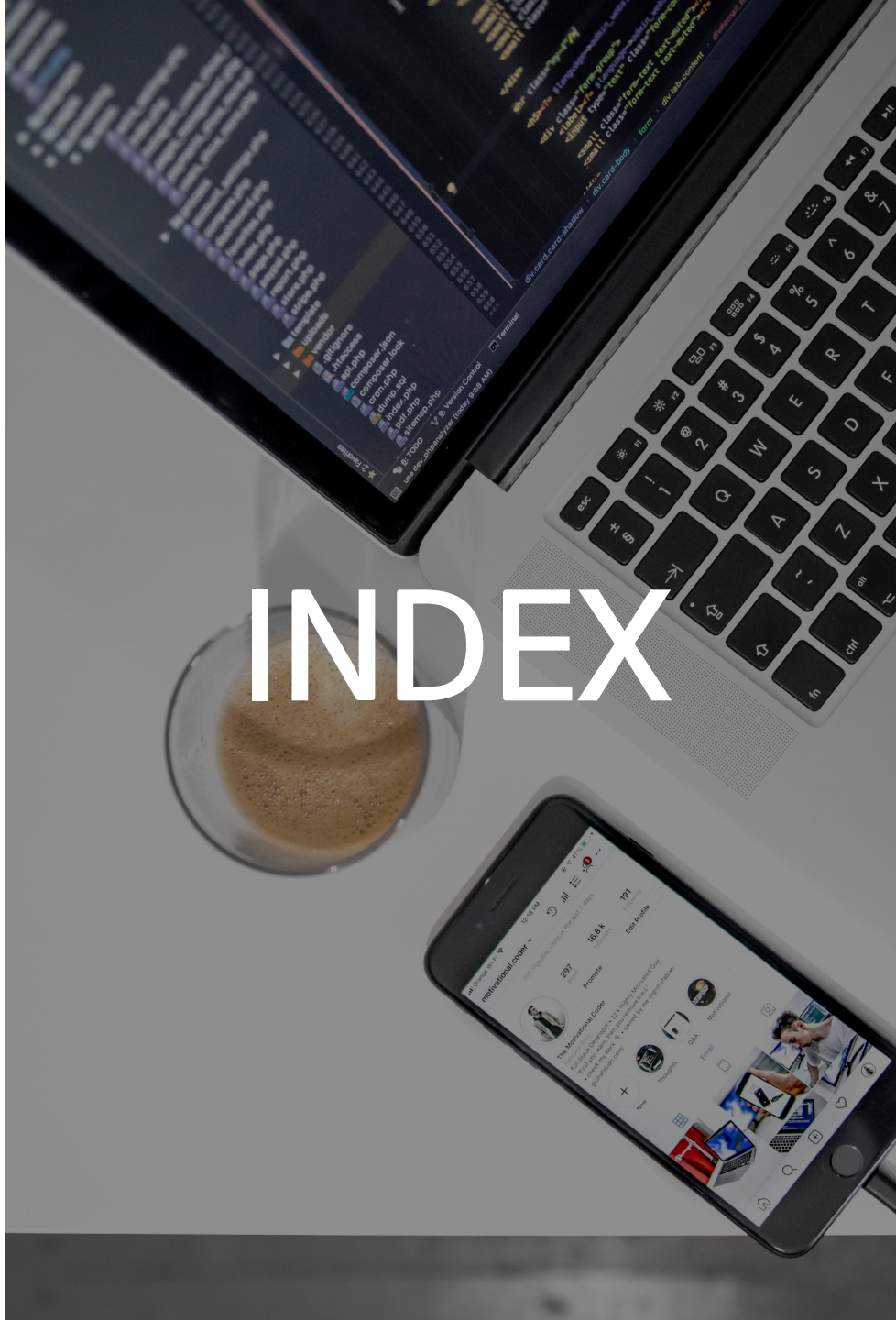
HAI Lecture

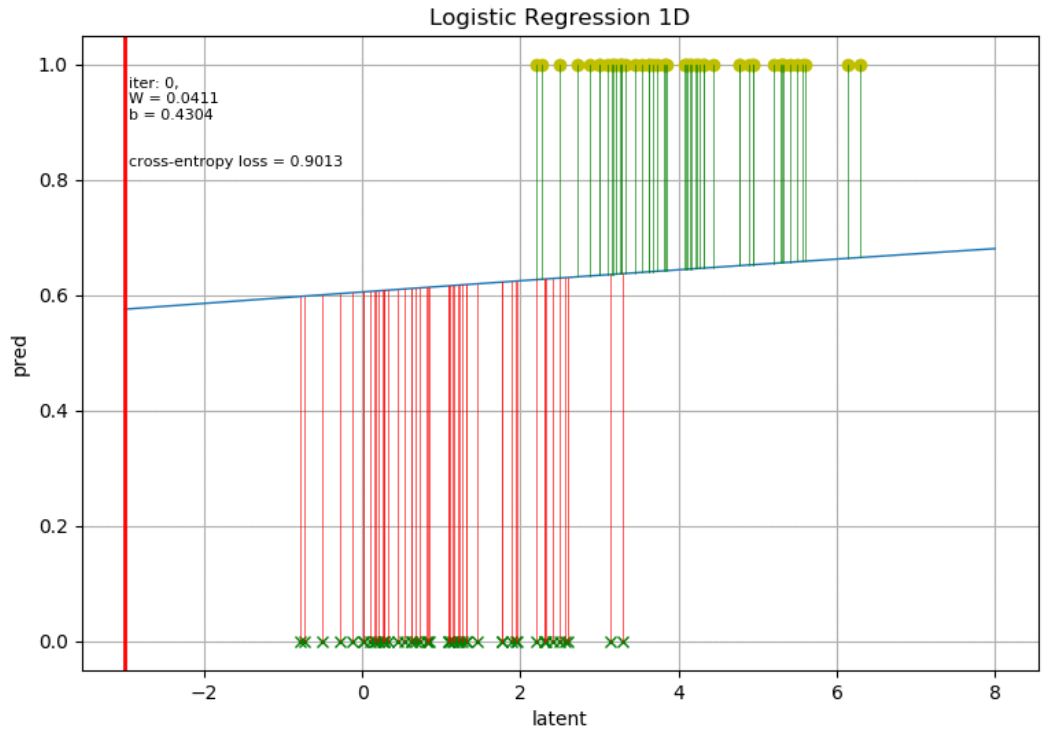
Lec 05.Multi Layer Perceptron

MLP

INDEX



Review : **Logistic Regression**



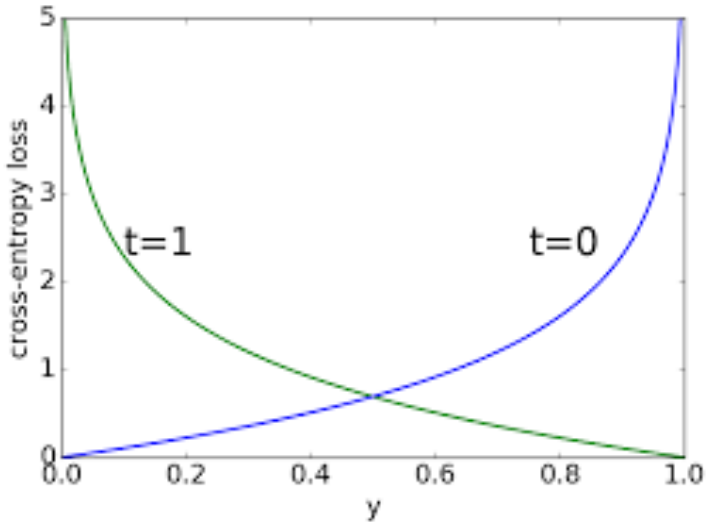
$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

Y = Sigmoid(wx + b)

Hypothesis as Bi

Review : **Linear Regression**



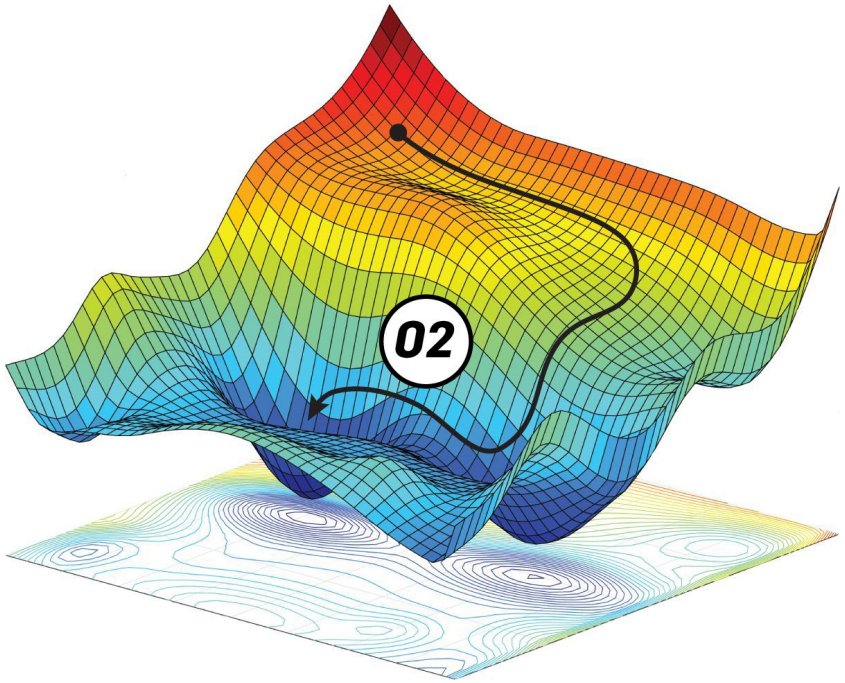
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Cross Entropy

Loss Function for Logistic Regression

Review : **Linear Regression**

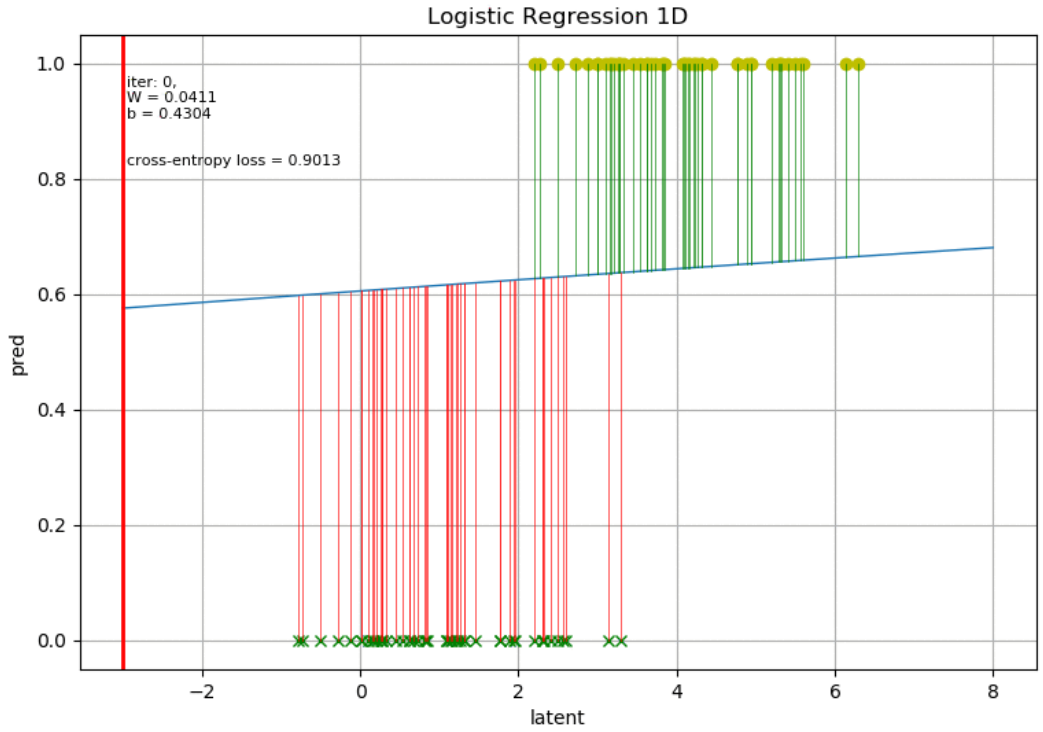


$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Gradient Descent

Algorithm for Optimizing the Model

Review : **Linear Regression**

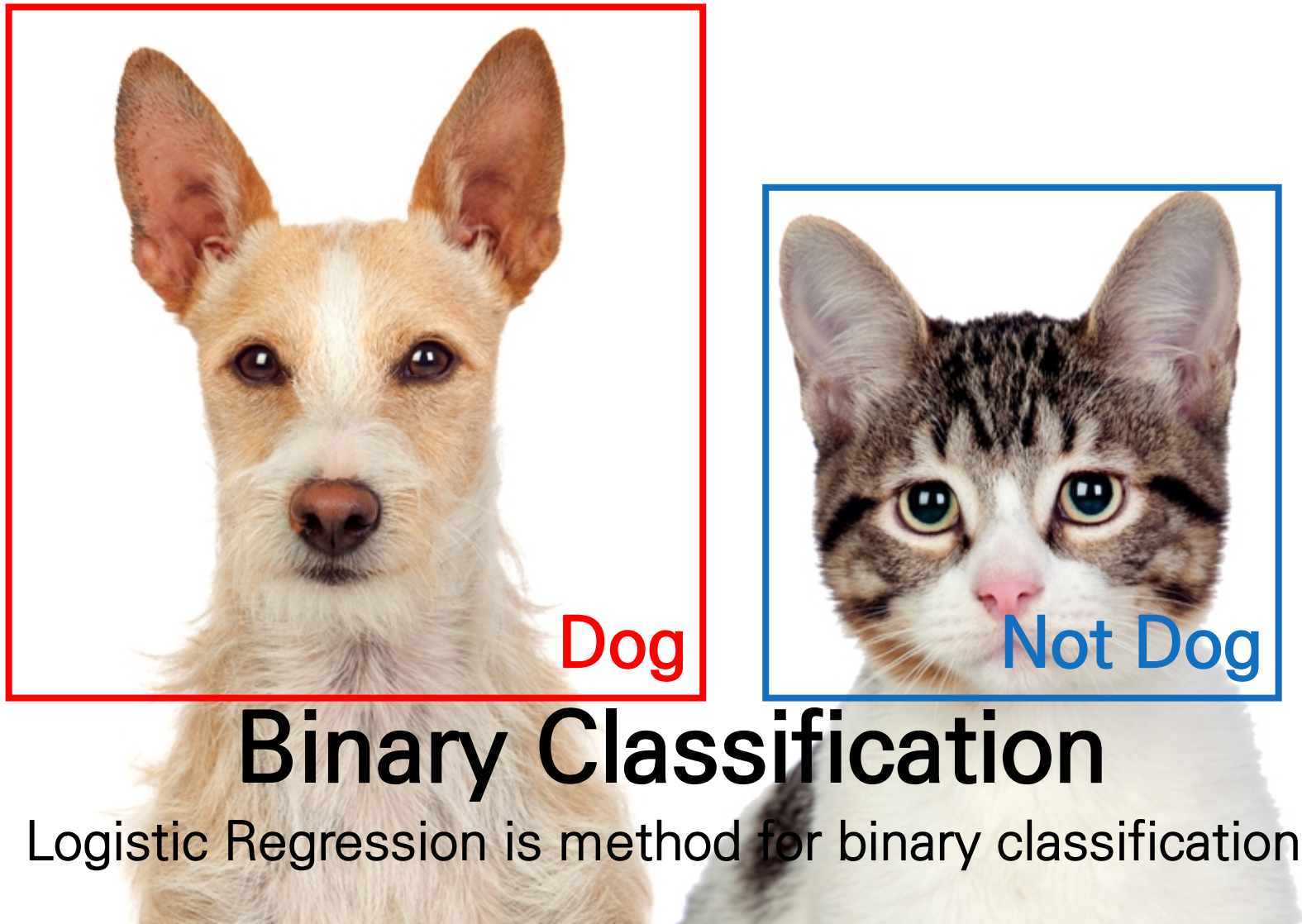


$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Gradient Descent

Algorithm for Optimizing the Model

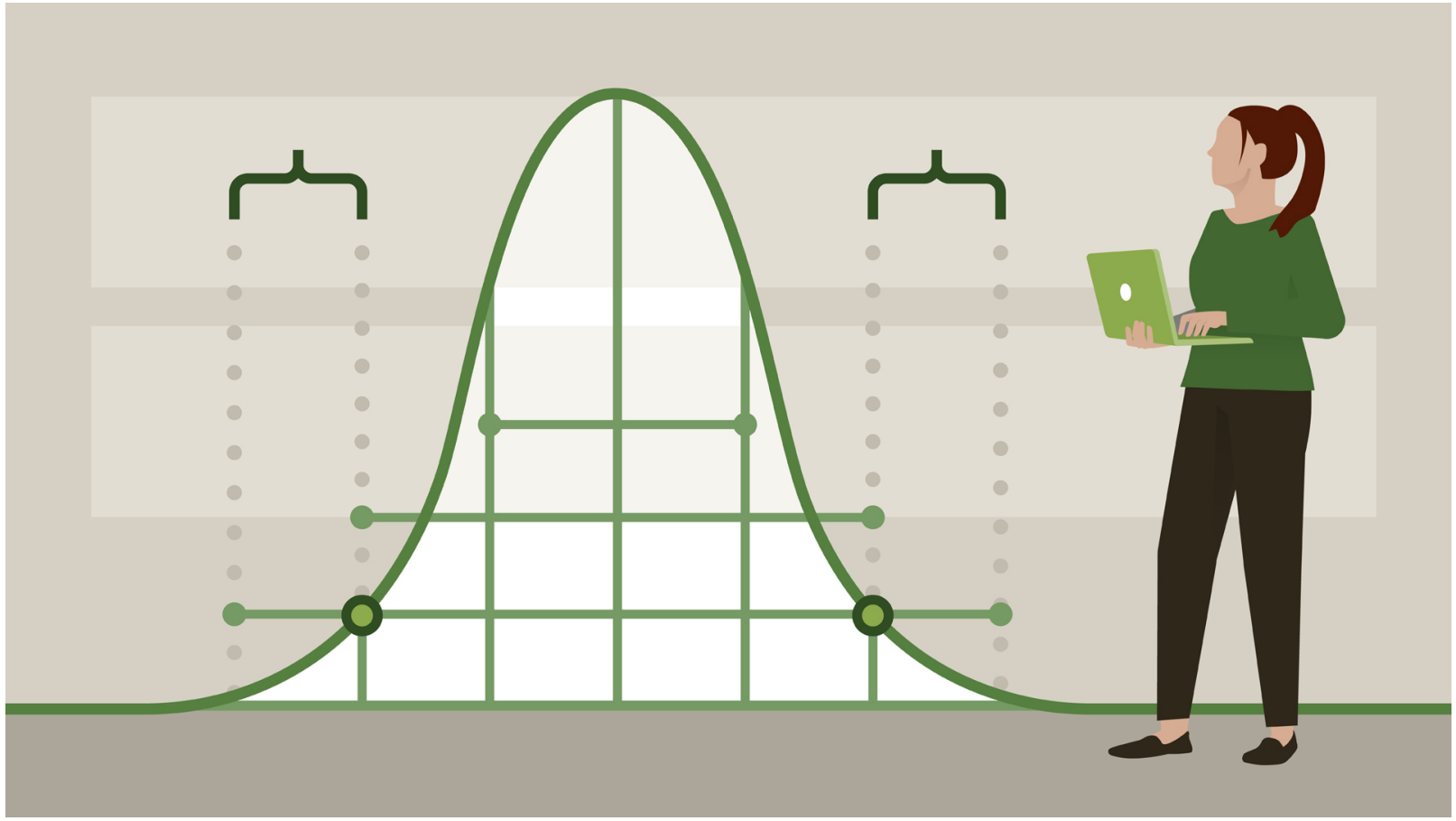
Review : Logistic Regression



Binary Classification

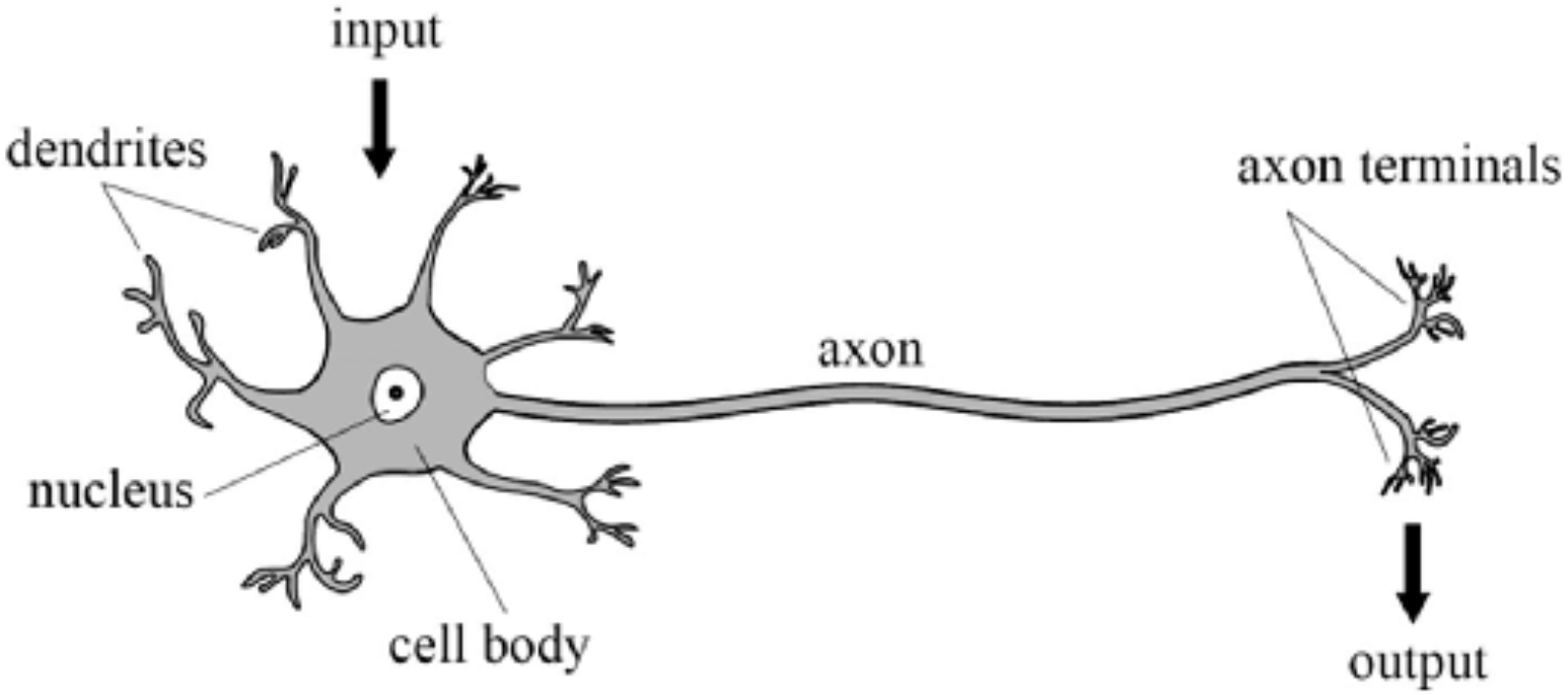
Logistic Regression is method for binary classification.

Introduction : **Perceptron**



잠시 Statistics는 내려놓도록 합시다

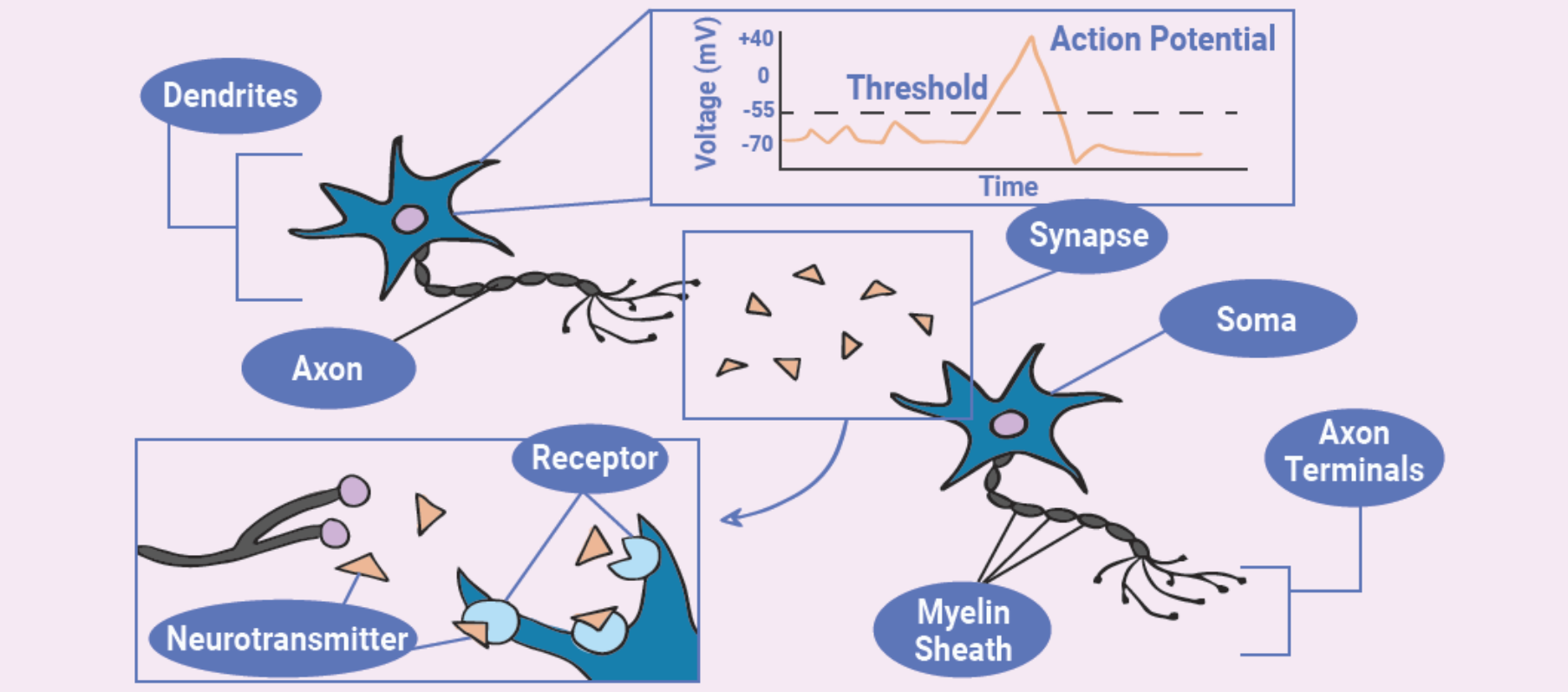
Introduction : **Perceptron**



Human Neuron

Processes Electrical Signals

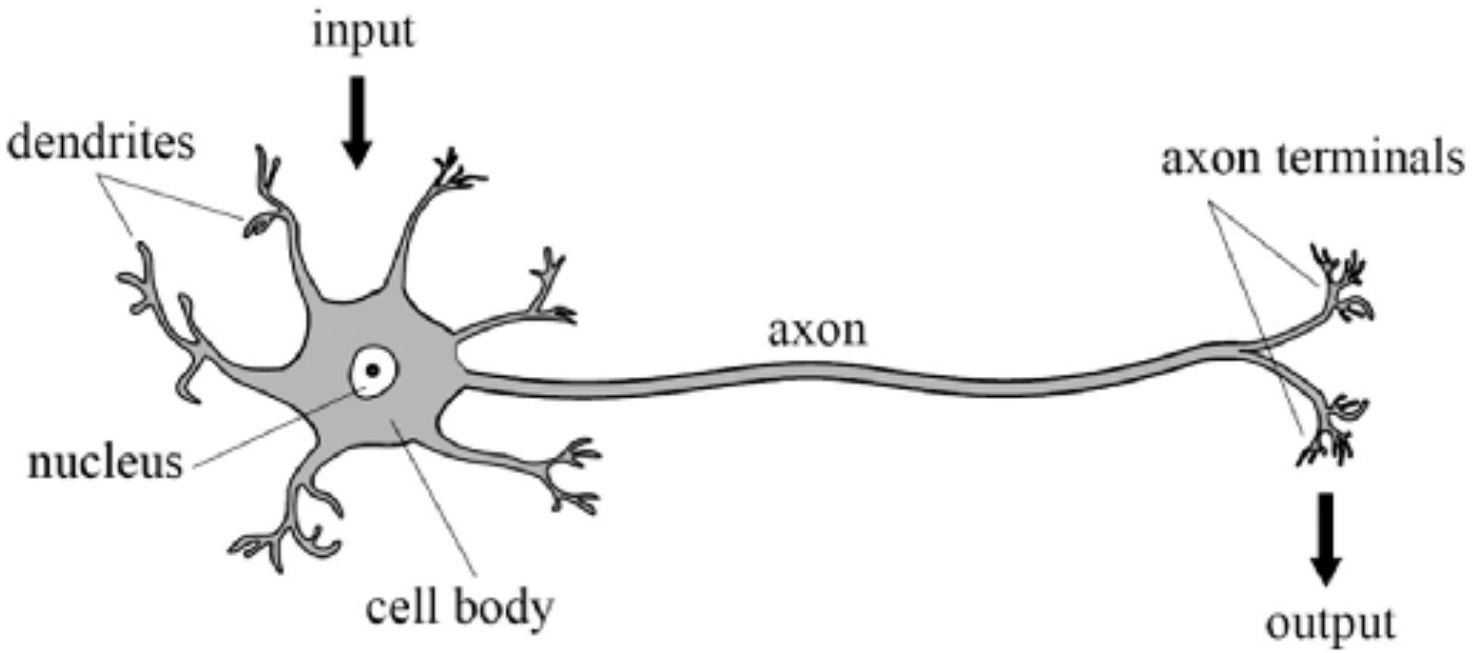
Introduction : **Perceptron**



If the mixture of signals exceeds threshold,

The Neuron fires, and sends signal to next neuron

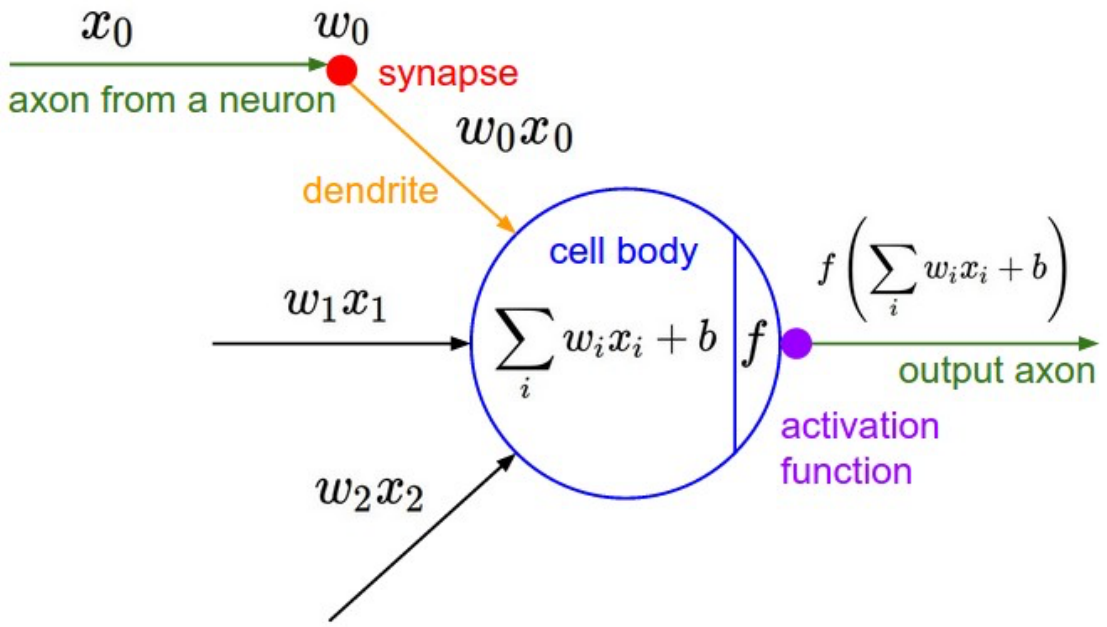
Introduction : **Perceptron**



If the mixture of signals exceeds threshold,

The Neuron fires, and sends signal to next neuron

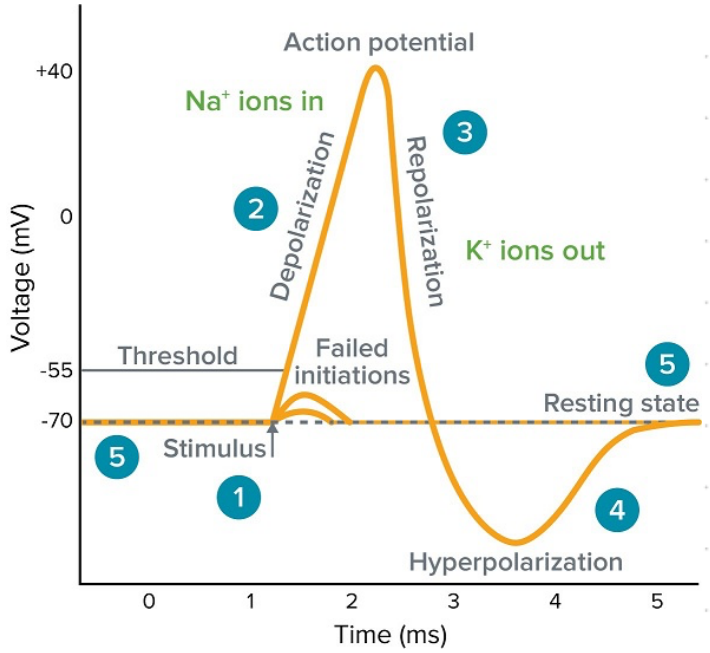
Introduction : **Perceptron**



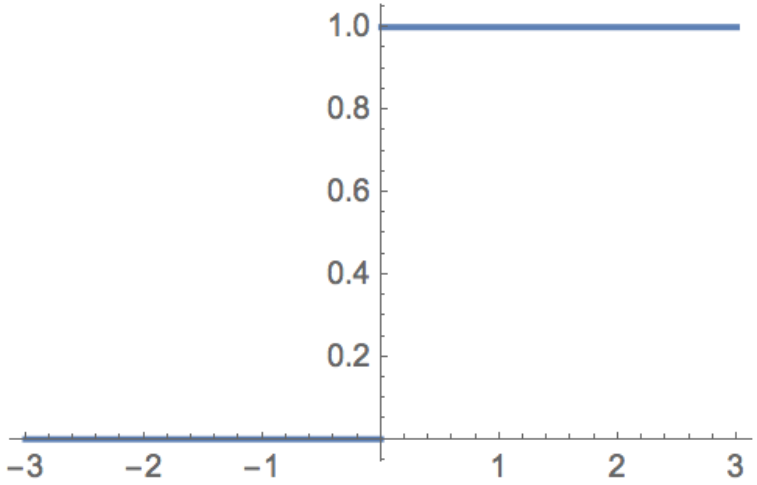
Perceptron

A mathematical Modeling of Biological Neurons

Introduction : **Perceptron**



Biological Threshold of neuron

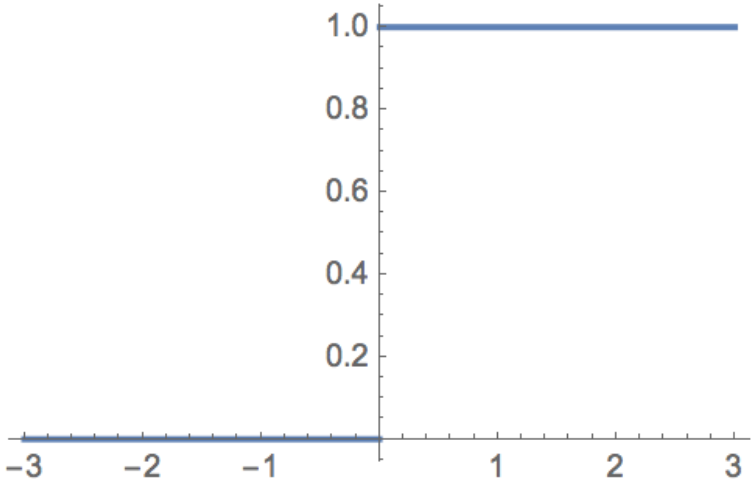


Mathematical Threshold of Perceptron

Perceptron

A mathematical Modeling of Biological Neurons

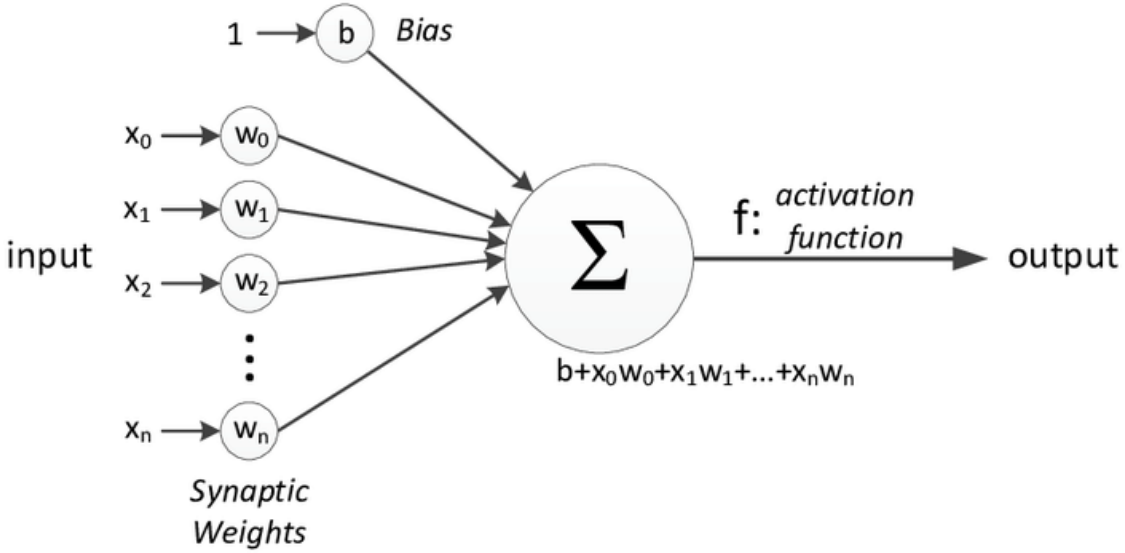
Introduction : **Perceptron**

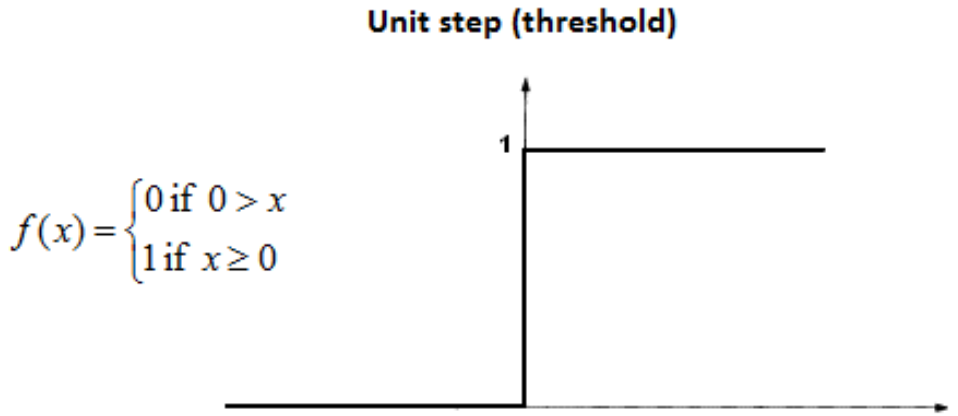


Mathematical Threshold of Perceptron

We call this threshold function,
Activation Function

Algorithm : **Perceptron**

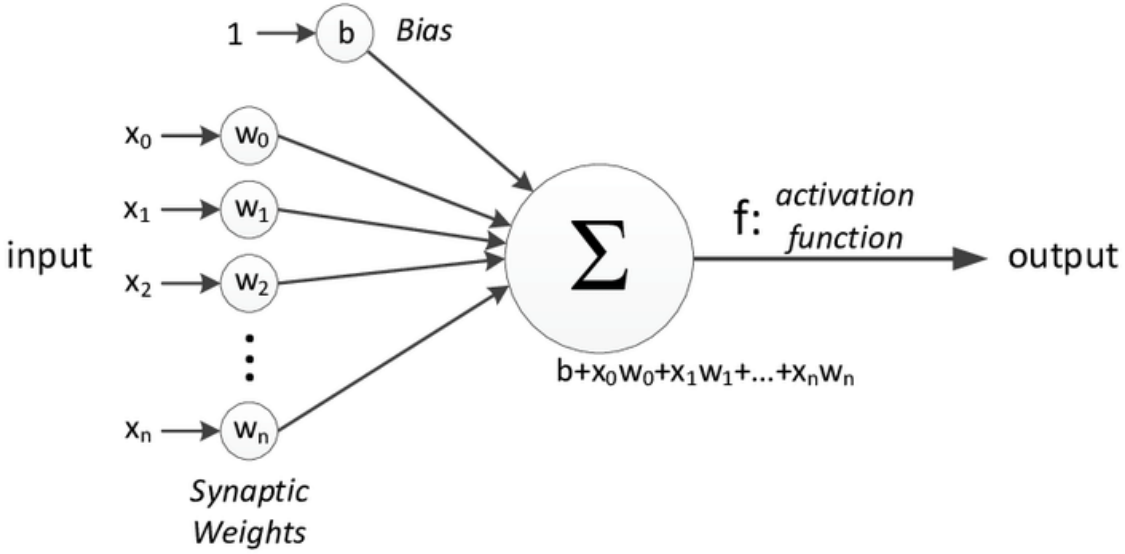




How Perceptron Makes Output

Multiply weights, and check if it exceeds its threshold.

Algorithm : **Perceptron**



$$output = \begin{cases} 0(w_1x_1 + w2x_2 + \ldots + w_ix_i \leq \theta) \\ 1(w_1x_1 + w2x_2 + \ldots + w_ix_i > \theta) \end{cases}$$

How Perceptron Makes Output

Multiply weights, and check if it exceeds its threshold.

Algorithm : **Perceptron**

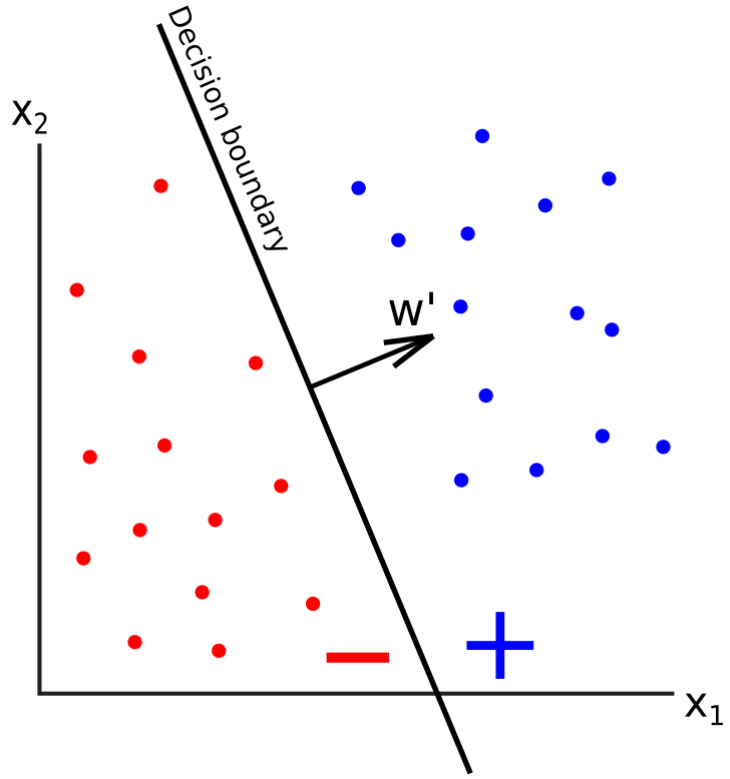
Algorithm: Perceptron Learning Algorithm

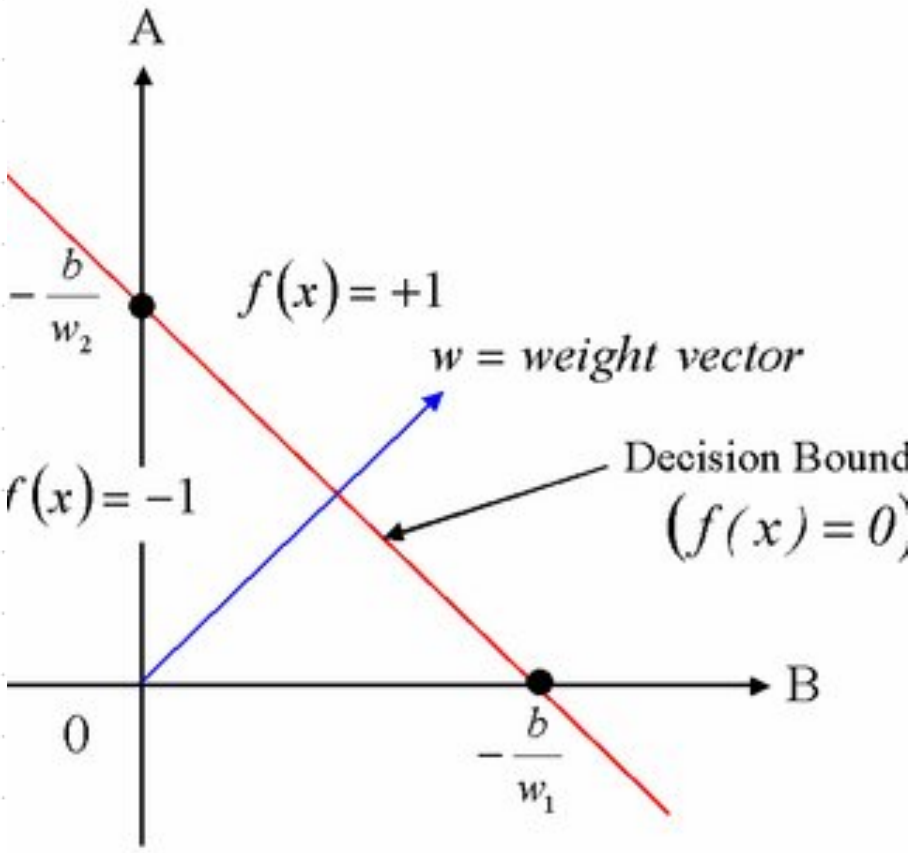
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

How Perceptron Learns

Weight adjustment

Algorithm : **Perceptron**

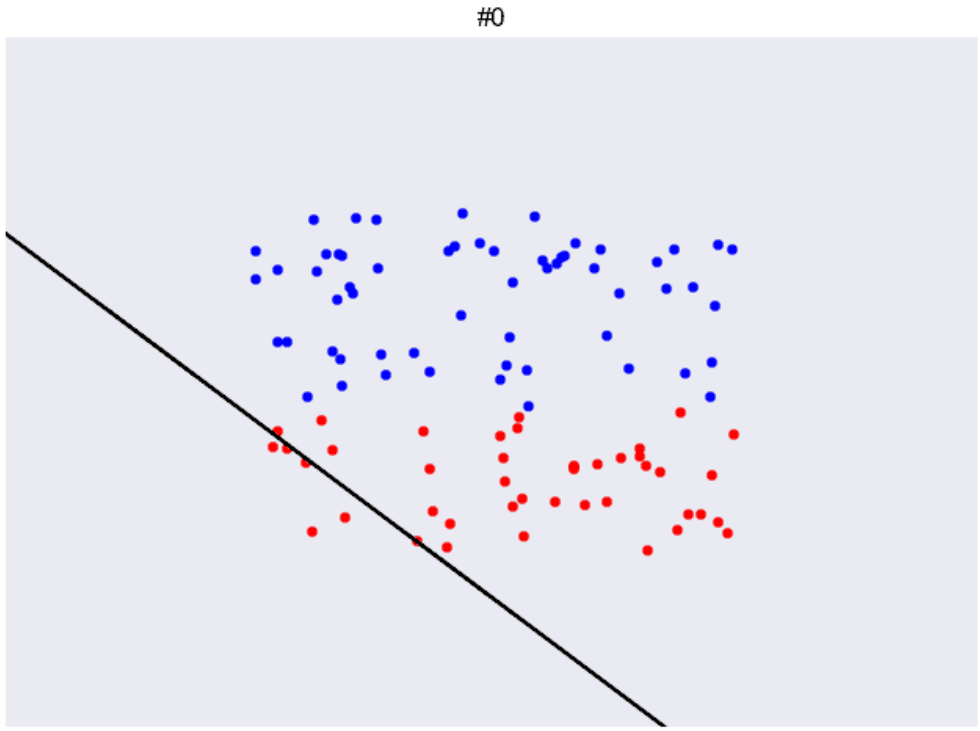




Decision Boundary of Perceptron

Linear Decision Boundary in Single-Layer Perceptron

Algorithm : **Perceptron**

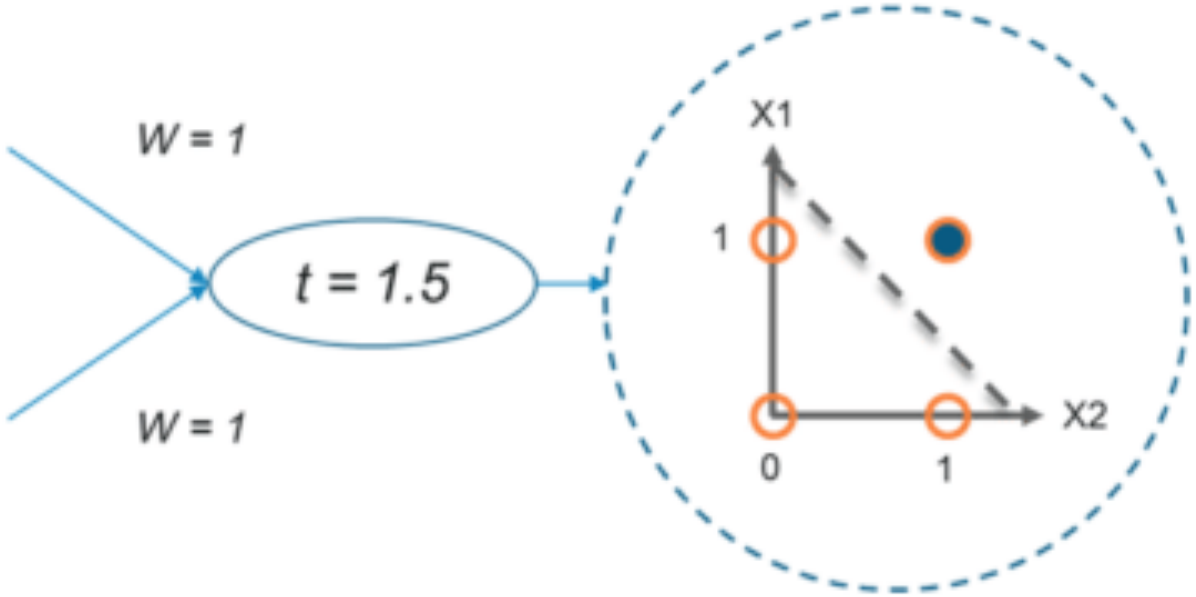


Decision Boundary of Perceptron

Linear Decision Boundary in Single-Layer Perceptron

Algorithm : **Perceptron**

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

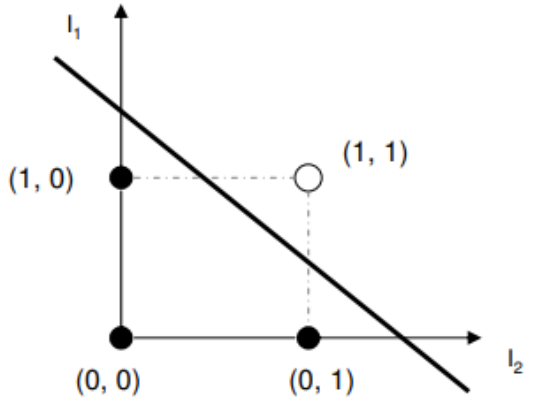


Example : AND Gate

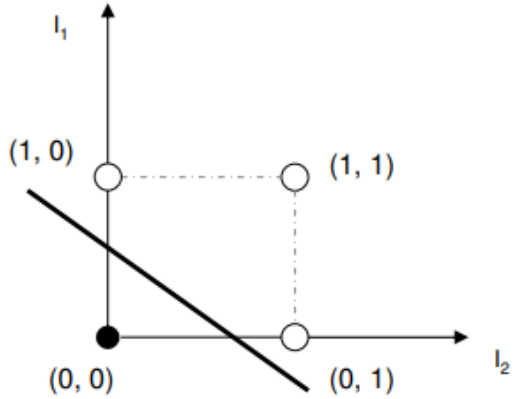
Perceptron Trained on AND Gate

Problem of Perceptron : **XOR Training?**

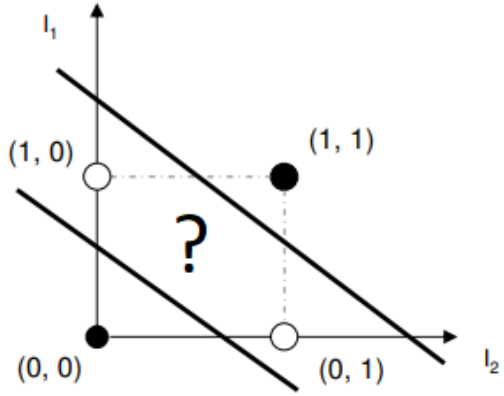
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



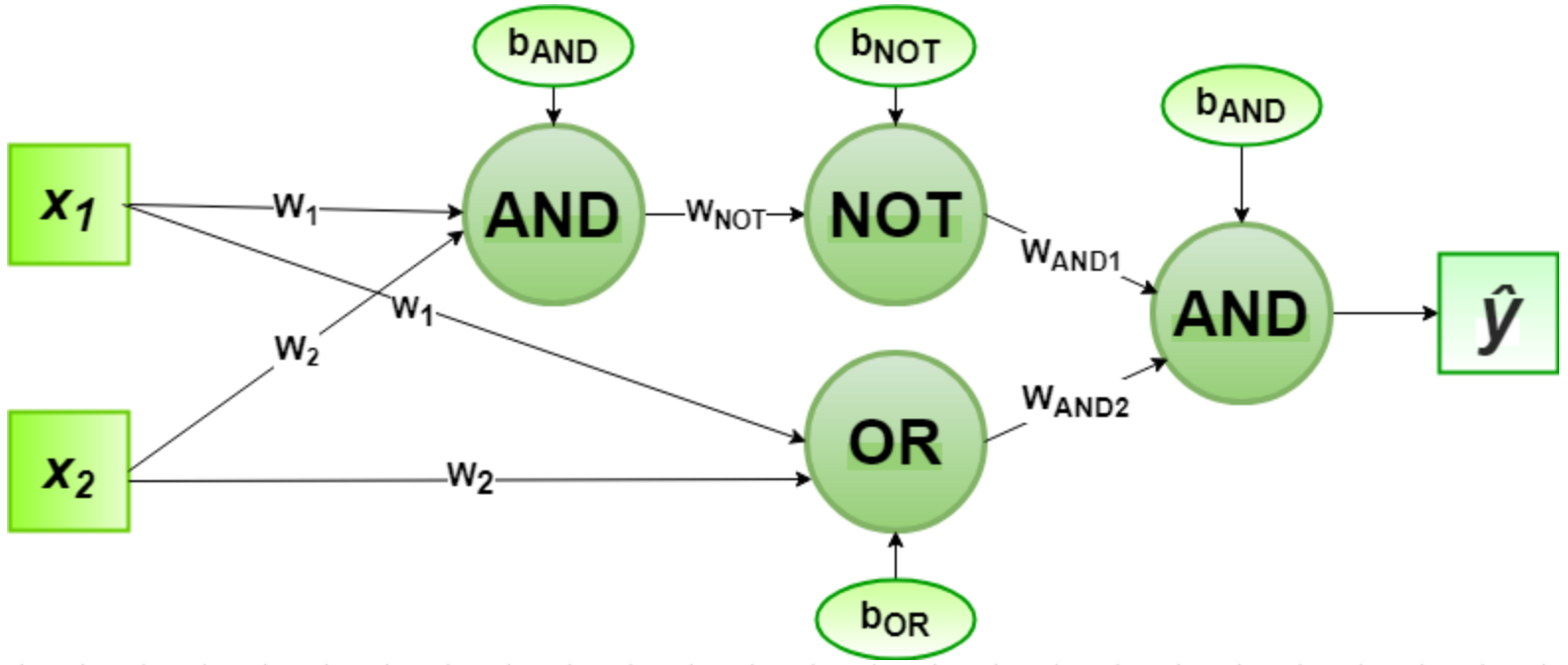
XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Training Logic Gates with Perceptron

How can we train XOR data?

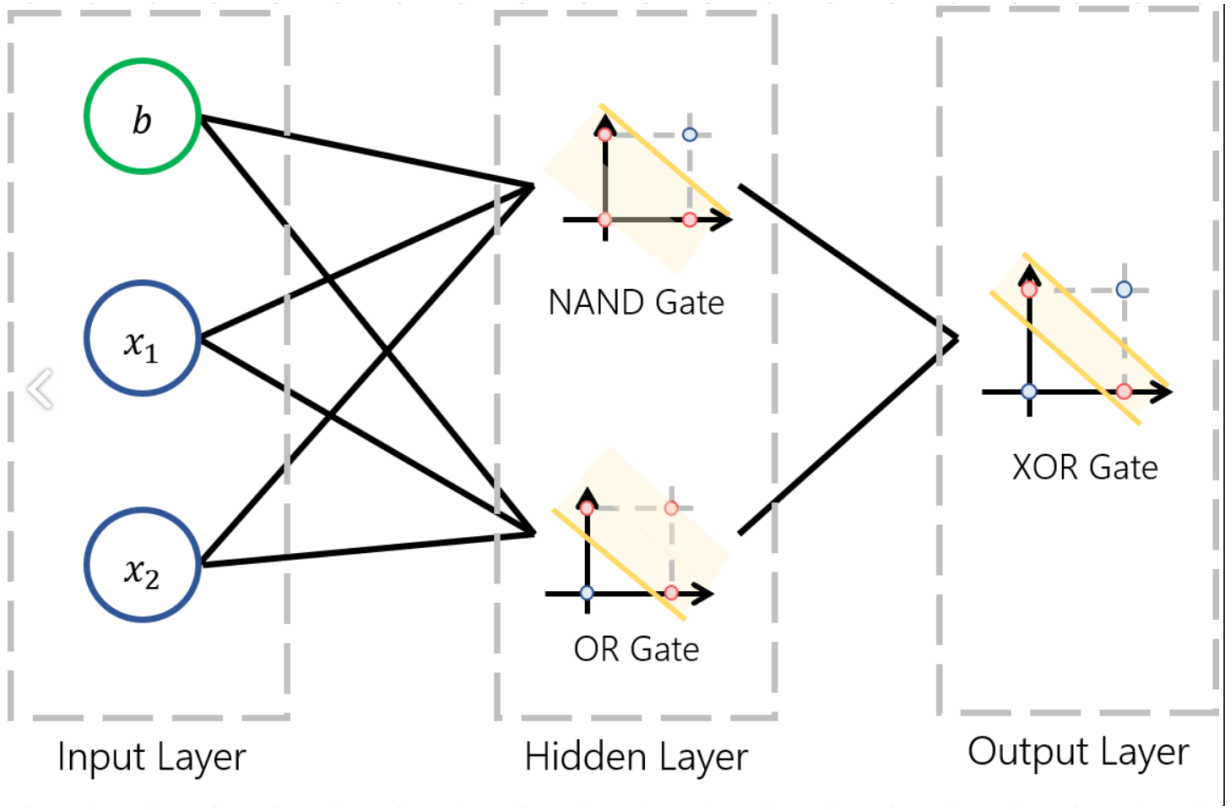
Overview : **Multi Layer Perceptron**



Mixture of Logic gates in Hierarchically

We can make XOR gate with NOT, OR, and AND

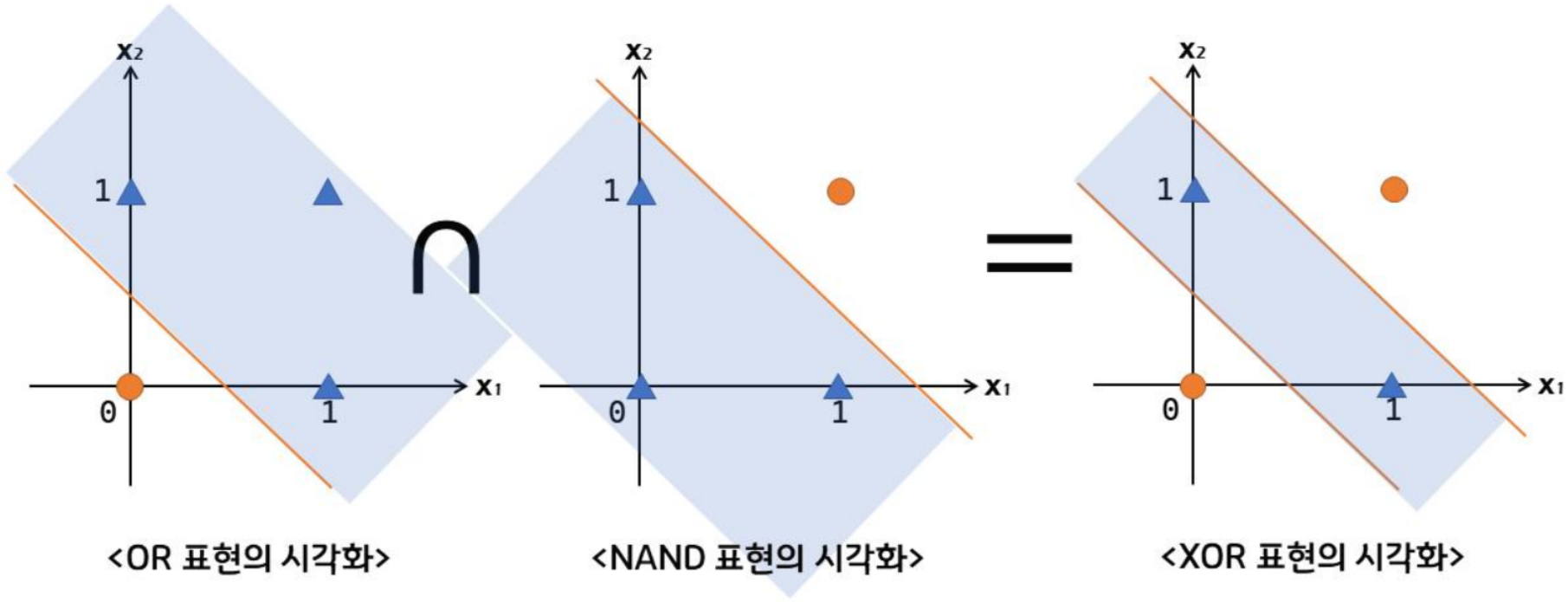
Overview : **Multi Layer Perceptron**



Mixture of Perceptrons in Hierarchically

Use the idea of "Mixture of Logic Gates"

Overview : **Multi Layer Perceptron**

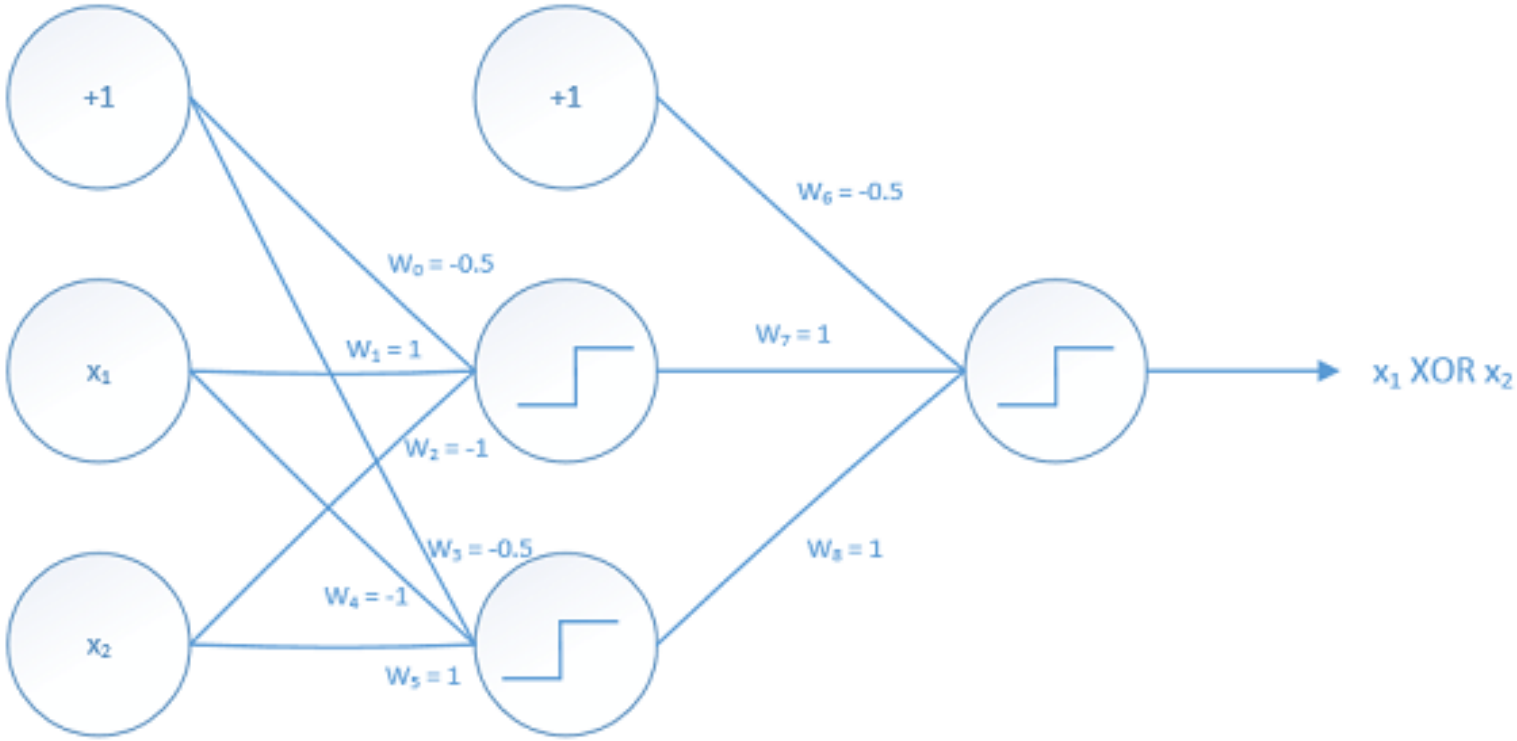


<OR 표현의 시각화> <NAND 표현의 시각화> <XOR 표현의 시각화>

Mixture of Perceptrons in Hierarchically

Use the idea of "Mixture of Logic Gates"

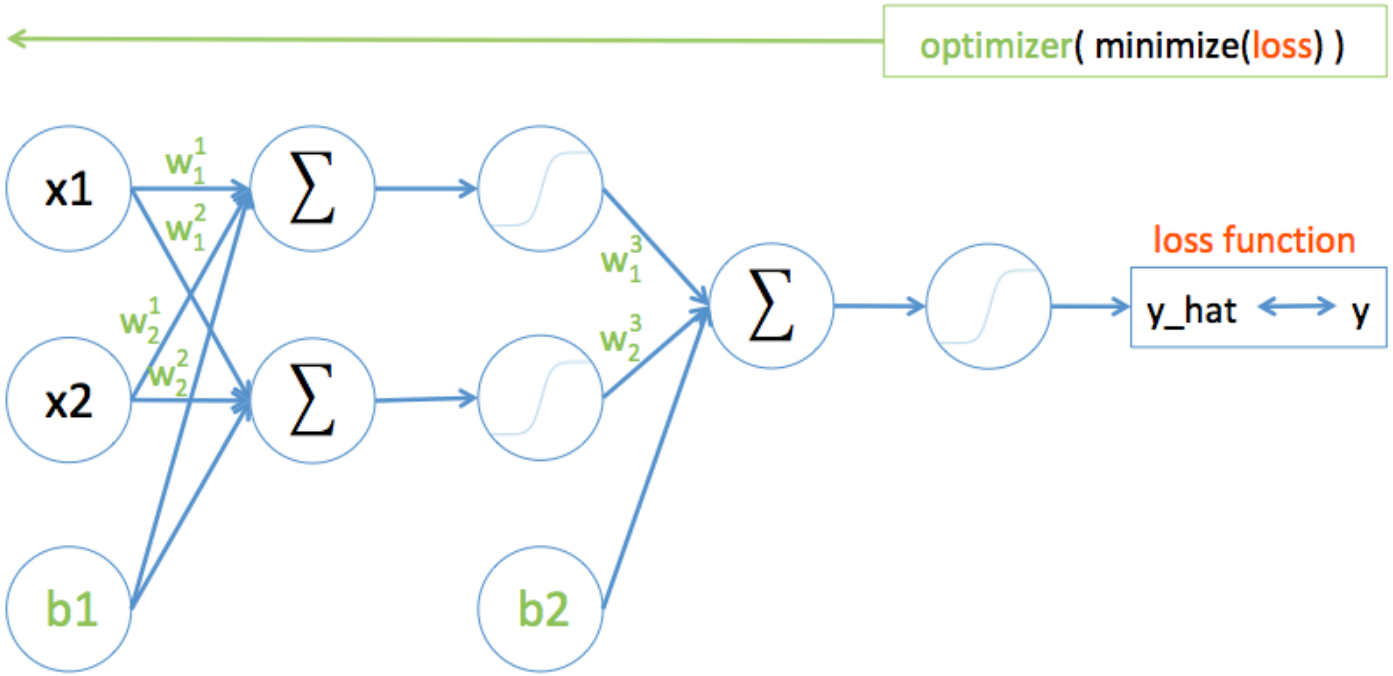
Overview : **Multi Layer Perceptron**



Mixture of Perceptrons in Hierarchically

Use the idea of "Mixture of Logic Gates"

Overview : **Multi Layer Perceptron**

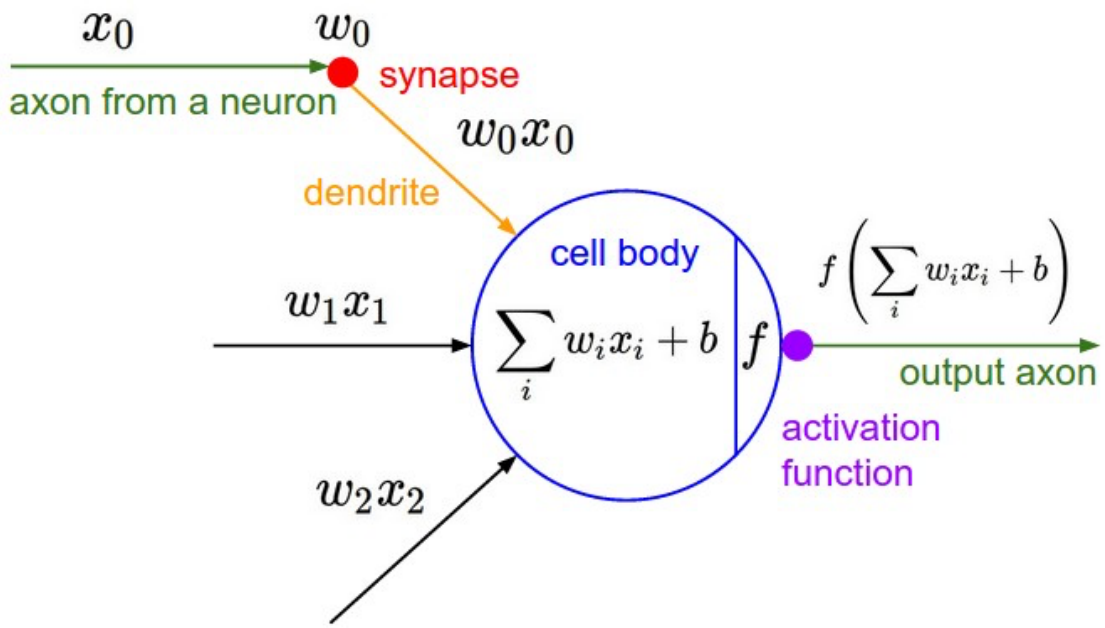


$$y = f(W'f(W\boldsymbol{x} + B) + B')$$

How MLP Works

Feed Forward, and Compare with Loss Function

Activation Function : **Why we use?**

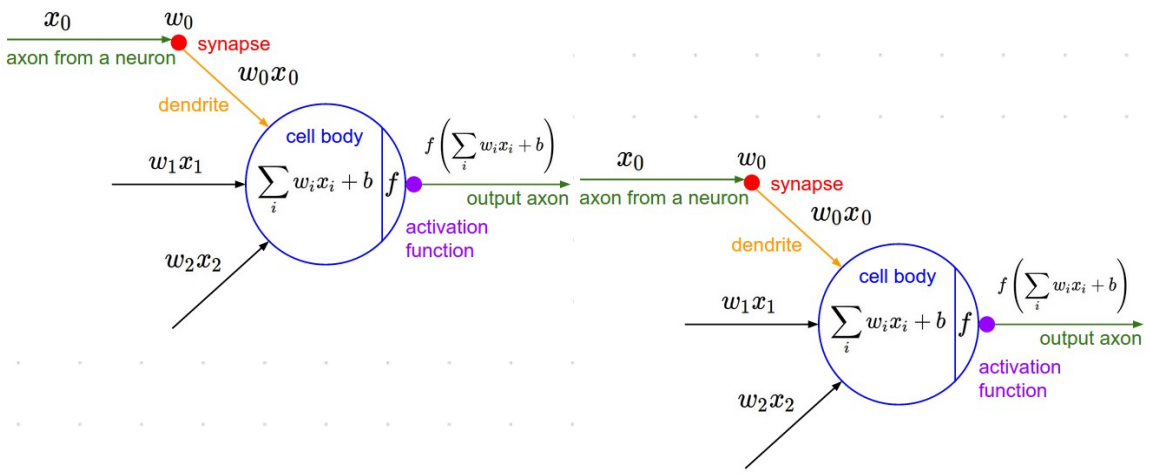


$$out1 = W_1x$$

Each Layer of Perceptrons Must Have Activation Function

Previously, we used "Step Function" as Activation Function.

Activation Function : **Why we use?**



$$out1 = W_1 x$$

$$out2 = W_2 out1$$

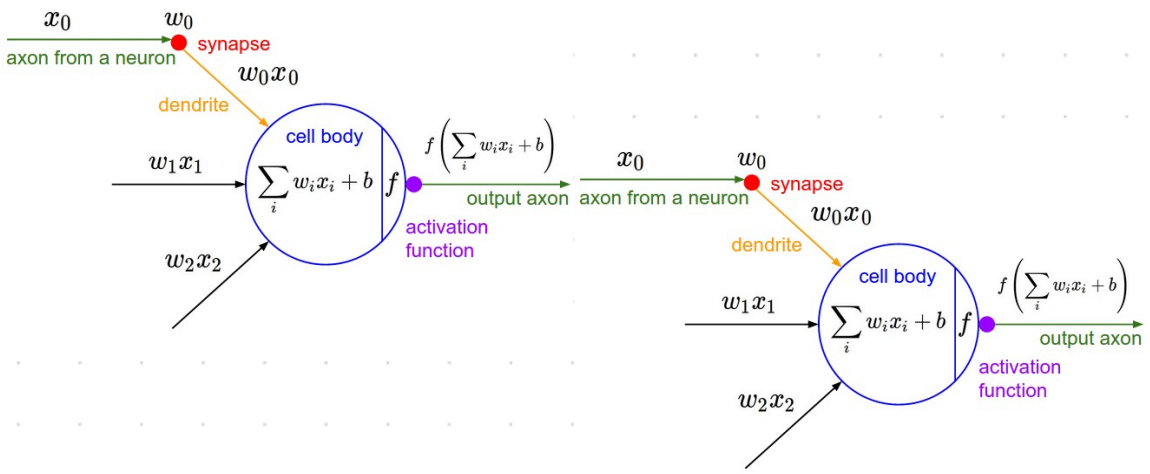
$$out = W_2(W_1 x)$$

$$out = Wx$$

Without Activation Function, it is just a Linear Model.

Same with Single Layered Model.

Activation Function : **Why we use?**



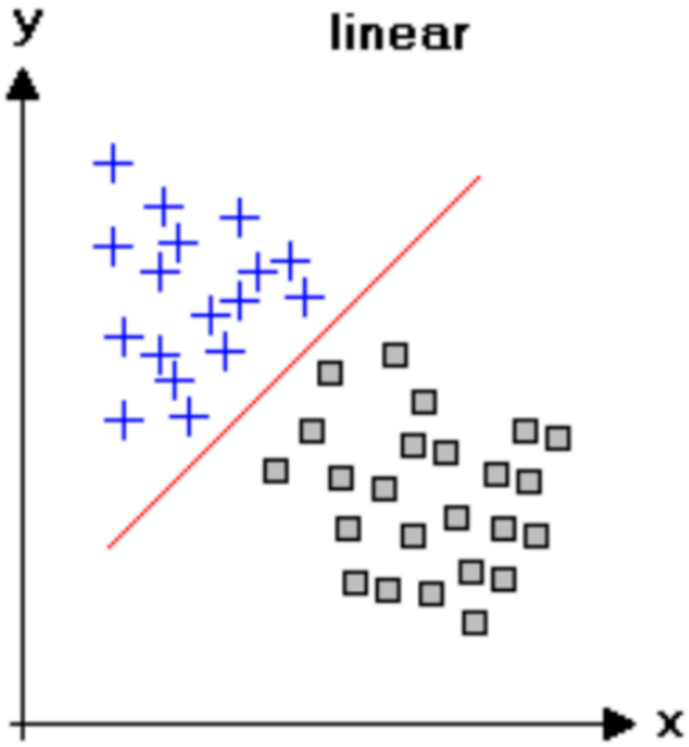
$$out1 = f(W_1x)$$
$$out2 = f(W_2out1)$$

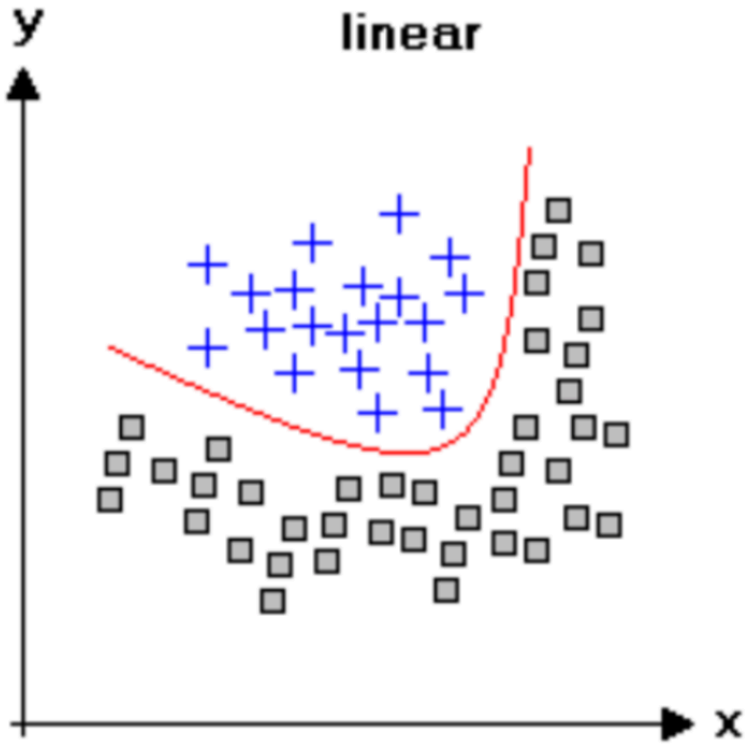
$$out = f(W_2(f(W_1x)))$$

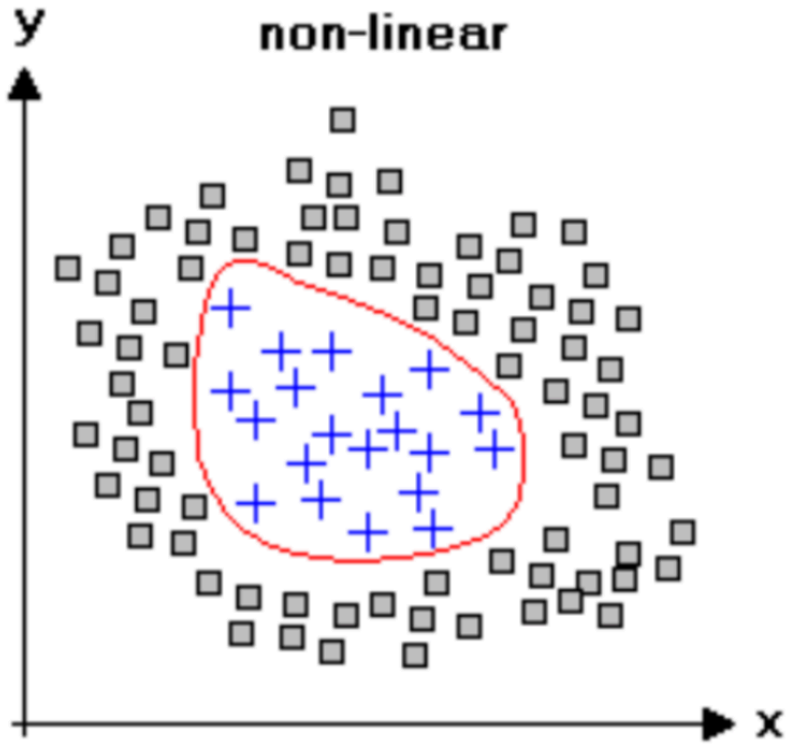
With Activation Function, Model can have non-linearity.

Now, model can fit into non-linear function.

Activation Function : **Why we use?**



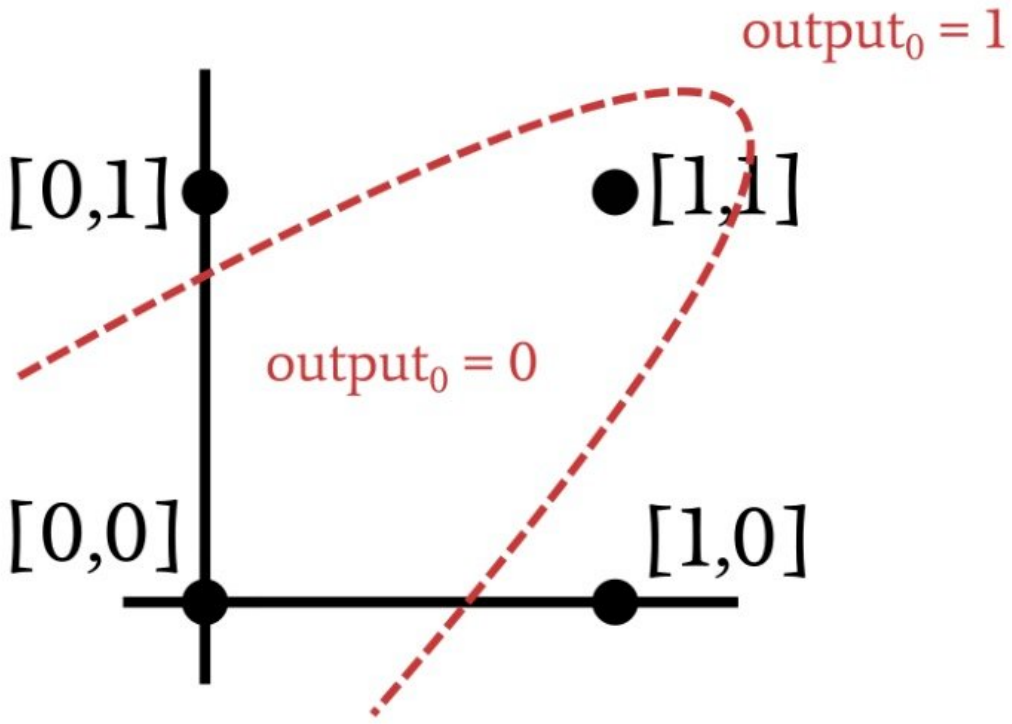




Non-Linear Multi Layer Perceptron makes Non-Linear Boundaries

Non-Linear Perceptron can learn non-linear boundaries

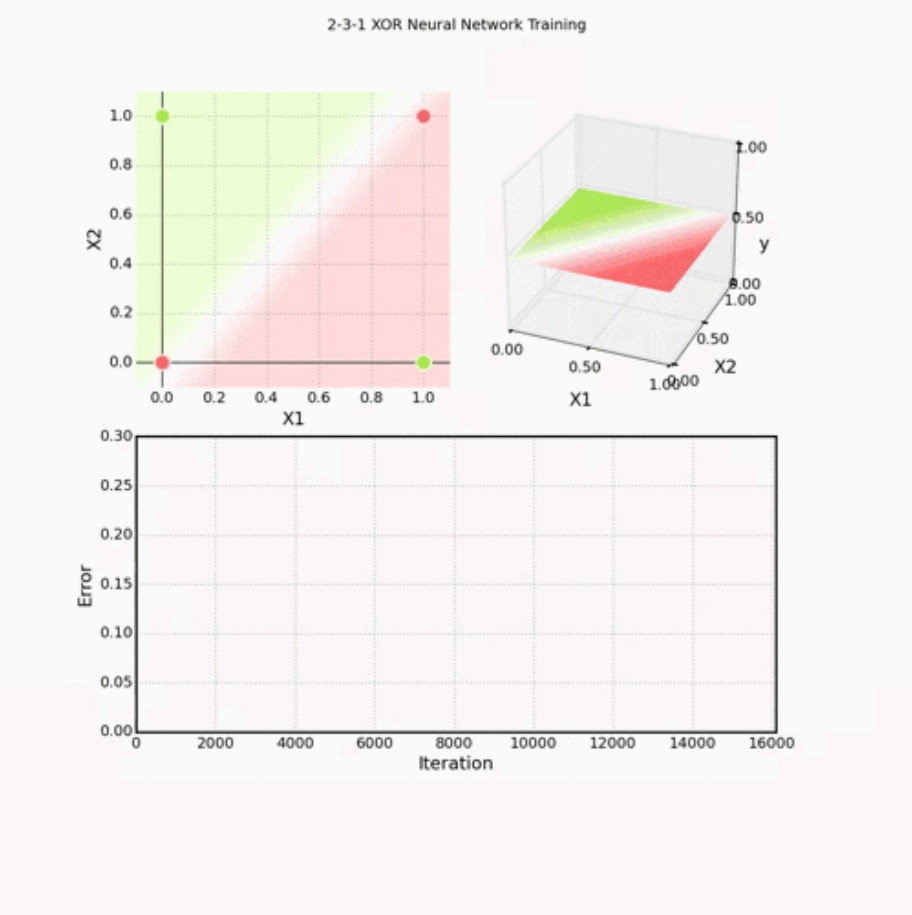
Activation Function : **Why we use?**



Non-Linear Multi Layer Perceptron makes Non-Linear Boundaries

Non-Linear Perceptron can learn non-linear boundaries

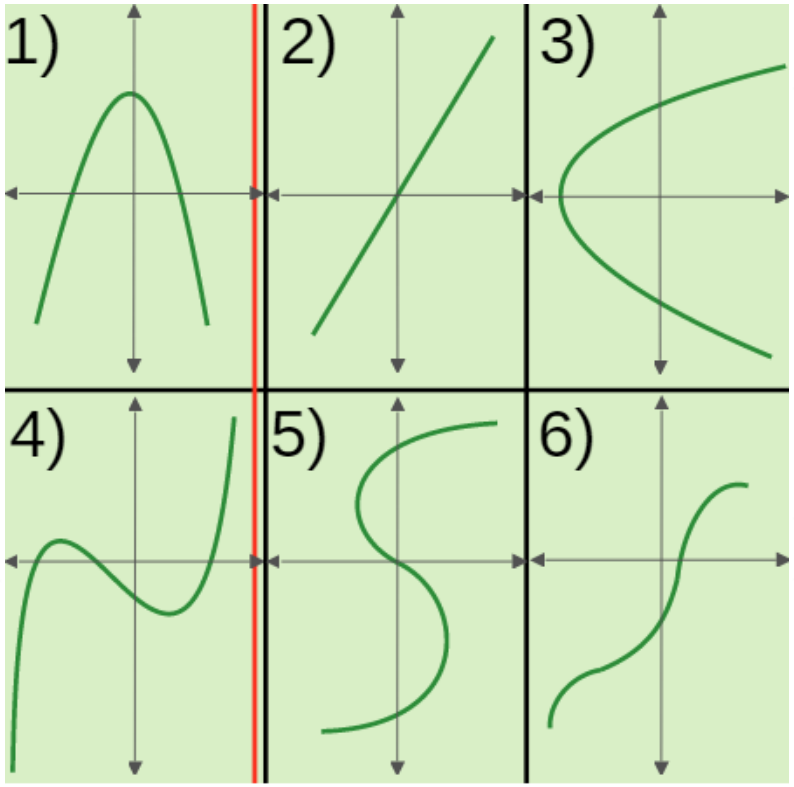
Activation Function : **Why we use?**



Non-Linear Multi Layer Perceptron makes Non-Linear Boundaries

Non-Linear Perceptron can learn non-linear boundaries

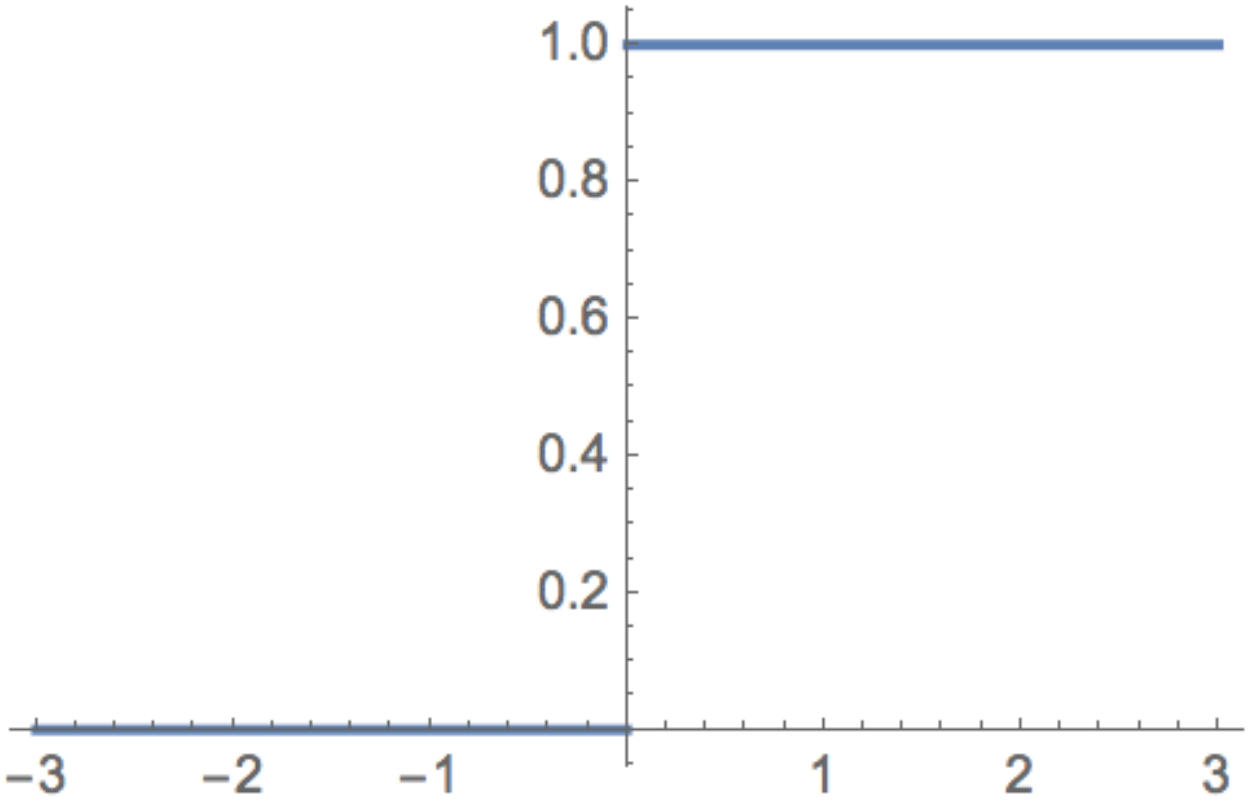
Activation Function : **Why we use?**



MLP can express any kind of "Non-Linear Functions"

Non-Linear Perceptron can learn non-linear boundaries

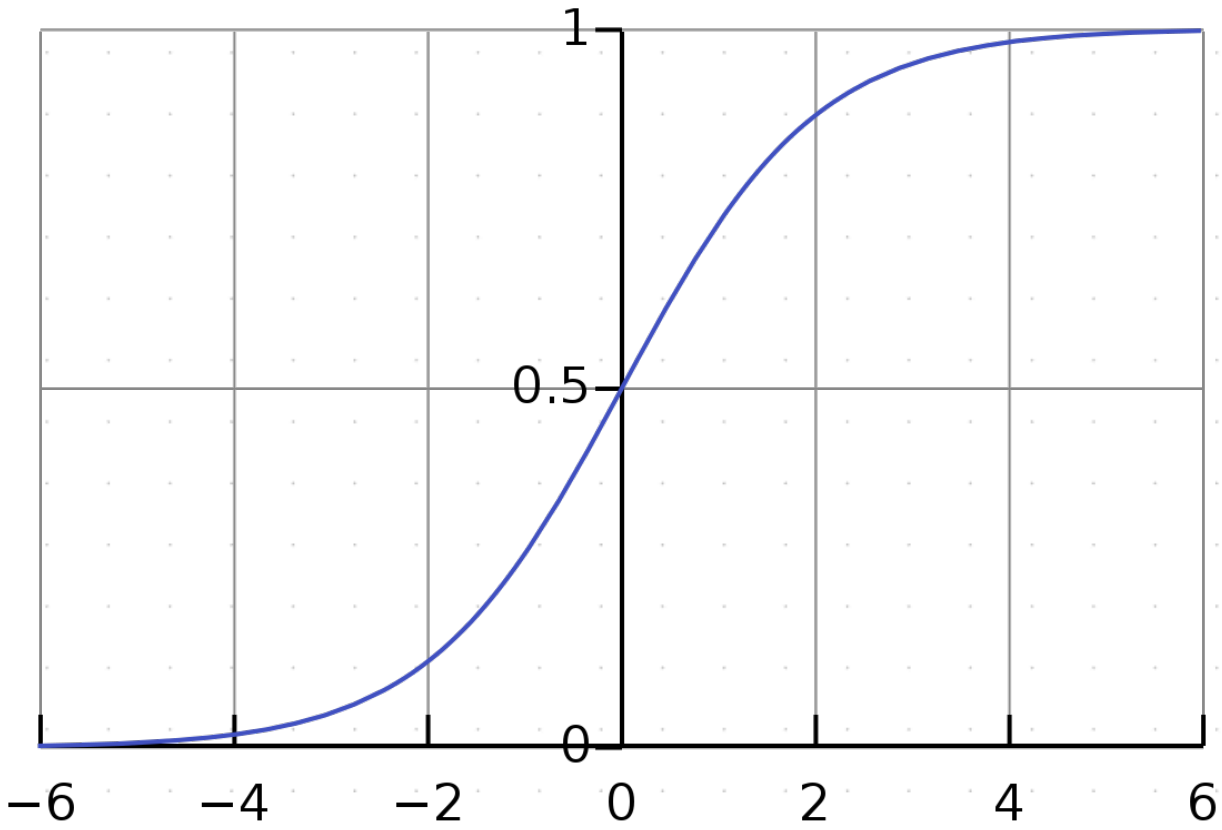
Activation Function : **What kinds are they?**



Step Function

Activation function from single perceptron

Activation Function : **What kinds are they?**

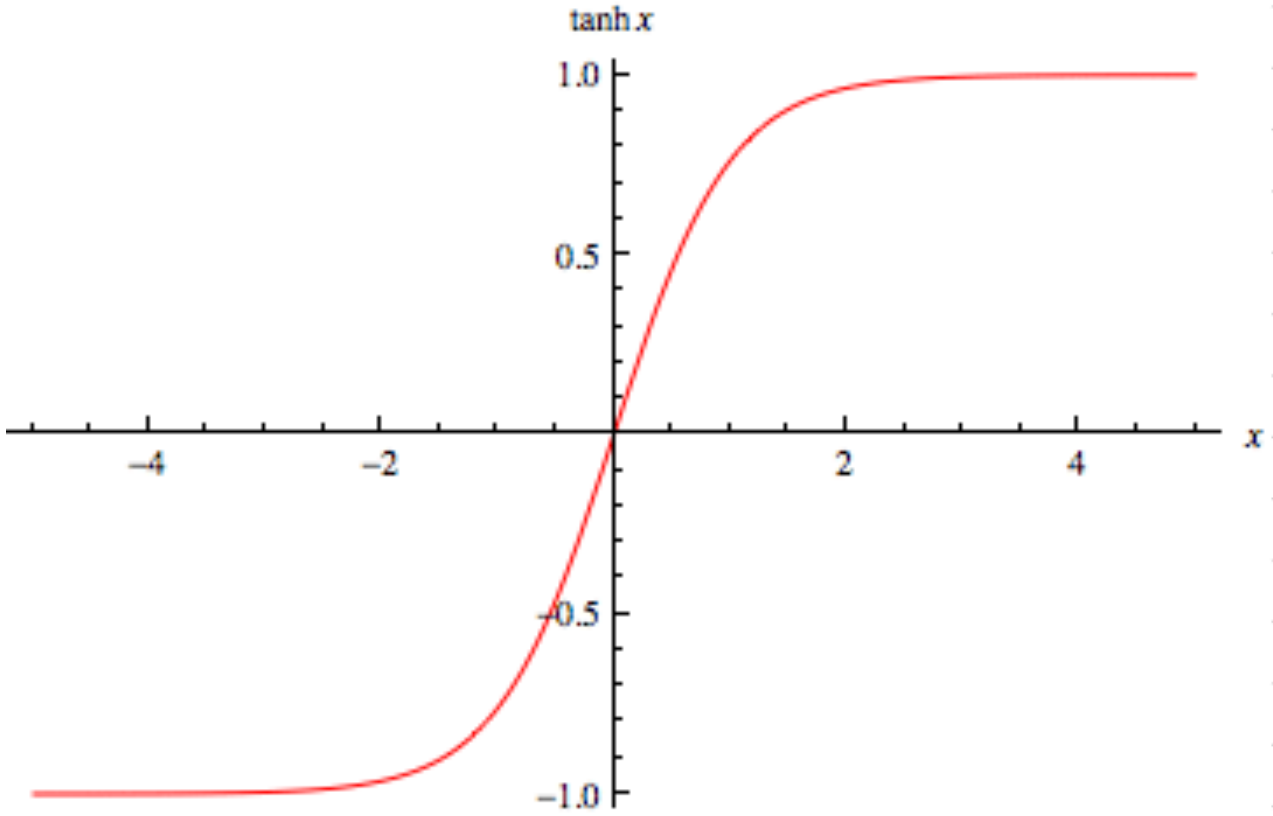


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid Function

Activation function that has derivatives

Activation Function : **What kinds are they?**

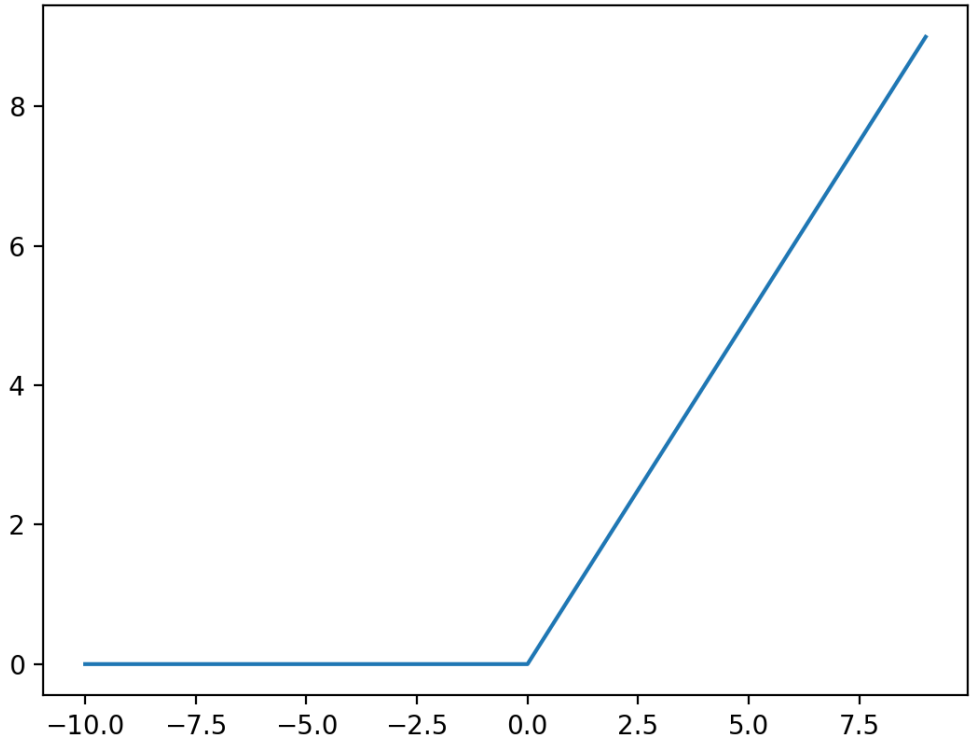


$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Sigmoid Function

Activation function that has derivatives

Activation Function : **What kinds are they?**

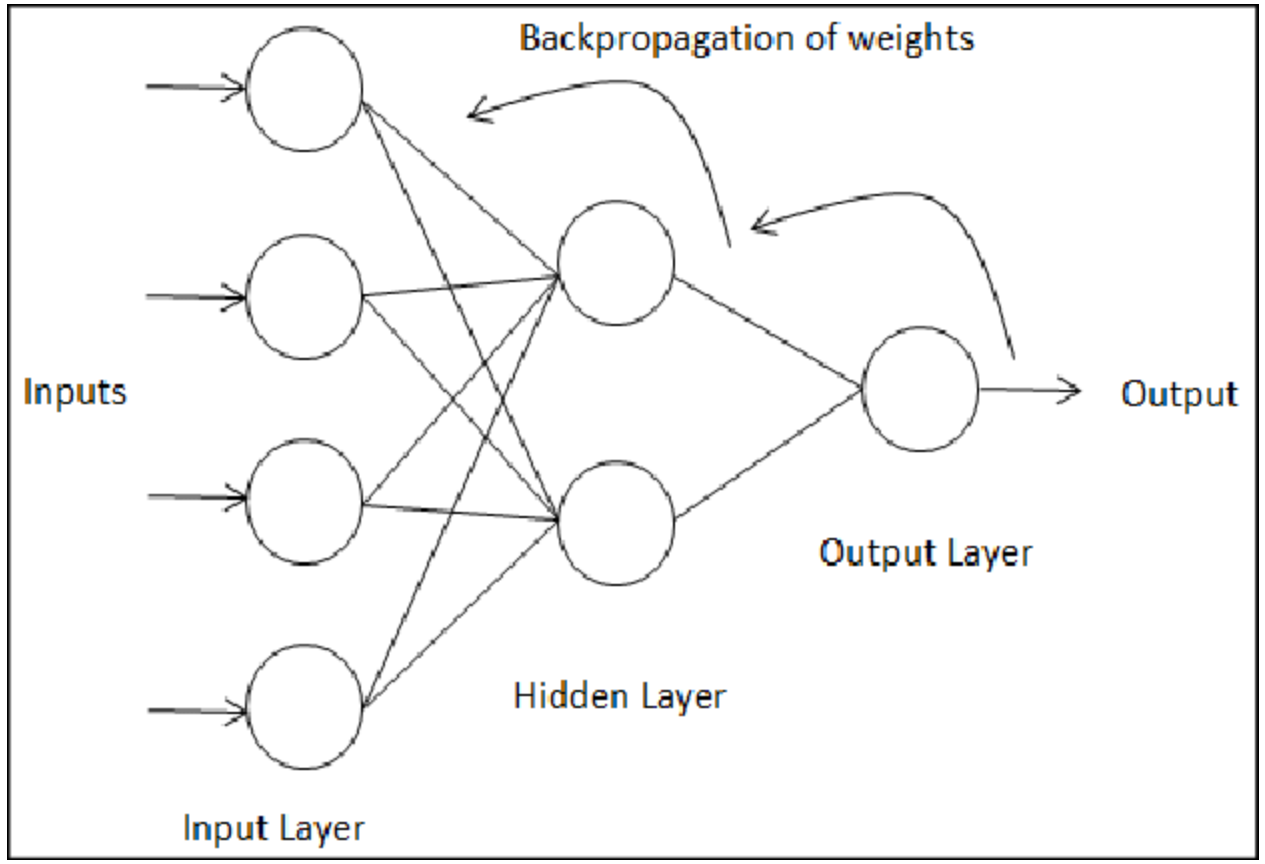


$$RELU(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x >= 0 \end{cases}$$

ReLU Function

Activation function recently used most

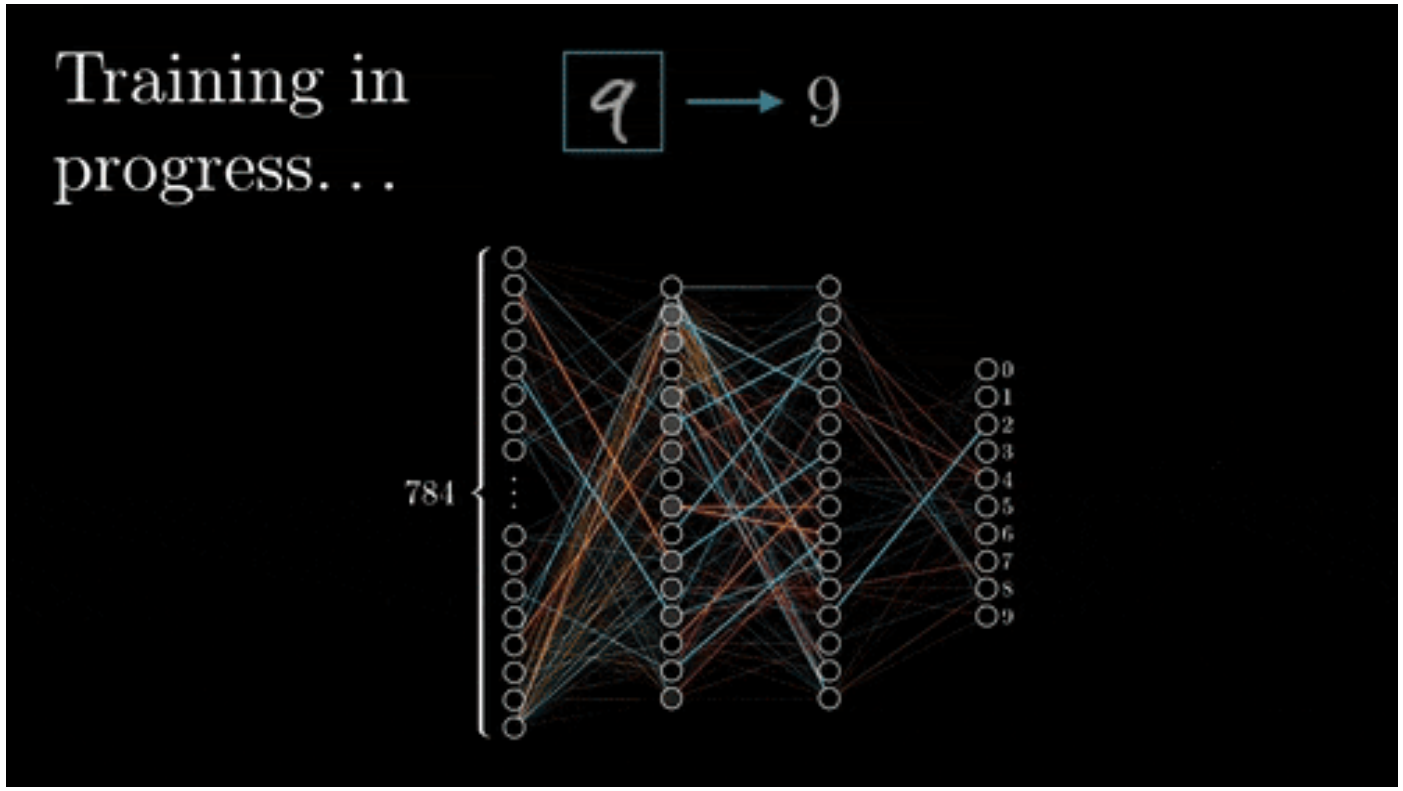
Error Back Propagation : **Why derivates?**



How can we train Multi Layer Perceptrons?

With Error Back Propagation

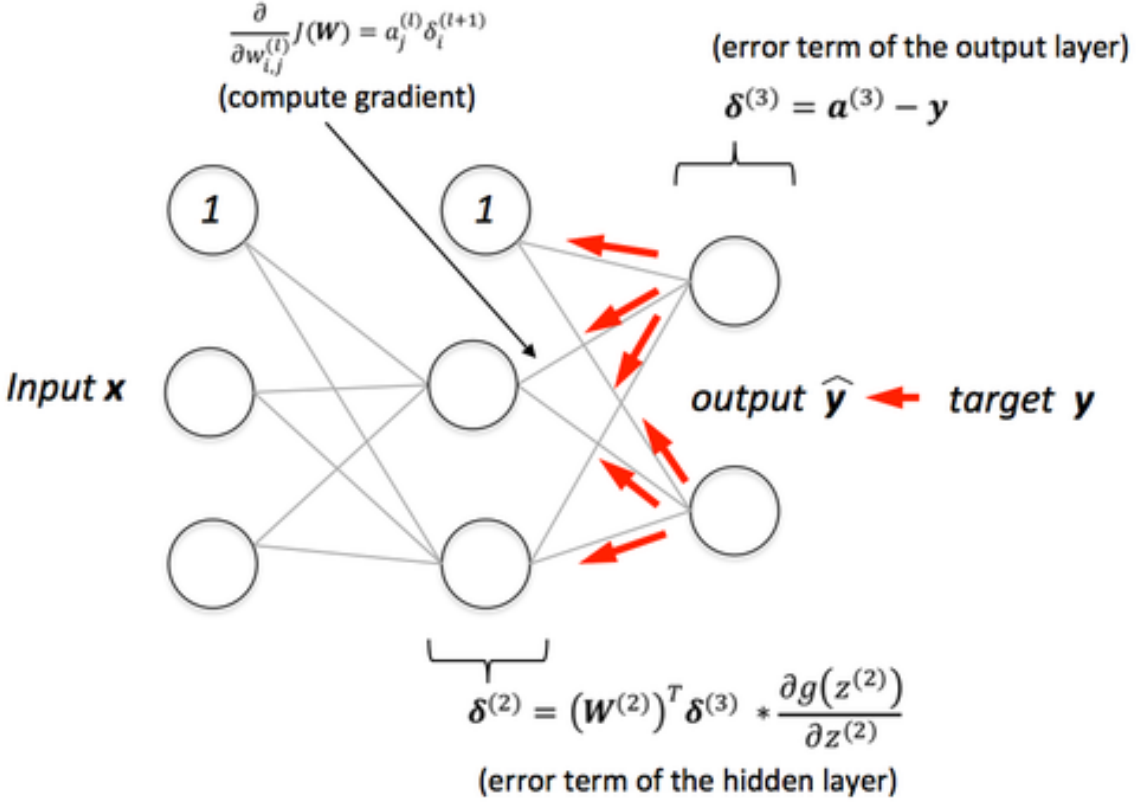
Error Back Propagation : **Why derivates?**

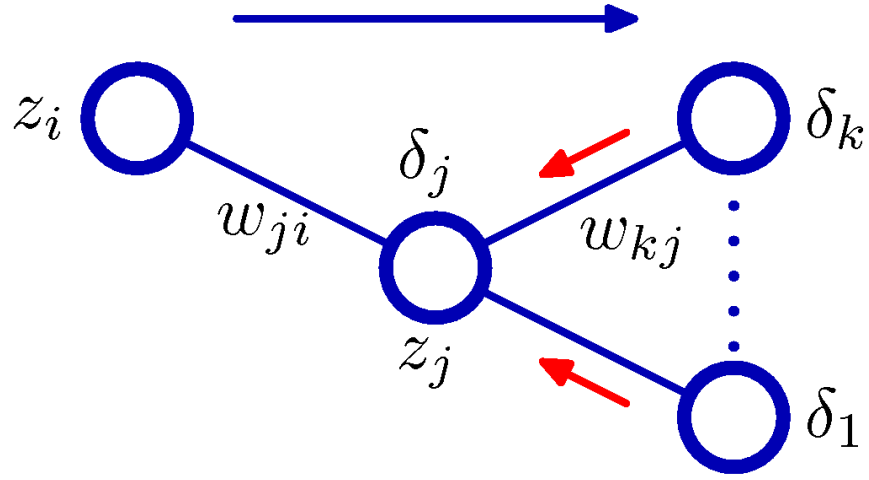


How can we train Multi Layer Perceptrons?

With Error Back Propagation

Error Back Propagation : **Why derivates?**





How can we train Multi Layer Perceptrons?

With Error Back Propagation

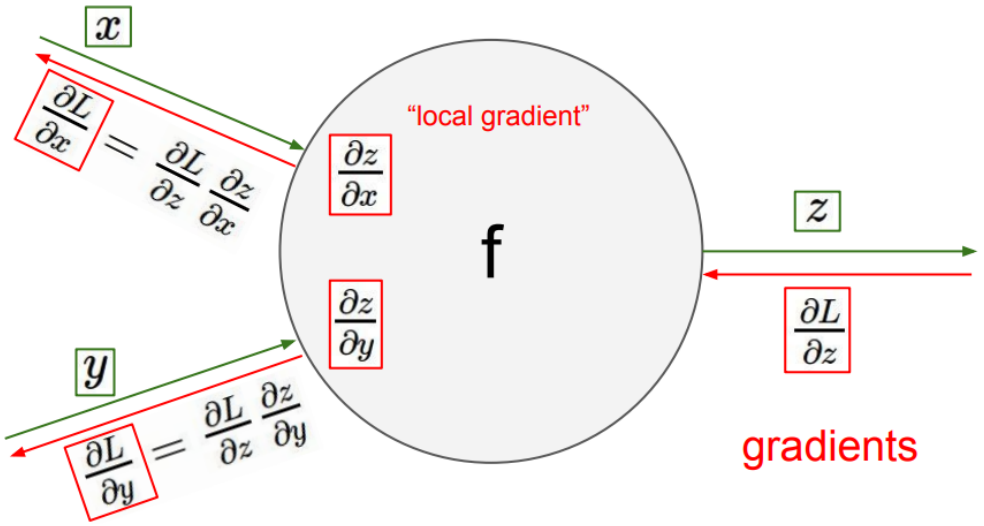
Error Back Propagation : **Why derivates?**

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial t} \\
&= (2x)(r\cos(t)) + (2)(2\sin(t)\cos(t)) \\
&= (2r\sin(t))(r\cos(t)) + 4\sin(t)\cos(t) \\
&= 2(r^2+2)\sin(t)\cos(t) \\
&= (r^2+2)\sin(2t).
\end{aligned}
$$

Chain Rule in Calculus

Error Backpropagation with Derivates

Error Back Propagation : **Why derivates?**



But , what if we want update w1, w2, w3 , w4 which will be having chain rule be like

$$\partial error\ (\ \partial pred\ (\ \partial h\ (\ \ \partial w1)\ )\ )$$

$$\frac{\partial(error)}{\partial(weight\ 1)} = \frac{\partial(error)}{\partial(pred)} * \frac{\partial(pred)}{\partial(h1)} * \frac{\partial(h1)}{\partial(w1)}$$

after ∂ these we will get

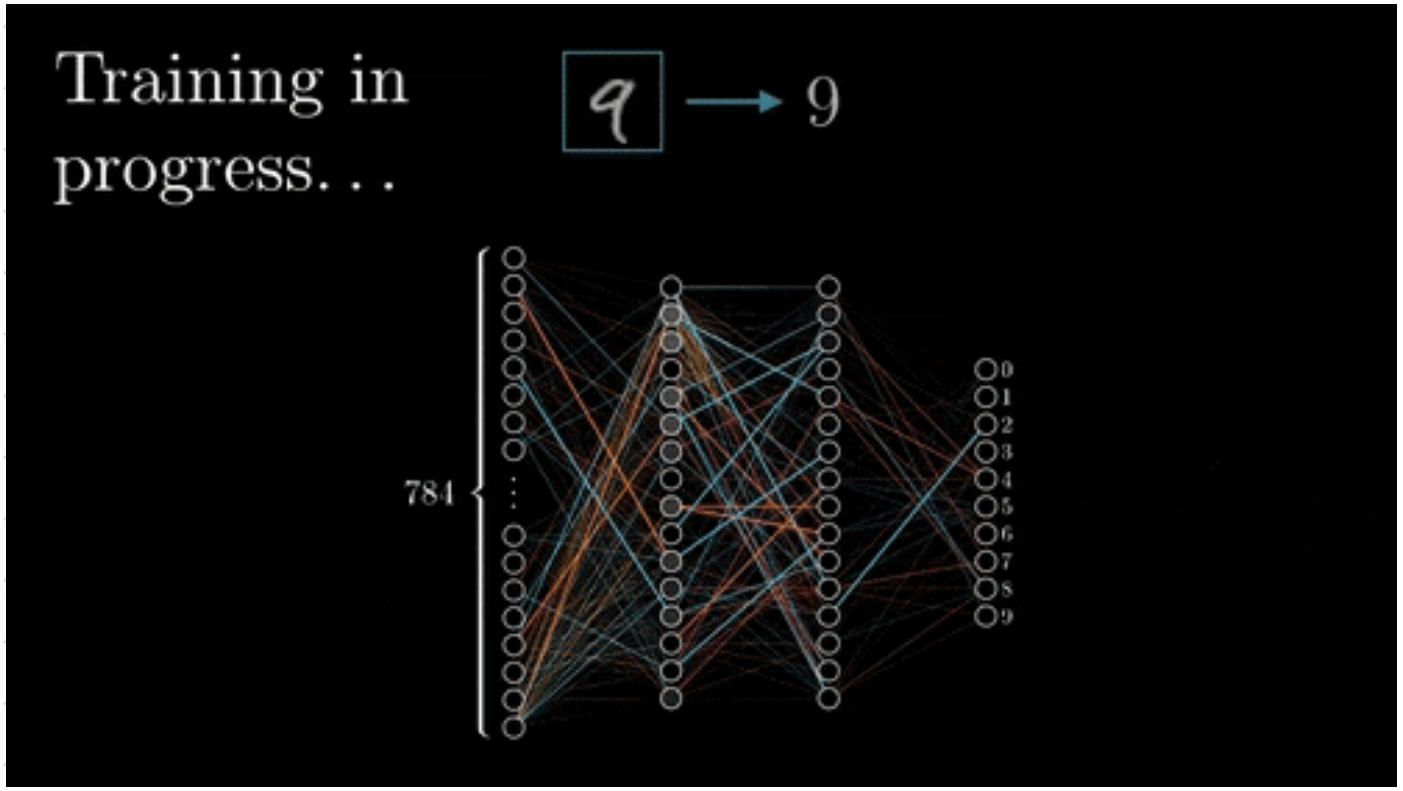
$$\frac{\partial(error)}{\partial(weight\ 1)} = \Delta\ .\ w5\ .\ i1$$

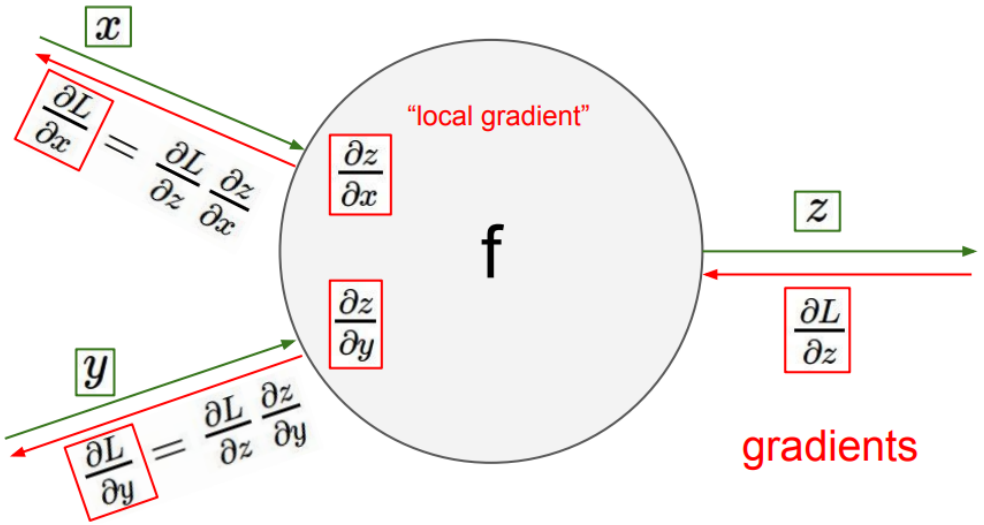
So , this is how the Chain rule plays a major role in Backpropagation

Chain Rule in Calculus

Error Backpropagation with Derivates

Error Back Propagation : **Why derivates?**





Chain Rule in Calculus

Error Backpropagation with Derivates

Error Back Propagation : **Why derivates?**

Assignment : MLP with MNIST Dataset

Lets Code!

With PyTorch


Thank You