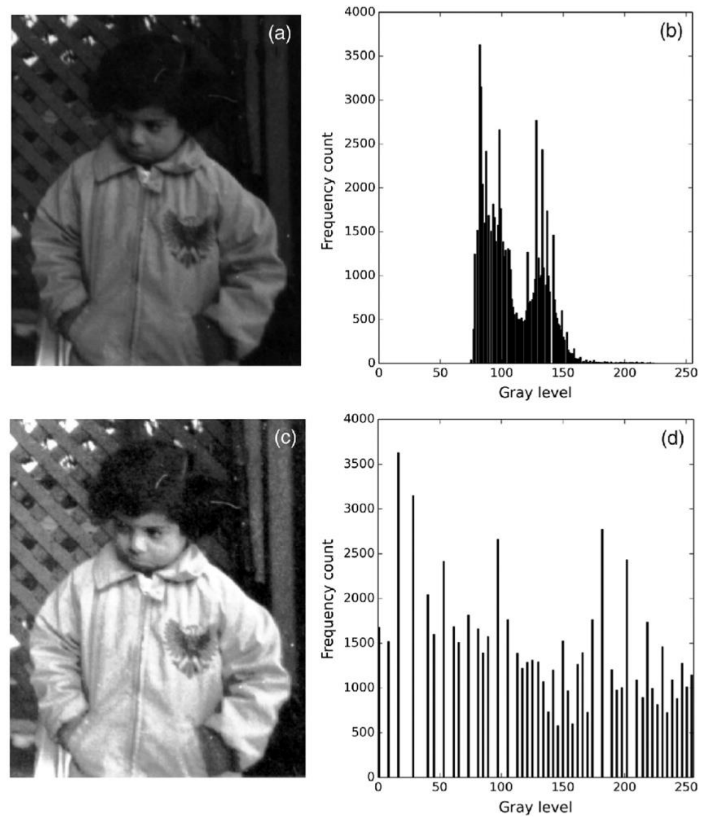
딥러닝을 활용한 **디지털 영상 처리**
Digital Image Processing via Deep Learning

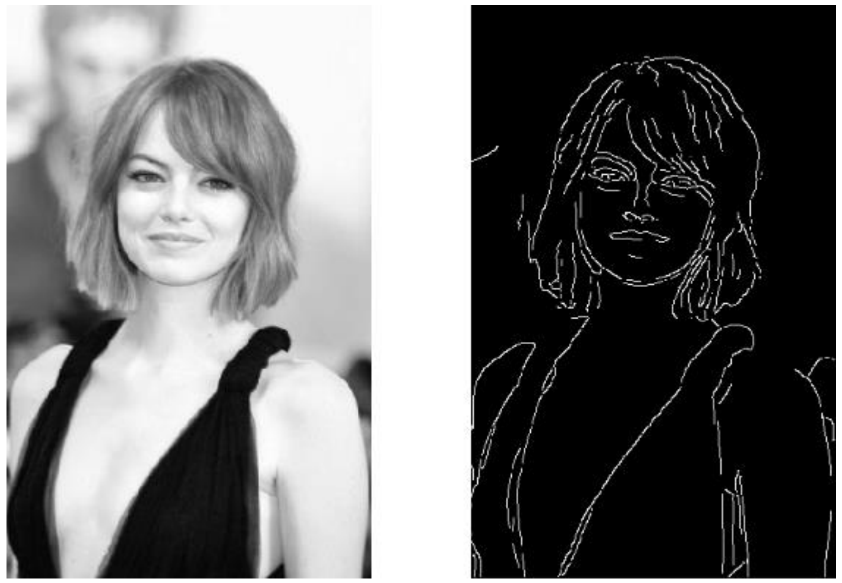
Lecture 2 – Perceptron and Introduction to Deep Learning

From Last Lecture

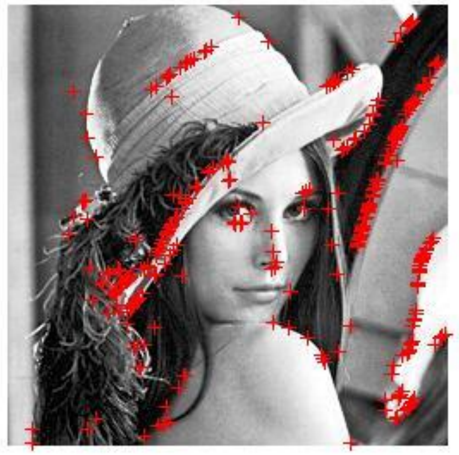
Histogram Equalization



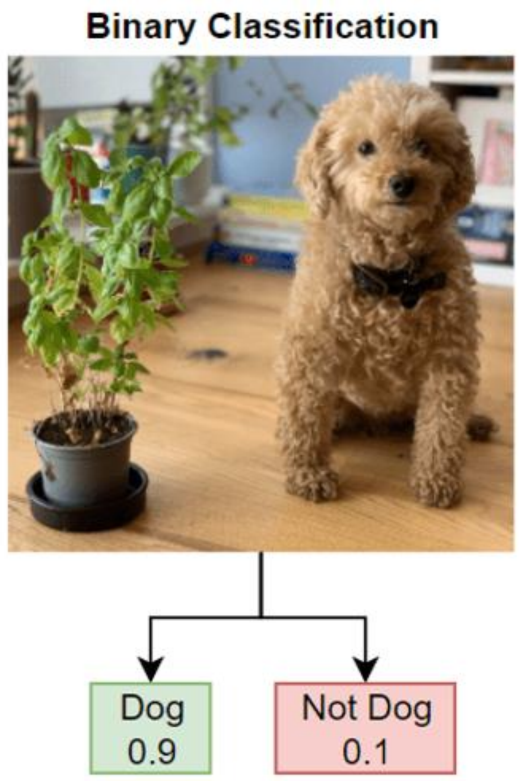
Edge Detection



Feature point extraction

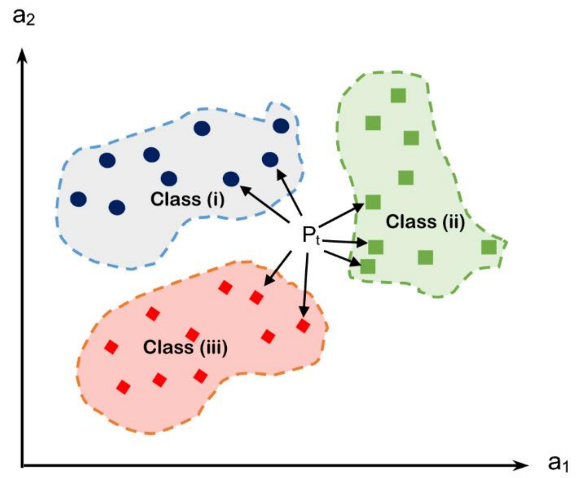


From Last Lecture



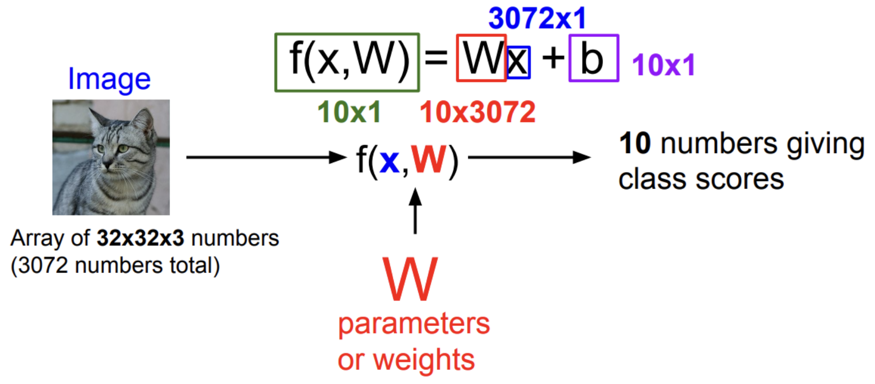
K Nearest Neighbor:

- Classifies test images by their distance metric with train images
- Fast Training but Long Testing → BAD!
- The distance metric is not informative



Linear Regression:

- Classifies test images by a linear function where the weights are to be determined.
- Slow Training but Fast Testing → OK!
- Only Linear classification is possible → BAD!
- How can we determine the weights?



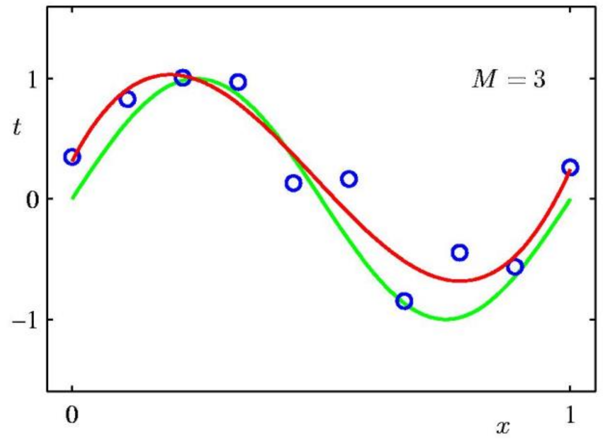
From Last Lecture

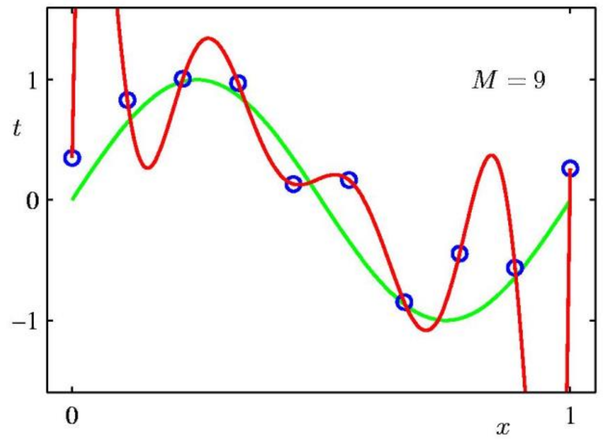
Linear Regression: Polynomial Curve Fitting

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

Our objective is to find the weights, $\boldsymbol{w}$, that minimises the error.
To prevent Overfitting, use more training data or use ridge regression.





From Last Lecture

How do we find the optimal weights?

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

$$\boldsymbol{w} = argmin_{\boldsymbol{w}} \sum_{n=1}^{N} \{y(x_n, \boldsymbol{w}) - t_n\}^2 = argmin_{\boldsymbol{w}} (\boldsymbol{x}\boldsymbol{w} - \boldsymbol{t})^T (\boldsymbol{x}\boldsymbol{w} - \boldsymbol{t}) \Longrightarrow \boxed{\boldsymbol{w} = (\boldsymbol{x}^T \boldsymbol{x})^{-1} \boldsymbol{x}^T \boldsymbol{y}}$$

The complexity of this computation is $O(n^3)$

As the size of the dataset increase, weights become extremely difficult to find.
We must seek more convenient methods to find the weights

Towards Deep Learning

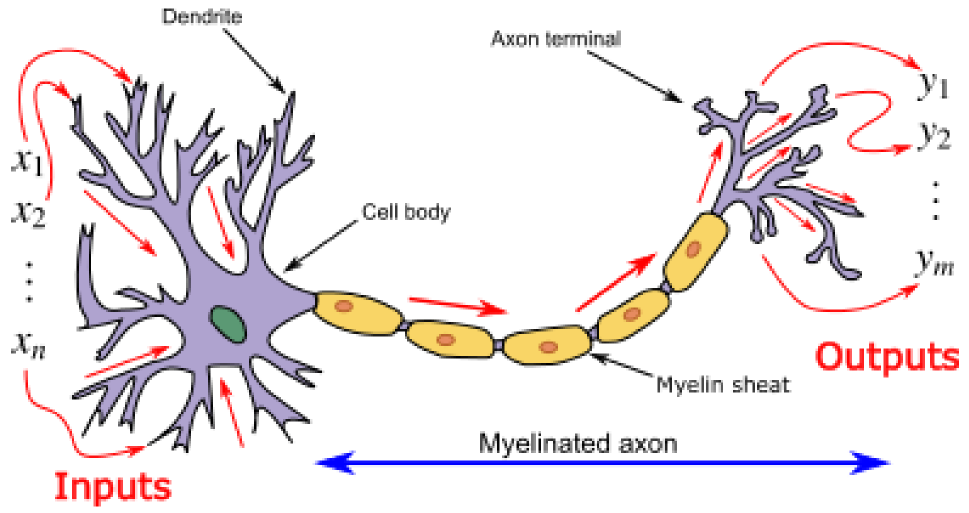
Can Linear Classifier classify all the images? How about non-linear relationships?

How can we find appropriate weights not by heavy computation?

Deep Learning methods remedy (partially in some sense) these shortcomings.

Perceptron

The idea of Perceptron was introduced by “Frank Rosenblatt(1957)”.
Old but still the key idea of Machine Learning and Deep Learning algorithms.
The Idea of Perceptron took its motivation from our neuro-cells



Perceptron

The circles containing data is called “**node**”.

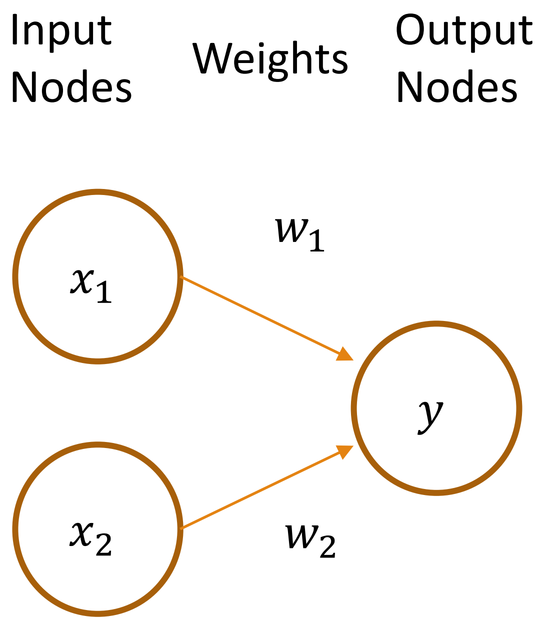
The Output node, containing y, may be expressed in 2 forms

1: Linear function of the input and weights:

$$y = w_1x_1 + w_2x_2$$

2: Binary Classification via a threshold value, θ, via activation function:

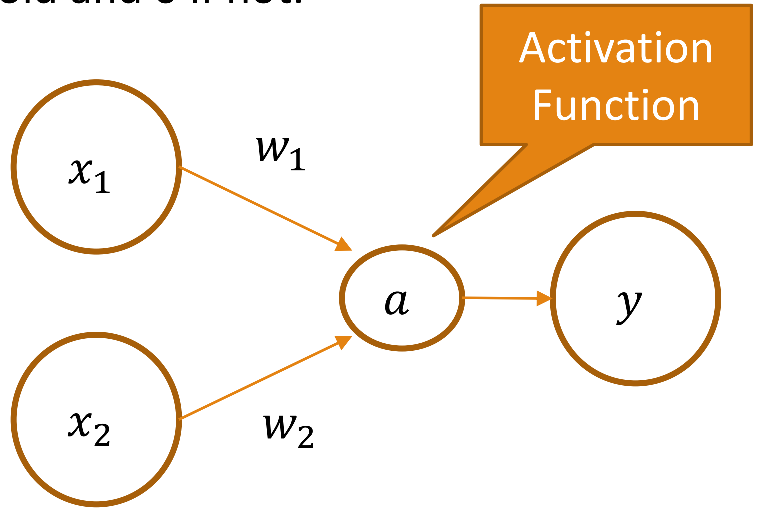
$$y = \begin{cases} 0, & w_1x_1 + w_2x_2 \leq \theta \\ 1, & w_1x_1 + w_2x_2 > \theta \end{cases}$$



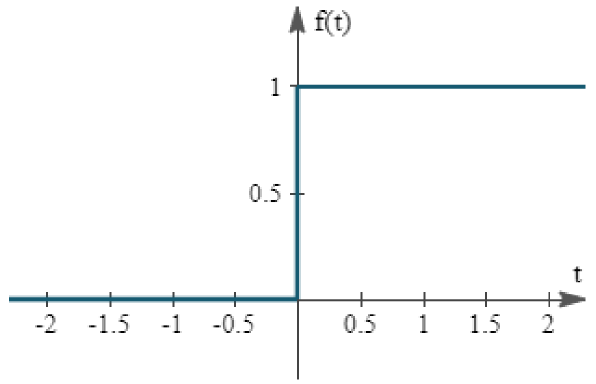
Activation Function

Activation function: A function that activates the next node from given information by previous nodes and weights.

An example: Step Function → Returns 1 if input is above certain threshold and 0 if not.



For simplicity, from now on, we will skip drawing the activation node in the diagram.

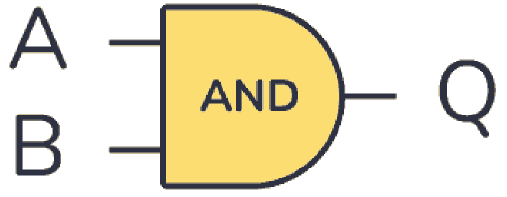


$y = h(w_1x_1 + w_2x_2)$,
where

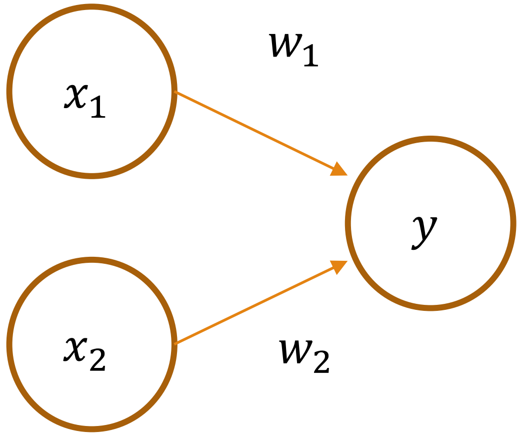
$$h(x) = \begin{cases} 0, & x \le \theta \\ 1, & x > \theta \end{cases}$$

$$y = \begin{cases} 0, & w_1x_1 + w_2x_2 \le \theta \\ 1, & w_1x_1 + w_2x_2 > \theta \end{cases}$$

Simple Logic Gate



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



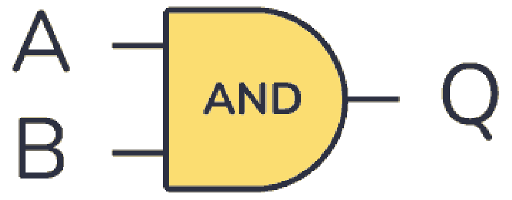
x_1	x_1	y
0	0	0
1	0	0
0	1	0
1	1	1

$$y = \begin{cases} 0, & w_1x_1 + w_2x_2 \leq \theta \\ 1, & w_1x_1 + w_2x_2 > \theta \end{cases}$$

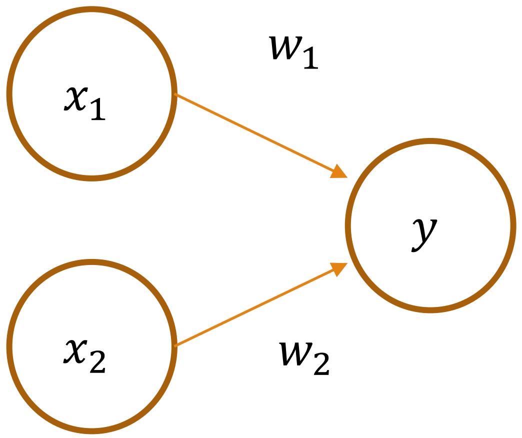
Various combinations of (w_1, w_2, θ) exist:
(0.5, 0.5, 0.7)
(0.5, 0.5, 0.8)
(1.0, 1.0, 1.0)

Try Out!!

Simple Logic Gate



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



x_1	x_1	y
0	0	0
1	0	0
0	1	0
1	1	1

Instead of threshold, we introduce a bias;

$$b = -\theta$$

$$y = \begin{cases} 0, & b + w_1x_1 + w_2x_2 \leq 0 \\ 1, & b + w_1x_1 + w_2x_2 > 0 \end{cases}$$

Various combinations of (w_1, w_2, b) exist:

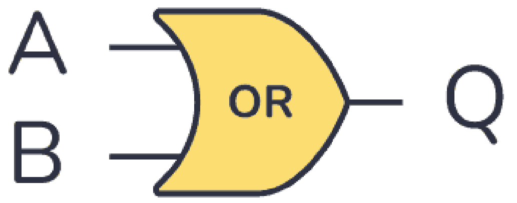
(0.5, 0.5, -0.7)

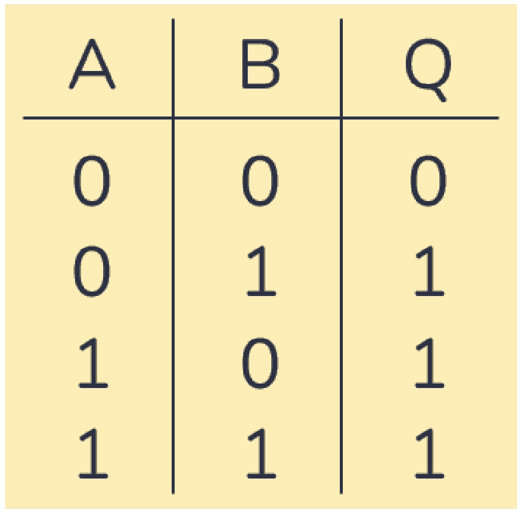
(0.5, 0.5, -0.8)

(1.0, 1.0, -1.0)

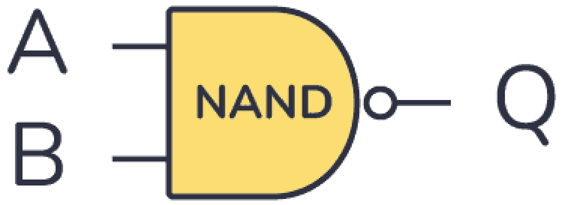
Try Out!!

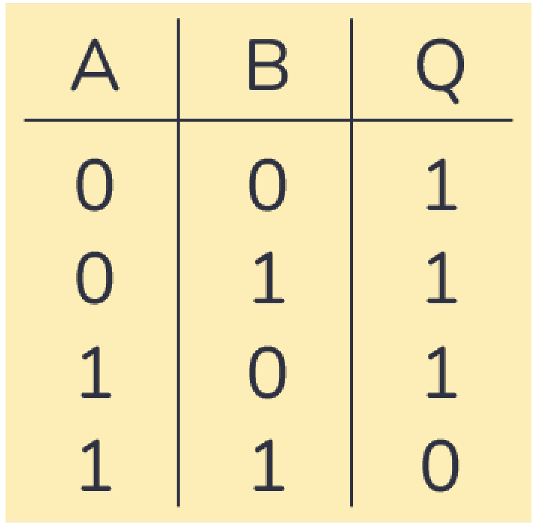
NAND and OR Gates





A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

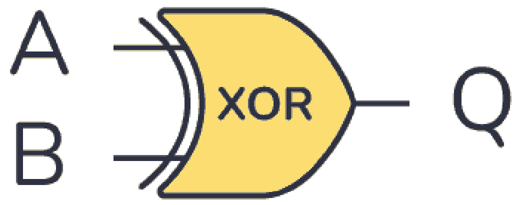




A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

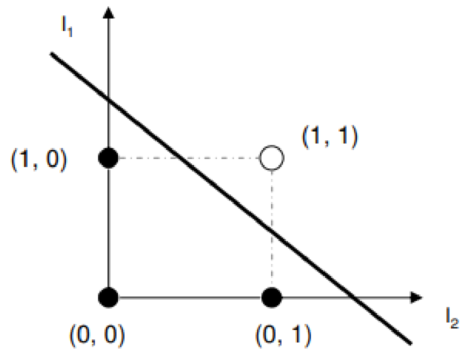
고키, p54~57

XOR Gate

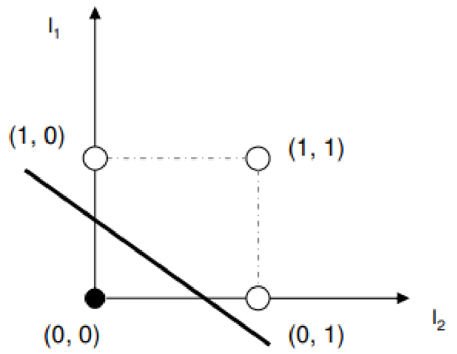


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

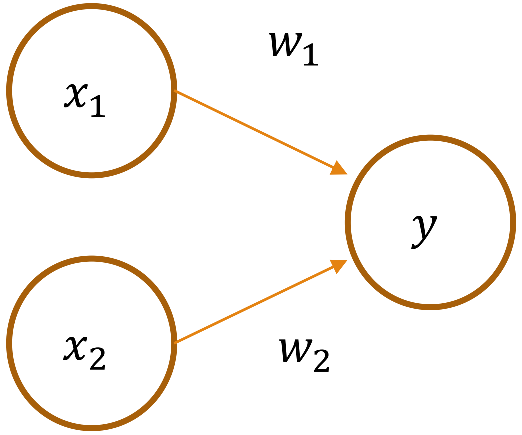
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



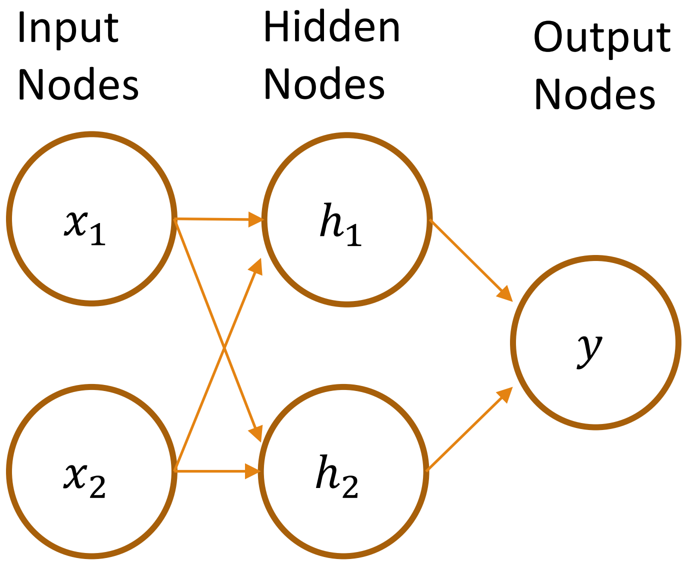
OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



Multi-layer Perceptron



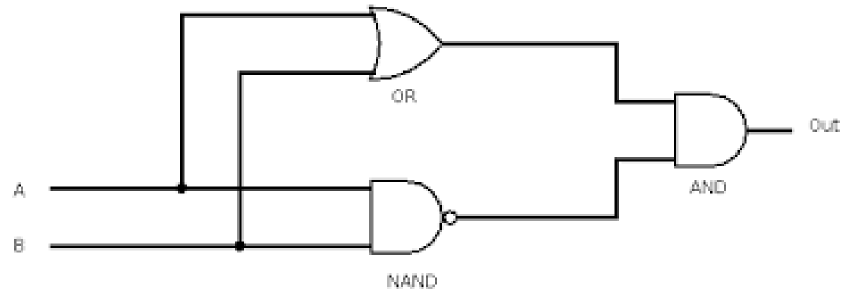
A perceptron can only classify data linearly. Just like linear regression.

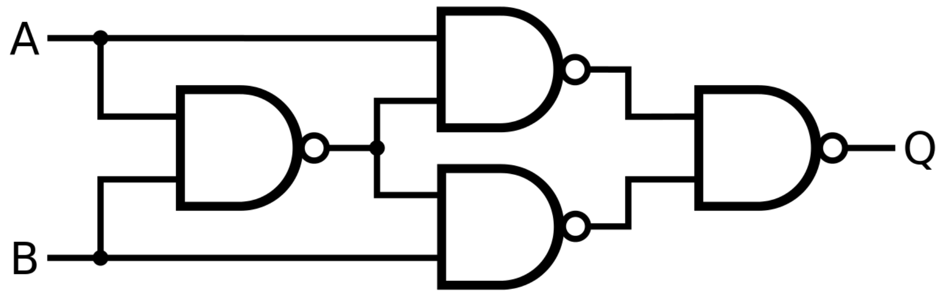


By adding an additional layer between the input and output layer, it can now classify data non-linearly.
This is called Multi-layer Perceptron

Stacking Logic Gates

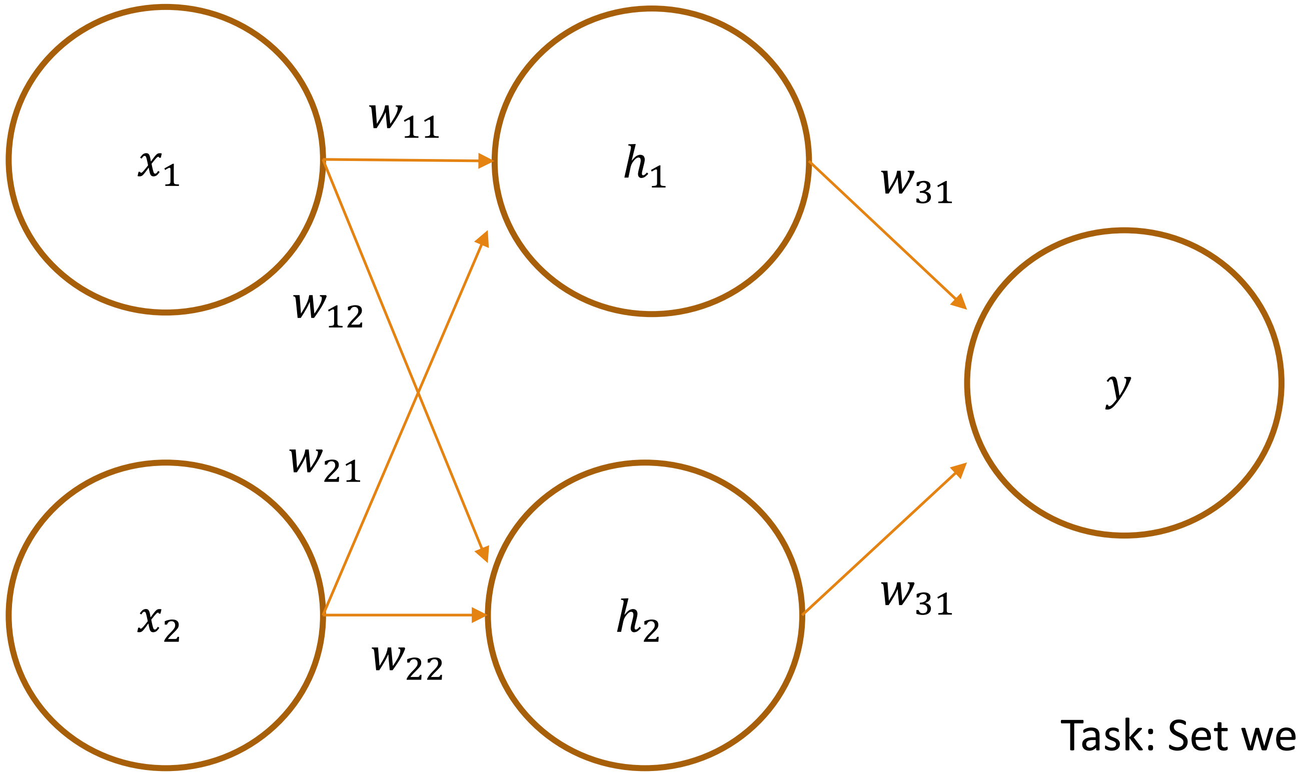
Given AND, OR and NAND logic gates can be designed by perceptrons, the logic circuit diagram below can be built and have the same effect as XOR gate.





This is identical to having multiple layers in a perceptron.

Multi-layer Perceptron



$$h_1 = \begin{cases} 0, & b + w_{11}x_1 + w_{12}x_2 \le 0 \\ 1, & b + w_{11}x_1 + w_{12}x_2 > 0 \end{cases}$$

$$h_2 = \begin{cases} 0, & b + w_{21}x_1 + w_{22}x_2 \le 0 \\ 1, & b + w_{21}x_1 + w_{22}x_2 > 0 \end{cases}$$

$$y = \begin{cases} 0, & b + w_{31}h_1 + w_{32}h_2 \le 0 \\ 1, & b + w_{31}h_1 + w_{32}h_2 > 0 \end{cases}$$

Task: Set weights and the bias $w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32}, b$

Multi-layer Perceptron

How does the Multi-layer Perceptron classify data non-linearly?

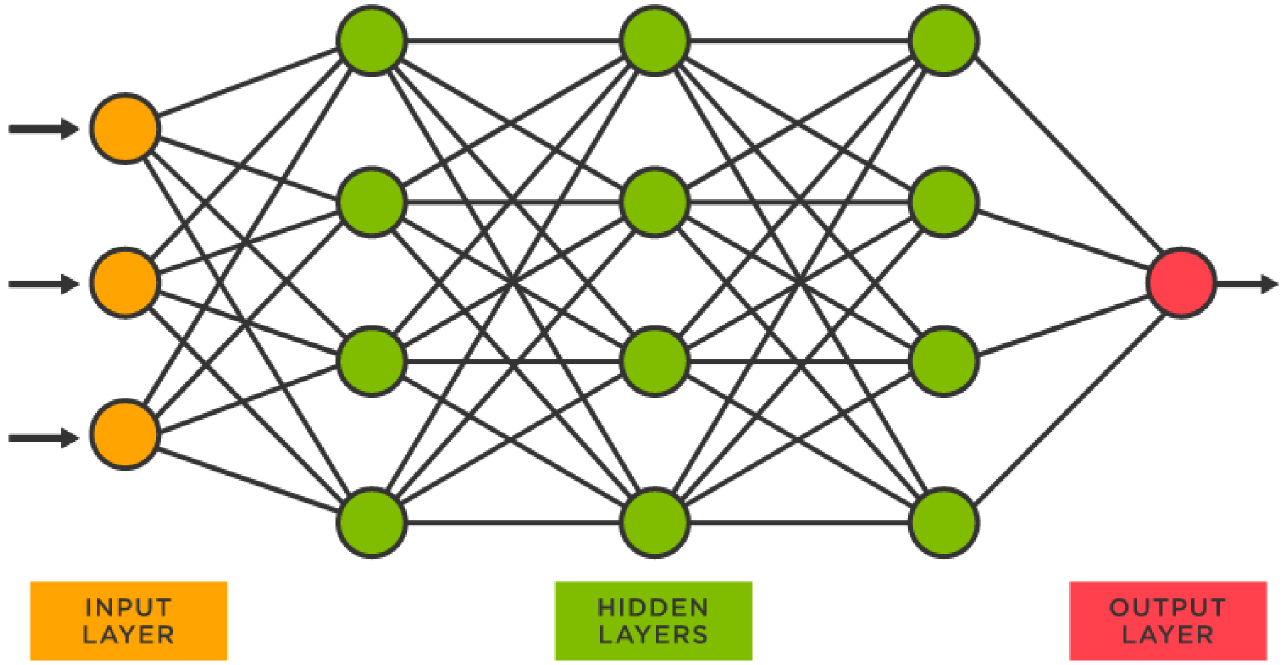
The activation function must be non-linear, and what we used, the Step Function is Non-Linear Function!

Stacking Non-Linear functions allow the system to behave non-linearly.

Imagine have a linear activation function; $h(x) = cx$, where c is a constant.
Then, stacking up 3 layers will result (roughly); $\mathrm{y}(x) = h\left(h\big(h(x)\big)\right) = c * c * c * x$
Which can be done by just one layer with an activation function of; $h(x) = c^3x$

Next Lecture

Deeper Neural Networks and more Activation Functions


INPUT LAYER
HIDDEN LAYERS
OUTPUT LAYER