{\partial y} j + \frac{\partial f}{\partial z} k \right]$$

f

$$[x, y, z] = \left[x - \frac{\partial f}{\partial x}, y - \frac{\partial f}{\partial y}, z - \frac{\partial f}{\partial z} \right]$$

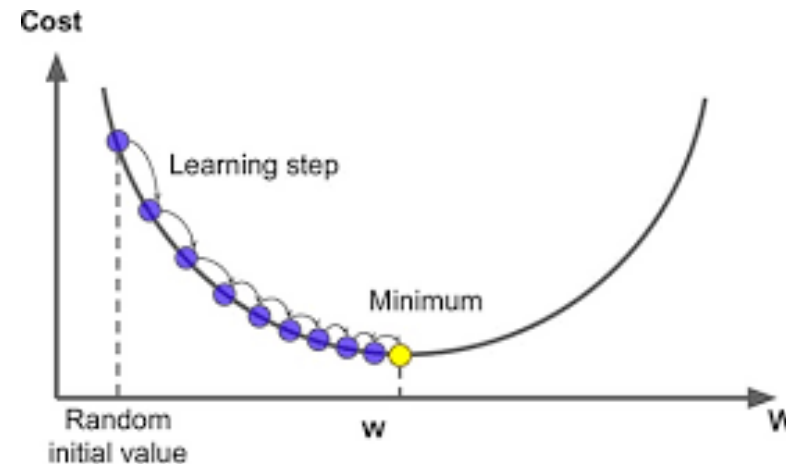
Training a network – Optimization of the loss

Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (loss). This can be done by iteratively moving in the direction of steepest descent as defined by the negative of the gradient

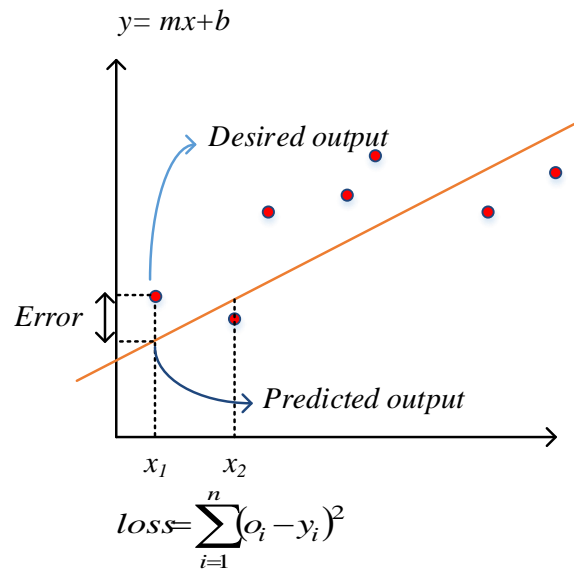
$$\vec{x}_0 = (x_0, y_0, z_0)$$
$$\vec{x}_{n+1} = \vec{x}_n - \eta \nabla f(\vec{x}_n)$$

$$[x, y, z] = \left[x - \eta \frac{\partial f}{\partial x}, y - \eta \frac{\partial f}{\partial y}, z - \eta \frac{\partial f}{\partial z} \right]$$



Optimization of the loss with gradient descent

Example: Linear Regression



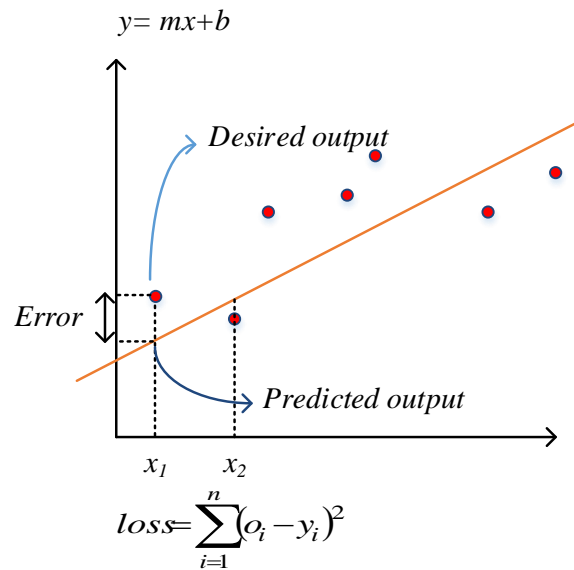
$$m = m + \Delta m$$

$$b = b + \Delta b$$

(loss) ↘

Optimization of the loss with gradient descent

Assignment 01: Logistic Regression

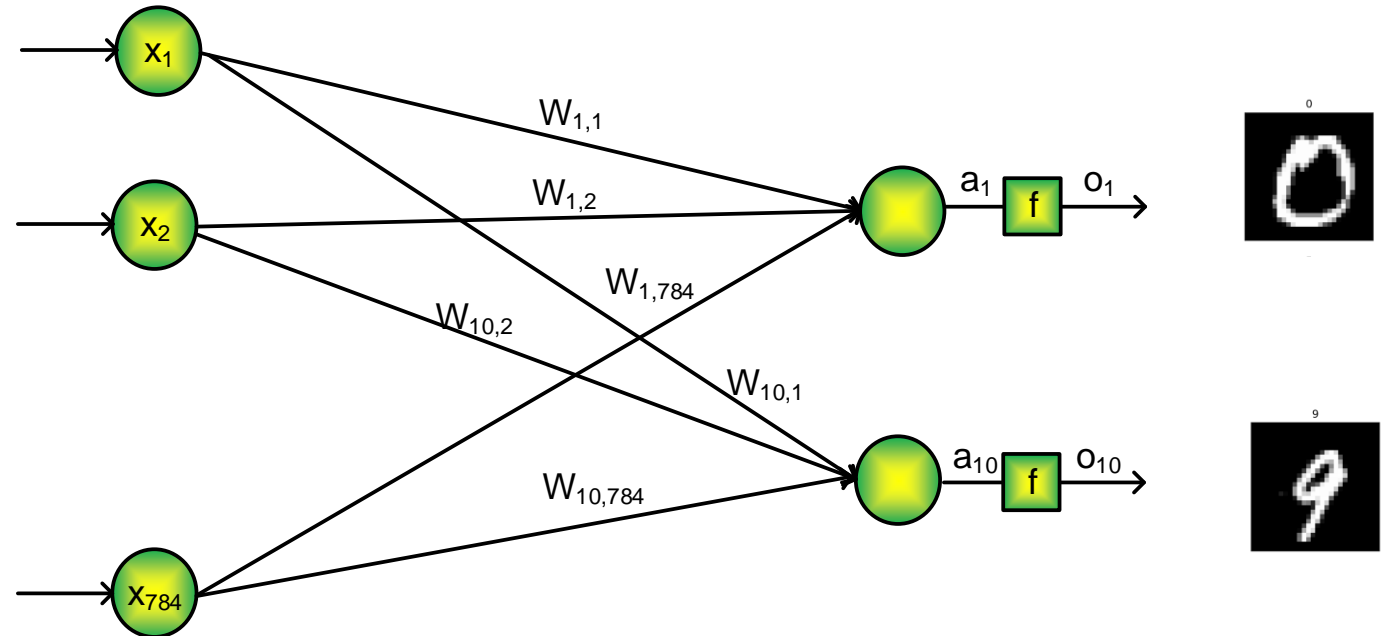
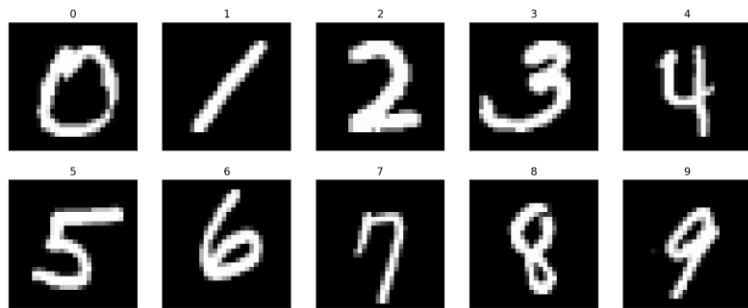


$$m = m + \Delta m$$

$$b = b + \Delta b$$

(loss) ↘

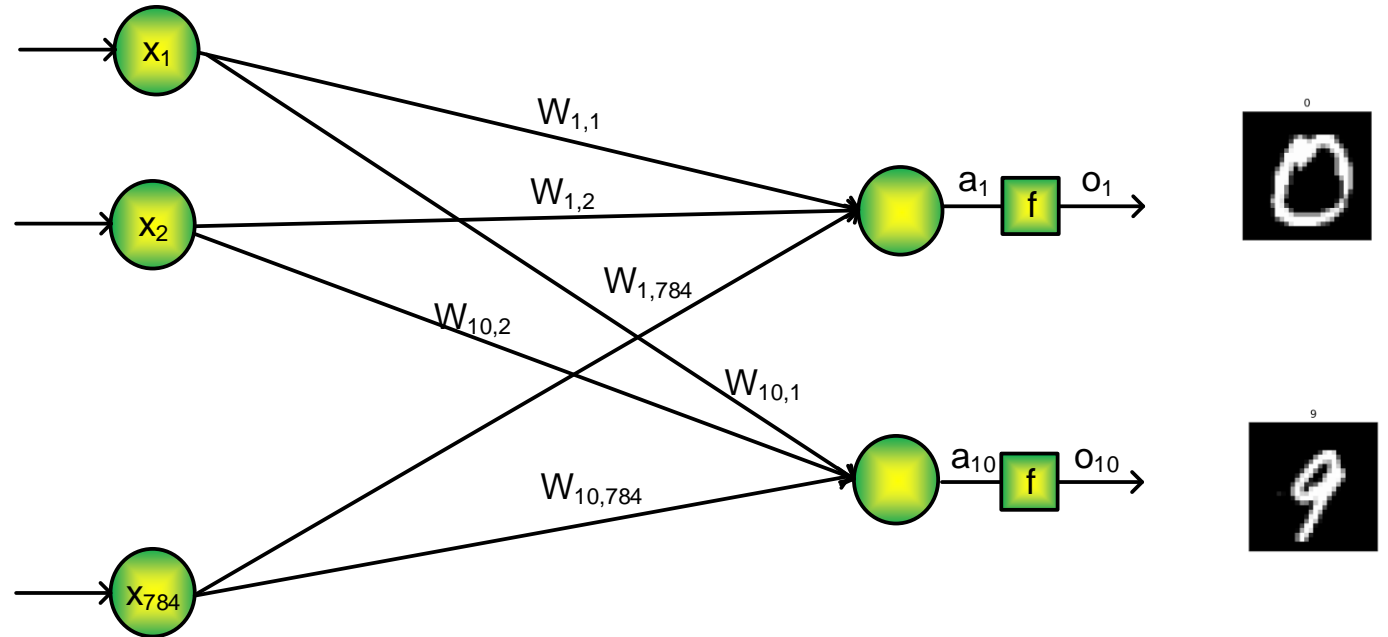
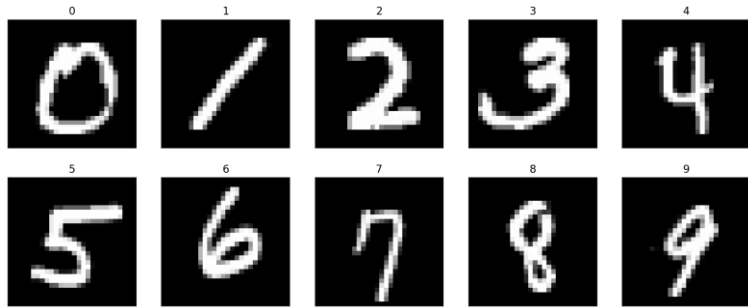
Training Neural networks – Optimization method



LOSS OPTIMIZATION WITH GRADIENT DESCENT

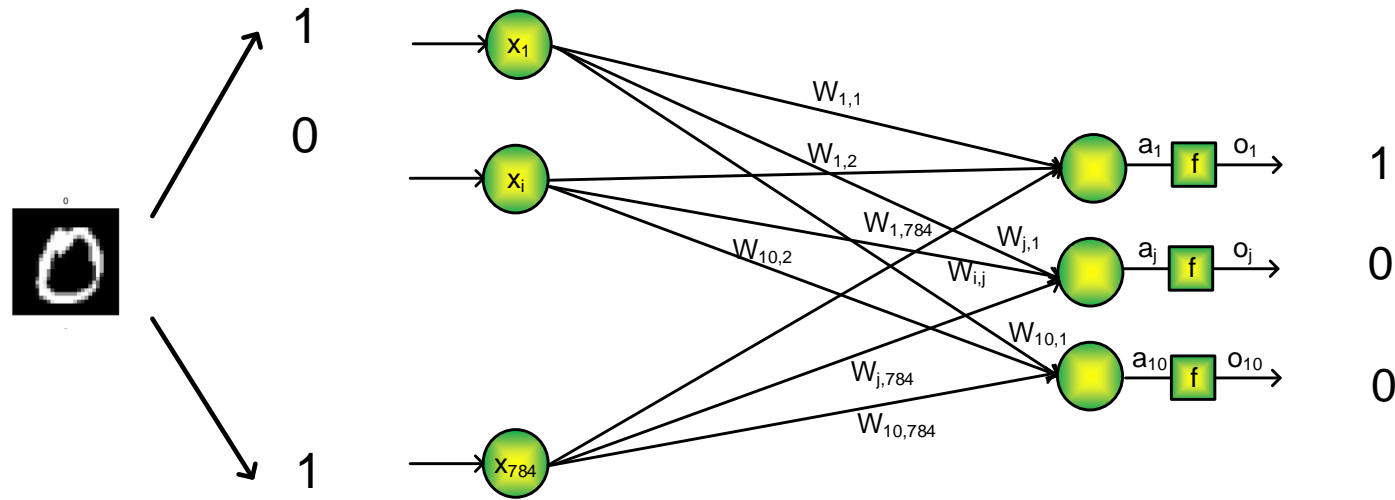
Mathematical modeling of Training Process

Example



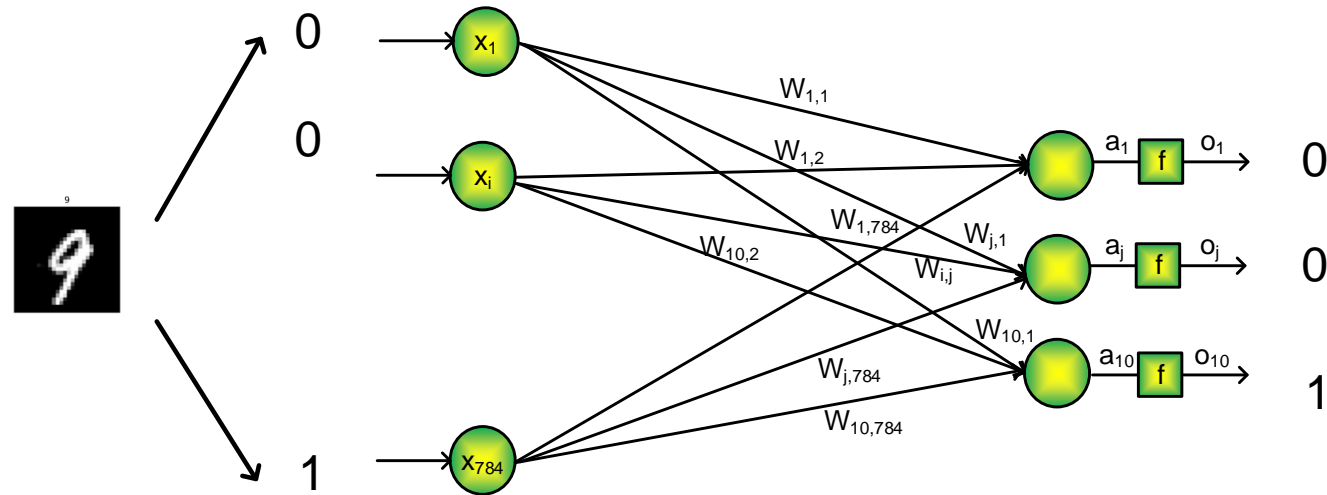
Mathematical modeling of Training Process

Desired outputs, labels



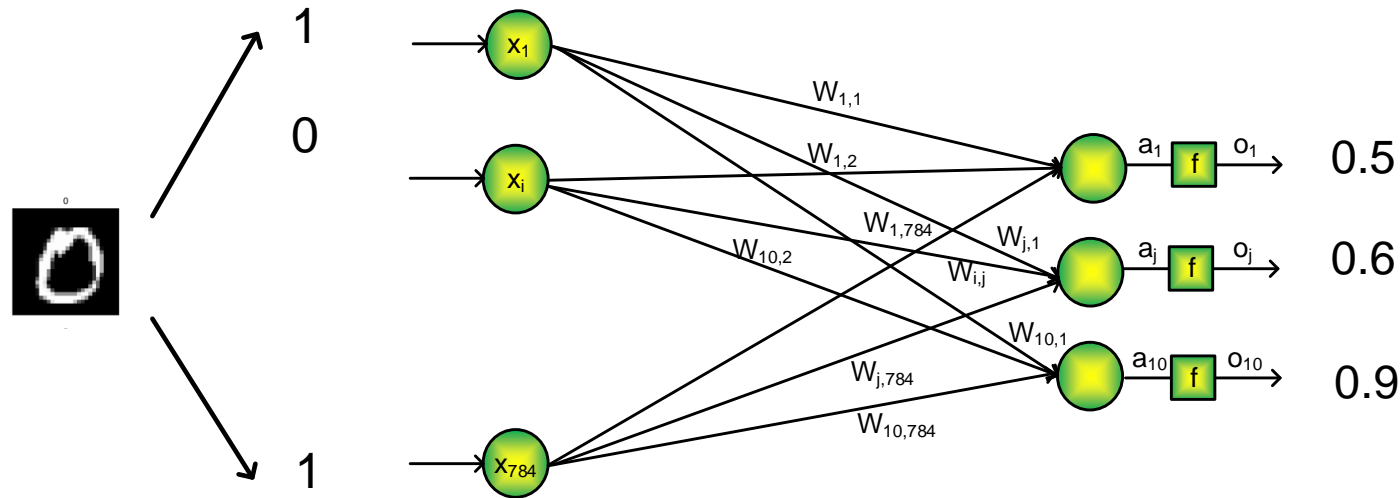
Mathematical modeling of Training Process

Desired outputs, labels



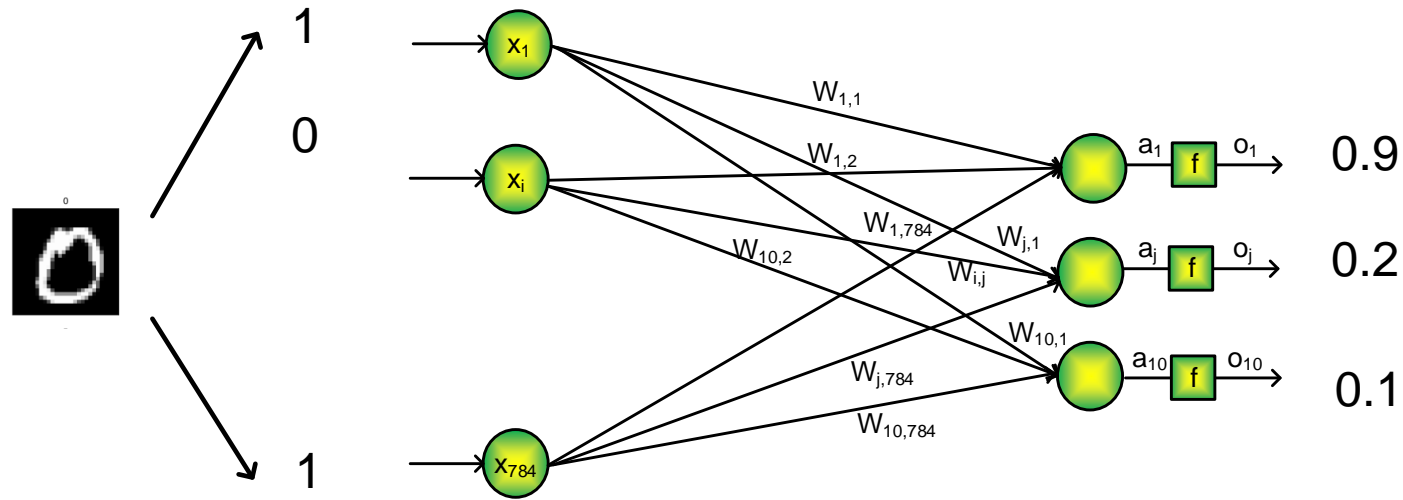
Mathematical modeling of Training Process

*Randomly initialize
Weights, W*



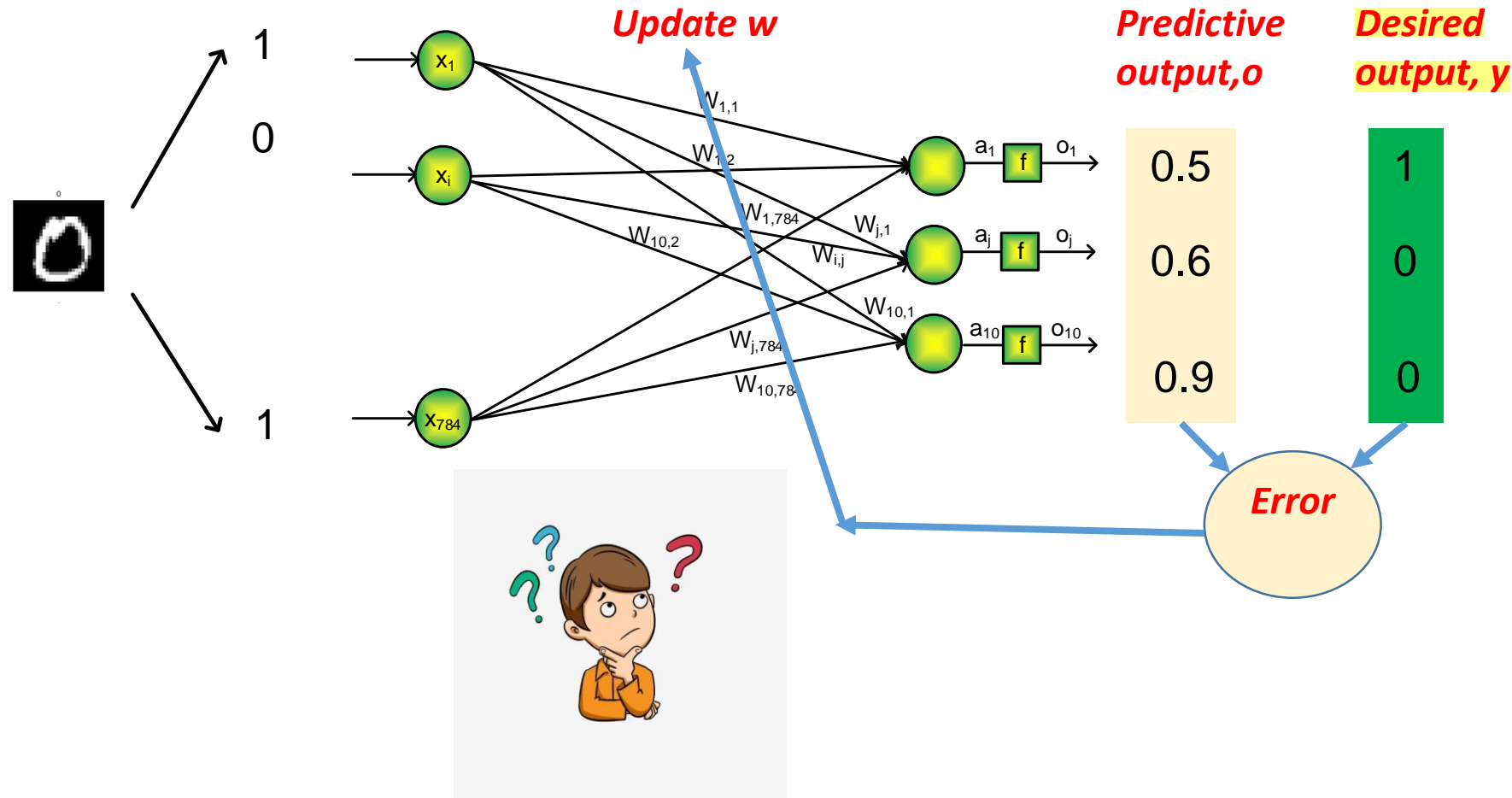
Mathematical modeling of Training Process

$$W = \text{ArgMin} (\text{Loss})$$

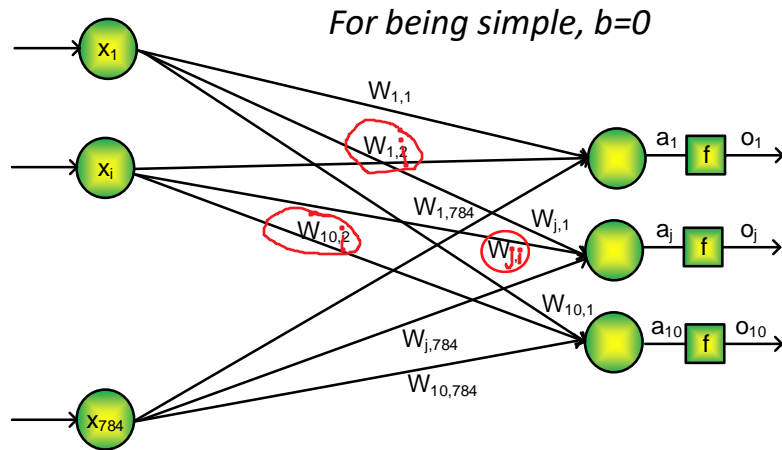


Mathematical modeling of Training Process

Training process



Mathematical modeling of Training Process



Formulate the output

$$a_j = \sum_{i=1}^{784} x_i w_{j,i}$$

$$o_j = \sigma(a_j) = \frac{1}{1 + e^{-a_j}}$$

For j^{th} output

For t^{th} sample

loss

$$L = \frac{1}{10} \sum_{t=1}^{10} \left[\sum_{j=1}^{10} (y_j^t - o_j^t)^2 \right]$$

For j^{th} output

$$L = \frac{1}{10} \sum_{t=1}^{10} (y_j^t - o_j^t)^2$$

For j^{th} output

$$a_j = \sum_{i=1}^{784} x_i w_{j,i}$$

$$o_j = \sigma(a_j) = \frac{1}{1 + e^{-a_j}}$$

Gradient descent

$$w_{j,i} \leftarrow w_{j,i} - \eta \frac{\partial L}{\partial w_{j,i}}$$

chain rule

$$\frac{\partial L}{\partial w_{j,i}} = -\frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) \left(\frac{\partial o_j^t}{\partial w_{j,i}} \right)$$

$$\frac{\partial L}{\partial w_{j,i}} = -\frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) \left(\frac{\partial o_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,i}} \right)$$

$$o_j^t = \sigma(a_j^t) = \frac{1}{1 + e^{-a_j^t}} \quad \frac{\partial o_j^t}{\partial a_j^t} = o_j^t (1 - o_j^t)$$

$$\frac{\partial L}{\partial w_{j,i}} = -\frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) o_j^t (1 - o_j^t) x_i^t$$

+ -

$$w_{j,i} = w_{j,i} + \Delta w_{j,i}$$

$$\Delta w_{j,i} = \eta \frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) o_j^t (1 - o_j^t) x_i^t$$

PYTHON CODE

Translating mathematics into code



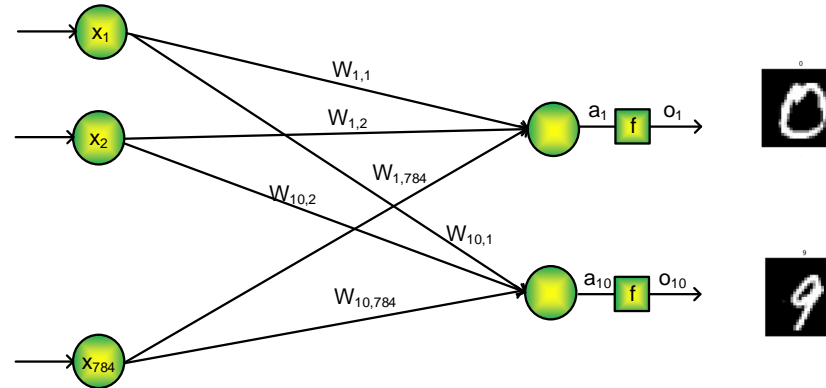
Translating mathematics into code



MNIST Dataset



60,000 training samples
10,000 testing samples



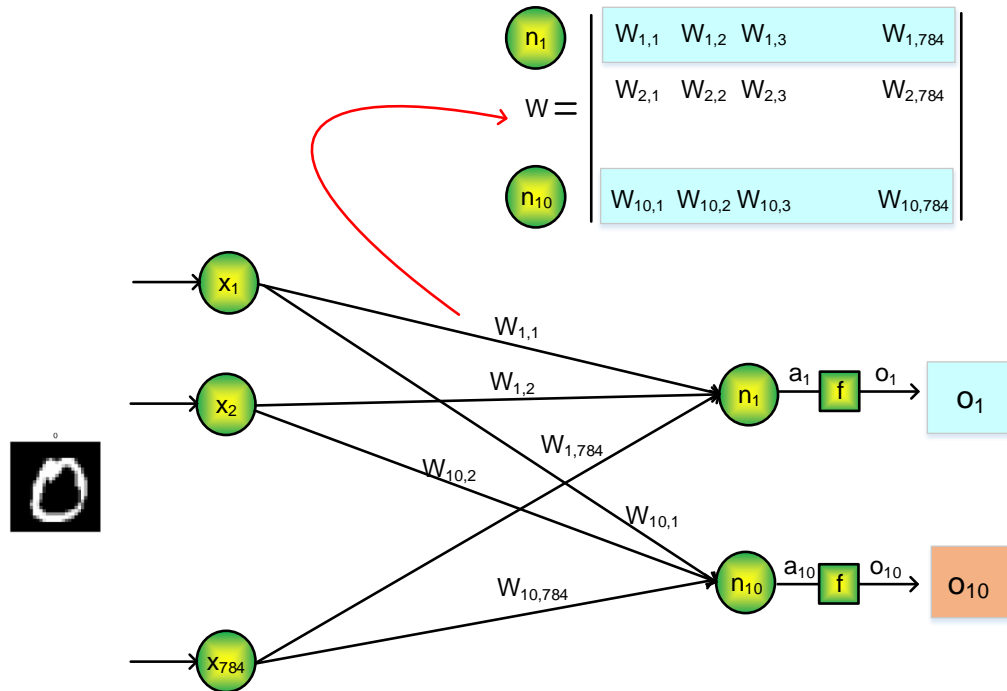
Gradient descent

$$\frac{\partial L}{\partial w_{j,i}} = -\frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) o_j^t (1 - o_j^t) x_i^t$$

$$w_{j,i} = w_{j,i} + \Delta w_{j,i}$$

$$\Delta w_{j,i} = \eta \frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) o_j^t (1 - o_j^t) x_i^t$$

Neuron's output



$$a_j = \sum_{i=1}^{784} x_i w_{j,i}$$

$$o_j = \sigma(a_j) = \frac{1}{1 + e^{-a_j}}$$

Matrix representation of the layer calculation:

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_{784} \end{bmatrix} \quad (1 \times 784)$$

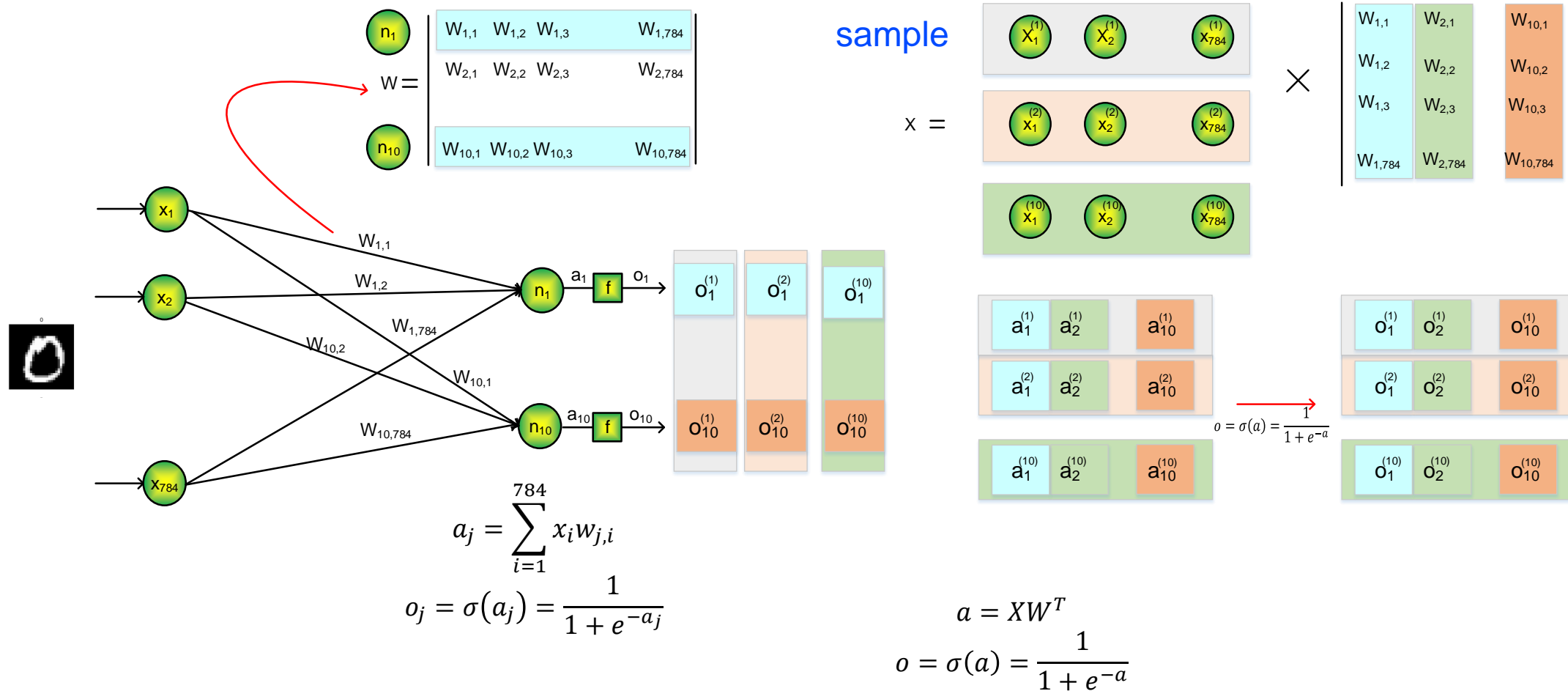
$$W^T = \begin{bmatrix} \begin{matrix} W_{1,1} & W_{1,2} & \dots & W_{1,784} \\ W_{2,1} & W_{2,2} & \dots & W_{2,784} \\ \vdots & \vdots & \ddots & \vdots \\ W_{10,1} & W_{10,2} & \dots & W_{10,784} \end{matrix} \end{bmatrix} \quad (784 \times 10)$$

$$a = XW^T$$

Resulting activation values: a_1, a_2, \dots, a_{10} (1 × 10)



Neuron's output - batch of neurons



Gradient calculating

$$\frac{\partial L}{\partial w_{j,i}} = -\frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) o_j^t (1 - o_j^t) x_i^t$$

$$w_{j,i} = w_{j,i} + \Delta w_{j,i}$$

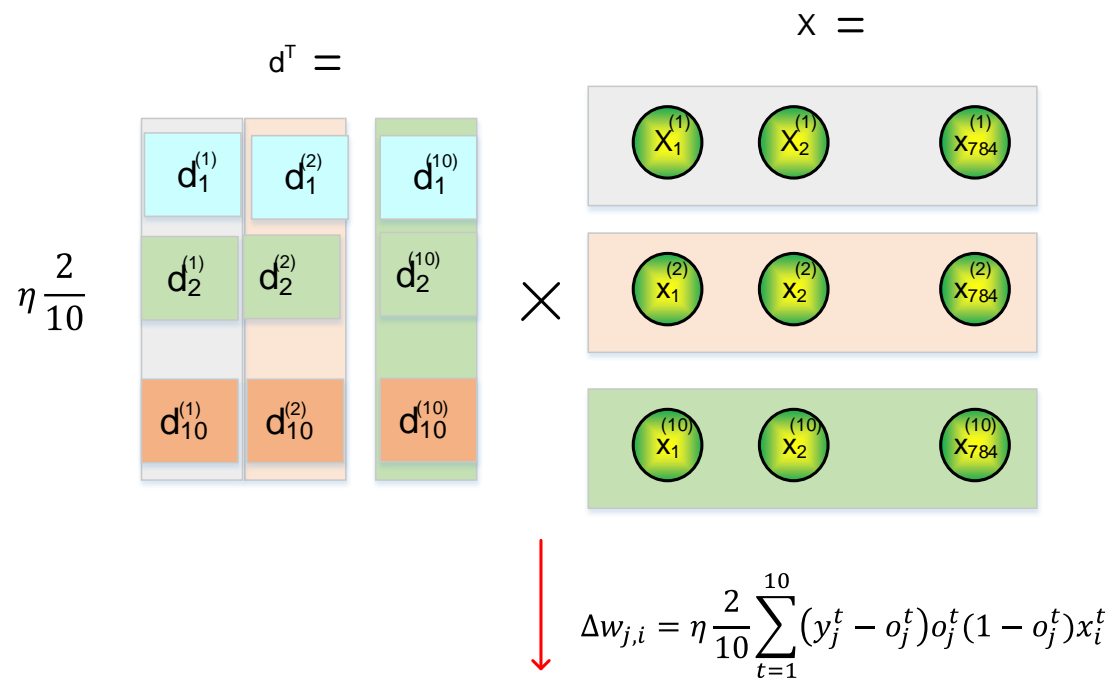
$$\Delta w_{j,i} = \eta \frac{2}{10} \sum_{t=1}^{10} (y_j^t - o_j^t) o_j^t (1 - o_j^t) x_i^t$$

d_j



$$d_j^t = (y_j^t - o_j^t) o_j^t (1 - o_j^t)$$

(element-wise product)



$$\Delta w = \eta \frac{2}{10} \begin{bmatrix} \Delta w_{1,1} & \Delta w_{1,2} & \Delta w_{1,3} & \Delta w_{1,784} \\ \Delta w_{2,1} & \Delta w_{2,2} & \Delta w_{2,3} & \Delta w_{2,784} \\ \Delta w_{10,1} & \Delta w_{10,2} & \Delta w_{10,3} & \Delta w_{10,784} \end{bmatrix}$$

$$d = (y - o) o (1 - o) \leftarrow \text{(element-wise product)}$$

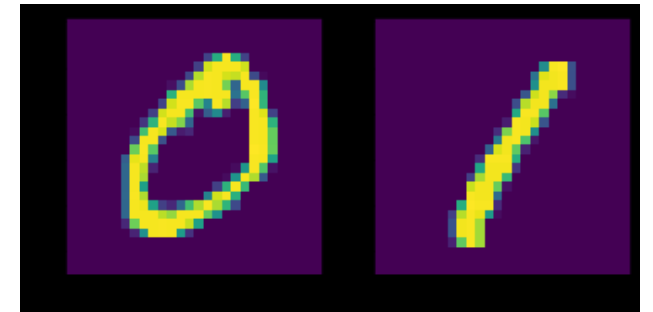
$$\Delta w = \eta \frac{2}{10} d^T X$$

Load data set, pick out 10 images

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
print('Load data from MNIST')
mnist = tf.keras.datasets.mnist
(x_train,y_train), (x_test,y_test) = mnist.load_data()
dig = np.array([1,3,5,7,9,11,13,15,17,19]) # get the digit 0 - 9
x = x_train[dig,:,:)
y = np.eye(10,10) ma trận đơn vị (chéo)
plt.subplot(121)
plt.imshow(x[0])
plt.subplot(122)
plt.imshow(x[1])
x = np.reshape(x, (-1, 784))/255 dữ liệu số nguyên -> 1 ảnh có giá trị từ
                                0 (đen) -> 255 (trắng)
```

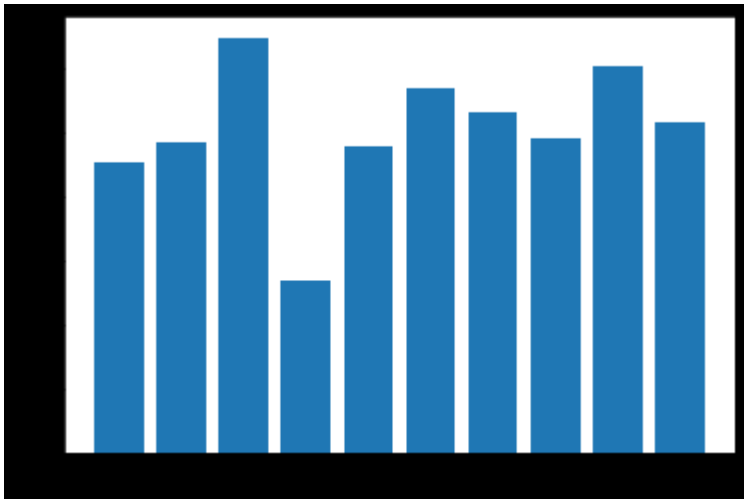
$$d = (y - o)o(1 - o)$$

$$\Delta w = \eta \frac{2}{10} d^T X$$



Define parameters and functions

```
def sigmoid(x):  
    return 1./(1.+np.exp(-x))  
  
W = np.random.uniform(-0.1,0.1,(10,784)) -0.1 -> 0.1  
o = sigmoid(np.matmul(x,W.transpose())) # matrix multiplication  
print('output of first neuron with 10 digits ', o[:,0])  
fig = plt.figure()  
plt.bar([i for i, _ in enumerate(o)],o[:,0])  
plt.show()
```



chưa huấn luyện

Training

```
#training process
n = 0.05
num_epoch = 10
for epoch in range(num_epoch):
    o = sigmoid(np.matmul(x,W.transpose()))
    loss =np.power(o-y,2).mean() mean: trung bình
    #calculate update for all wegihts in matrix
    dw =np.transpose((y-o)*o*(1-o))@x
    #update
    W=W+n*dw
    print(loss)

o = sigmoid(np.matmul(x,W.transpose()))
print('output of the first neuron with 10 input digits ', o[:,0])

fig = plt.figure()
plt.bar([i for i, _ in enumerate(o)],o[:,0])
plt.show()
```

$\Delta w = \eta \frac{2}{10} d^T X$

@: np.matmul 2 thông số ngõ vào
*: np.multiply



This is just a simple example to intuitively understand how to translate math into python code

