

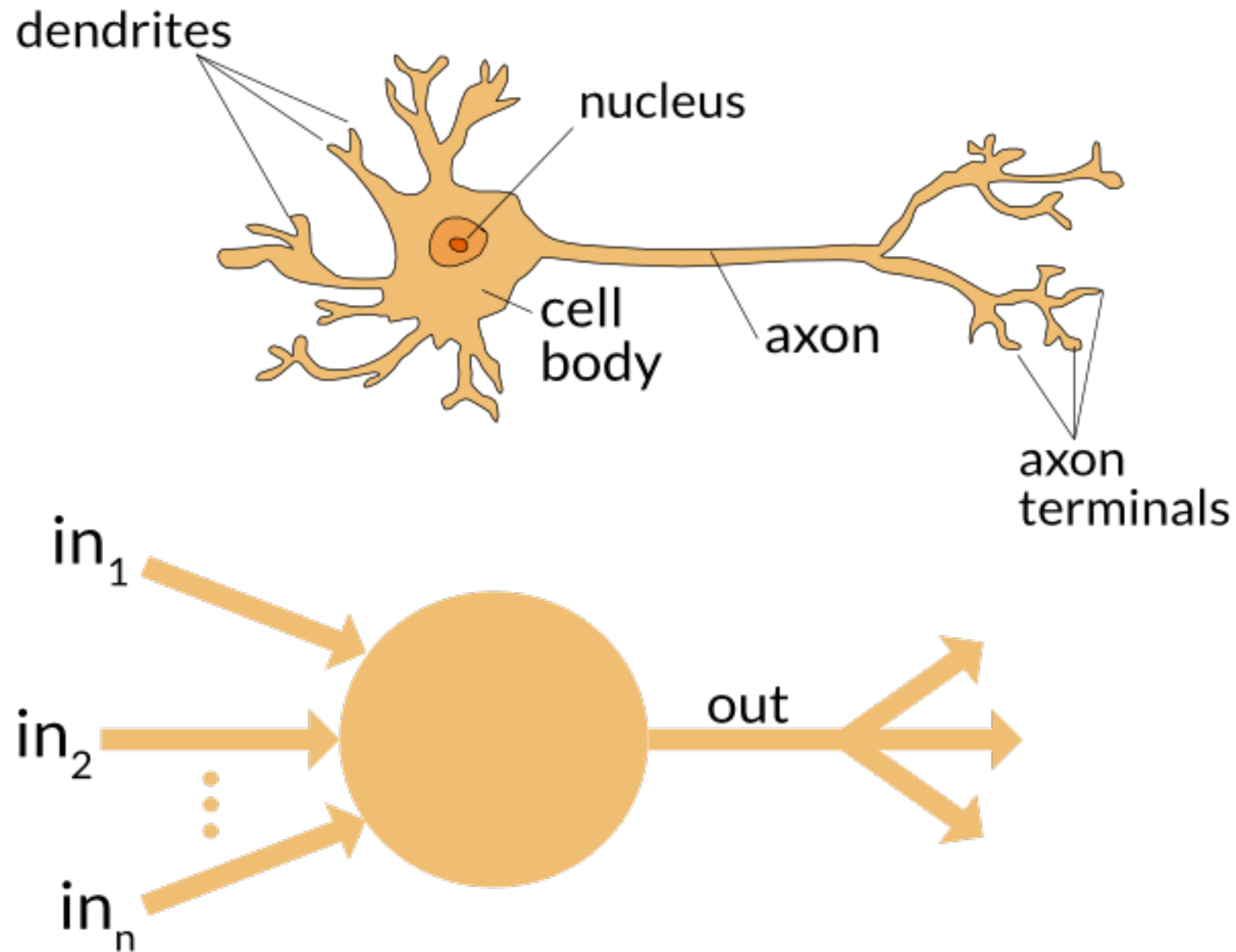
chapter 10

Introduction to Neural Networks

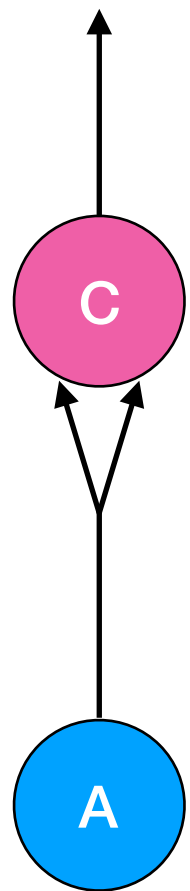
YongJin Jeon

Artificial Neural Network?

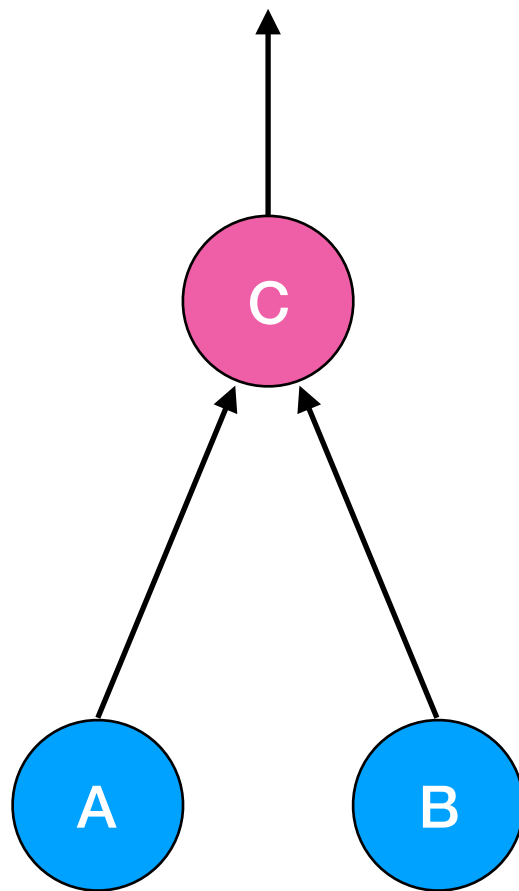
- Inspired by Human Brain



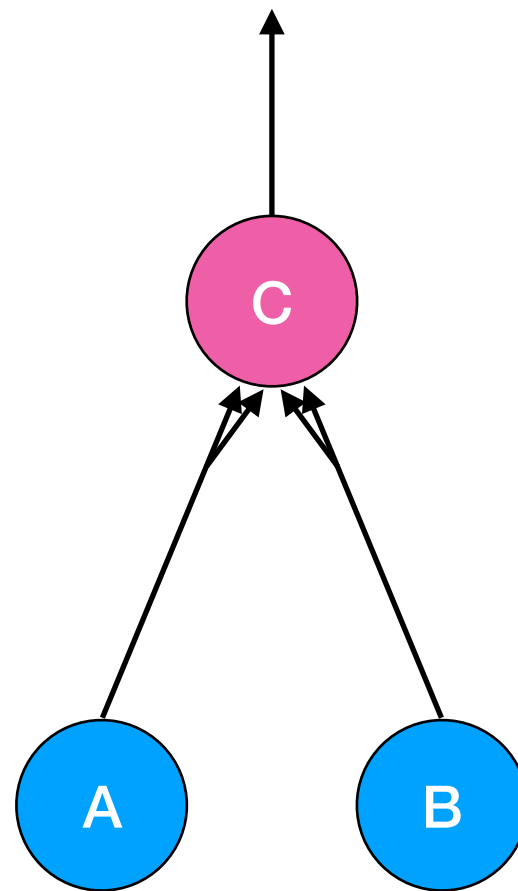
Logical Operation



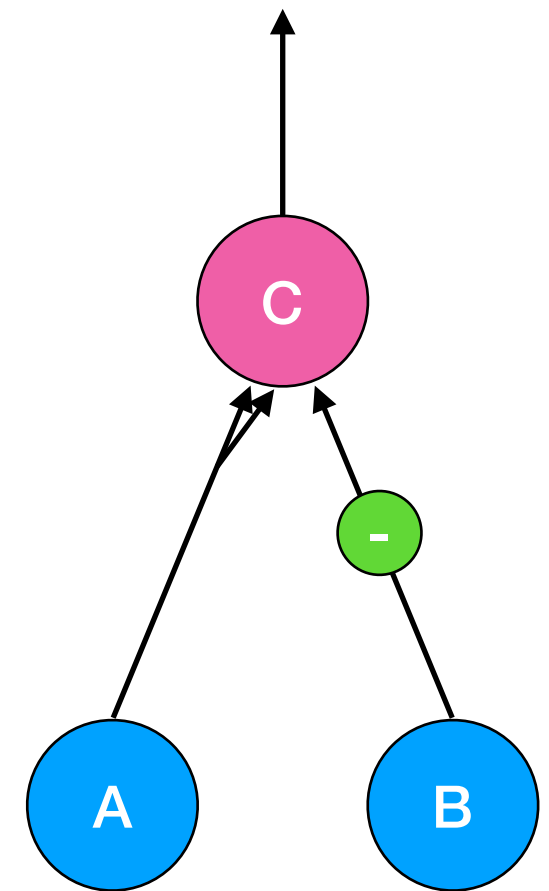
$$C = A$$



$$C = A \wedge B$$

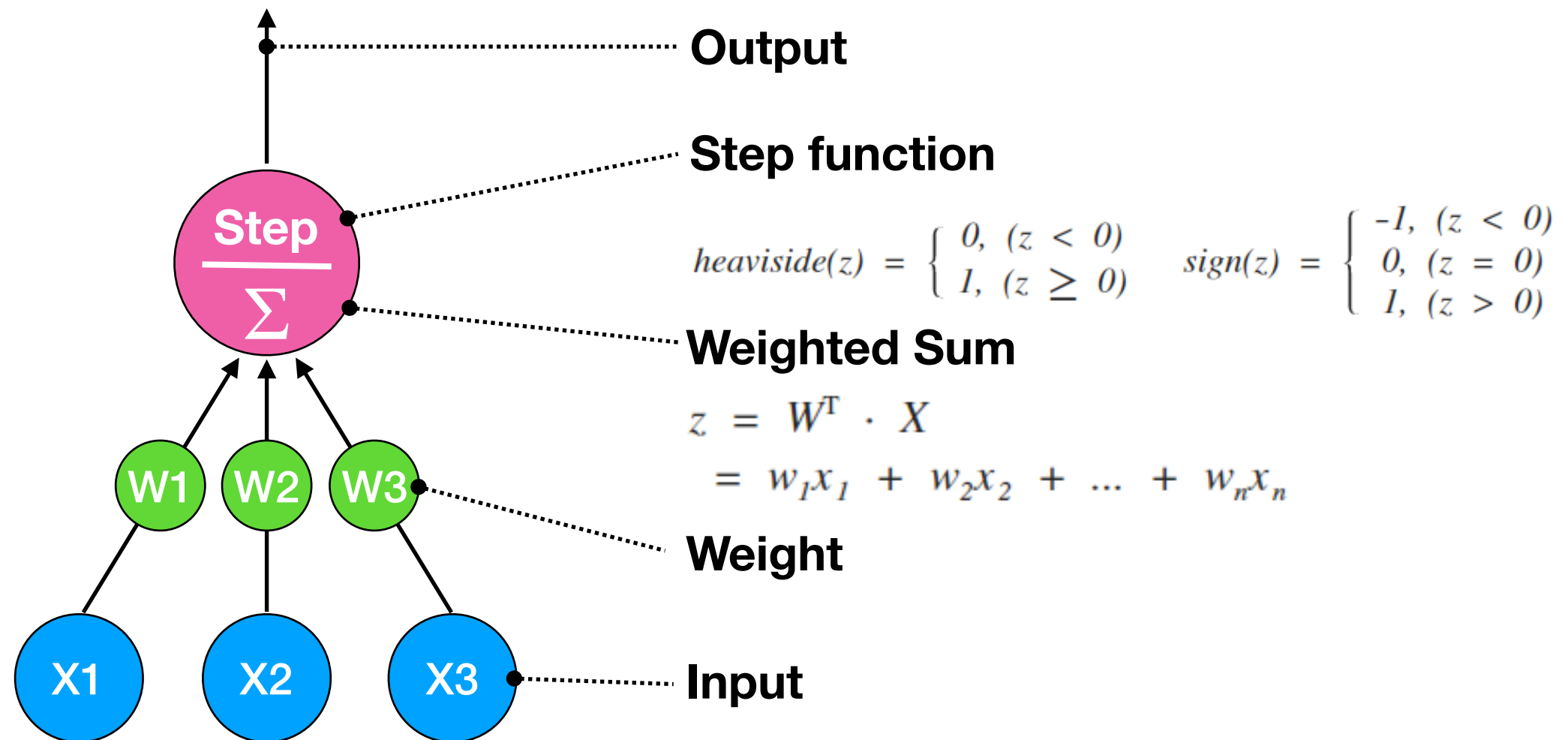


$$C = A \vee B$$



$$C = A \wedge \neg B$$

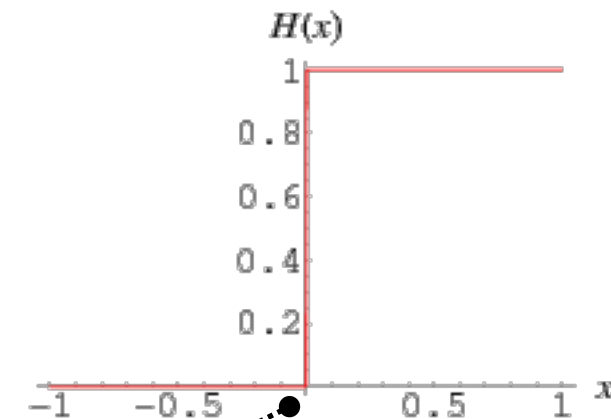
TLU(threshold logic unit)



TLU - Threshold

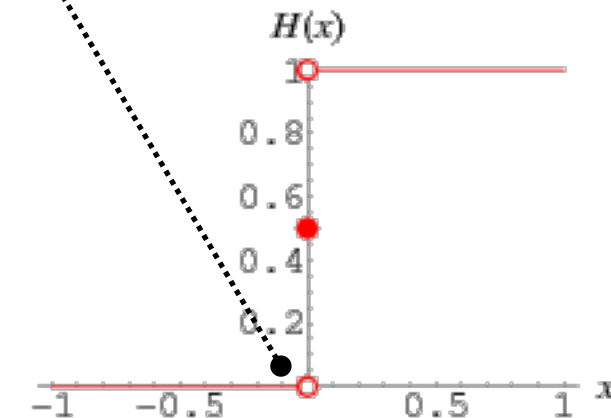
- Step Functions

$$\text{heaviside}(z) = \begin{cases} 0, & (z < 0) \\ 1, & (z \geq 0) \end{cases}$$



Threshold point

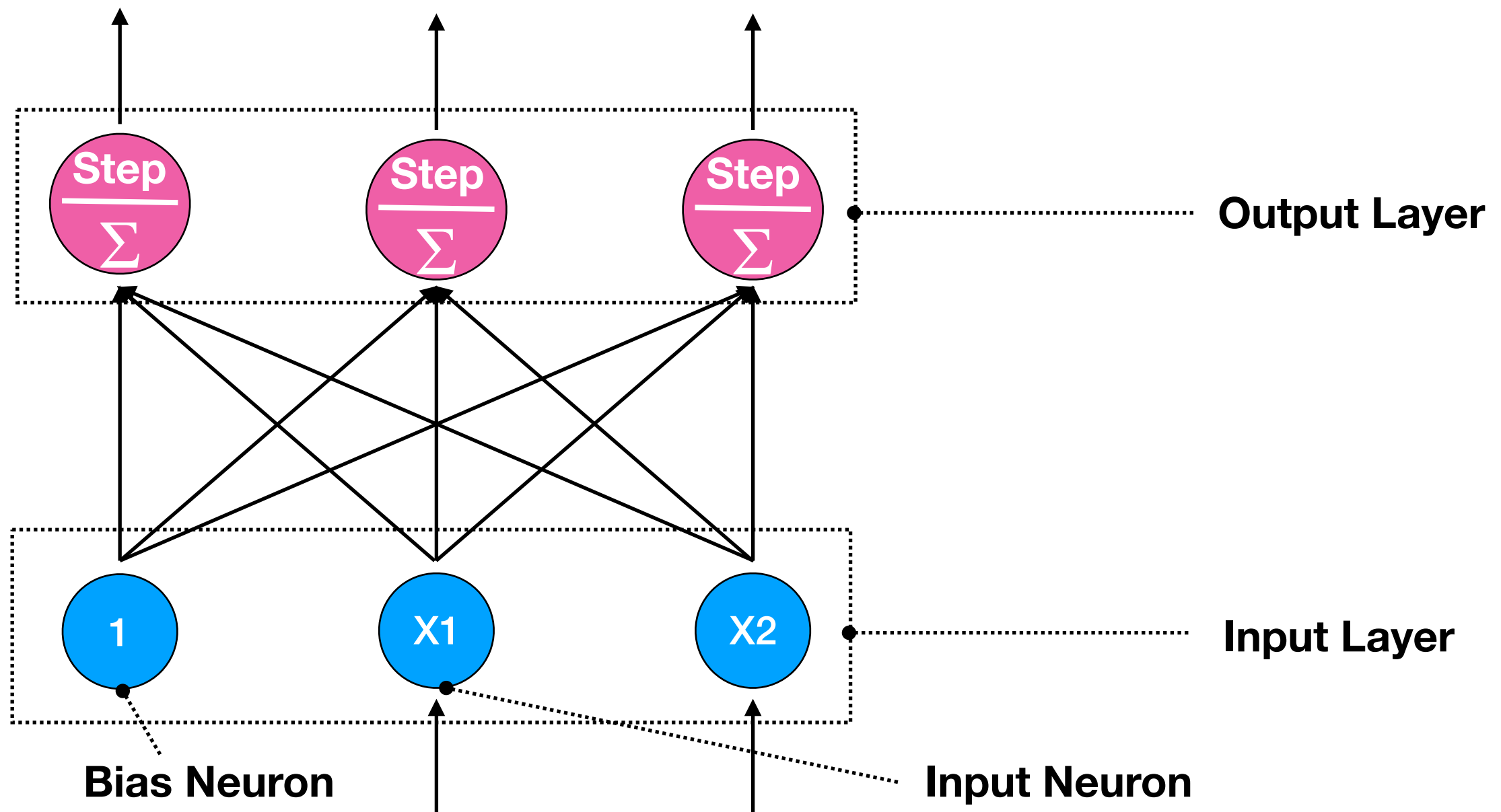
$$\text{sign}(z) = \begin{cases} -1, & (z < 0) \\ 0, & (z = 0) \\ 1, & (z > 0) \end{cases}$$



- One TLU could be used for simple Binary Linear Classification

Perceptron - Threshold

- One of the simplest Artificial Neural Network Structures
- Consist of one TLU layer



Perceptron - Training

- The Organization of Behavior, Donald Hebb(1949)
 - > connection between two neurons are getting stronger when one biological neuron activates the other one.
- Hebb's Rule, Siegrid Lowel
 - > neuron's that are activated by each other are connecte
 - d to each other
- Update rule of Perceptron

$$w_{ij}^{(next\ step)} = w_{ij} + \eta(y_j - \hat{y}_j)x_i$$

w_{ij} : weight of connection between i th input node and j th output node

x_i : value of i th input node

\hat{y}_j : value of j th output node

y_j : target value of j th output node

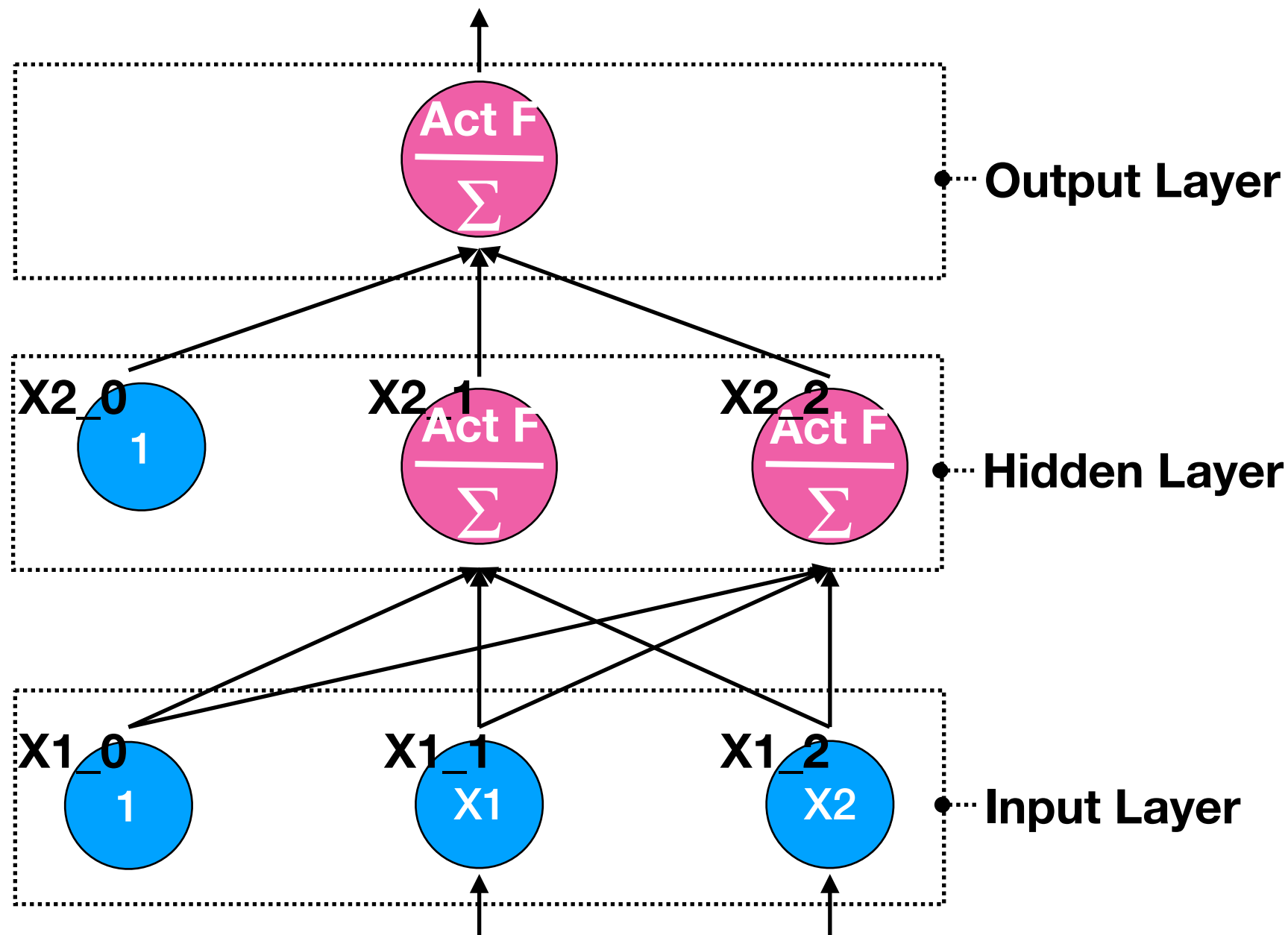
η : learning rate

Perceptron

- Each output neuron's decision boundary shows Linearity
 - > Perceptron can not learn complex pattern
 - > however Hebb's learning algorithm shows converging to the answer if training data could be separate by linear function (Perceptron convergence theorem)
- Perceptron predicts the class based on fixed threshold value and doesn't calculate probabilities unlike logistic regression
 - > for this reason most people prefer logistic regression over perceptron
- Limit of the perceptron
 - > can't solve simple problems such as 'XOR classification problem'

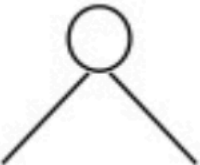
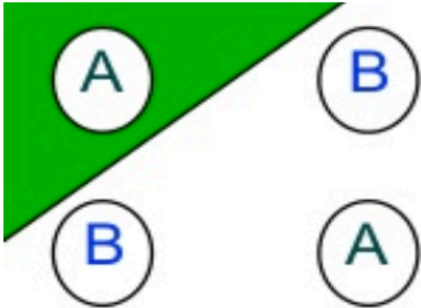
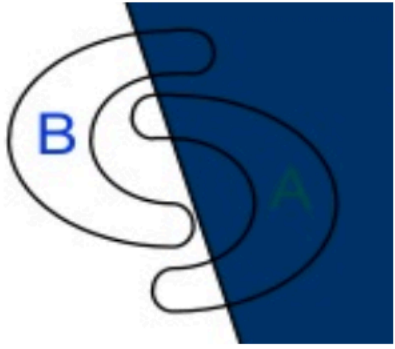

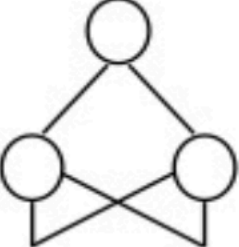
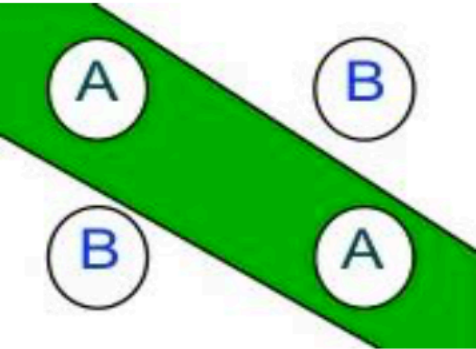
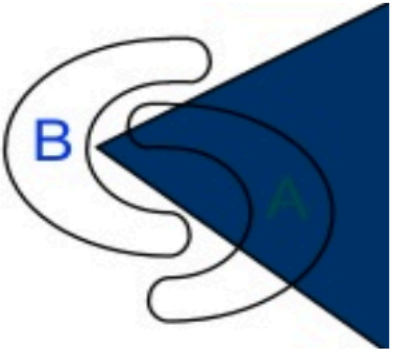

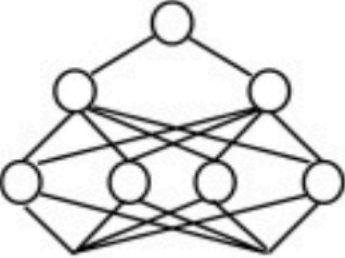
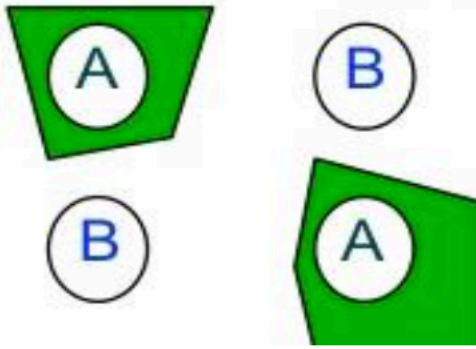

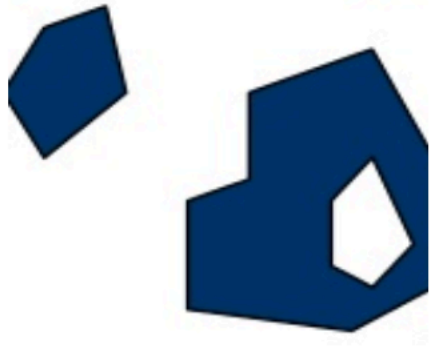
Multi Layer Perceptron

- Include more than one hidden layer
-> DNN : more than two hidden layers



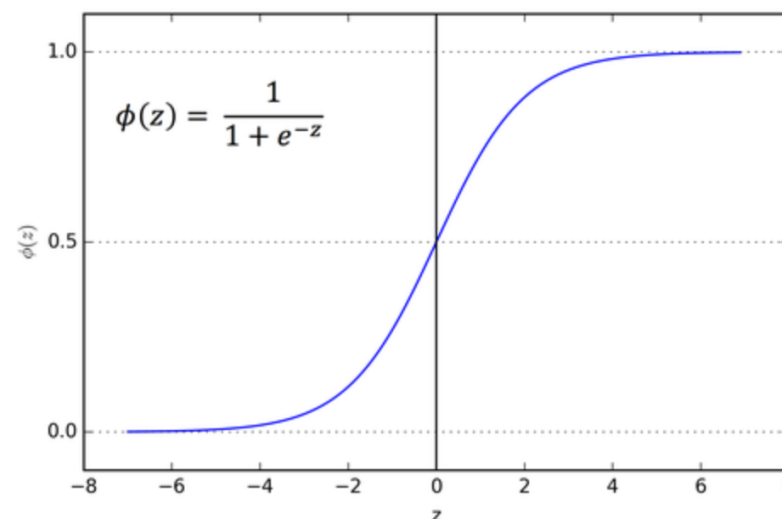
Multi Layer Perceptron

- Non Linearly Separable Problems

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

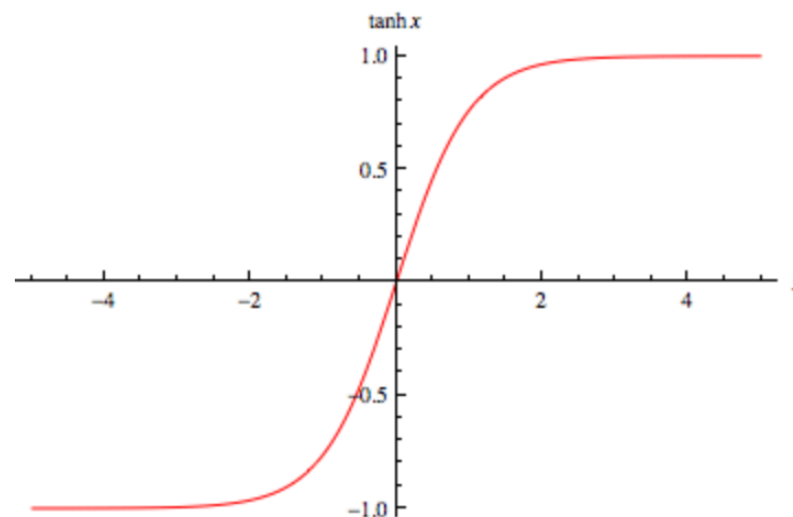
MLP - Activation Function

- Limit of step function
 - > binary input and output
 - > linear summation
 - > non stochastic
 - > Impossible temporal information processing
- Sigmoid function
 - > having S shaped curve
 - > differentiable function with bounded derivative and always having positive value



MLP - Activation Function

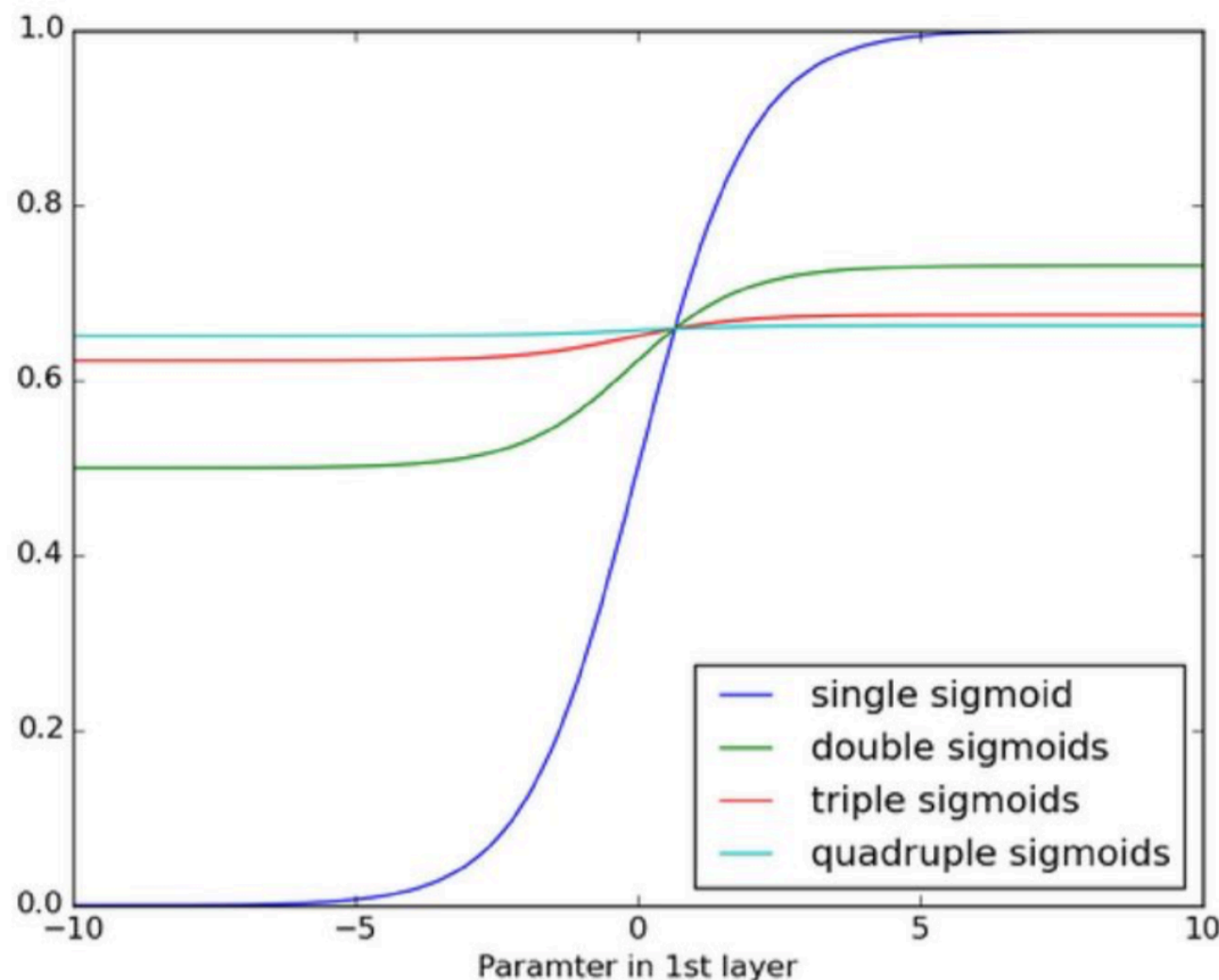
- Hyperbolic tangent function



- Artificial neuron usually takes sigmoid(logistic) or hyperbolic function cause logistic regression's cost function is convex and guaranteed to find the global cost minimum

MLP - Activation Function

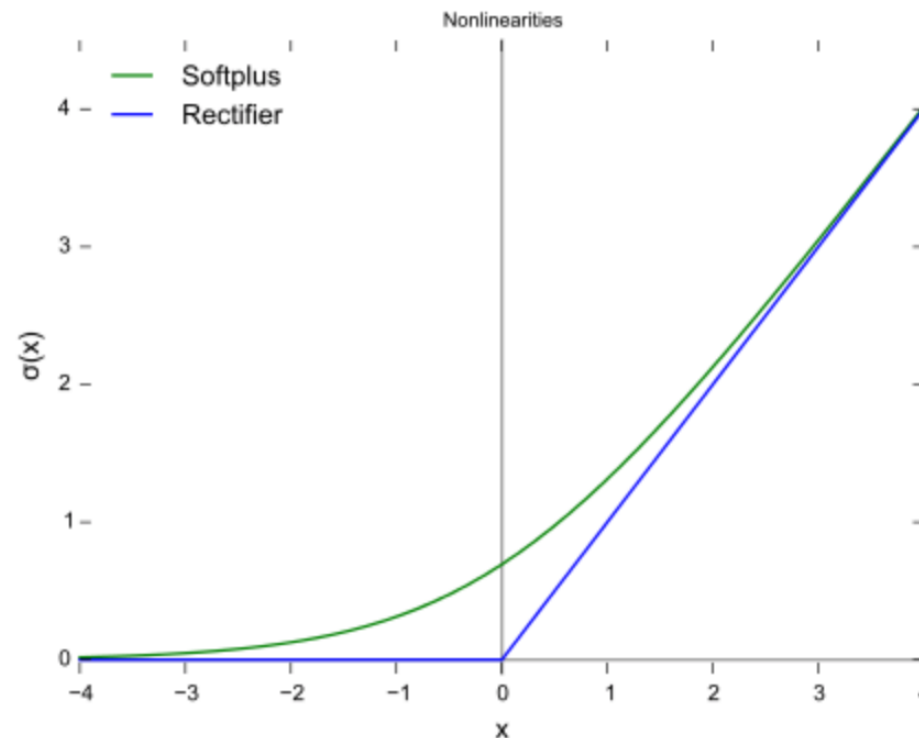
- Limit of sigmoid function
 - > can cause vanishing gradient (slow and inefficient learning) because of stacked multiple layers
 - > shows local minimum of the error function



with each subsequent layer the magnitude of the gradients gets exponentially smaller(vanishing), thus making the steps very small which results in very slow learning of the weights in the lower layers of a deep network

MLP - Activation Function

- ReLU(Rectified Linear Unit)

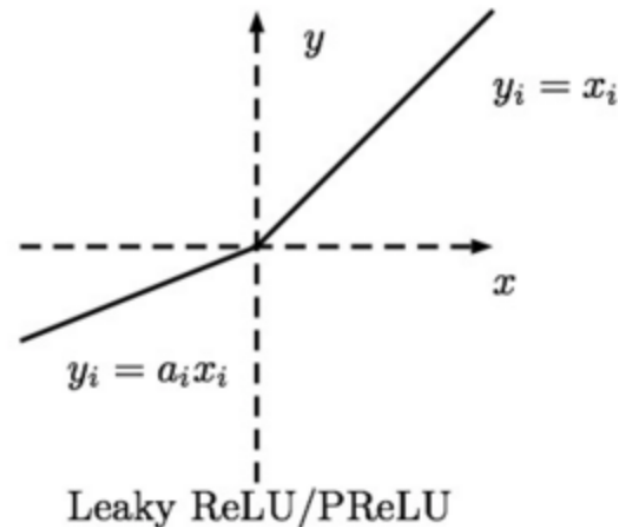


$$f = \begin{cases} (x < 0) & f(x) = 0 \\ (x \geq 0) & f(x) = x \end{cases}$$

- > simple implementation and low computation
- > accelerate the convergence(no vanishing gradient)
- > however relu can make model die(filled with zeros)

MLP - Activation Function

- Leaky ReLU(Rectified Linear Unit)



$$f(x) = \alpha x, (x < 0)$$

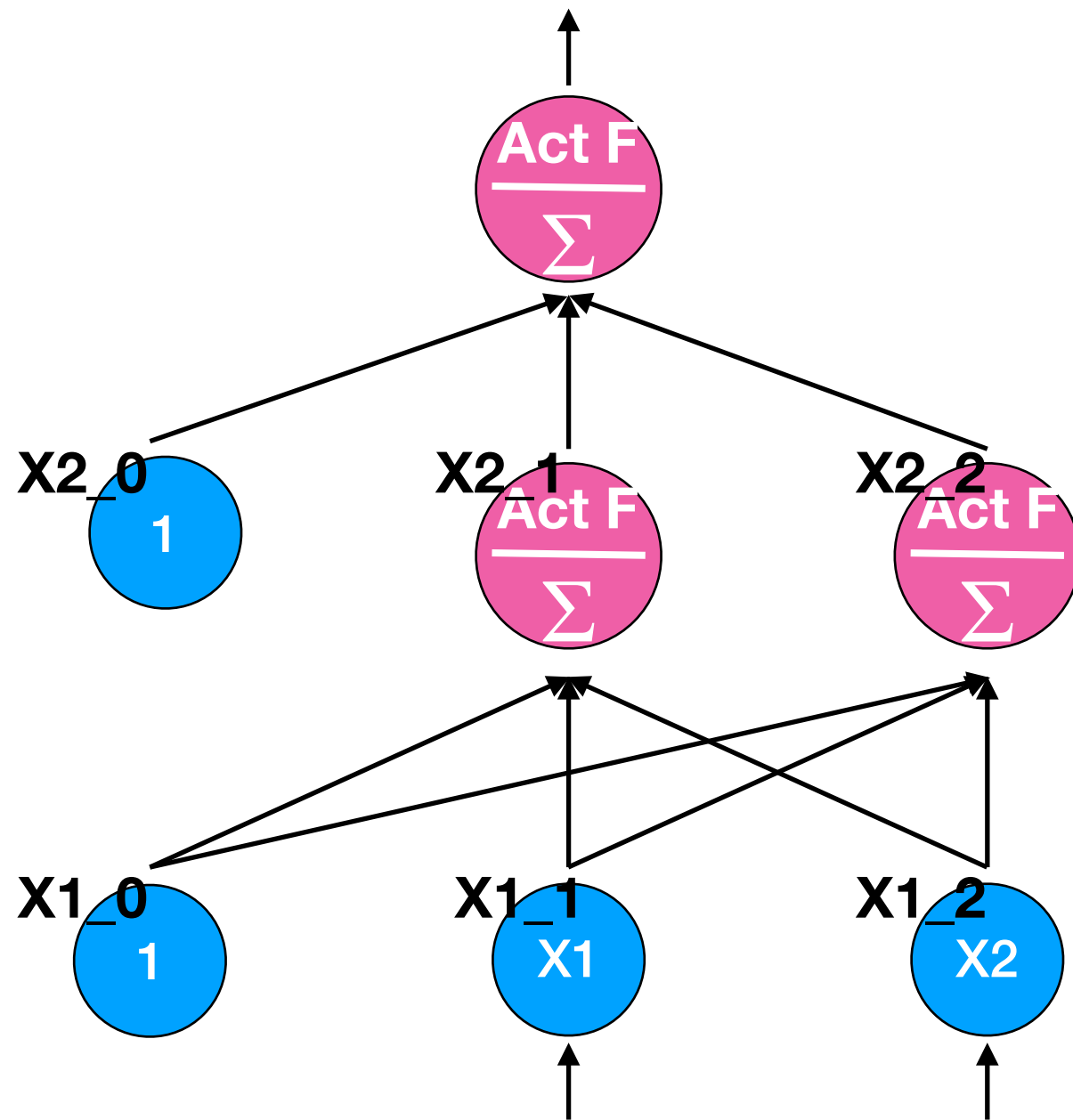
$$f(x) = x, (x \geq 0)$$

-> Instead of the function being zero when $x < 0$, it has a small negative slope to prevent dying problem

MLP - Backpropagation Algorithm

- Initialize weights
- Keep doing epochs
 - for each data in training set
 - * forward pass to compute $\rightarrow O(\text{output})$
 - * calculate loss $\rightarrow (d-O)$
 - : difference between expected out and real output
 - * backward pass to calculate all delta of each weight
 - * update all weights
- Until reach to the satisfying error rate

MLP - Training



Input : (1, 0)

Initial Weight

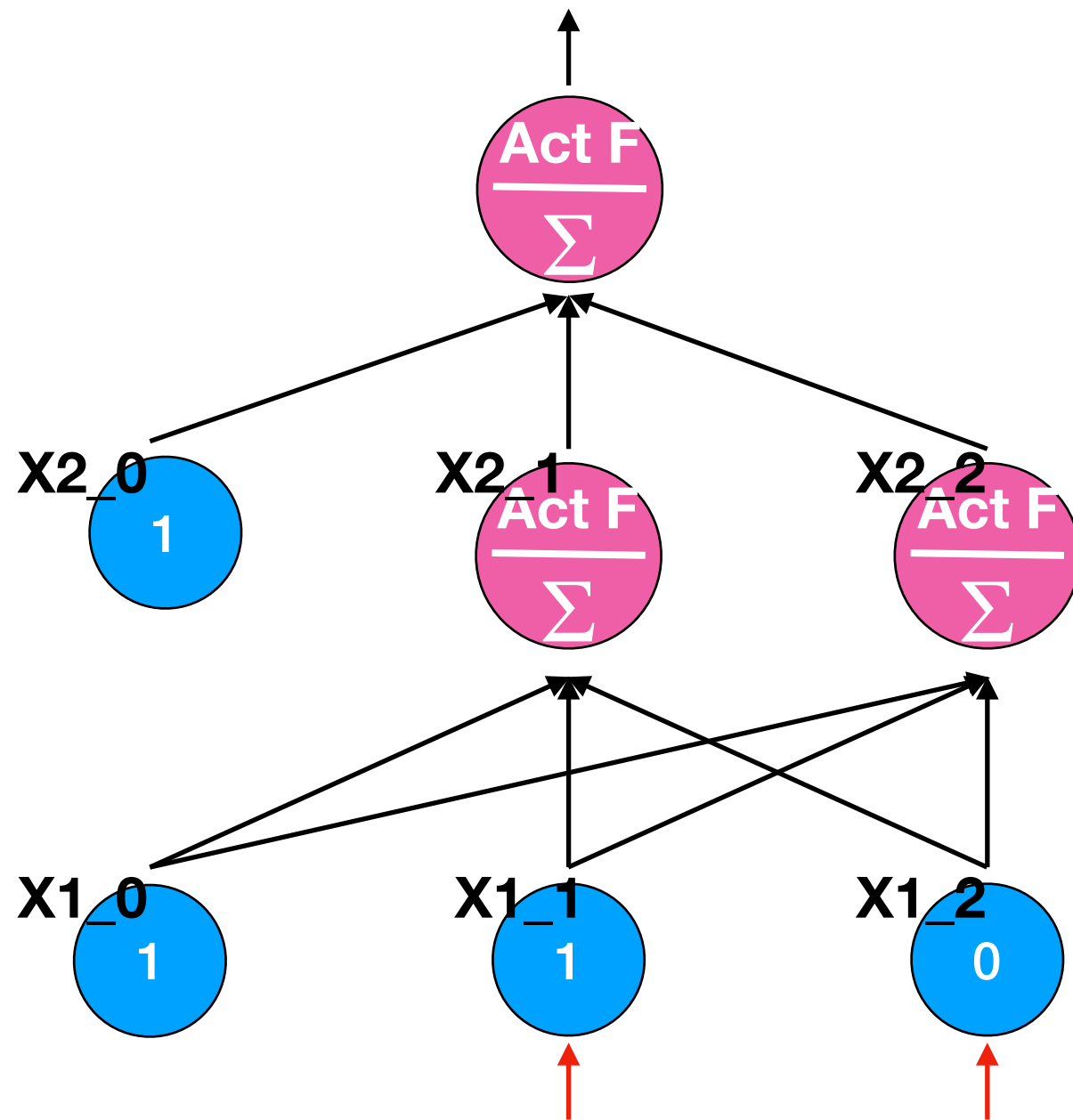
$w1_01 : 1, w1_02 : 0, w1_11 : 1$

$w1_12 : 0, w1_21 : 1, w1_22 : 0$

$w2_0 : 1, w2_1 : 0, w2_2 : 1$

Target : 0

MLP - Training



Input : (1, 0)

Initial Weight

$w1_01 : 1, w1_02 : 0, w1_11 : 1$

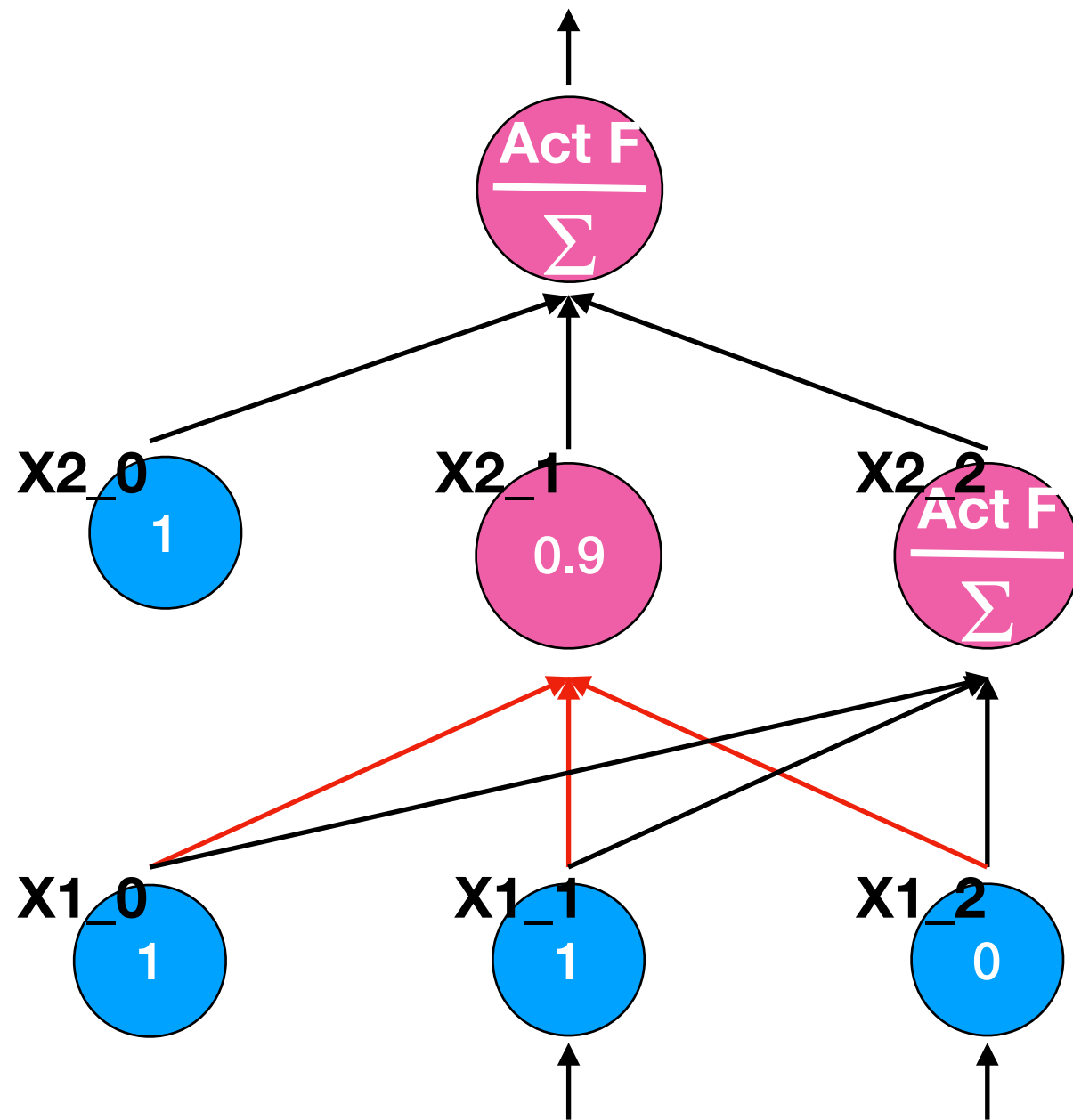
$w1_12 : 0, w1_21 : 1, w1_22 : 0$

$w2_0 : 1, w2_1 : 0, w2_2 : 1$

Target : 0

MLP - Training

$$h_i = \frac{1}{(1+e^{-S_x})} , \quad S_x = \sum x_k \cdot w_{ki}$$



Input : (1, 0)

Initial Weight

w1_01 : 1, w1_02 : 0, **w1_11 : 1**
w1_12 : 0, **w1_21 : 1**, w1_22 : 0
w2_0 : 1, w2_1 : 0, w2_2 : 1

Target : 0

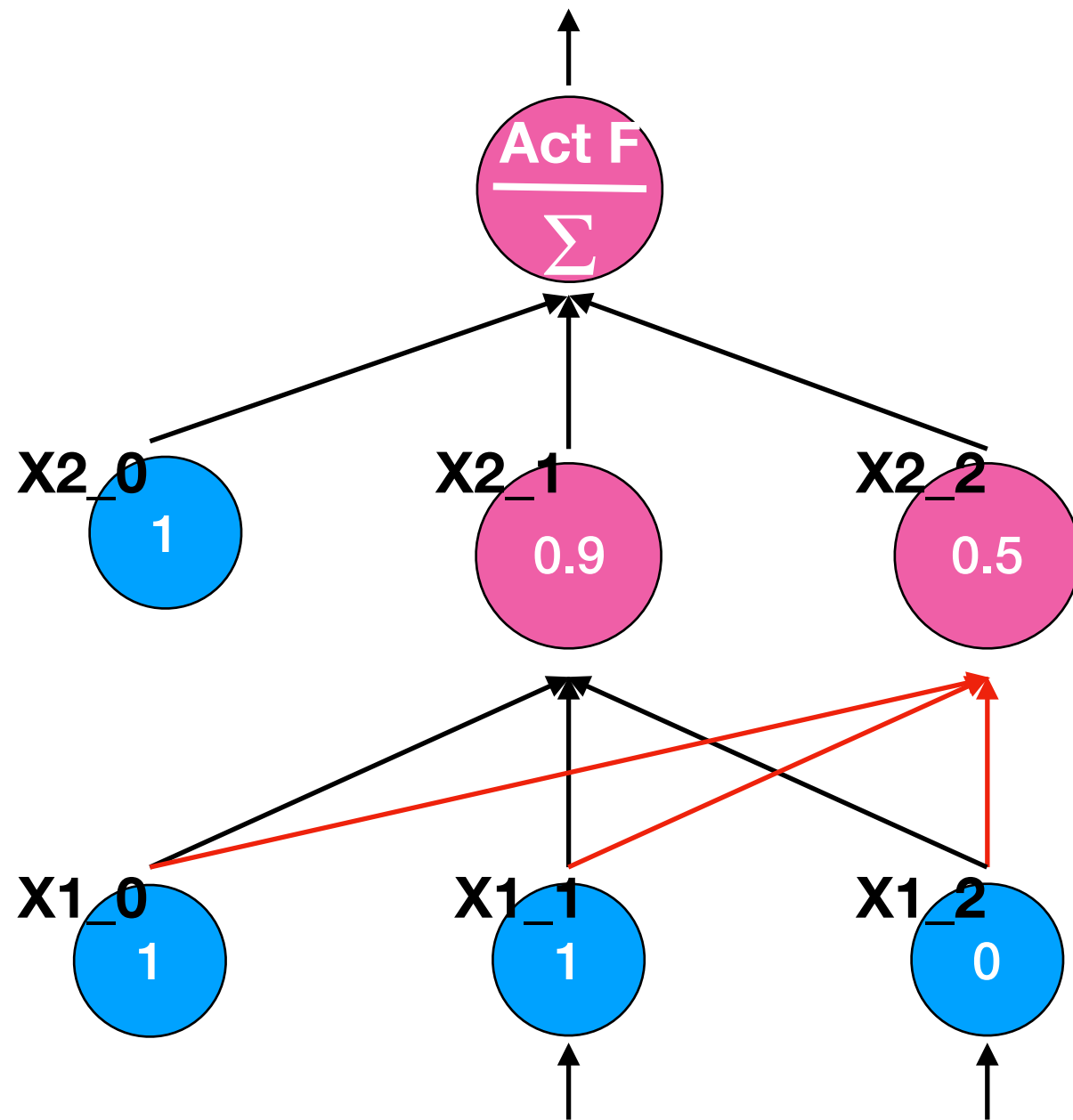
Input : (1, 0)

$$z1_1 = 1x1 + 1x1 + 0x1 \\ = 2$$

X2_2 = 0.9(0.88)
(use sigmoid function)

MLP - Training

$$h_i = \frac{1}{(1+e^{-S_x})} , \quad S_x = \sum x_k \cdot w_{ki}$$



Input : (1, 0)

Initial Weight

w1_01 : 1, **w1_02 : 0**, w1_11 : 1
w1_12 : 0, w1_21 : 1, **w1_22 : 0**
w2_0 : 1, w2_1 : 0, w2_2 : 1

Target : 0

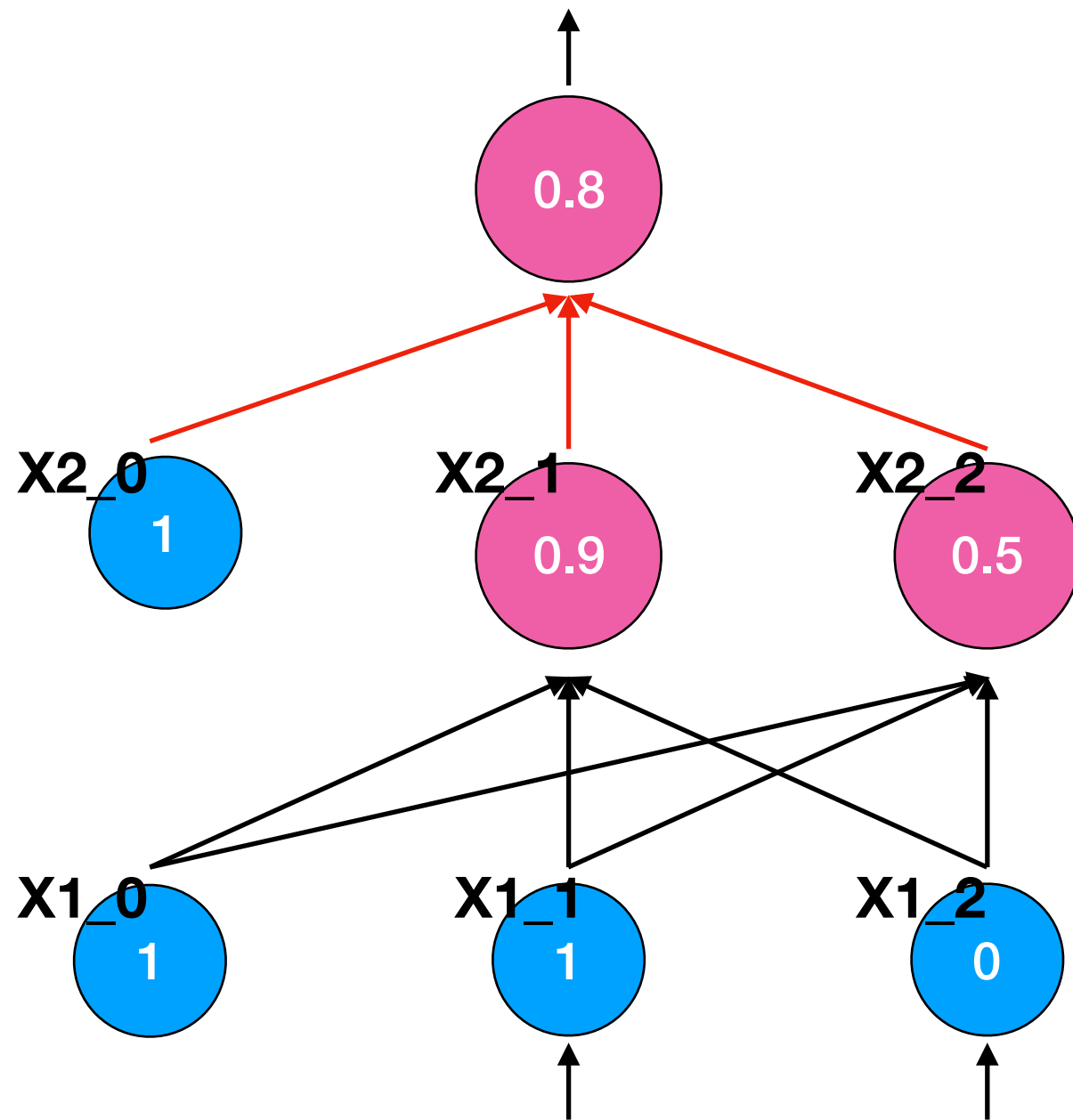
Input : (1, 0)

$$z1_2 = 1x0 + 1x0 + 0x0 \\ = 0$$

X2_2 = 0.5
 (use sigmoid function)

MLP - Training

$$O = \frac{1}{(1+e^{-S_h})} , \quad S_h = \sum h_i \cdot w_{ij}$$



Input : (1, 0)

Initial Weight

w1_01 : 1, w1_02 : 0, w1_11 : 1

w1_12 : 0, w1_21 : 1, w1_22 : 0

w2_0 : 1, w2_1 : 0, w2_2 : 1

Target : 0

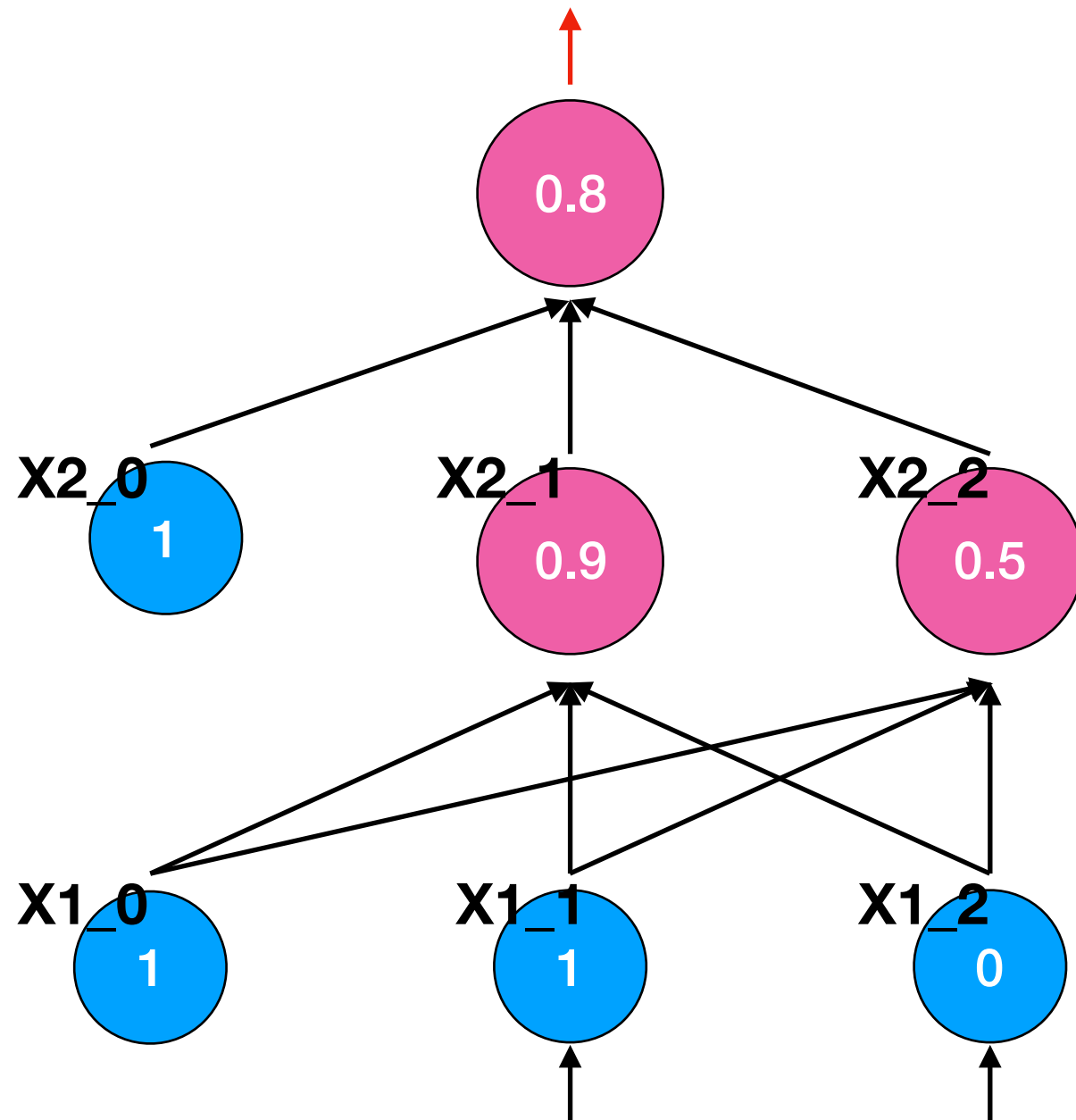
Input : (1, 0)

$$z2 = 1x1 + 0.9x0 + 0.5x1 \\ = 1.5$$

Y = 0.8(0.82)
(use sigmoid function)

MLP - Training

$$E = \frac{1}{2} \sum_i^n (d_j - O_j)^2, \quad E_j = \frac{1}{2} (d_j - O_j)^2$$



Input : (1, 0)

Initial Weight

w1_01 : 1, w1_02 : 0, w1_11 : 1

w1_12 : 0, w1_21 : 1, w1_22 : 0

w2_0 : 1, w2_1 : 0, w2_2 : 1

Target : 0

Output : 0.8

$$E = 0.5 \times (0 - 0.8)^2 \\ = 0.32$$

MLP - Training

$$\text{Let } E = \frac{1}{2} \sum_j^n (d_j - O_j)^2, \quad E_j = \frac{1}{2} (d_j - O_j)^2$$

- Adjust weights to output layers

$$\begin{aligned} \Delta w_{ij} &= -c \cdot \frac{\partial E}{\partial w_{ij}} = -c \cdot \frac{\partial E_j}{\partial O_j} \cdot \frac{\partial O_j}{\partial S_h} \cdot \frac{\partial S_h}{\partial w_{ij}} \\ &= c \cdot (d_j - O_j) \cdot O_j \cdot (1 - O_j) \cdot h_i \end{aligned}$$

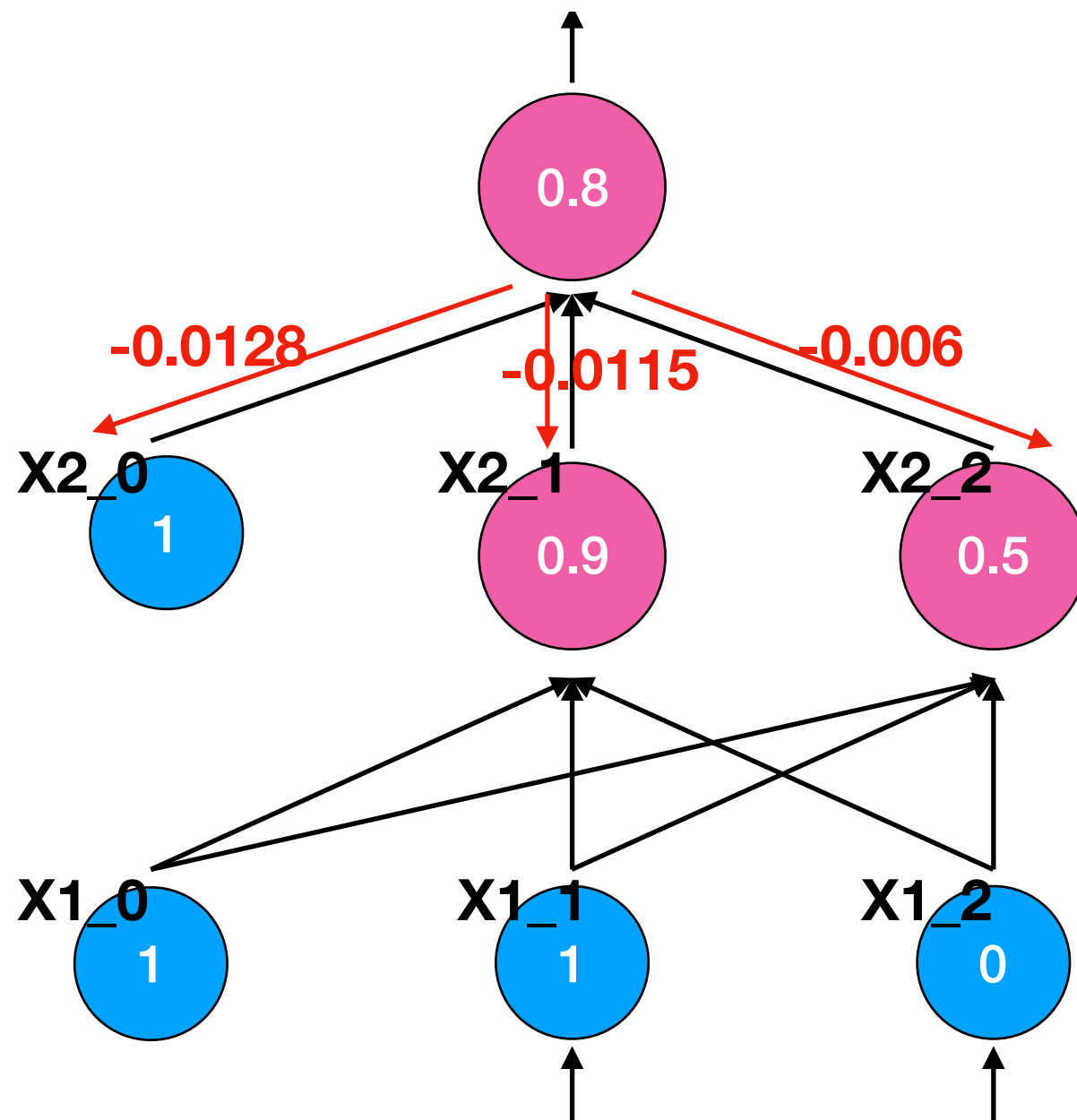
- Adjust weights to hidden layers

$$\begin{aligned} \Delta w_{ki} &= -c \cdot \frac{\partial E}{\partial w_{ki}} = -c \cdot \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial S_x} \cdot \frac{\partial S_x}{\partial w_{ki}} = -c \cdot \frac{\partial E}{\partial h_i} \cdot h_i \cdot (1 - h_i) \cdot x_k \\ &= -c \cdot \sum_j \frac{\partial E_j}{\partial h_i} \cdot h_i \cdot (1 - h_i) \cdot x_k = -c \cdot \sum_j \frac{\partial E_j}{\partial O_j} \cdot \frac{\partial O_j}{\partial S_h} \cdot \frac{\partial S_h}{\partial h_i} \cdot h_i \cdot (1 - h_i) \cdot x_k \\ &= c \cdot \left(\sum_j ((d_j - O_j) \cdot O_j \cdot (1 - O_j) \cdot w_{ij}) \right) \cdot h_i \cdot (1 - h_i) \cdot x_k \end{aligned}$$

MLP - Training

$$\Delta w_{ij} = -c \cdot \frac{\partial E}{\partial w_{ij}} = -c \cdot \frac{\partial E_j}{\partial O_j} \cdot \frac{\partial O_j}{\partial S_h} \cdot \frac{\partial S_h}{\partial w_{ij}}$$

$$= c \cdot (d_j - O_j) \cdot O_j \cdot (1 - O_j) \cdot h_i$$



Initial Weight

w1_01 : 1, w1_02 : 0, w1_11 : 1

w1_12 : 0, w1_21 : 1, w1_22 : 0

w2_0 : 0.99, w2_1 : -0.01, w2_2 : 0.99

Target : 0

Learning rate = 0.1

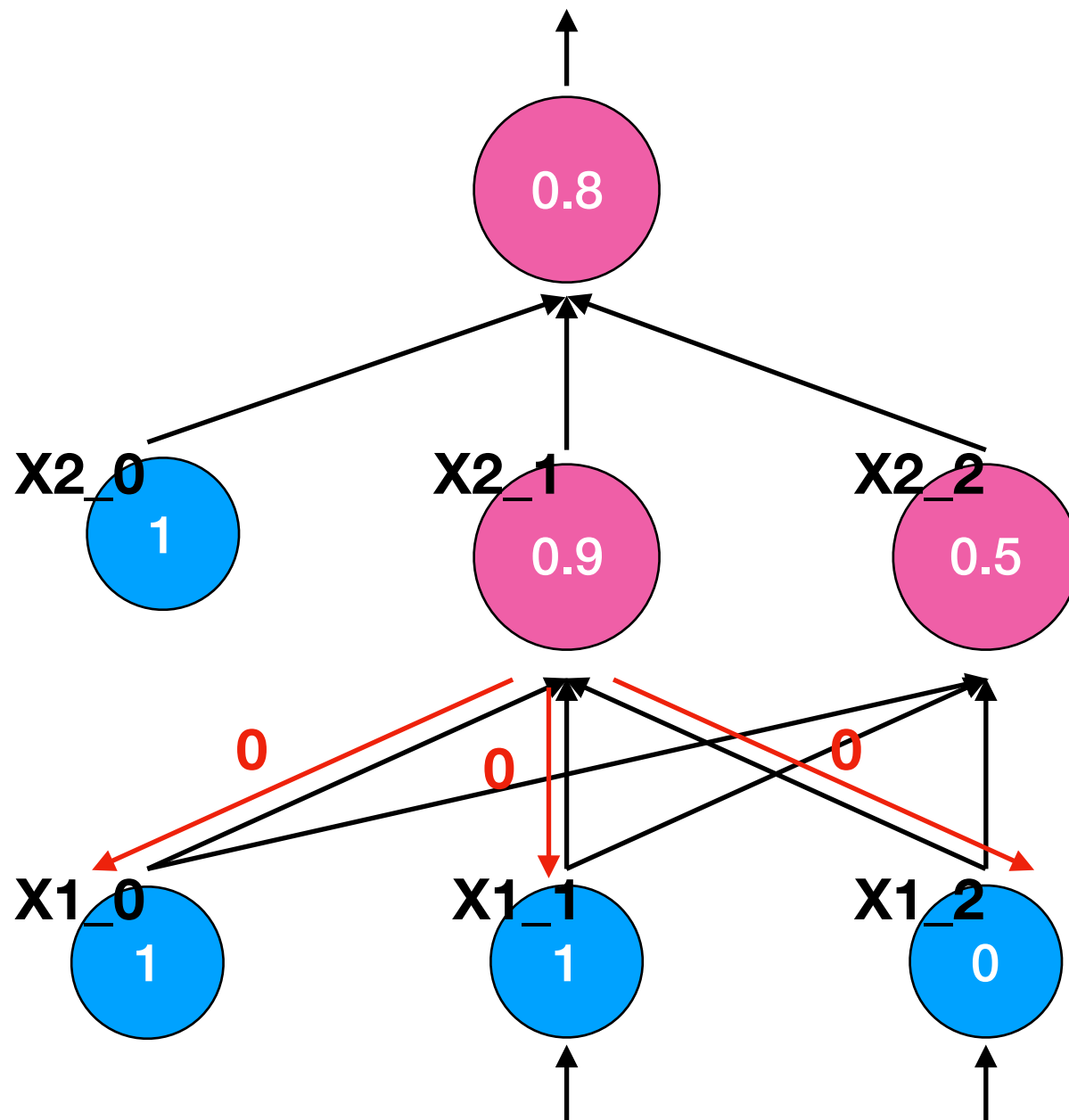
delta2_0 : 0.1 x (-0.8) x 0.8 x 0.2 x 1
= -0.0128

delta2_1 : 0.1 x (-0.8) x 0.8 x 0.2 x 0.9
= -0.0115

delta2_2 : 0.1 x (-0.8) x 0.8 x 0.2 x 0.5
= -0.006

MLP - Training

$$\Delta w_{ki} = c \cdot \left(\sum_j ((d_j - O_j) \cdot O_j (1 - O_j) \cdot w_{ij}) \right) \cdot h_i \cdot (1 - h_i) \cdot x_k$$



Initial Weight

w1_01 : 1, w1_02 : 0, **w1_11 : 1**

w1_12 : 0, **w1_21 : 1**, w1_22 : 0

w2_0 : 0.99, w2_1 : -0.01, w2_2 : 0.99

Target : 0

Learning rate = 0.1

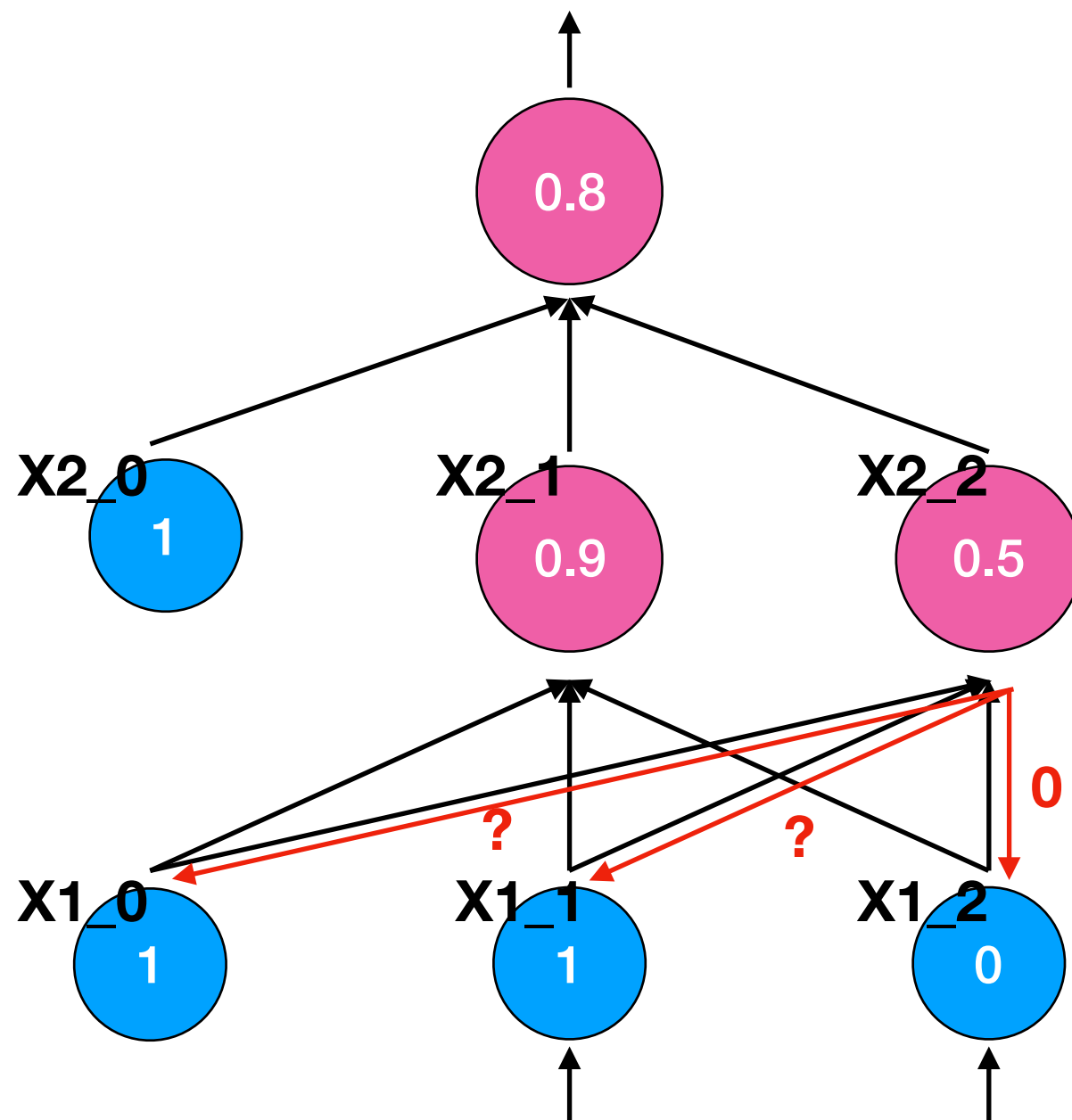
delta1_01 : 0.1 x (-0.8) x 0.8 x (-0.01) x
0 x 0.9 x 0.1 x 1
= 0

delta1_11 : 0.1 x (-0.8) x 0.8 x (-0.01) x
0 x 0.9 x 0.1 x 1
= 0

delta1_21 : 0.1 x (-0.8) x 0.8 x (-0.01) x
0 x 0.9 x 0.1 x 0

MLP - Training

$$\Delta w_{ki} = c \cdot \left(\sum_j ((d_j - O_j) \cdot O_j (1 - O_j) \cdot w_{ij}) \right) \cdot h_i \cdot (1 - h_i) \cdot x_k$$



Initial Weight

w1_01 : 1, w1_02 : 0, w1_11 : 1

w1_12 : 0, w1_21 : 1, w1_22 : 0

w2_0 : 0.99, w2_1 : -0.01, w2_2 : 0.99

Target : 0

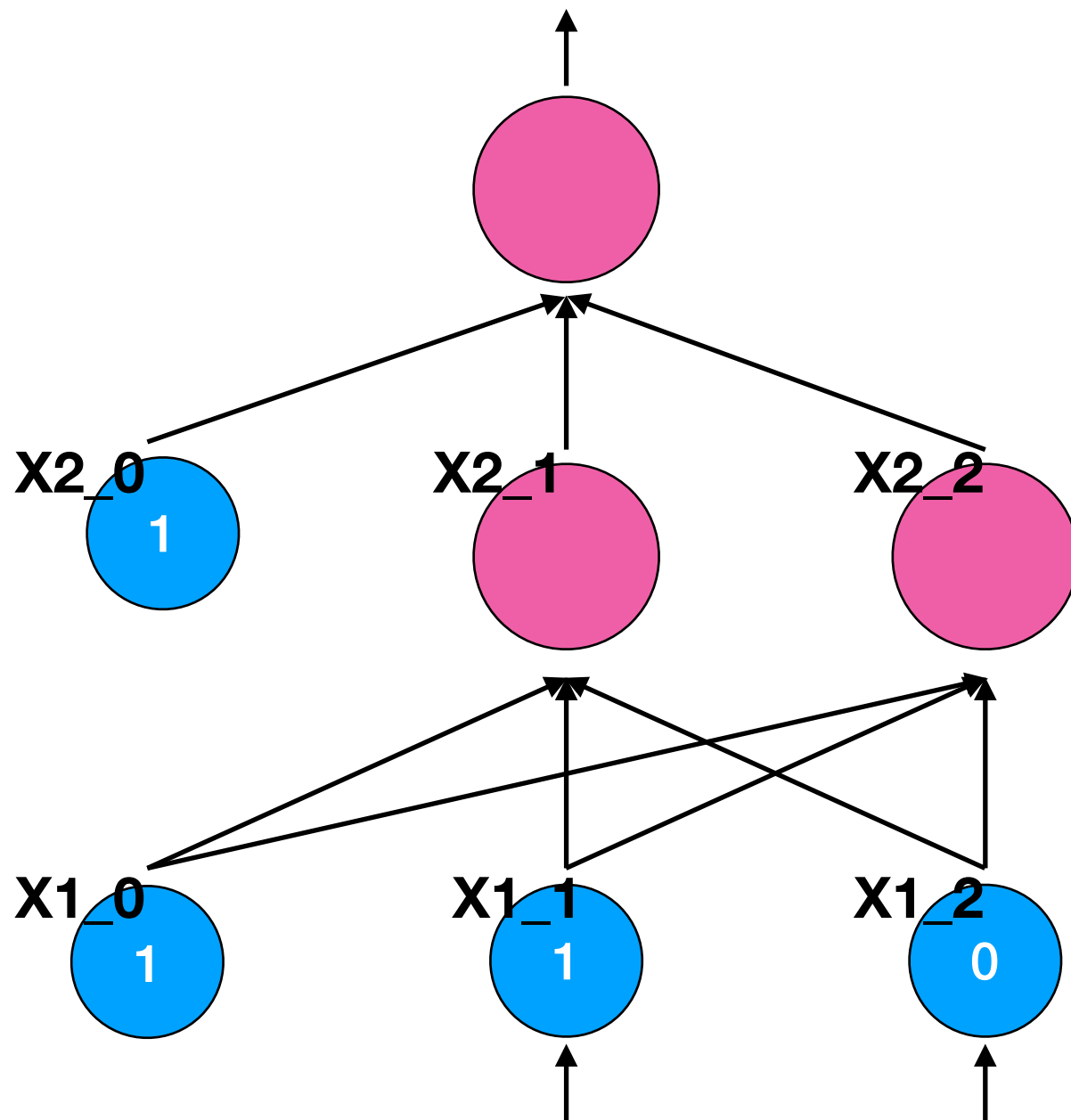
Learning rate = 0.1

delta1_02 : ?

delta1_12 : ?

delta1_22 : 0

MLP - Training



First step Initial Weight

$w1_01 : 1, w1_02 : 0, w1_11 : 1$

$w1_12 : 0, w1_21 : 1, w1_22 : 0$

$w2_0 : 1, w2_1 : 0, w2_2 : 1$



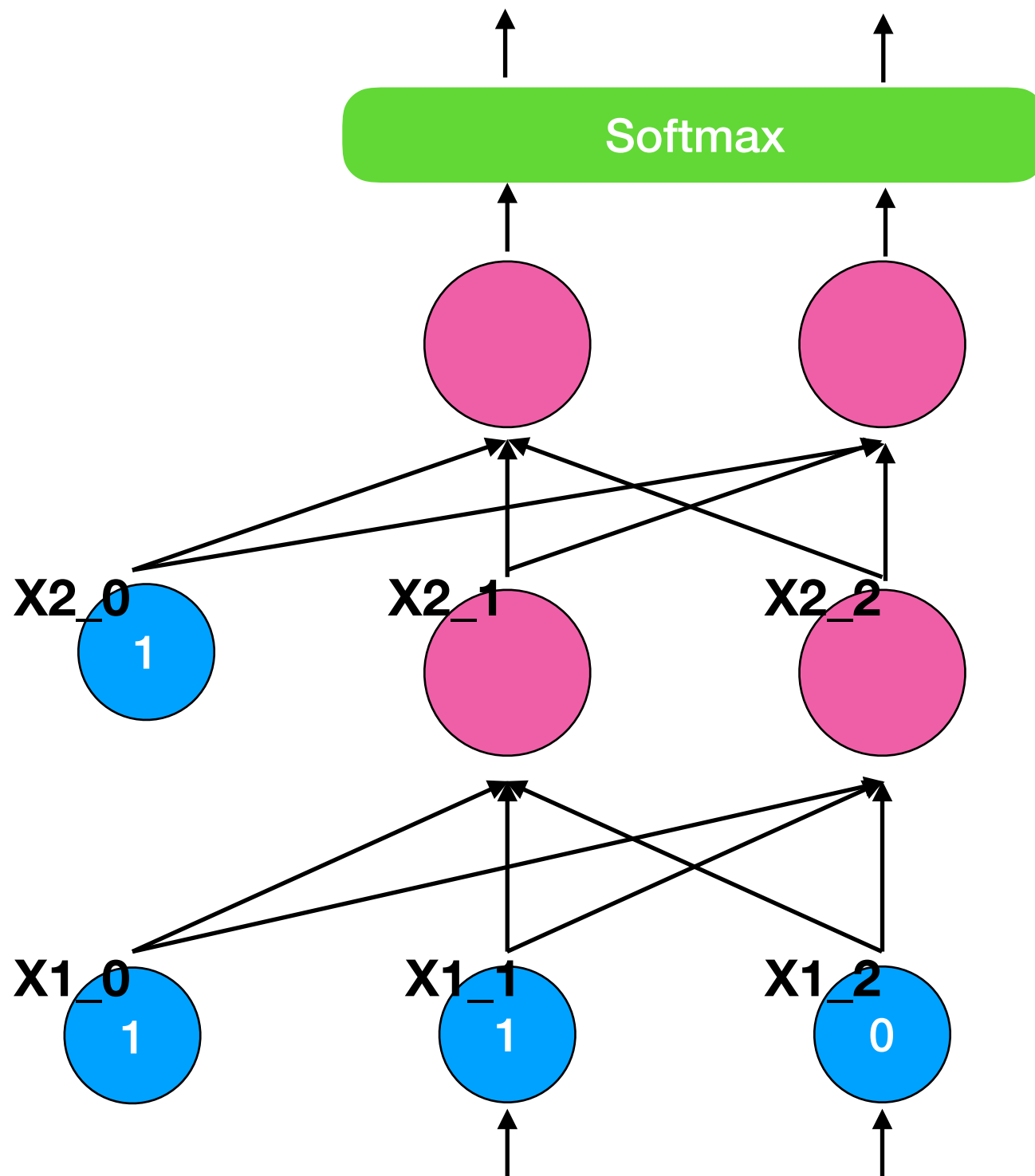
Second step Initial Weight

$w1_01 : 1, w1_02 : ?, w1_11 : 1$

$w1_12 : ?, w1_21 : 1, w1_22 : ?$

$w2_0 : 0.99, w2_1 : -0.01, w2_2 : 0.99$

MLP - Softmax



$$\hat{p}_k = \sigma(s(x)) = \frac{e^{(s_k(x))}}{\sum_{j=1}^K e^{(s_j(x))}}$$

K : number of class

$s(x)$: score vector of the class for data x

$\sigma(s(x))_k$: probability that data x classify into class k

Hyperparameter Tuning

- Number of hidden layer
 - > just one hidden layer is enough if the layer has enough neurons to compute complex functions
 - > however deep models have much more higher parameter efficiency
 - * deep models takes more less time to train the complex function cause they use fewer neurons.
 - * fast convergence, increase generalization ability
- Number of neurons in hidden layer
 - > depends on input and output size

Parameter Initialization

- Break the Symmetry
 - > Initialize each neuron to compute a different function from all of the other neurons

what happens if we set weights of the FCN with same value?

-> If weights are all same, the neurons of FCN output the same value. Then, update value based on delta rule for every neuron of FCN are also same. so training has no meaning in this situation. This situation is called **Symmetry unbreakable**.

- Gaussian Distribution & Uniform Distributions

