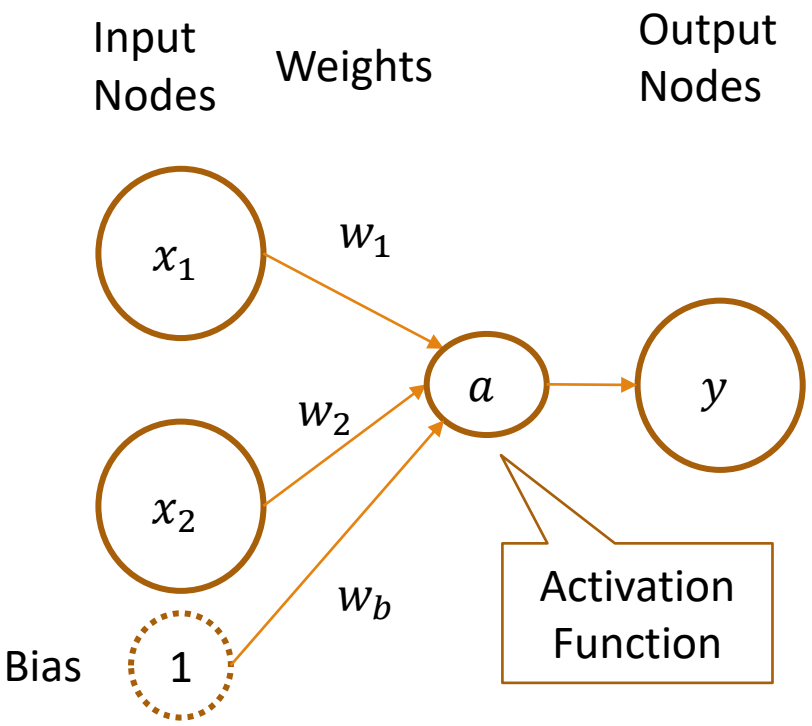
딥러닝을 활용한 **디지털 영상 처리**

Digital Image Processing via Deep Learning

Lecture 3 – Artificial Neural Network and Learning

From Last Lecture

A Perceptron without hidden layers can be used to compute a simple logic gate and classification but more complex non-linear classifications, such as XOR gate representation, were impossible to be represented.



For Step Function activation:

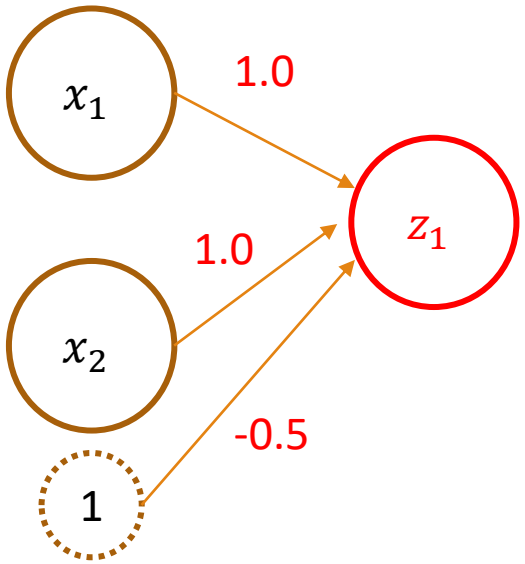
$$y = \begin{cases} 0, & w_b + w_1 x_1 + w_2 x_2 \leq 0 \\ 1, & w_b + w_1 x_1 + w_2 x_2 > 0 \end{cases}$$

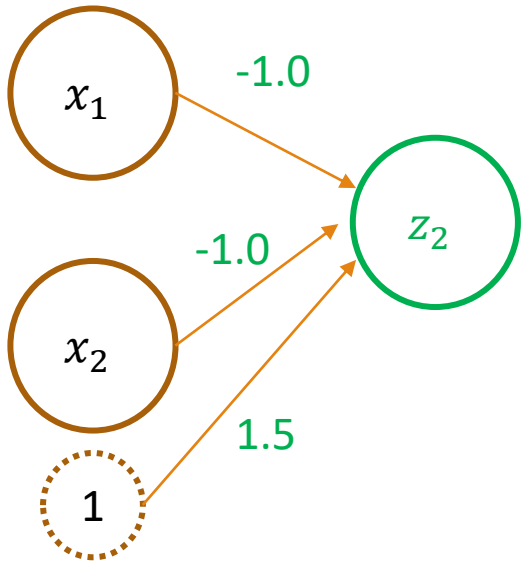
Note we have also added a weight for the bias, w_b, and we usually set the bias to 1.

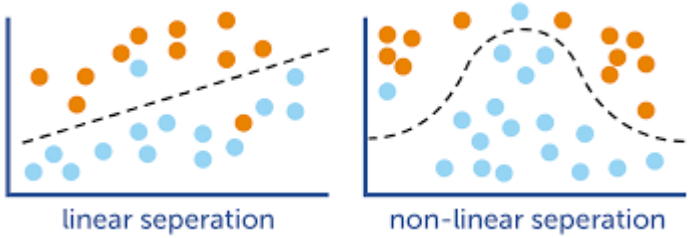
For simplicity, we omit representing the bias and activation function in the diagram from now on.

From Last Lecture

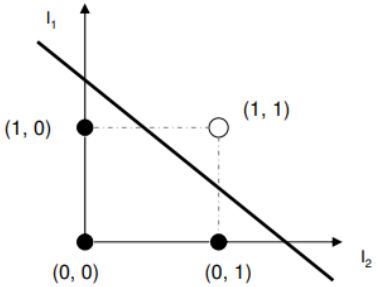
A single-layer perceptron is capable of only classifying data linearly.



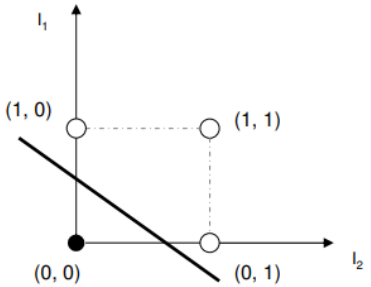




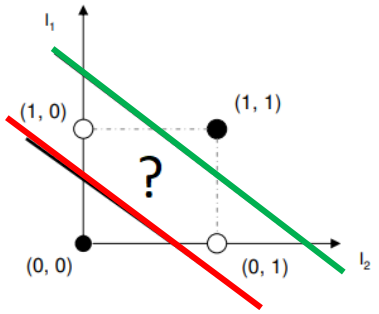
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



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

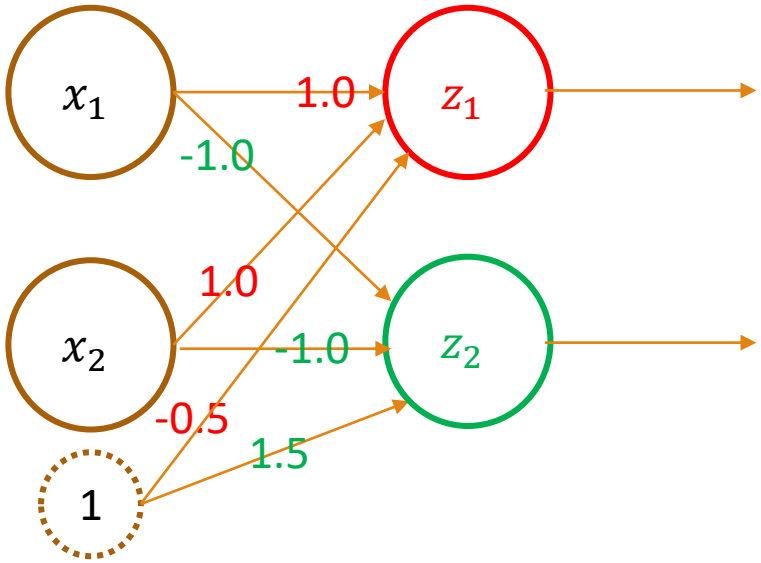


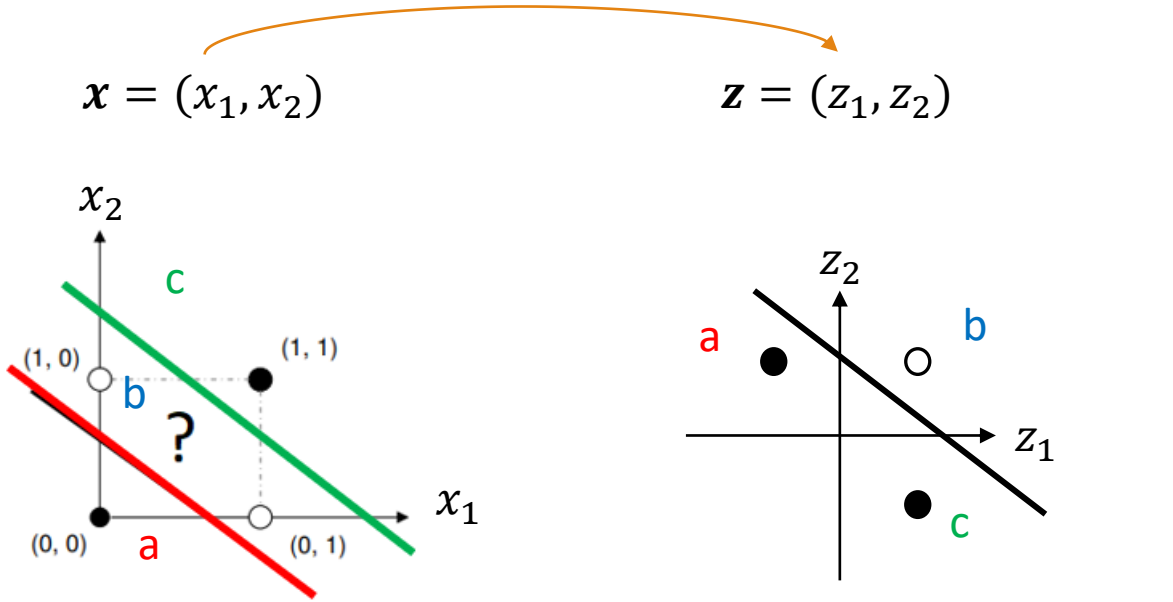
XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



From Last Lecture

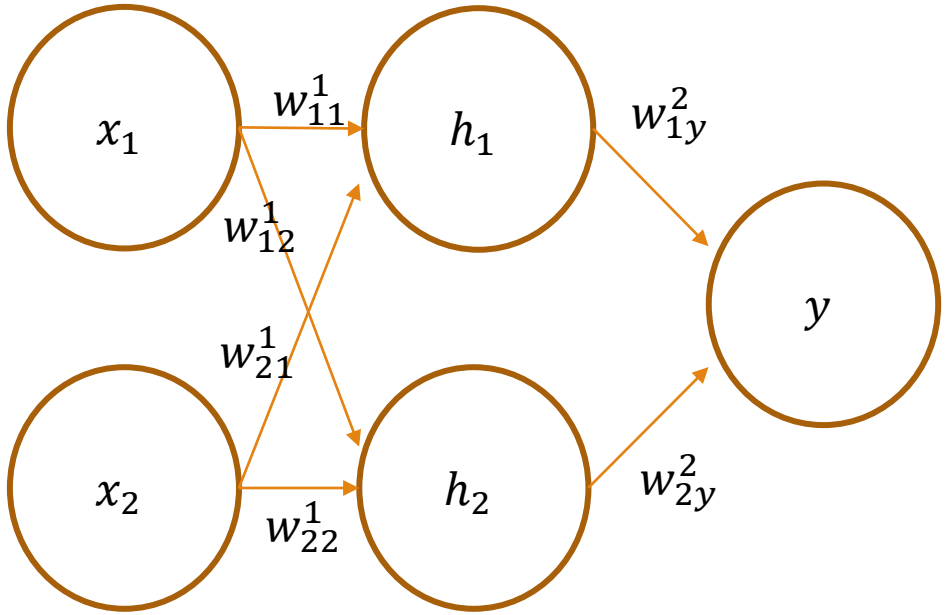
By adding an additional layer, now the transformed space is linearly separable.





From Last Lecture

Multi-layer Perceptron enables non-linear classification due to stacking non-linear activation functions.
However, not we inevitably have more weight parameters to determine.
We call this structure a **Fully-Connected Neural Network**
Notice that the notations have changed.



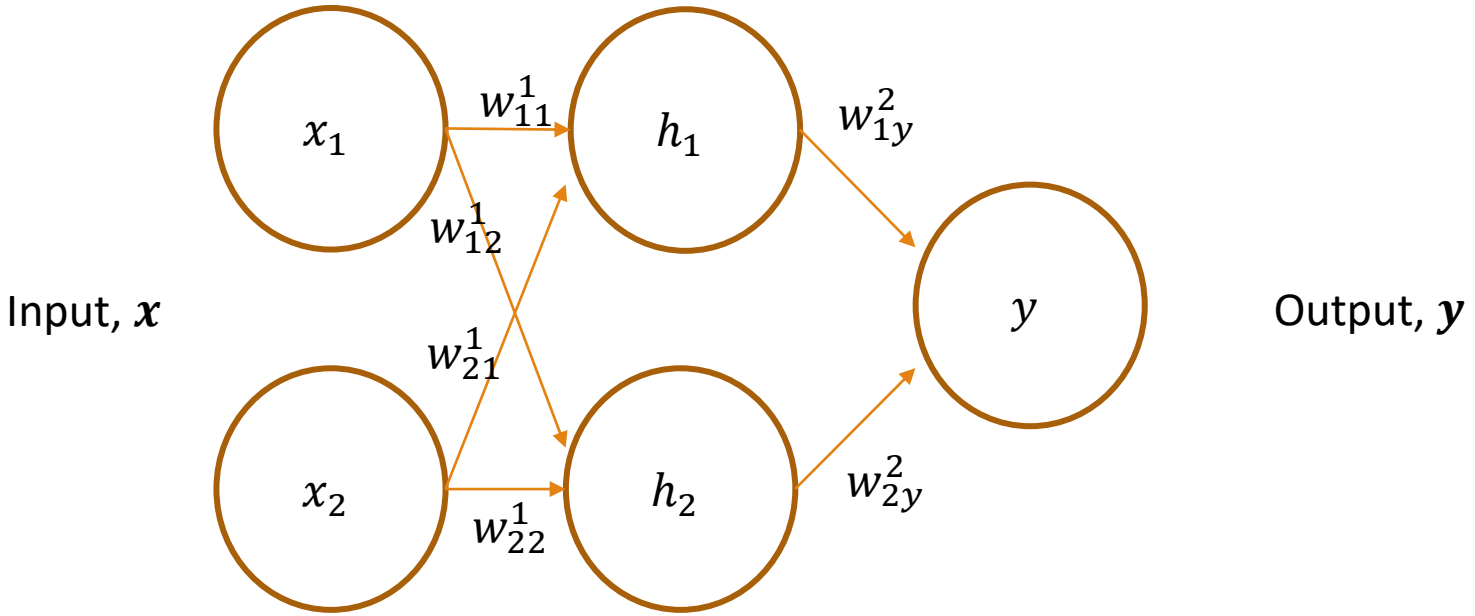
$$h_1 = \begin{cases} 0, & w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2 \leq 0 \\ 1, & w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2 > 0 \end{cases}$$

$$h_2 = \begin{cases} 0, & w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2 \leq 0 \\ 1, & w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2 > 0 \end{cases}$$

$$y = \begin{cases} 0, & w_{by}^2 + w_{1y}^2 h_1 + w_{2y}^2 h_2 \leq 0 \\ 1, & w_{by}^2 + w_{1y}^2 h_1 + w_{2y}^2 h_2 > 0 \end{cases}$$

Forward Propagation

Forward Propagation: Input → Output



Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$h_1 = a\left(w_{b1}^1 + w_{11}^1 x_1 + w_{21}^1 x_2\right)$$
$$h_2 = a\left(w_{b2}^1 + w_{12}^1 x_1 + w_{22}^1 x_2\right)$$

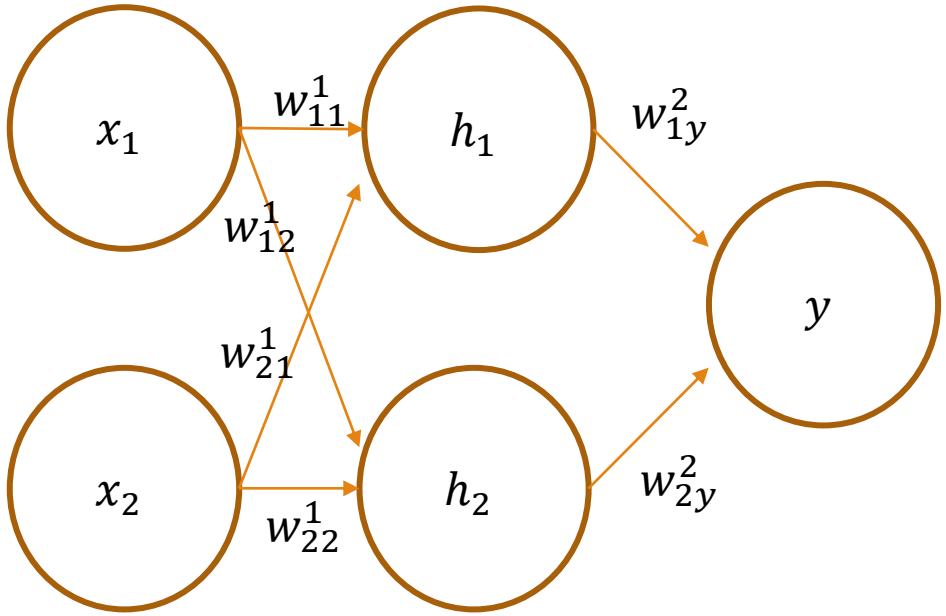
And we can combine these expressions,

$$\boldsymbol{h} = a(\boldsymbol{w}^{(1)}\boldsymbol{x})$$

Where

$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Where (1) denotes that the weights are between input and the first hidden layer. (Not power)



We denote the activation function as $a()$

Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$ ← bias

Then,

$$\boldsymbol{h} = a(\boldsymbol{w}^{(1)}\boldsymbol{x})$$

Where

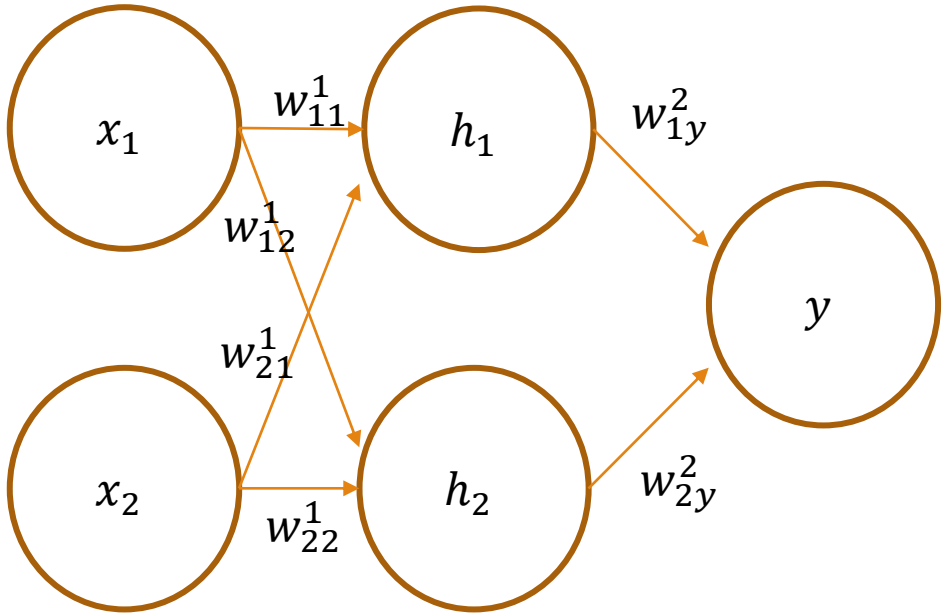
$$\boldsymbol{w}^{(1)} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 \\ w_{b2}^1 & w_{12}^1 & w_{22}^1 \end{pmatrix}$$

Then,

$$y = a(\boldsymbol{w}^{(2)}\boldsymbol{h})$$

Where

$$\boldsymbol{w}^{(2)} = \begin{pmatrix} w_{by}^2 & w_{1y}^2 & w_{2y}^2 \end{pmatrix}$$



We denote the activation function as $a()$

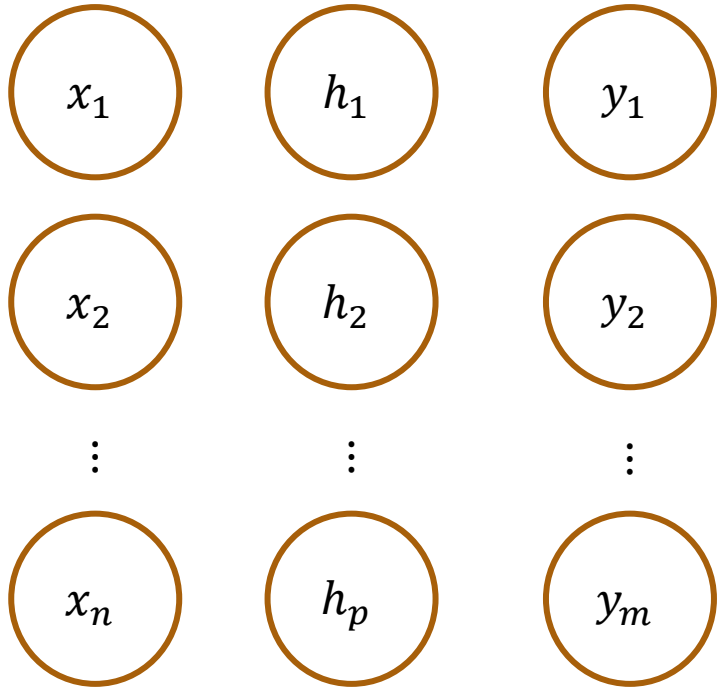
Forward Propagation

Let Input vector, $\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$

$$\boldsymbol{w^{(1)}} = \begin{pmatrix} w_{b1}^1 & w_{11}^1 & w_{21}^1 & \cdots & w_{n1}^1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{bp}^1 & w_{1p}^1 & w_{2p}^1 & \cdots & w_{np}^1 \end{pmatrix} \qquad p \times (n+1)$$

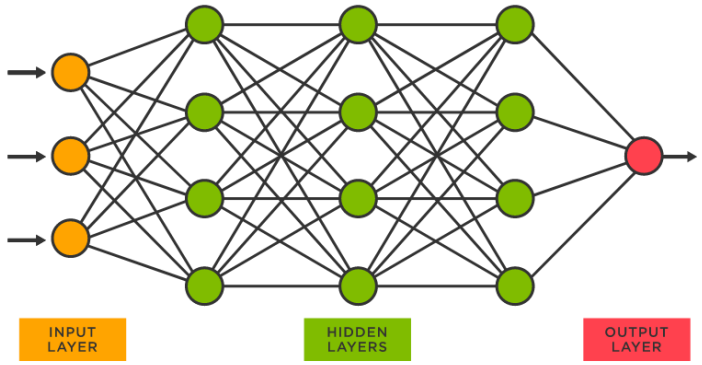
$$\boldsymbol{w^{(2)}} = \begin{pmatrix} w_{b1}^2 & w_{11}^2 & w_{21}^2 & \cdots & w_{p1}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{bm}^2 & w_{1m}^2 & w_{2m}^2 & \cdots & w_{pm}^2 \end{pmatrix} \qquad m \times (p+1)$$

$$y = a_2(\boldsymbol{w^{(2)}} \underbrace{a_1(\boldsymbol{w^{(1)}} \boldsymbol{x})}_{\boldsymbol{h}})$$



QUIZ!

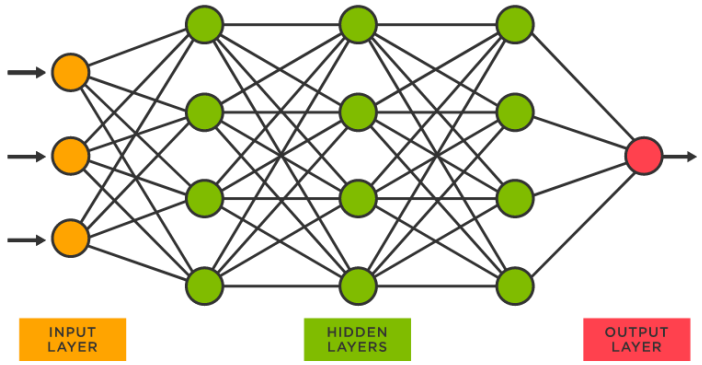
Consider a fully connected neural network with 3 hidden layers of 4 nodes each. The input is an RGB color-coded image with 256 pixels in both axis, all pixel data will be reformatted into a single-column vector. The input image is expected to show a person and the network is used to output a label of the image, who the person is out of 4 options → the size of the output vector is 4.
Q: State the size of the weight matrix between each layer and the total number of weight parameters to be deduced/learned.



WHO?

QUIZ!

각각 4개의 hidden node로 구성된 3개의 hidden layer(은닉층)를 가진 fully connected neural network를 고려하시오. 우리는 걸그룹 Aespa의 멤버 중 한 명의 사진을 입력으로 받아 해당 멤버가 누구인지를 맞추는 인공 신경망을 디자인 하려고 한다. 입력 사진은 RGB로 색상을 표기하며 양측으로 256개의 픽셀을 가진다, 해당 이미지의 모든 픽셀 정보는 하나의 column vector로 변형되어 들어간다. 출력층의 길이는 4이다(카리나, 윈터, 닝닝, 지젤).
Q: 각 층 사이에 weight matrix(가중치 행렬)의 크기와 모든 weight parameter의 개수를 제시하시오.



WHO?

Activation Functions

Step Function:

$$a(x) = \begin{cases} 1, & x \geq 0 \\ \beta, & x < 0 \end{cases}$$

Where $\beta = 0$ or $\beta = -1$

Logistic Sigmoid:

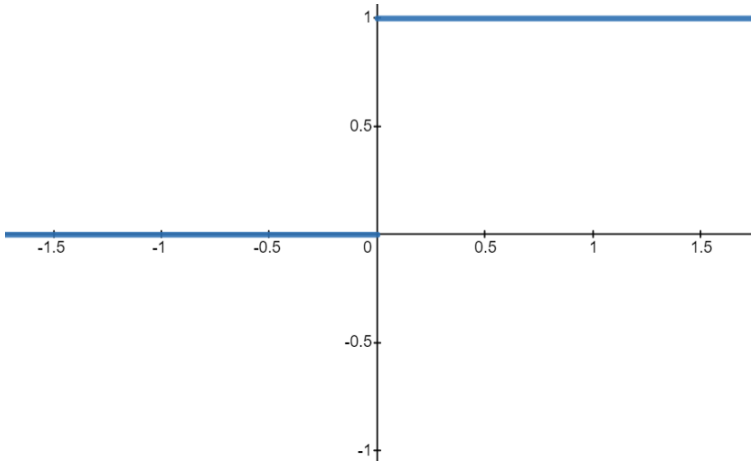
$$a(x) = \frac{1}{1 + e^{-\beta x}}$$

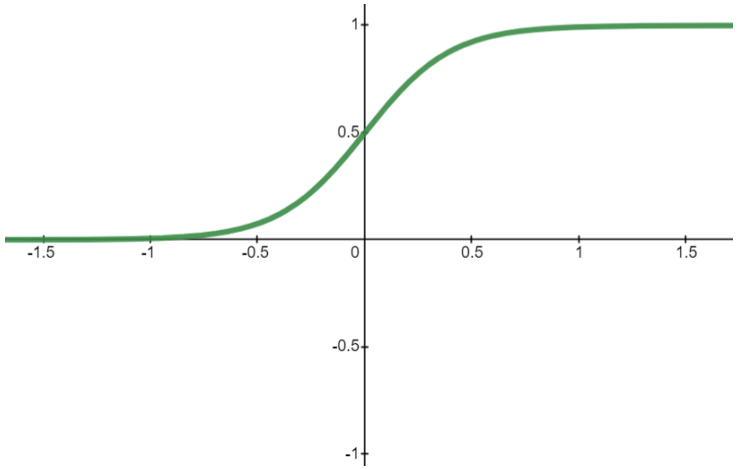
Where $\beta > 0$

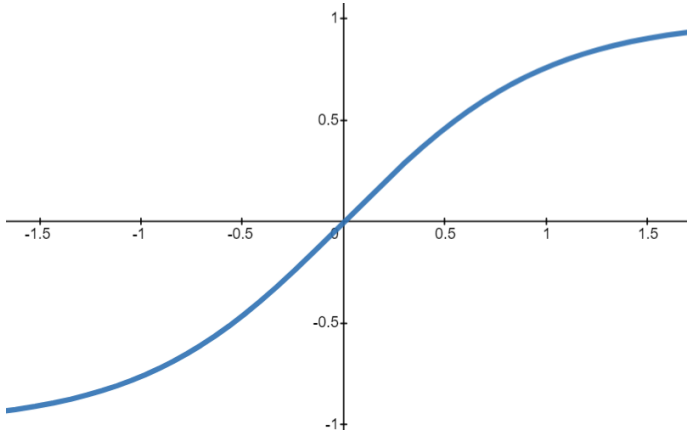
Hyperbolic Tangent:

$$a(x) = \tanh \beta x$$

Where $\beta > 0$



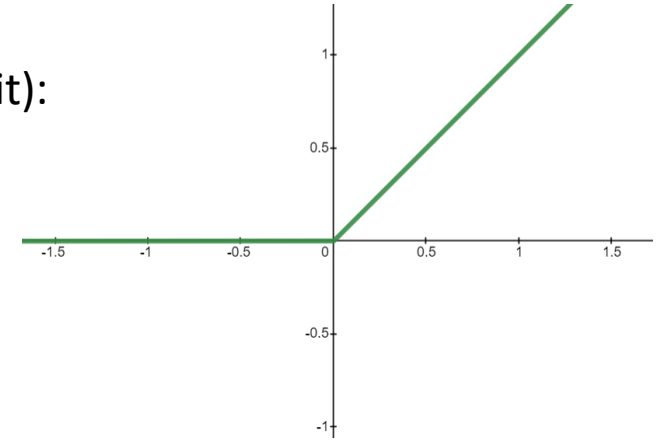




Activation Functions

ReLU(Rectified Linear Unit):

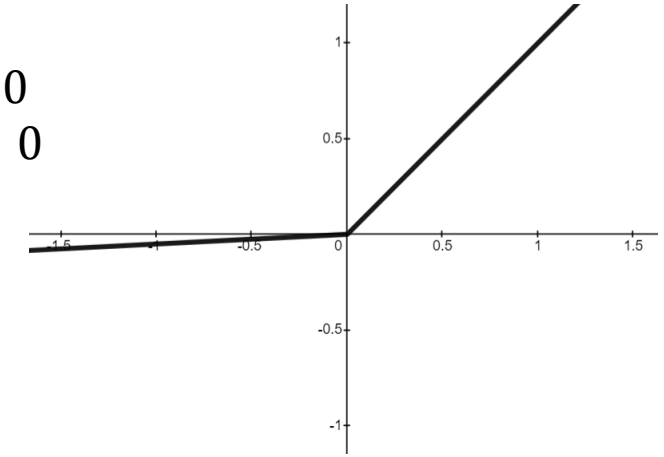
$$a(x) = \max(0, x)$$



Leaky ReLU:

$$a(x) = \begin{cases} x, & x \geq 0 \\ \beta x, & x < 0 \end{cases}$$

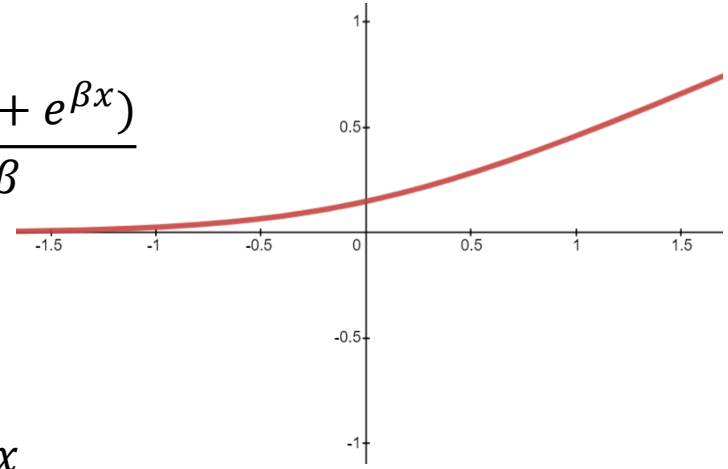
Where $0 < \beta \ll 1$



Softplus:

$$a(x) = \frac{\log(1 + e^{\beta x})}{\beta}$$

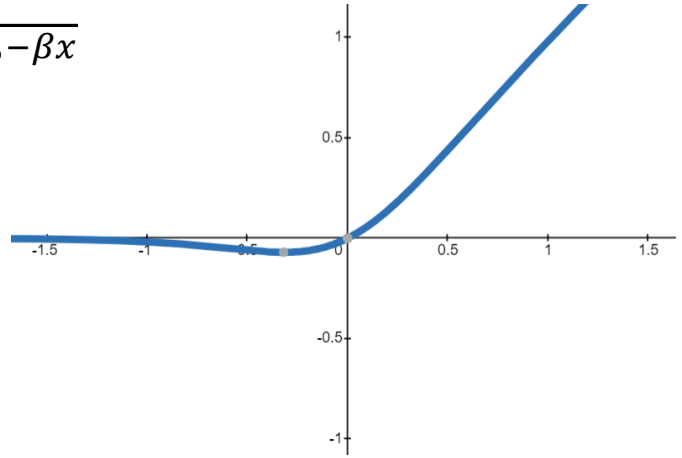
Where $\beta > 0$



Swish:

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

Where $\beta > 0$



Activation Functions

Activation function	Mathematical notation	Features
Step	$\begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$	Can only take 2 values; limited representation. The gradient is 0 for all input (ignoring the origin).
Logistic Sigmoid	$\frac{1}{1+e^{-\beta x}}$	Smooth, takes continuous values. The gradient converges to 0 for extreme input values.
Hyperbolic Tangent	$\tanh \beta x$	
ReLU	$\max(0, x)$	Gradient is 1 for all positive input values. Enables classification.
Leaky ReLU	$\begin{cases} x, & x \geq 0 \\ 0.001x, & x < 0 \end{cases}$	Prevents “dying neuron” by having a small non-zero gradient for negative input values
Softplus	$\frac{\log(1+e^{\beta x})}{\beta}$	
Swish	$\frac{x}{1+e^{-\beta x}}$	

Output Layer Activation - Softmax

The neural network mostly outputs a number indicating a “score” of the input data to a certain label.
In other words, how likely is this input to have a certain label.
This is typically expressed in probability format.

$$\begin{pmatrix} 1.058 \\ 0.013 \\ 0.568 \\ 1.345 \end{pmatrix} \Rightarrow \begin{pmatrix} 0.303 \\ 0.107 \\ 0.186 \\ 0.404 \end{pmatrix}$$

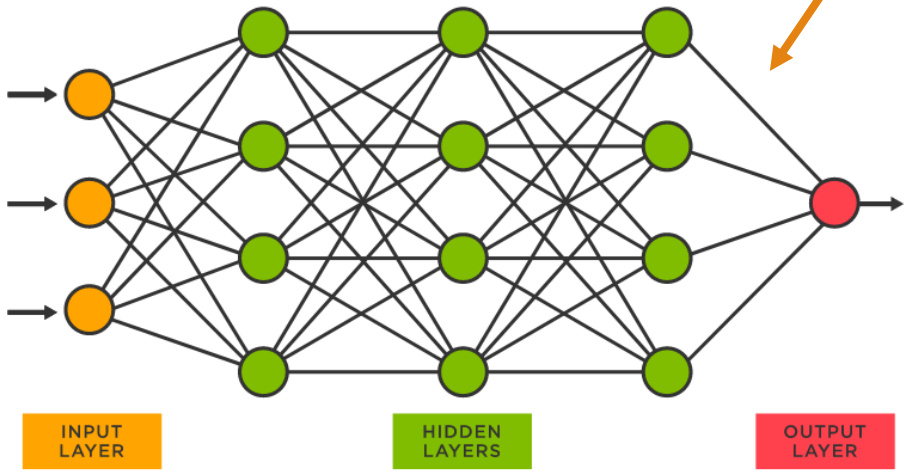
We use a softmax function for this purpose. It takes a portion of the exponential of a given indexed output out of the total → equivalent to probability.

$$softmax(i, \boldsymbol{x}) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

Output Layer Activation - Softmax

Softmax used right before the final layer.

Softmax used here!!



$$\begin{pmatrix} P(x = \text{"}Karina\text{"}) \\ P(x = \text{"}Winter\text{"}) \\ P(x = \text{"}NingNing\text{"}) \\ P(x = \text{"}Giselle\text{"}) \end{pmatrix}$$

Output Layer Activation - Softmax

Why Softmax, not Standard Normalization?

$$norm(i, \boldsymbol{x}) = \frac{x_i}{\sum_i x_i}$$

$$norm([1, 2]) = [0.333, 0.667]$$

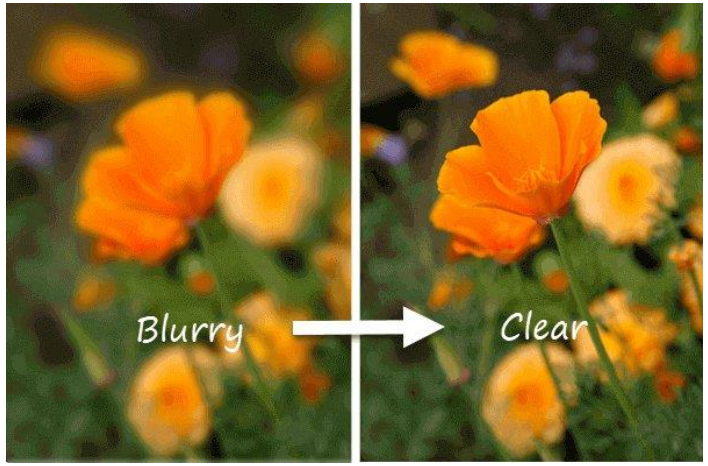
$$norm([10, 20]) = [0.333, 0.667]$$

$$softmax(i, \boldsymbol{x}) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

$$softmax([1, 2]) = [0.2689, \quad 0.7311]$$

$$softmax([10, 20]) = [0.00004539, \quad 0.99995460]$$

Blur images have low differences between pixel intensities. Thus, it may not be clear to be classified.



Clear images have high differences between pixel intensities. Thus, it is clear to be classified.

Output Layer Activation - Softmax

Why Softmax, not Standard Normalization?

1. It reacts well to stimulations of data.
2. We do not need to transform negative numbers.
3. Prevents gradient vanishing during optimization (beyond the scope of this course).

$$norm(i, \boldsymbol{x}) = \frac{x_i}{\sum_i x_i}$$

$$softmax(i, \boldsymbol{x}) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

Finding optimal weights – Loss function

As discussed so far, the key of deep learning and using neural networks lies on being able to find optimal/near-optimal weight parameters, $\boldsymbol{w}$.

Consider a function expression of neural network:

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w})$$

If wrong weights are used, the outputs will be far from our expectations and correct answers.
Therefore, we need a measure of how good the weights are!
We call this a **Loss function**.

Loss Function: a function that represents differences between network outputs and given answers.

Eg; $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$, where t is the given answer label

Finding optimal weights – Loss function

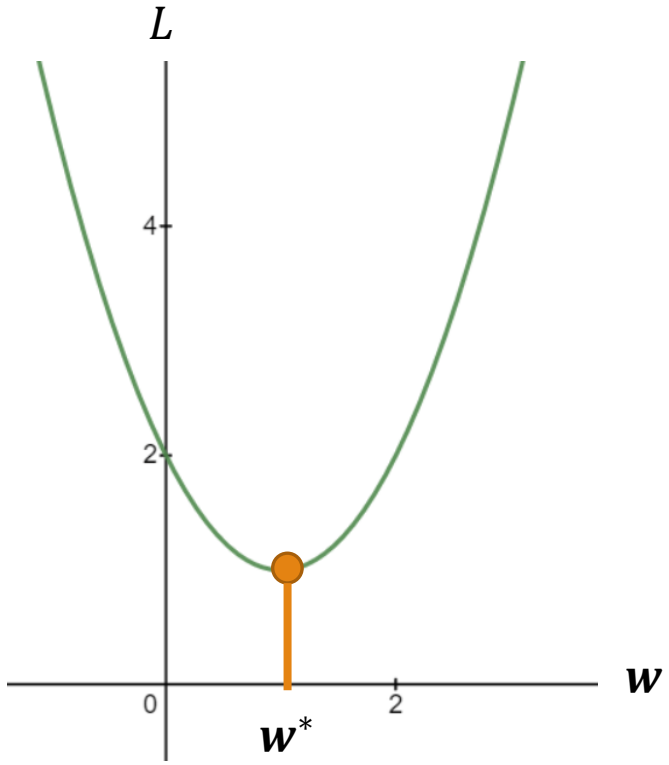
Consider: $L(\boldsymbol{x}, \boldsymbol{w}) = (y - t)^2 = (f(\boldsymbol{x}, \boldsymbol{w}) - t)^2$

Our task is finding weights, $\boldsymbol{w}$, that minimizes the loss, L.

From intuition, we can simply find the optimal weights is finding the turning points of the loss function by differentiating it.

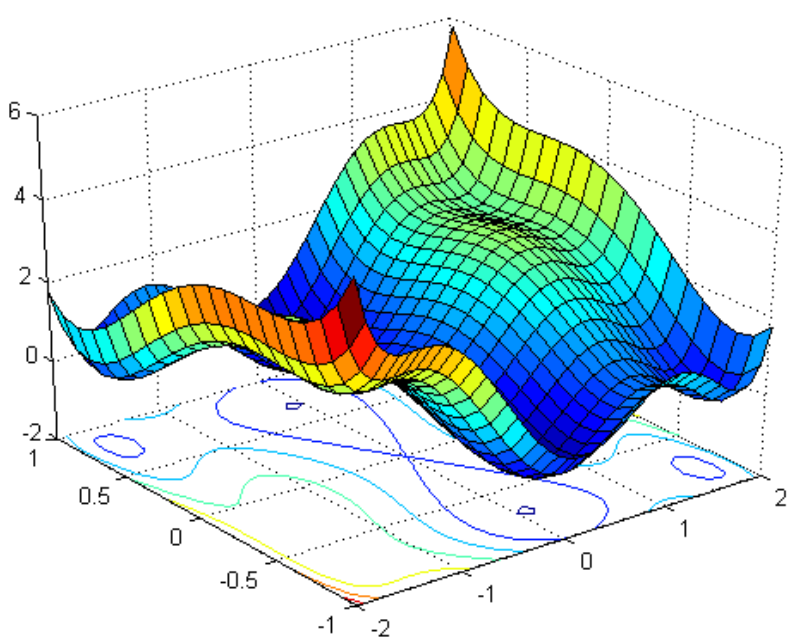
$$\boldsymbol{w}^* = \boldsymbol{argmin}_{\boldsymbol{w}}(L(\boldsymbol{x}, \boldsymbol{w}))$$

$$0 = \frac{\partial L}{\partial \boldsymbol{w}}|_{\boldsymbol{w}=\boldsymbol{w}^*}$$



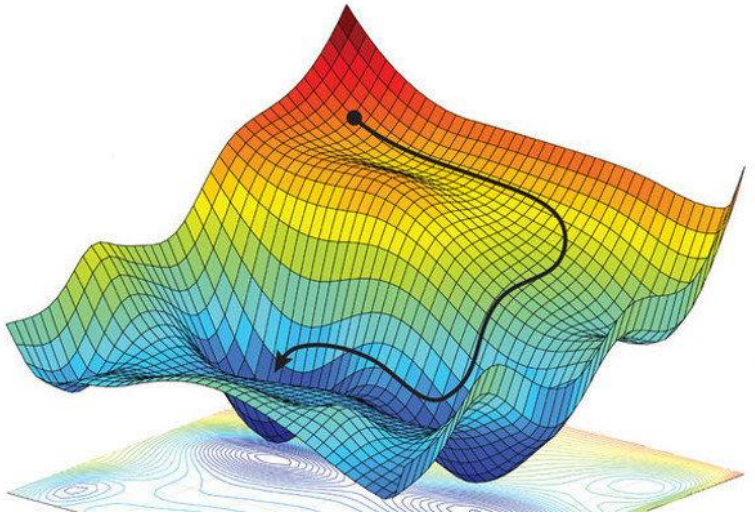
Finding optimal weights – Loss function

Example loss function with two weights



Actual loss functions with billions of weight parameters are computationally impossible to compute the full derivative.

So, without the derivative, how can we find the optimal weights?



Next Lecture

Gradient Descent Algorithms