



# COS10022 – DATA SCIENCE PRINCIPLES

Dr Pei-Wei Tsai (Lecturer, Unit Convenor)  
ptsai@swin.edu.au, EN508d

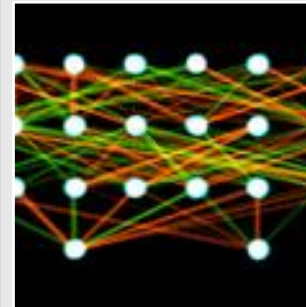
SWIN  
BUR  
NE

SWINBURNE  
UNIVERSITY OF  
TECHNOLOGY





# Week 10



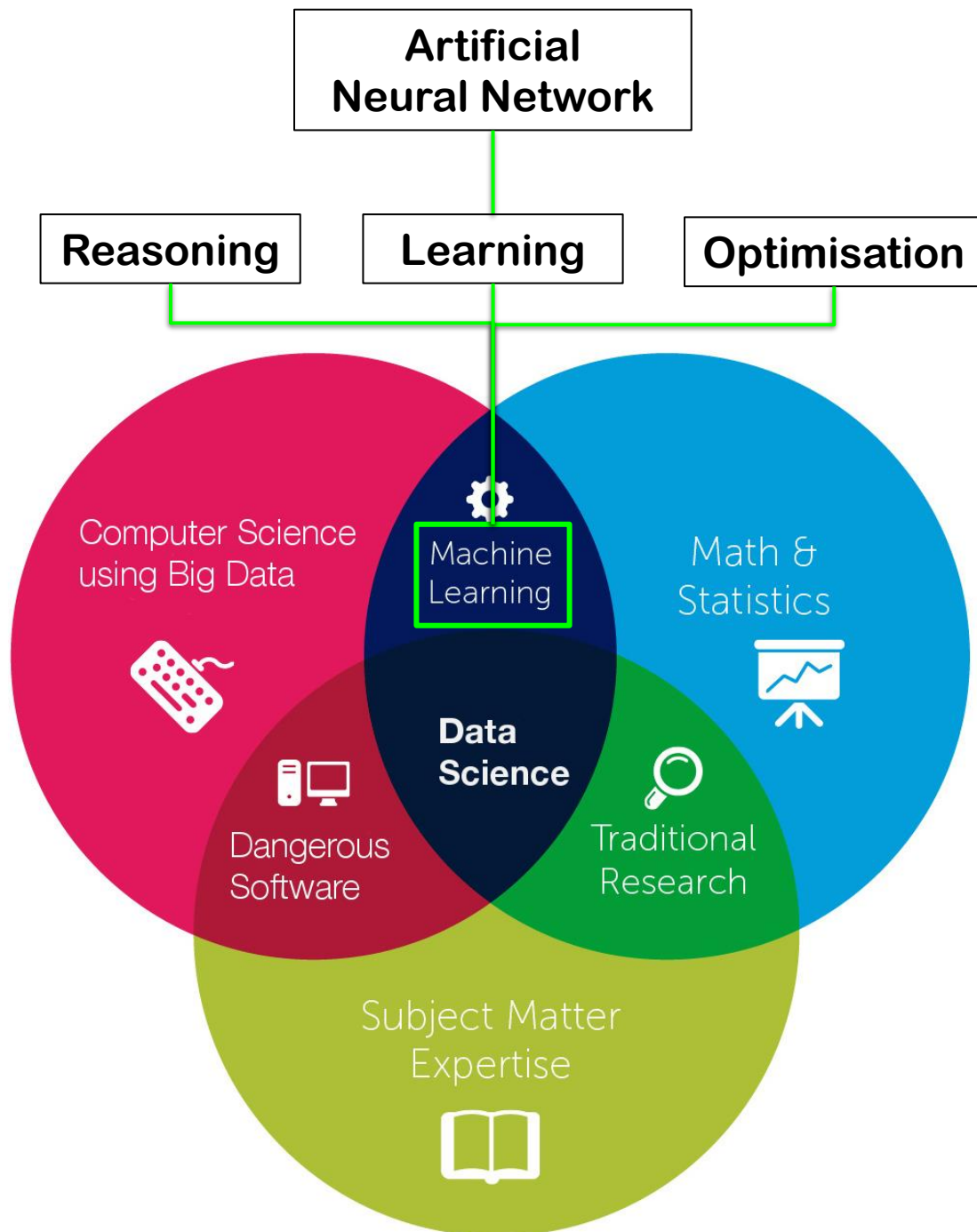
**Advanced Predictive Model:**

- Artificial Neural Network





# ARTIFICIAL NEURAL NETWORK (ANN)

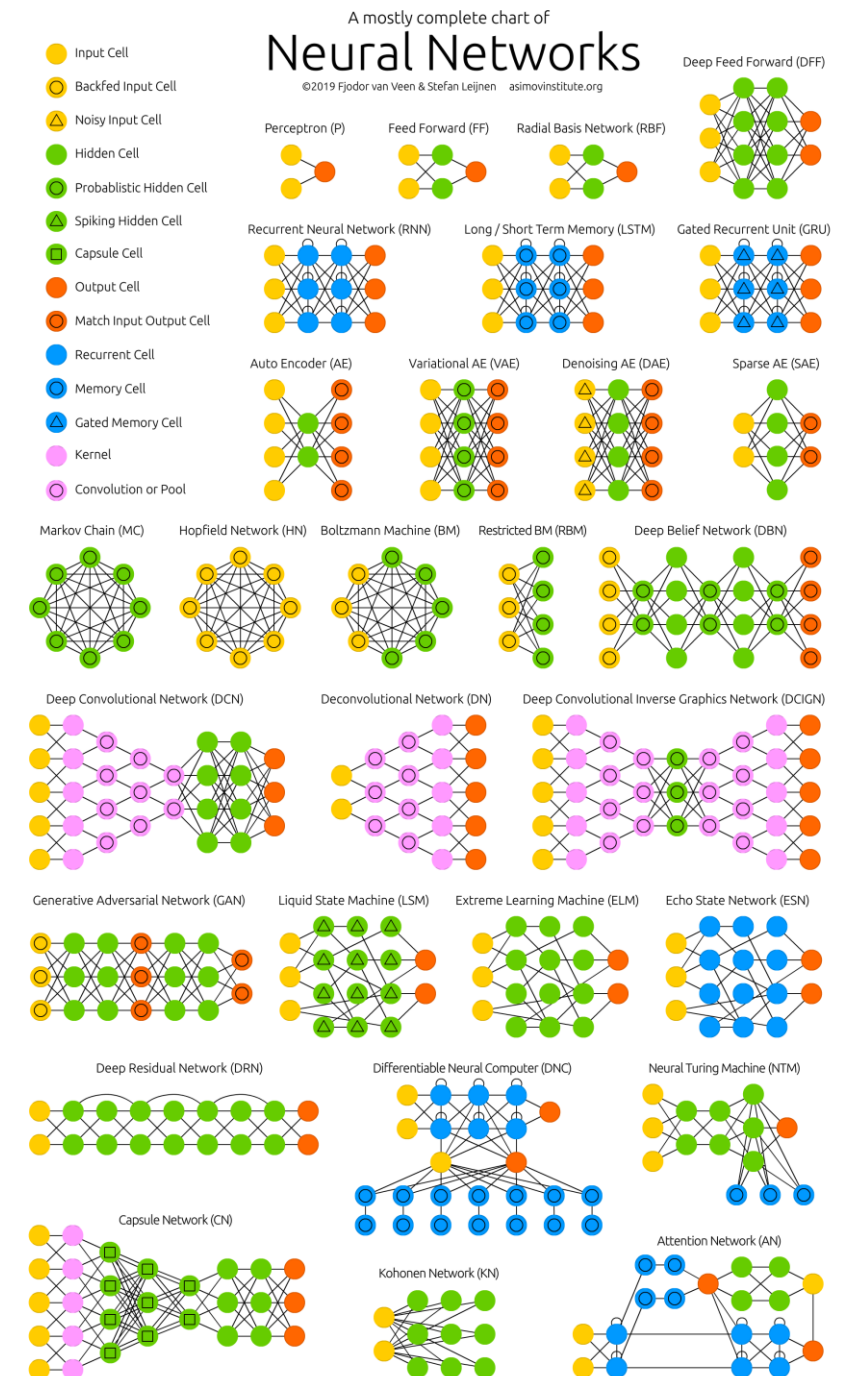


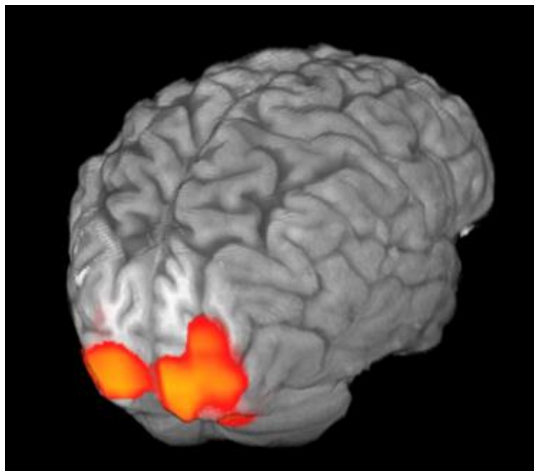
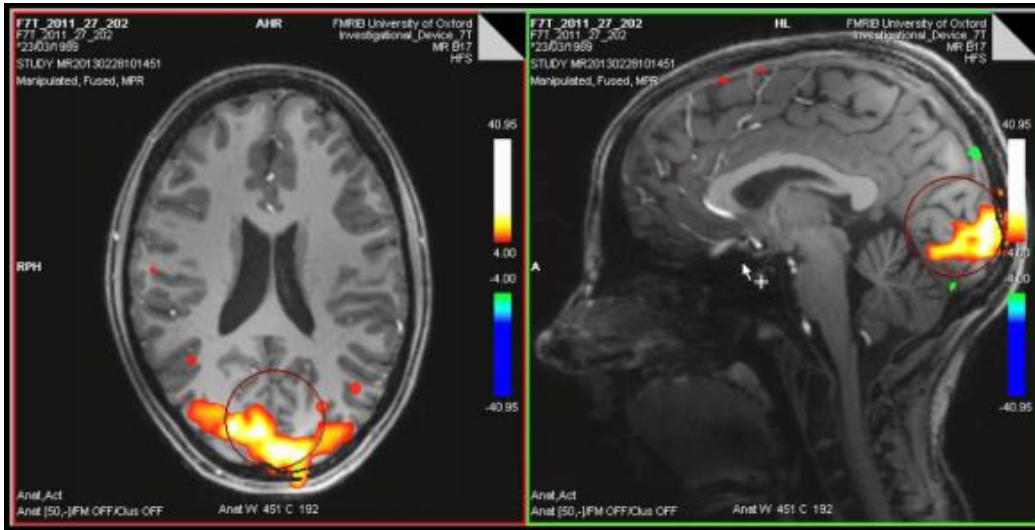
**WHERE  
DOES  
ANN  
STAND?**



# TYPES OF ARTIFICIAL NEURAL NETWORKS

- Artificial Neural Network (ANN) can be considered as a set of nodes (neurons) combined with the designed patterns.
- The weights on the connections between nodes are adjusted in the training process.
- Alike other classifiers introduced before,
- In IDS, we will focus on the shallow networks instead of the deep structures.



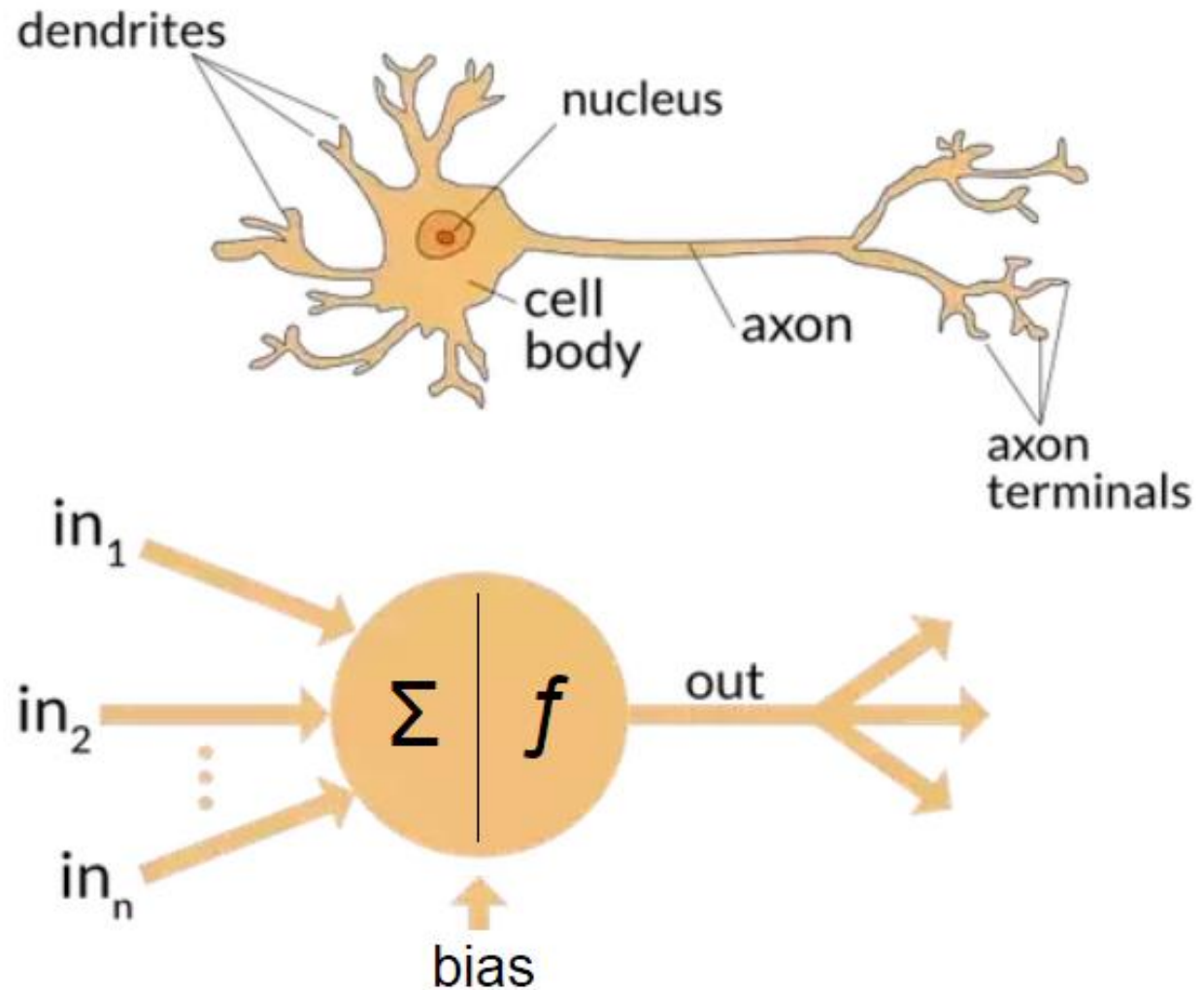


# Neural Network in Human Brain

- Our brains, in average, contain 86 billion neurons (nerve cells).
- A single neuron may receive one to multiple inputs from other neurons and sending the biocurrent to the corresponding output(s) when the certain criteria are satisfied.
- Nowadays, the actual mechanism of how human brain works is still a mystery.
- However, artificial neural network somehow achieves learning by imitating the human brain neuron behaviour.

# How Does a Neuro Work?

- Input signals and the bias value are collected and the defined calculation is performed in the neuron.
- If the calculation result satisfies the output criteria, the neuron sends out the output signal to the desired destination.

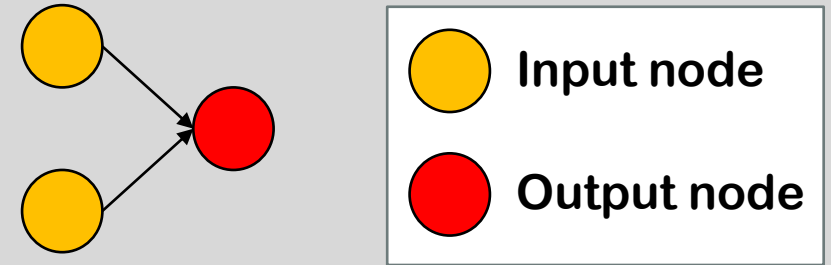








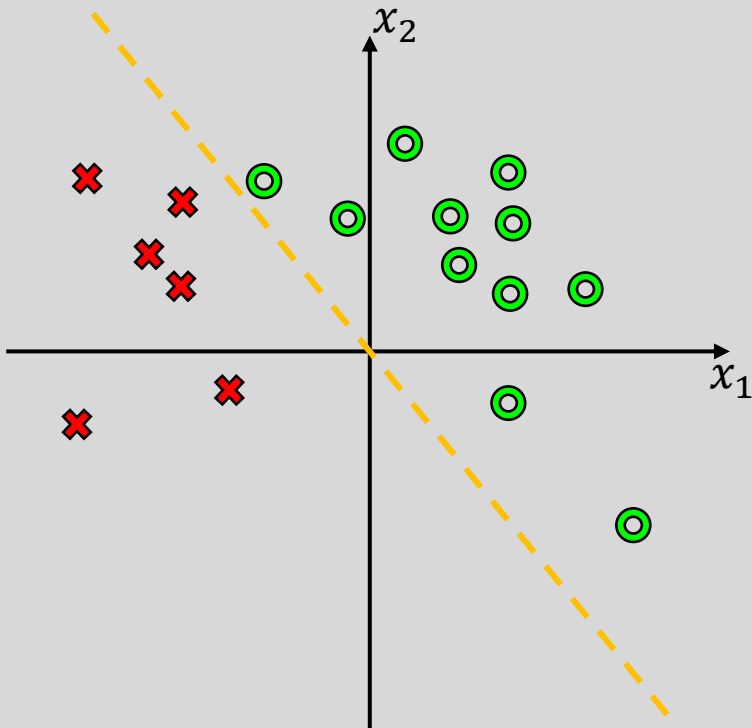
# Perceptron (P)



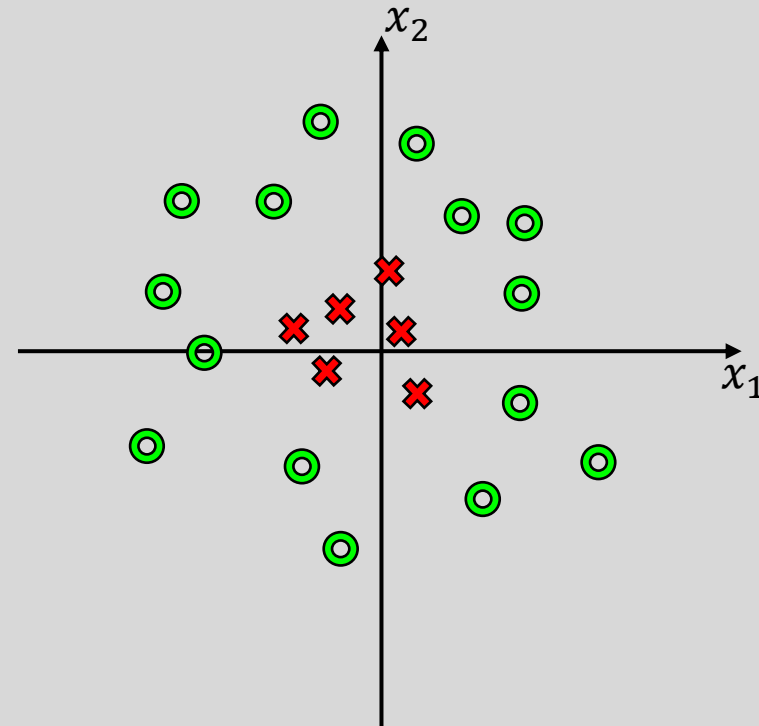
- Straight forward: Feed information from the front to the back (input to the output).
- This type of simplest but practical Neural Network (NN) can be used to model logic gates for performing the logical operation.
- Training
  - Type: **supervised learning**.
  - Method: **back-propagation**. (The back-propagated error is usually the variation of the difference between the input and the output such as the linear difference, MSE, etc.)
- Application
  - The simple structure limits the applications, but they are popularly combined with other networks to form new NN structures.
- Origin
  - Frank Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review* 65(6), pp. 386-408, 1958.

# Problem Definition

Linearly Separable



Not Linearly Separable





# Parameters (Coefficients) of a Perceptron

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_n \end{bmatrix}$$

$\omega$ : A vector, which length is the no. of dimension you have plus one.

$\nu$ : The step size controlling how fast the learning process will terminate.

A constant  $X_j = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix},$   
 $j = 1, \dots, m$  and  
 $X = \{X_1, \dots, X_m\}$

Input

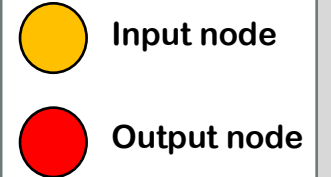
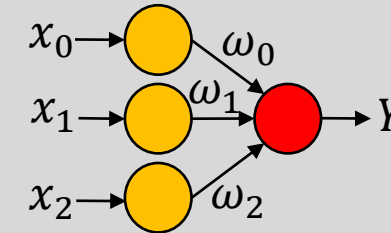
Weights

Learning Rate

Input Data

Output

Final result

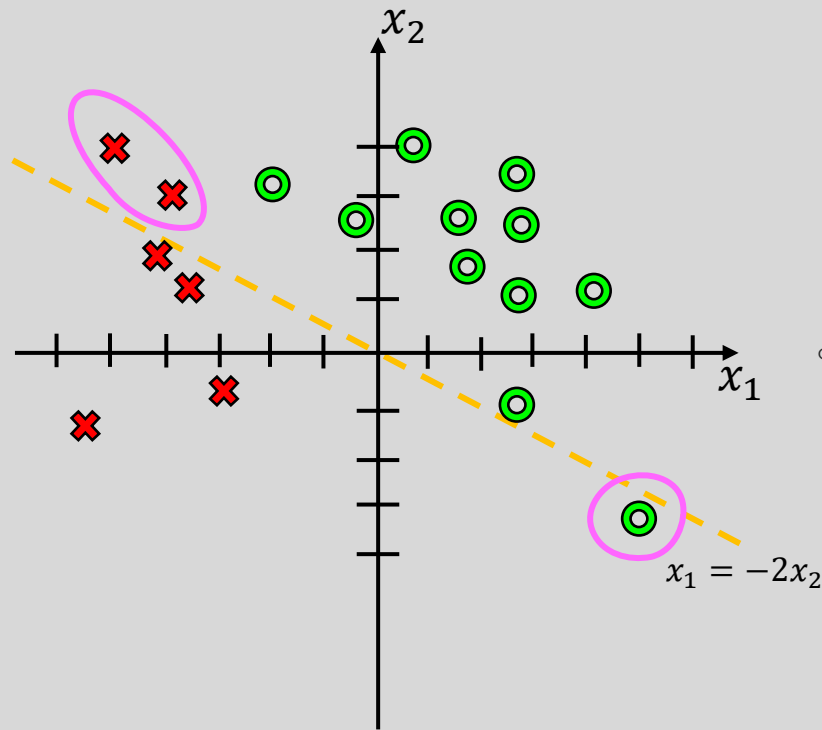


$$y = \omega^T X = \sum_{i=0}^n \omega_i \cdot x_i$$

$$Y = \begin{cases} \text{green circle}, & \text{if } y > 0 \\ \text{red X}, & \text{if } y \leq 0 \end{cases}$$

$$X_j = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \text{ where } x_0 = 1$$

# Let's Get Back to the Given Example



- Initialisation

- Randomly assign the values to the vector  $\omega$  and decide a value for  $v$ .

$$\omega = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \rightarrow y = \omega^T X = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = x_1 + 2x_2 > 0 \rightarrow \text{green circle}$$

$v = 0.06$

$$\rightarrow x_1 > -2x_2$$

- Update

- Update  $\omega$  to adjust the line into the right location by  $\omega'_i = \omega_i + \sum_{j=1}^m v \cdot d_j \cdot x_i$ , where  $\omega'_i$  and  $\omega_i$  represents the new and the old  $\omega$ , respectively.

$$d_j = \begin{cases} 1, & \text{if } X_j \text{ should be in the upper plan} \\ -1, & \text{if } X_j \text{ should be in the lower plan} \\ 0, & \text{if } X_j \text{ is already in the correct plan} \end{cases}$$



# Let's Get Back to the Given Example

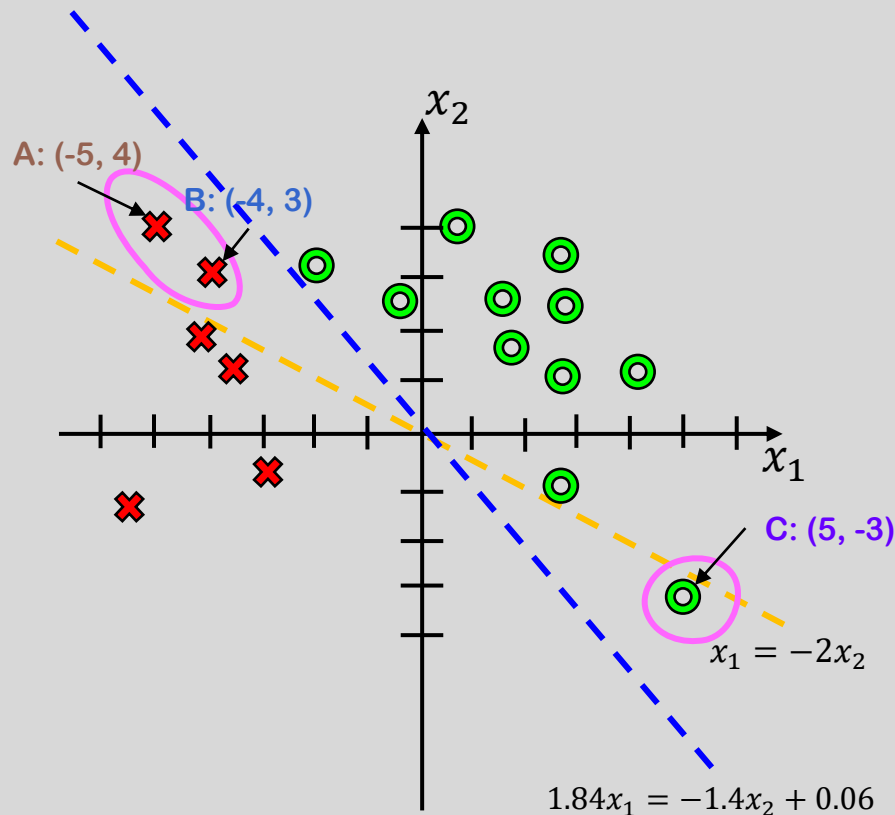
## Update

- Update  $\omega$  to adjust the line into the right location by  $\omega'_i = \omega_i + \sum_{j=1}^m v \cdot d_j \cdot x_i$ , where  $\omega'_i$  and  $\omega_i$  represents the new and the old  $\omega$ , respectively.

$$d_j = \begin{cases} 1, & \text{if } X_j \text{ should be in the upper plan} \\ -1, & \text{if } X_j \text{ should be in the lower plan} \\ 0, & \text{if } X_j \text{ is already in the correct plan} \end{cases}$$

- $\omega'_0 = \omega_0 + [(0.06)(-1)(1)] + [(0.06)(-1)(1)] + [(0.06)(1)(1)] = -0.06$
- $\omega'_1 = \omega_1 + [(0.06)(-1)(-5)] + [(0.06)(-1)(-4)] + [(0.06)(1)(5)] = 1.84$
- $\omega'_2 = \omega_2 + [(0.06)(-1)(4)] + [(0.06)(-1)(3)] + [(0.06)(1)(-3)] = 1.4$

- $\rightarrow y = \omega^T X = [-0.06 \quad 1.84 \quad 1.4] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = 1.84x_1 + 1.4x_2 - 0.06$



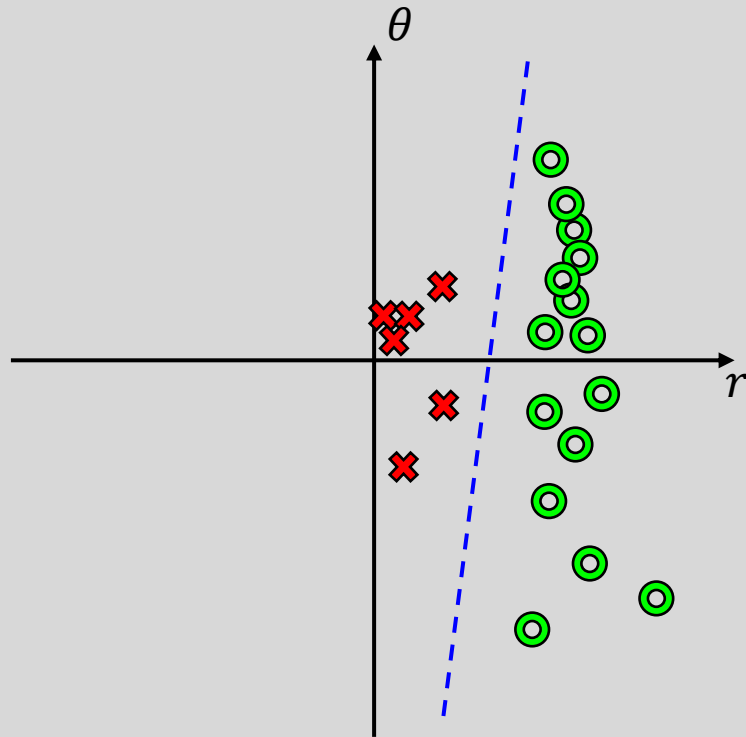
$x_0$

$x_1$

$x_2$

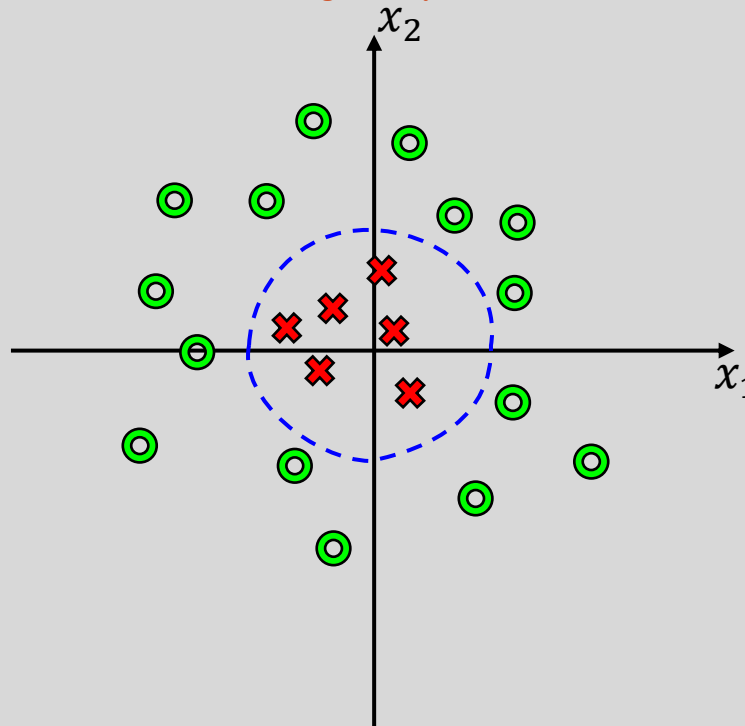
# What About the Not Linearly Separable Case?

Convert to Different  
Coordination System

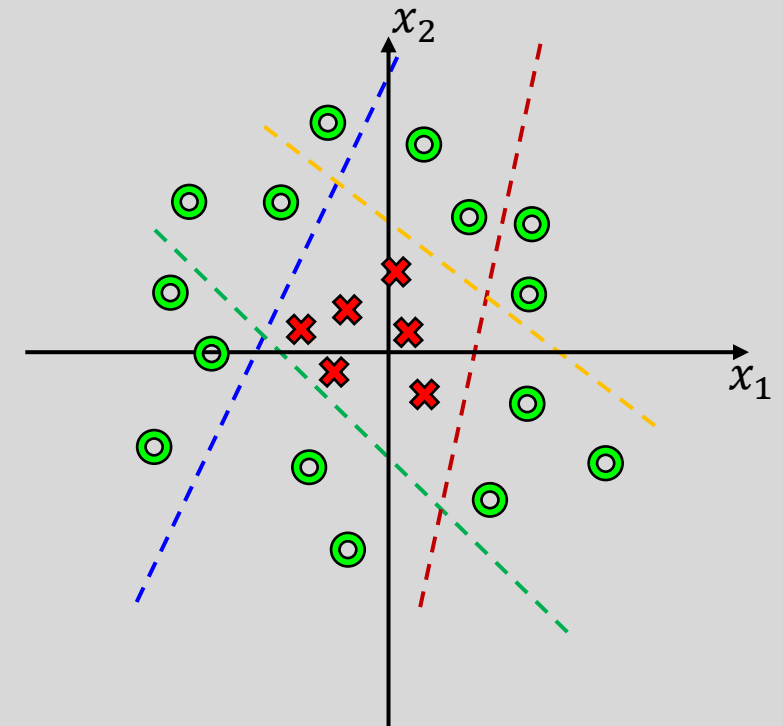


$(x_1, x_2) \rightarrow (r, \theta)$  in Polar Coordination

Not Linearly Separable

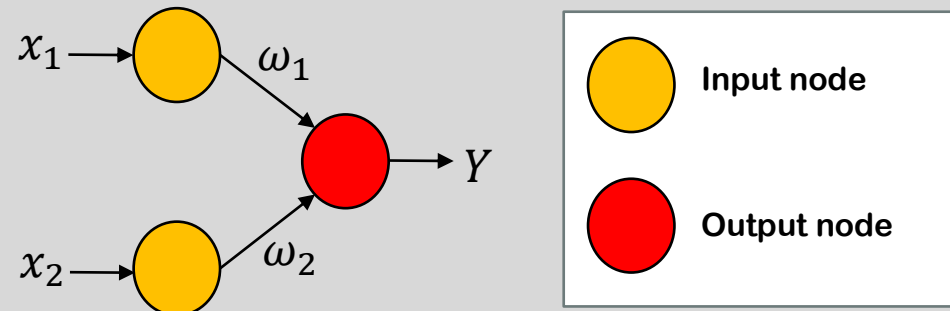
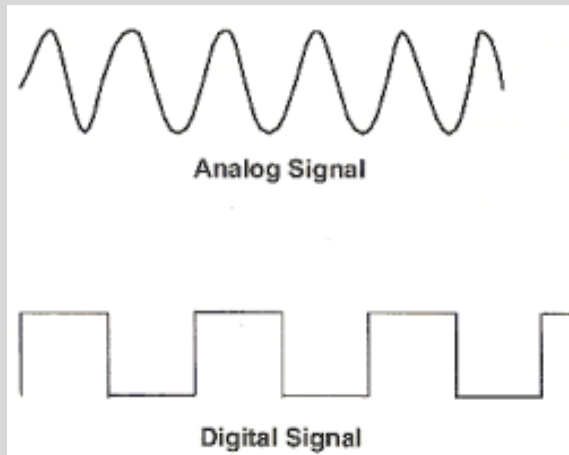


Stacking of Perceptrons





# Example 2: Logical Gates



- The perceptron is not only capable of making simple classifications but also suitable for imitating logical gates.
- It can be used at the final layer of other ANNs to make decisions.
- The perceptron is not only suitable to deal with the **digital** signal but also preserves the ability of taking **analog** signals as inputs.

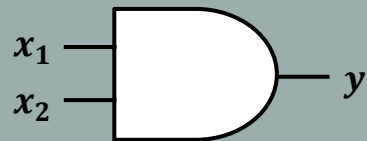
# Background Knowledge: Boolean Algebra

## AND

Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

$$y = x_1 \text{ AND } x_2$$

$$y = x_1 \times x_2$$



## OR

Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

$$y = x_1 \text{ OR } x_2$$

$$y = x_1 + x_2$$



## Exclusive OR (XOR)

Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

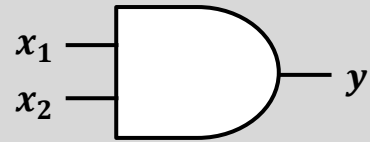
$$y = x_1 \text{ XOR } x_2$$

$$y = x_1 \oplus x_2$$



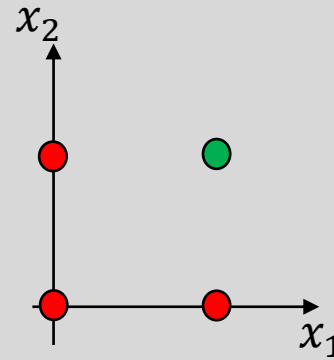


# Example 2-1: AND Gate



$$y = x_1 \times x_2$$

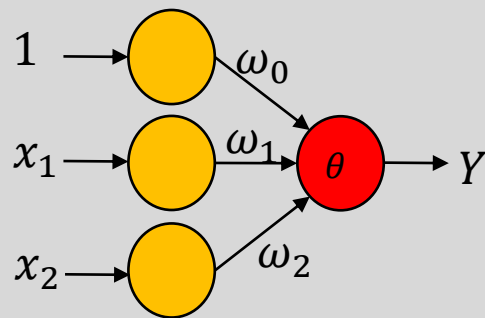
Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1



$$y = (\omega_0 \times 1) + (\omega_1 \times x_1) + (\omega_2 \times x_2)$$

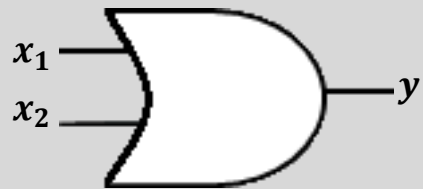
$$\theta = 0.2$$

$$Y = \begin{cases} 0, & \text{if } y < \theta \\ 1, & \text{if } y \geq \theta \end{cases}$$



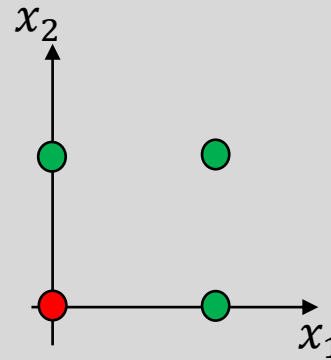
Input		Weight			Value	Output
$x_1$	$x_2$	$\omega_0$	$\omega_1$	$\omega_2$	$y$	$Y$
0	0	-0.4	0.4	0.4	$(-0.4) + (0.4 \times 0) + (0.4 \times 0) = -0.4 < \theta$	0
0	1				$(-0.4) + (0.4 \times 0) + (0.4 \times 1) = 0 < \theta$	0
1	0				$(-0.4) + (0.4 \times 1) + (0.4 \times 0) = 0 < \theta$	0
1	1				$(-0.4) + (0.4 \times 1) + (0.4 \times 1) = 0.4 \geq \theta$	1

# Example 2-2: OR Gate



$$y = x_1 + x_2$$

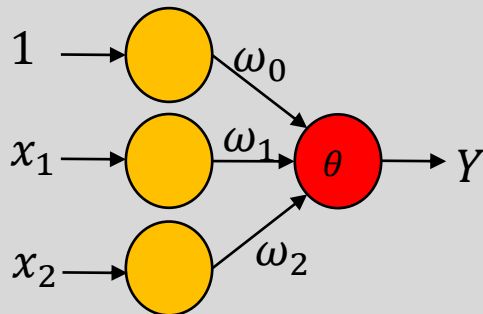
Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1



$$y = (\omega_0 \times 1) + (\omega_1 \times x_1) + (\omega_2 \times x_2)$$

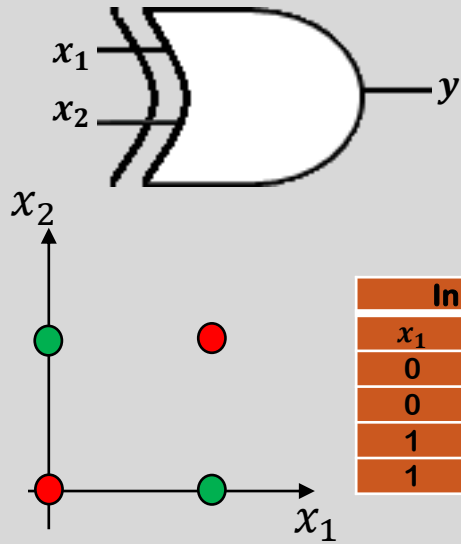
$$\theta = 0.5$$

$$Y = \begin{cases} 0, & \text{if } y < \theta \\ 1, & \text{if } y \geq \theta \end{cases}$$



Input		Weight			Value	Output
$x_1$	$x_2$	$\omega_0$	$\omega_1$	$\omega_2$	$y$	$Y$
0	0	0.2	0.4	0.4	$(0.2) + (0.4 \times 0) + (0.4 \times 0) = 0.2 < \theta$	0
0	1				$(0.2) + (0.4 \times 0) + (0.4 \times 1) = 0.6 \geq \theta$	1
1	0				$(0.2) + (0.4 \times 1) + (0.4 \times 0) = 0.6 \geq \theta$	1
1	1				$(0.2) + (0.4 \times 1) + (0.4 \times 1) = 1.0 \geq \theta$	1

# Example 2-3: XOR Gate

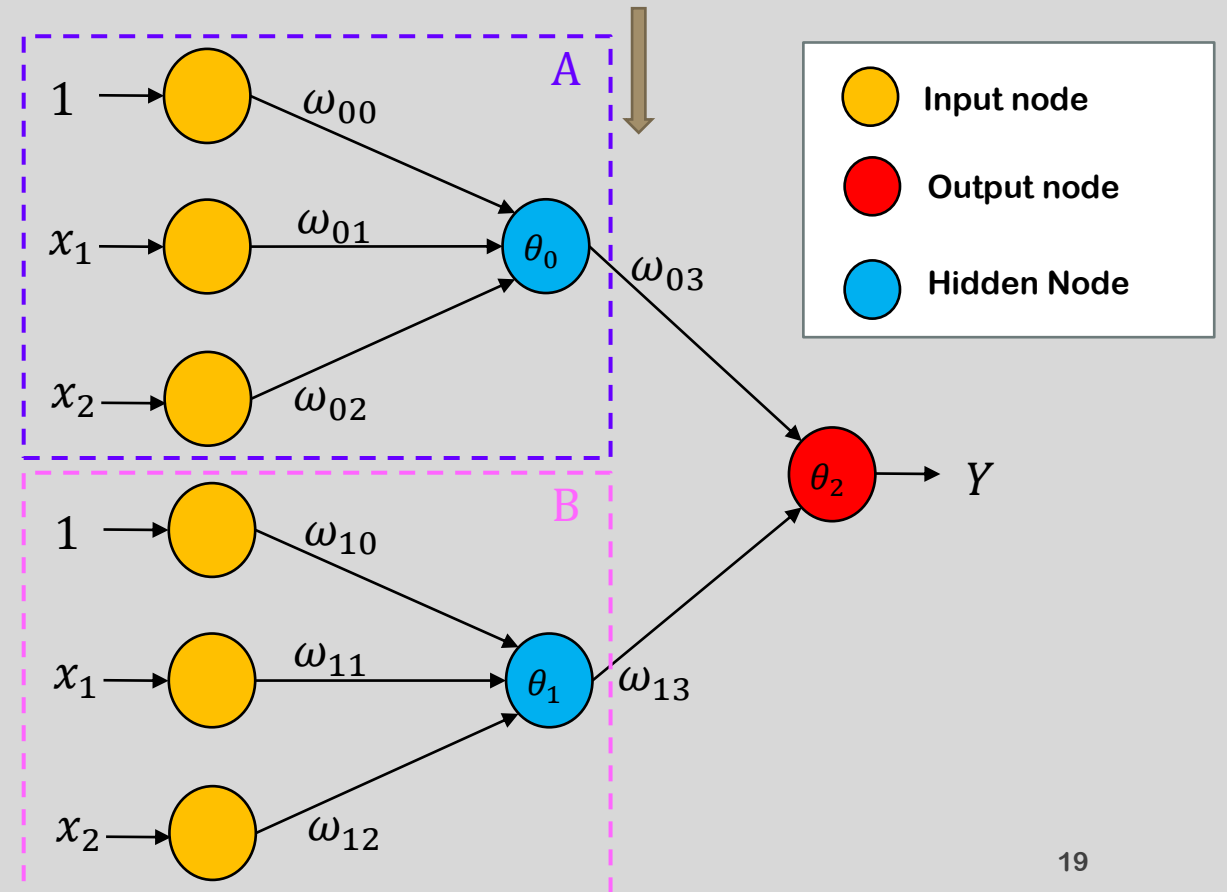
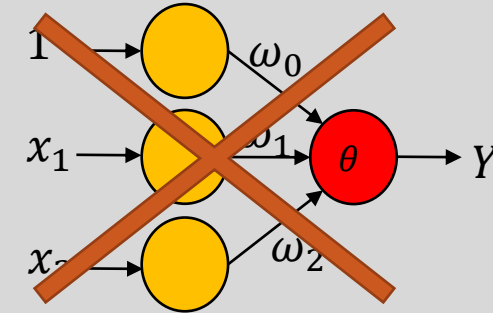


$$y = x_1 \oplus x_2$$

Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

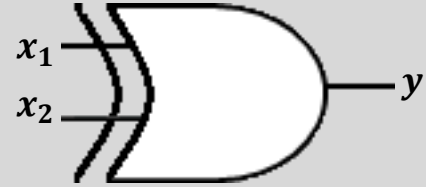
Input		A	B	B-A
$x_1$	$x_2$	$x_1 \times x_2$	$x_1 + x_2$	$x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- The single layer perceptron can no longer take care of this case.
- To solve the exclusive or operation problem, we need to move on to the **multilayer perceptron model**.



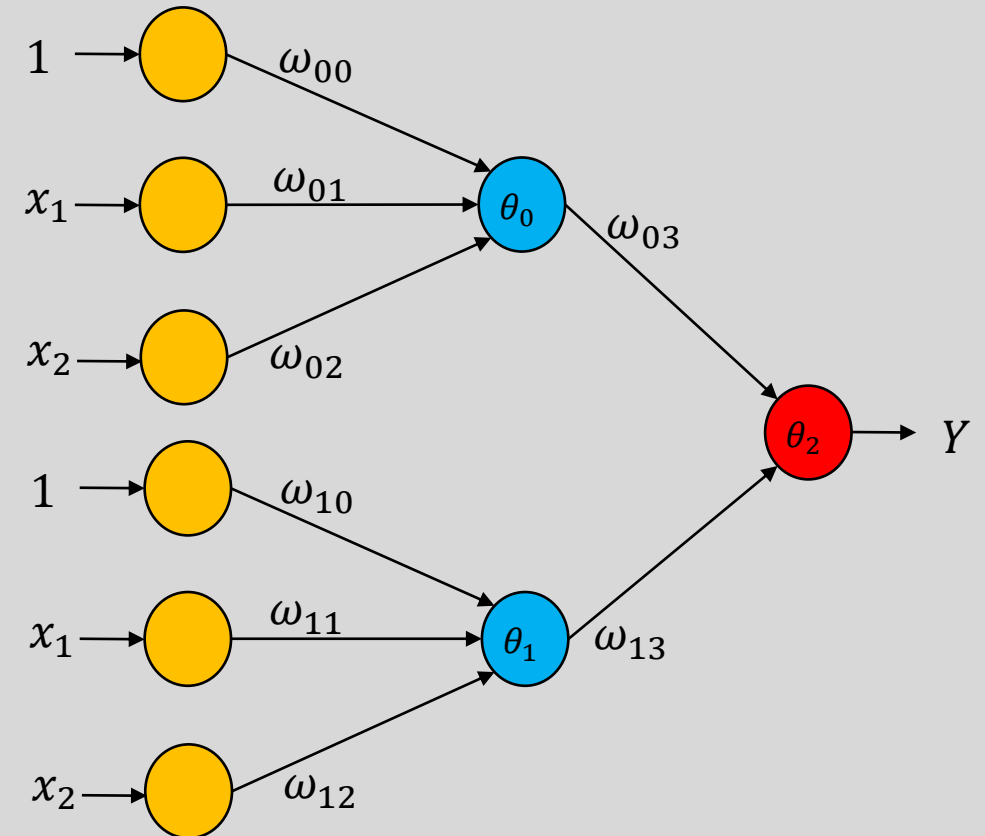
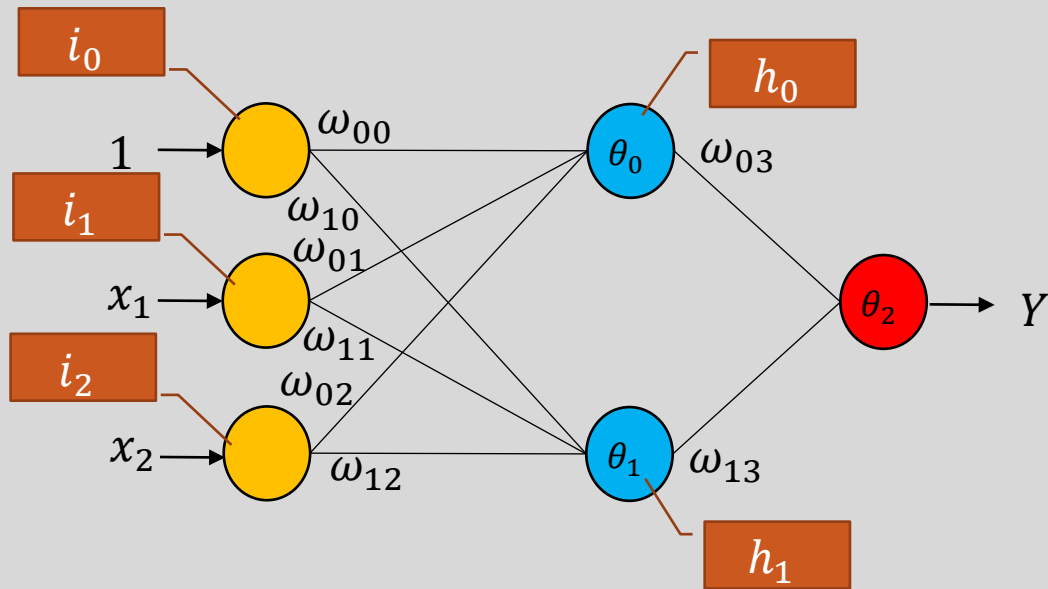


# Example 2-3: XOR Gate



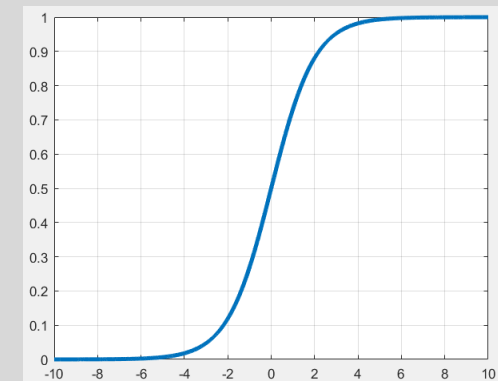
$$y = x_1 \oplus x_2$$

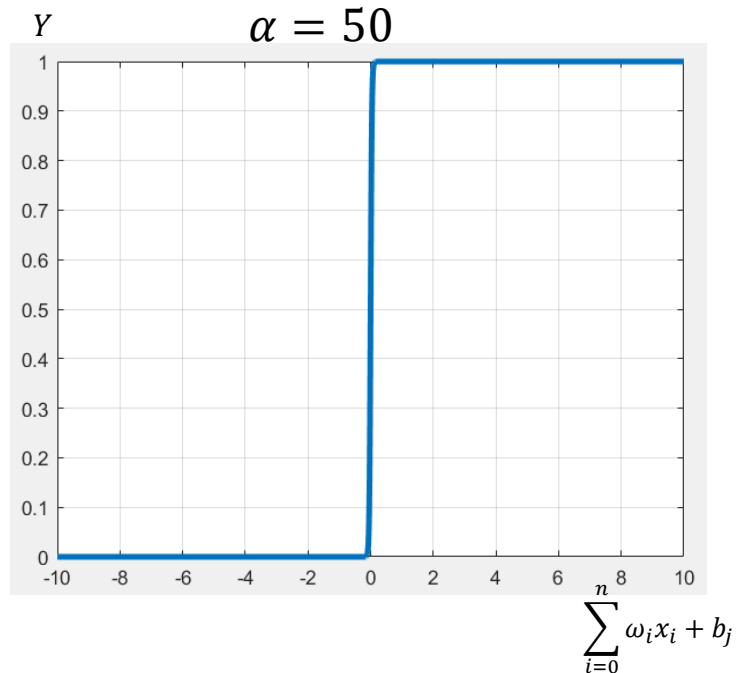
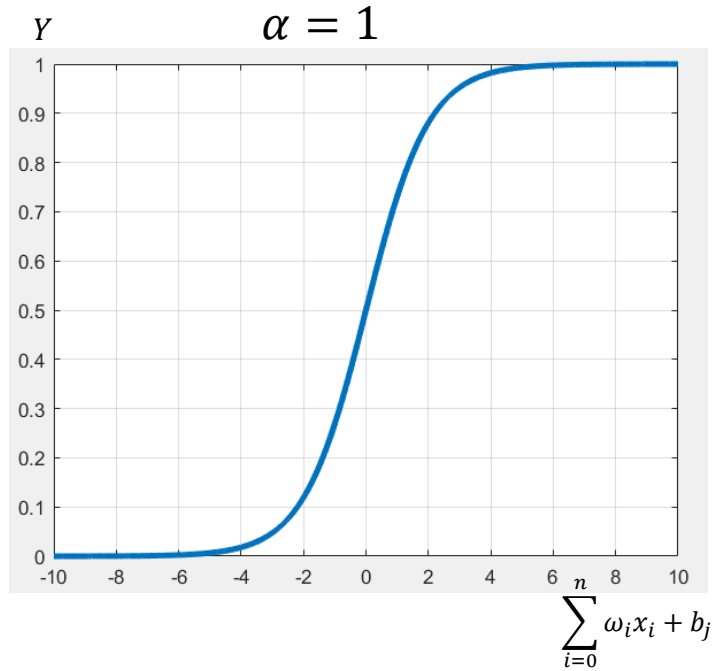
Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0



$$h_m = \sum_{k=0}^n i_k \times \omega_{mk}$$

$$\text{sigmoid}(x) = \frac{1}{1+e^{-ax}}$$



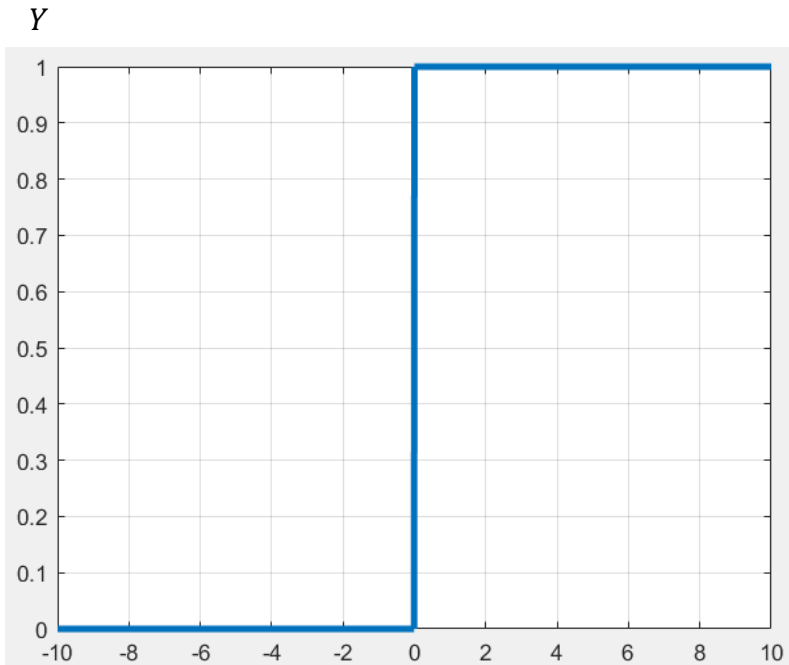


# Background Knowledge

- **Sigmoid function:**

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

- **Usually used to round the output of a node.**
- **$\alpha$  is a coefficient controlling the sharpness of the curve.**



$$\sum_{i=0}^n \omega_i x_i + b_j$$

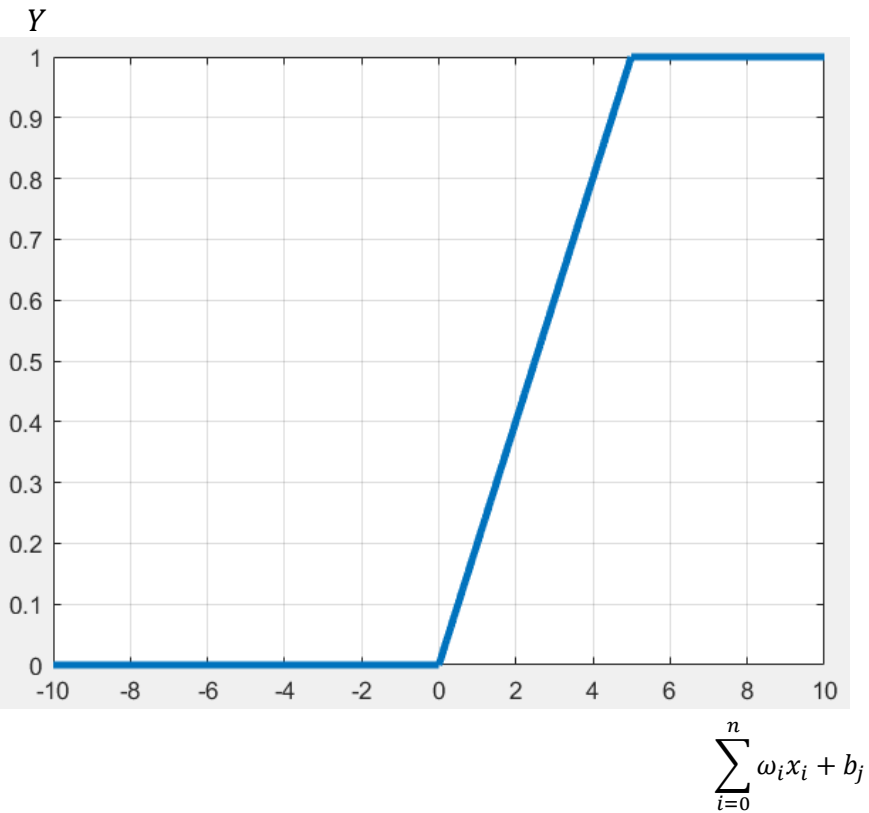
# Background Knowledge

- **Threshold function:**

$$\emptyset(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- **Usually used to determine whether a neuro will fire (activate the output).**
- **Work based on the combination of a step function and a threshold.**



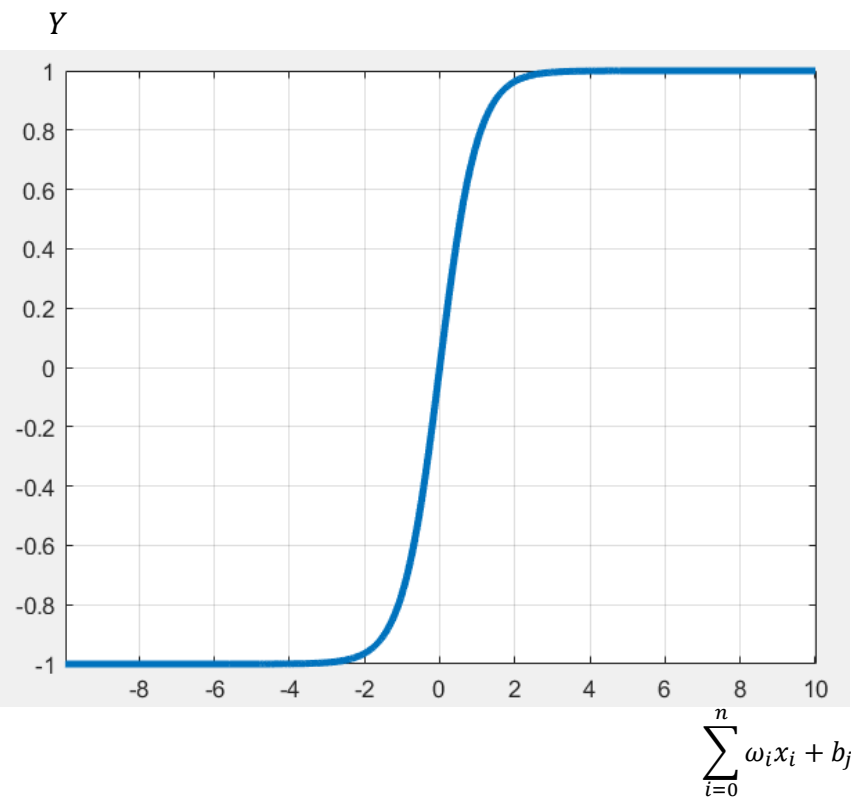


# Background Knowledge

- **Relu function:**

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

- **Cut off anything exceed 1 and anything below 0.**
- **Allow anything between [0,1].**
- **Feeds the actual value to the next layer.**



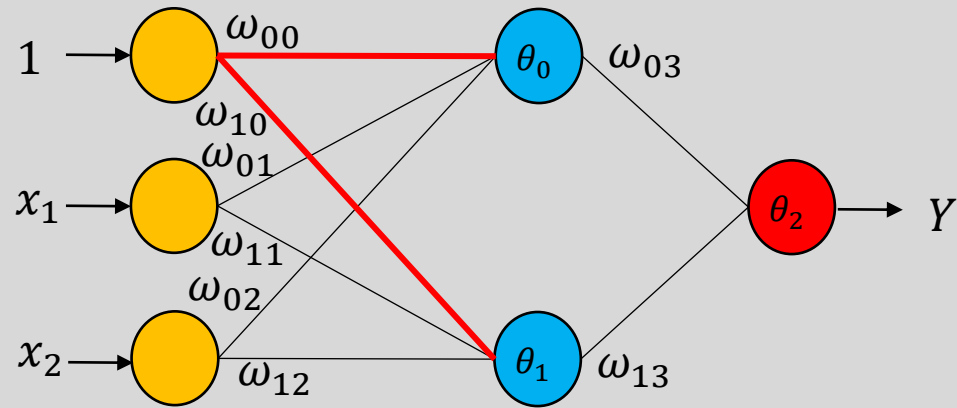
# Background Knowledge

- Hyperbolic Tangent function:

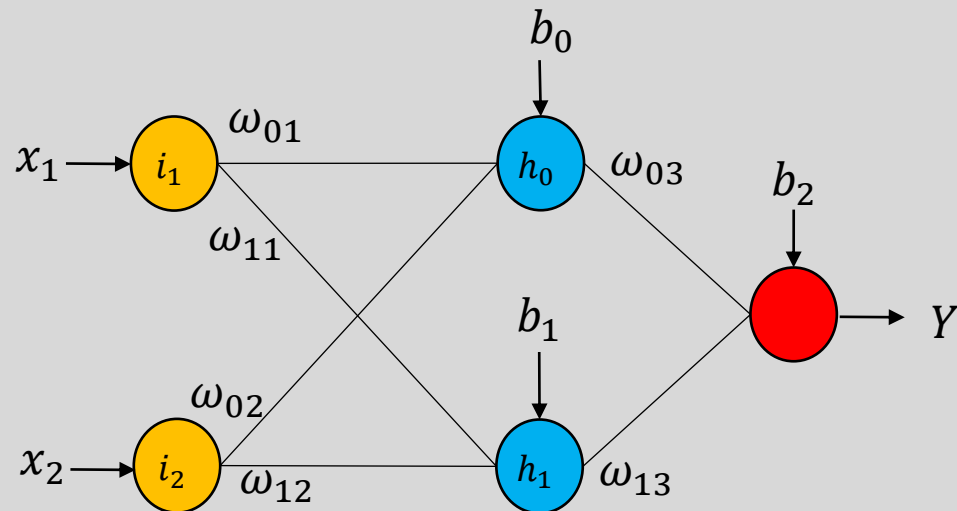
$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

- Very similar to the sigmoid function but the range is between  $[-1, 1]$ .

# Example 2-3: XOR Gate



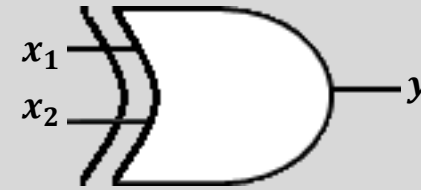
||



$$h_m = \sum_{k=1}^n i_k \times \omega_{mk} + b_m$$

If  $\omega = \begin{bmatrix} b_0 \\ \omega_{01} \\ \omega_{02} \\ \omega_{03} \\ b_1 \\ \omega_{11} \\ \omega_{12} \\ \omega_{13} \\ b_2 \end{bmatrix} = \begin{bmatrix} -10 \\ 20 \\ 20 \\ 20 \\ 30 \\ -20 \\ -20 \\ 20 \\ -30 \end{bmatrix}$ , the result values will be

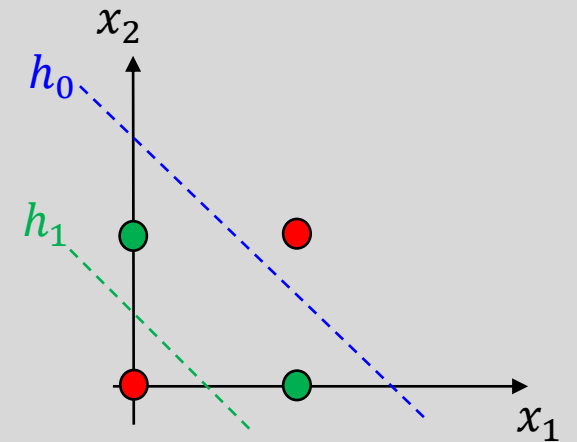
$4.5 \times 10^{-5}$  and  $9.9 \times 10^{-1}$ .



$$y = x_1 \oplus x_2$$

Input		Output
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

$h_0 = \text{sigmoid}(\omega_{01}x_1 + \omega_{02}x_2 + b_0) = 1.5$   
 $h_1 = \text{sigmoid}(\omega_{11}x_1 + \omega_{12}x_2 + b_1) = 0.5$   
 $\rightarrow$  For  $h_0: x_1 + x_2 \approx 1.5$   
 For  $h_1: x_1 + x_2 \approx 0.5$

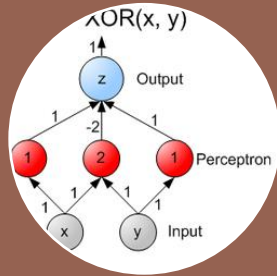






# FEED FORWARD NEURAL NETWORK (FFNN)

# Feed Forward Neural Network (FFNN)

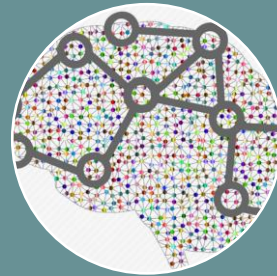


Perceptron is also a type of the FFNN.



## Single Direction

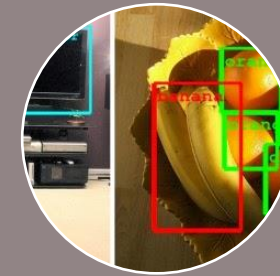
- The connection (data travel direction) is all straight forward from the **input** moving to the **output**.
- There is no loop going back to the previous layers.



The only time using the backpropagation is in the training phase to minimise the differences between the target value and the ANN output.



When deploying the trained ANN in an application, the computation is very fast and thus is capable of using in real-time scenario.



## Applications

- Vision Recognition
- Speech Recognition
- Etc.

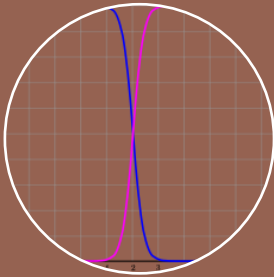






# RADIAL BASIS FUNCTION (RBF)

# Radial Basis Function Neural Network (RBF)



## Characteristics

- Classifies the data point based on its distance from a centre point.
- No complicated training but simply gathering things together



## Unsupervised Learning

- Since it is an unsupervised model, you may discover things are connected together in a way you never noticed before.



## Applications

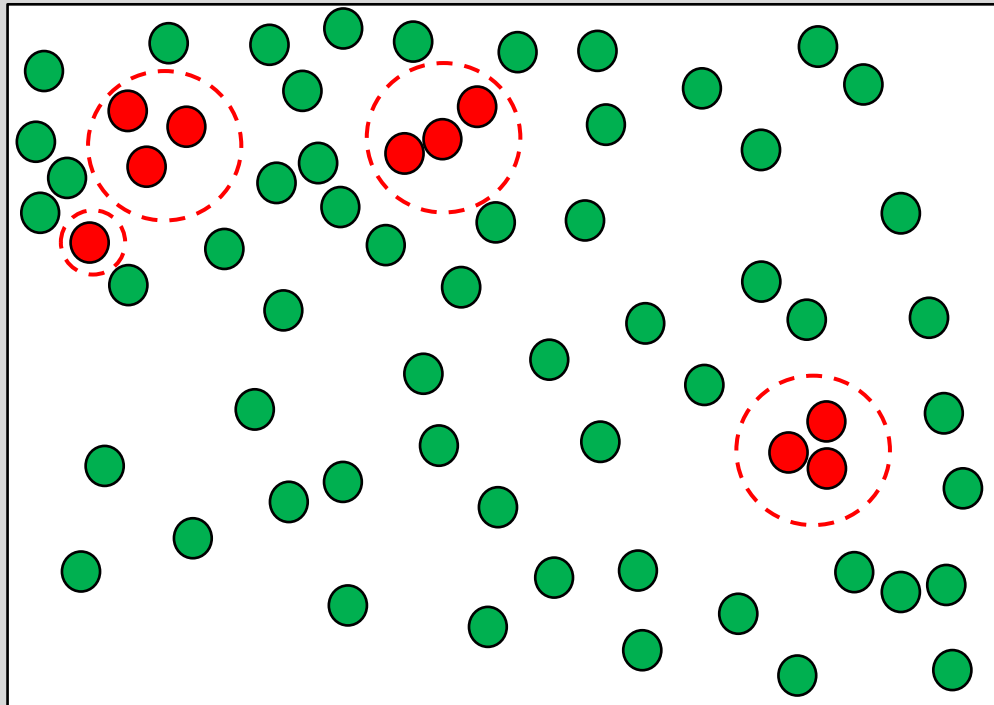
- Power restoration systems.
- Hand writing digit recognition.
- Interpretation (predicting data in the gaps).





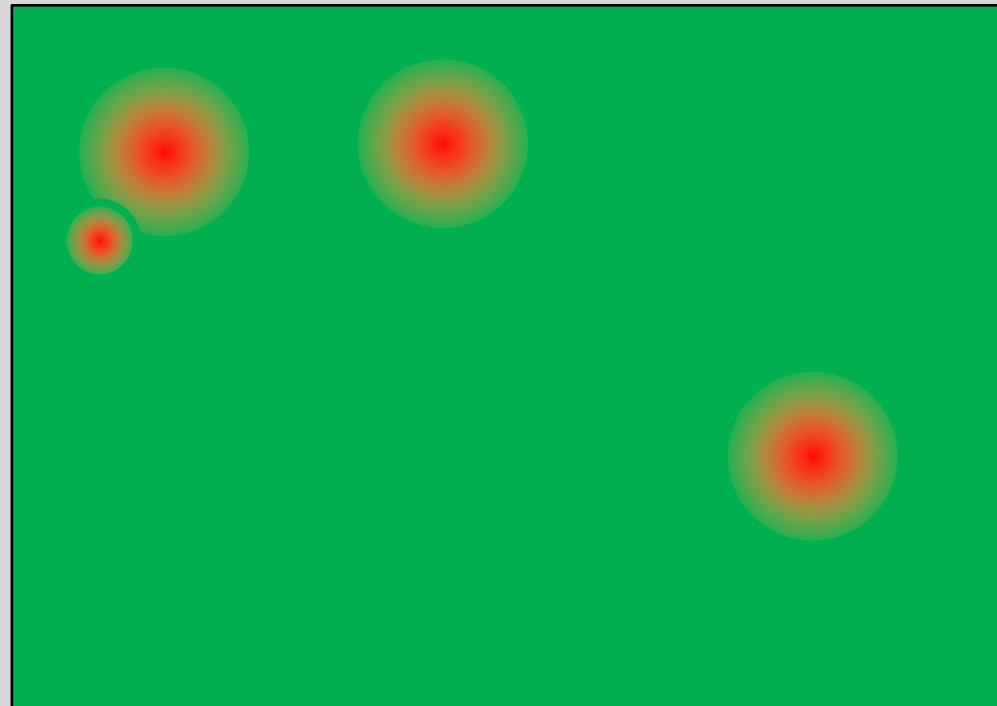
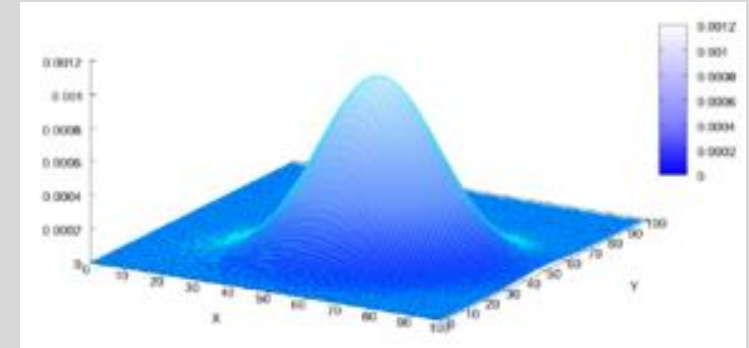
# How RBF Works?

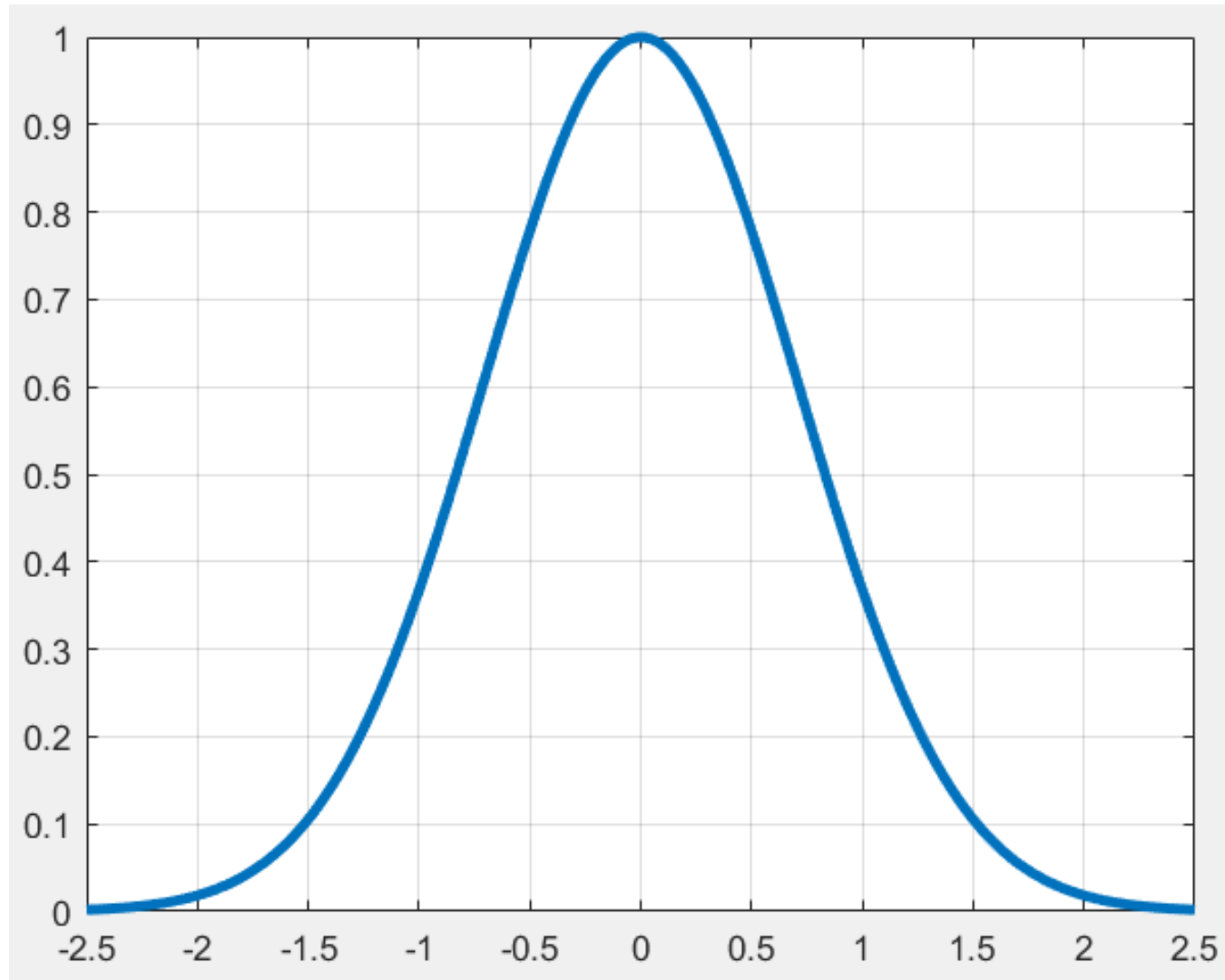
- Assume we have a pool of data
- RBF draws circles instead of lines to split them.



# How RBF Works?

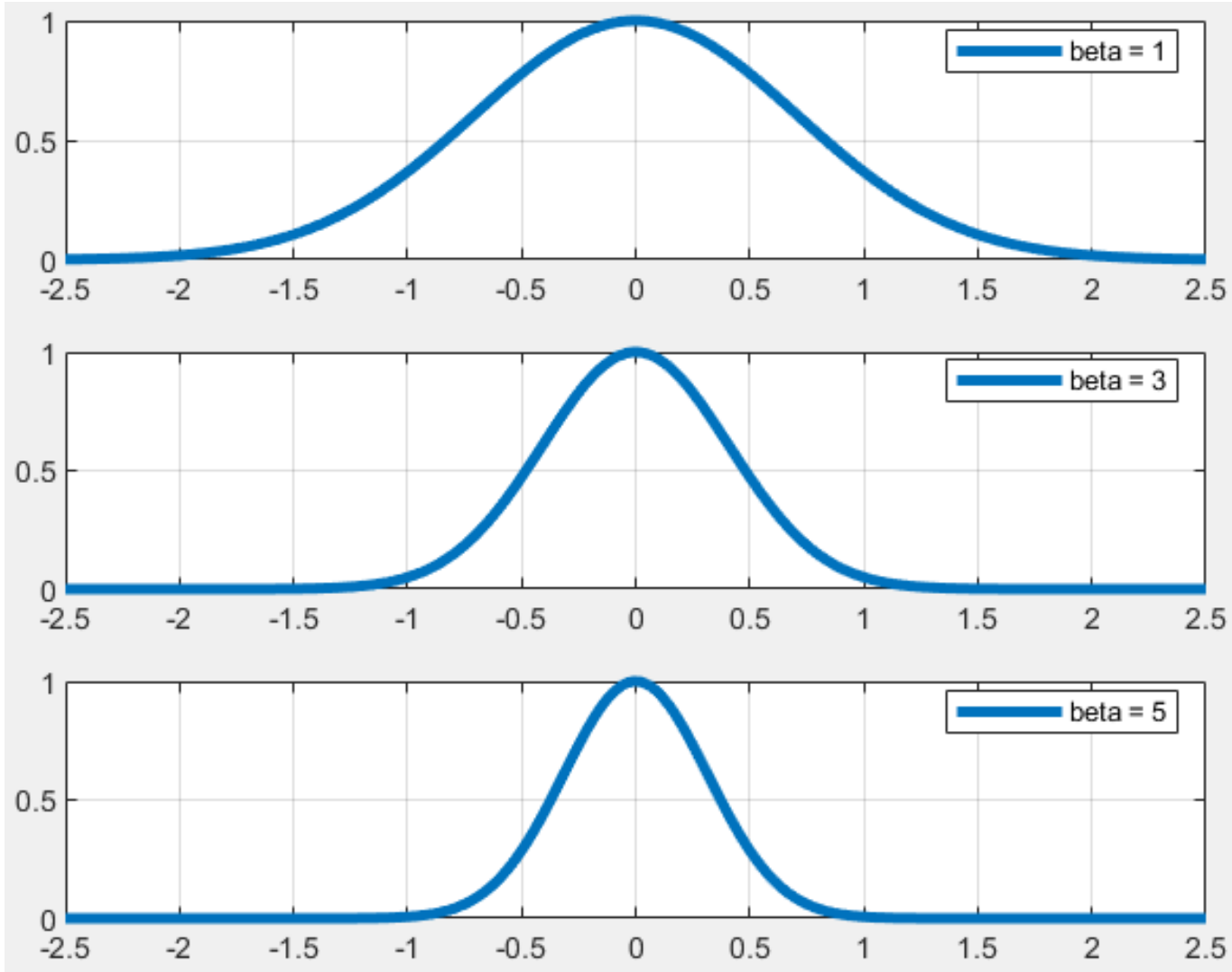
- RBF cluster boundaries are not sharp.
- They follow the Gaussian distribution.
- Now we can model “how confident we are” to classify a data point into a specific group.





## Radial Basis Function

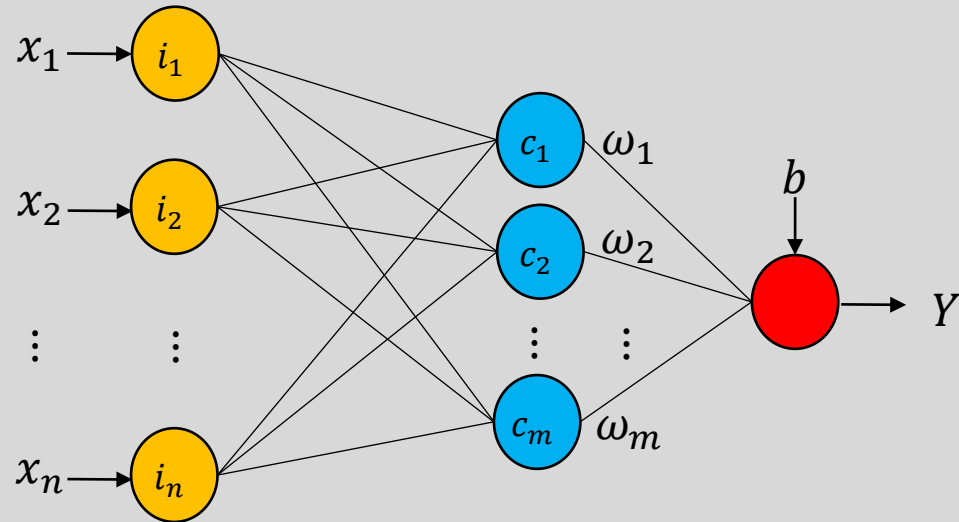
- $\phi(D) = e^{-D^2}$   
where  $D$  is the distance between the data point to the centre of a circle.
- The distance can be calculated based on different measurements.



## RBF: How to decide the radius of a circle?

- $\phi(D) = e^{-\beta D^2}$   
where  $D$  is the distance between the data point to the centre of a circle and  $\beta$  is the parameter controls the radius of the circle.

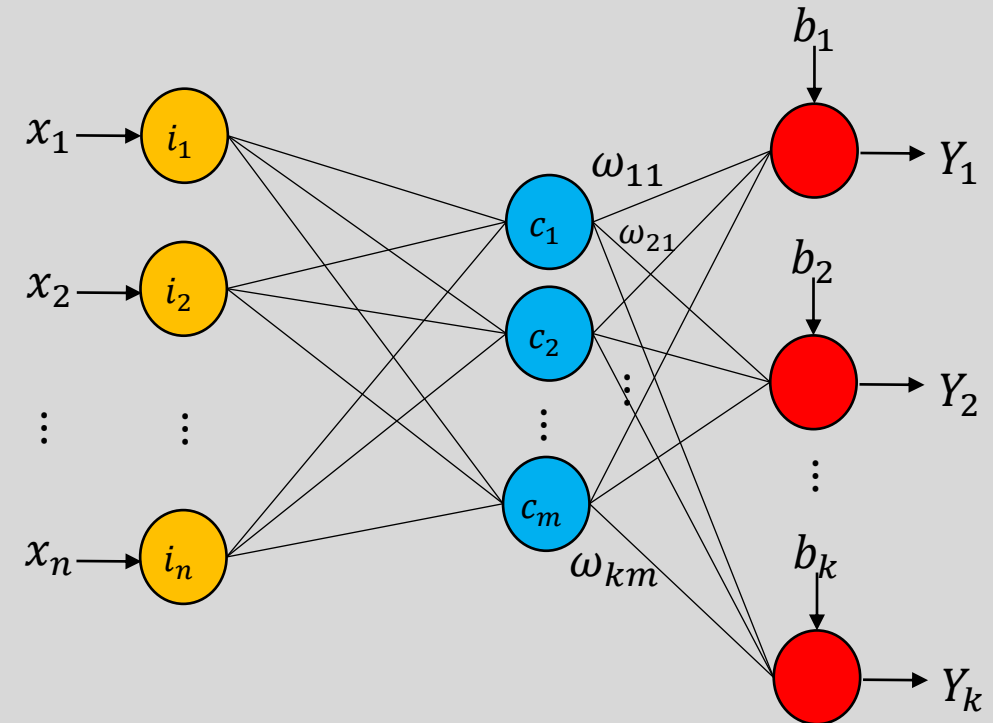
# RBF Model



$$c_j = e^{-\beta \cdot D(x, c_j)^2}$$

$$Y = \sum_{j=1}^m \omega_j \cdot c_j + b$$

- If you have more outputs, you'll have multiple output nodes.





# RBF Training Methods

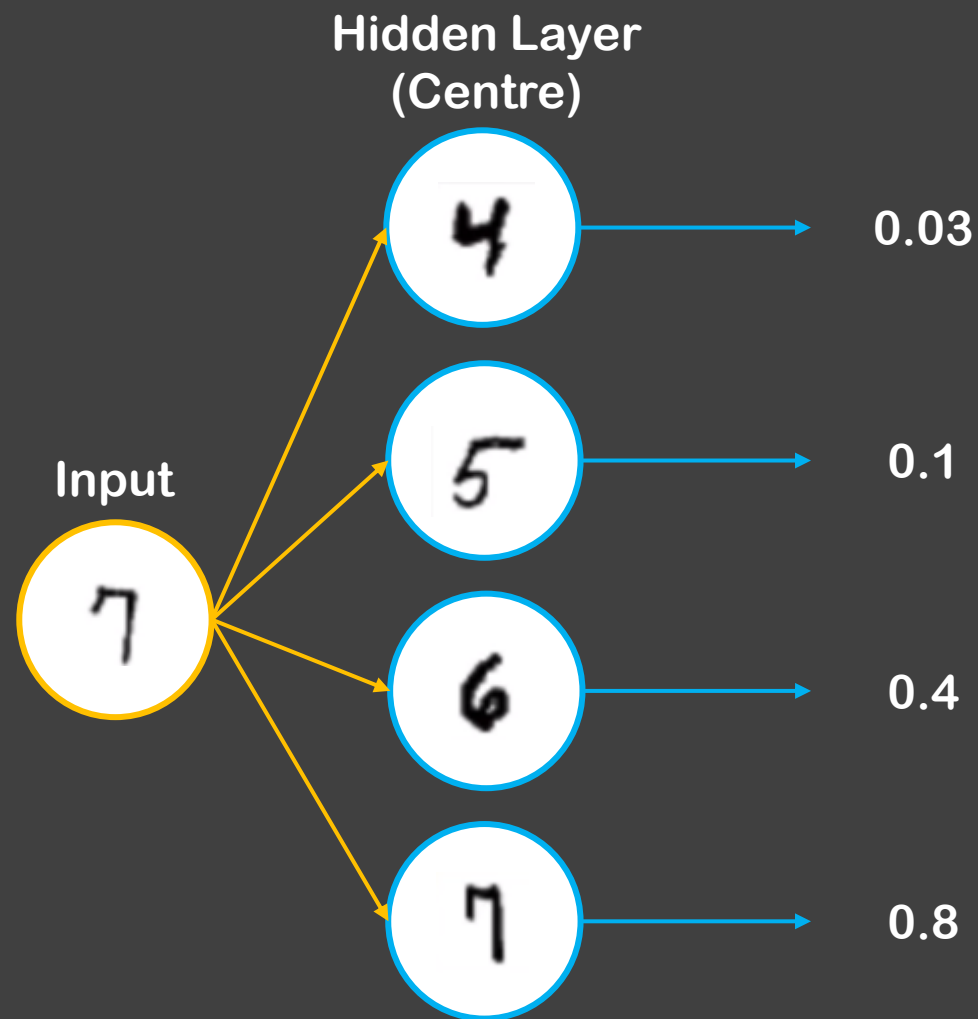
## For Centres

- Backpropagation
- Select random data
- Clustering algorithms

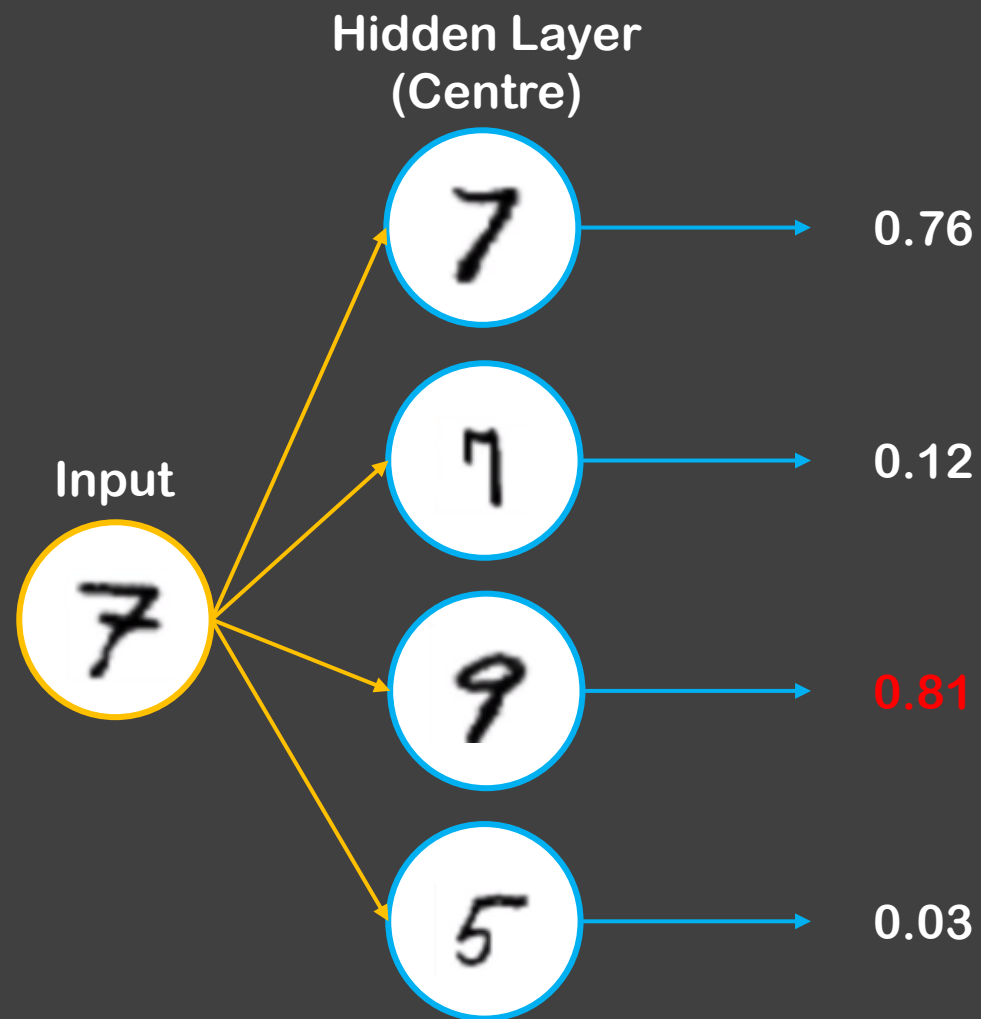
## For Output Weights

- Backpropagation
- System of equations
- Least-squares regression

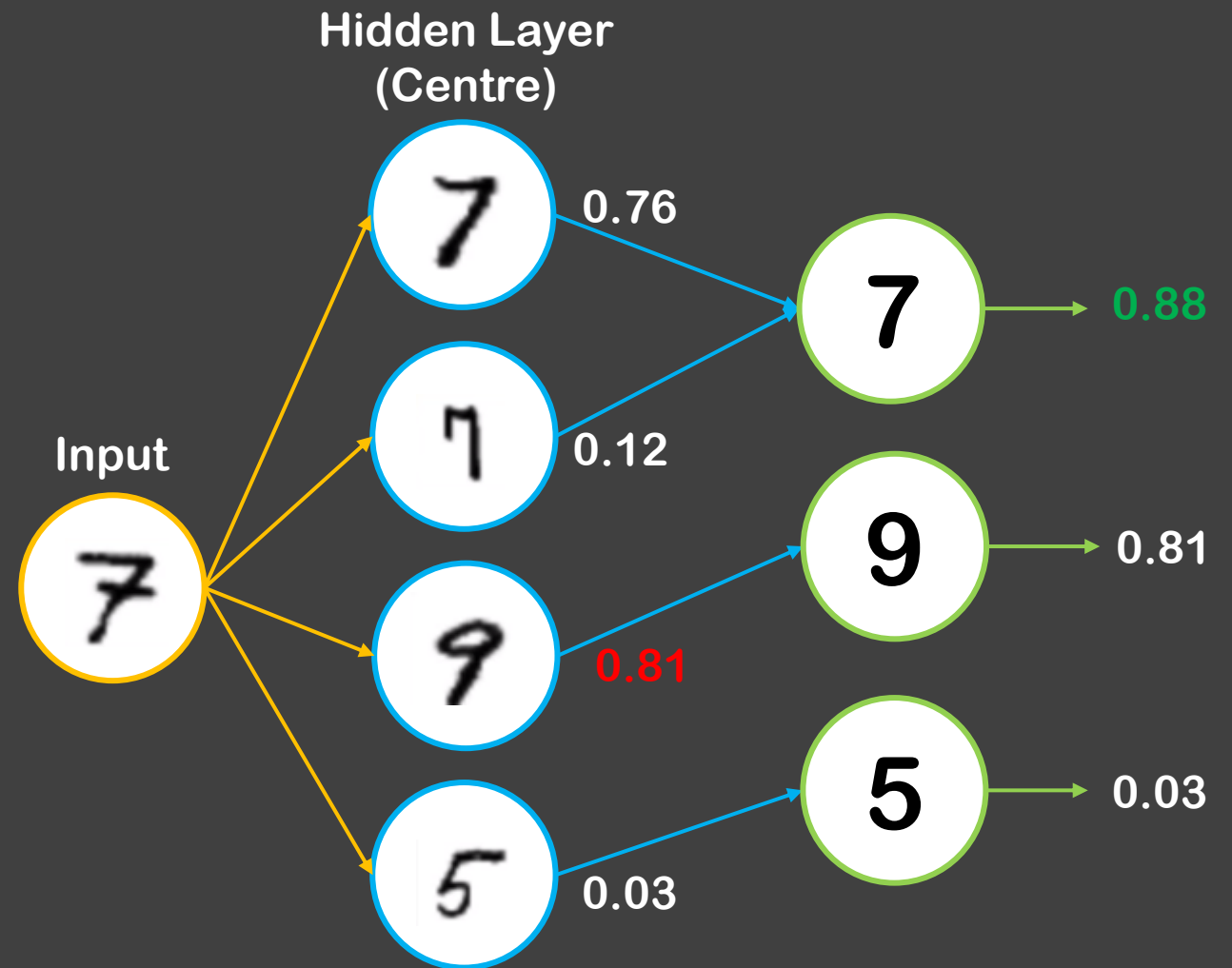
**RBF NN  
EXAMPLE:  
HAND-WRITTEN  
DIGIT  
RECOGNITION**



**RBF NN  
EXAMPLE:  
HAND-WRITTEN  
DIGIT  
RECOGNITION**



**RBF NN  
EXAMPLE:  
HAND-WRITTEN  
DIGIT  
RECOGNITION**

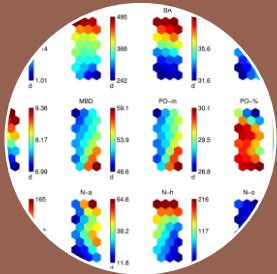




# KOHONEN SELF ORGANIZING NEURAL NETWORK



# Kohonen Self Organising Neural Network



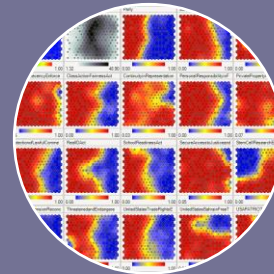
## Nickname

- Sometimes, it is also called the self-organising map.



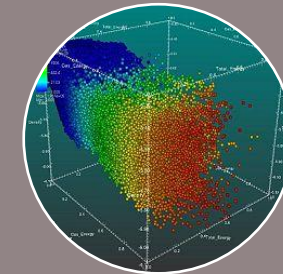
## How SOM works?

- Self-organising maps are **unsupervised** NNs that cluster high-dimensional data.
- It transforms the complex inputs from high-dimensional space into easy to understand two-dimensional outputs.



## Where to use it?

- Self organising map can be used for clustering, compression, and visualisation.

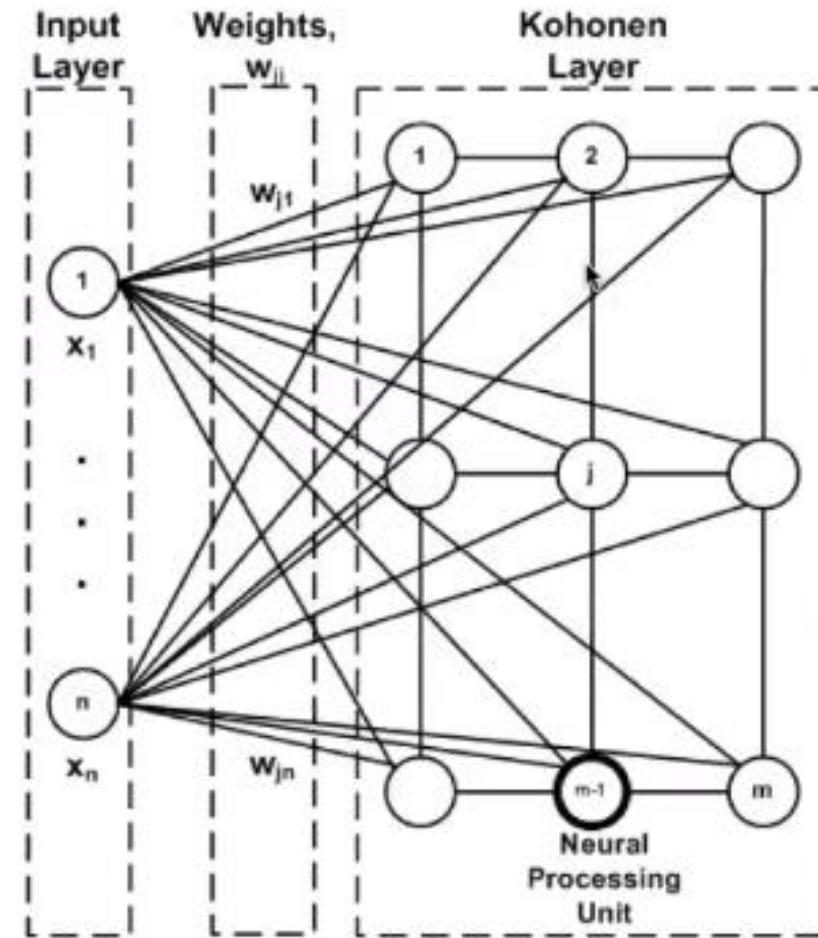


## Applications

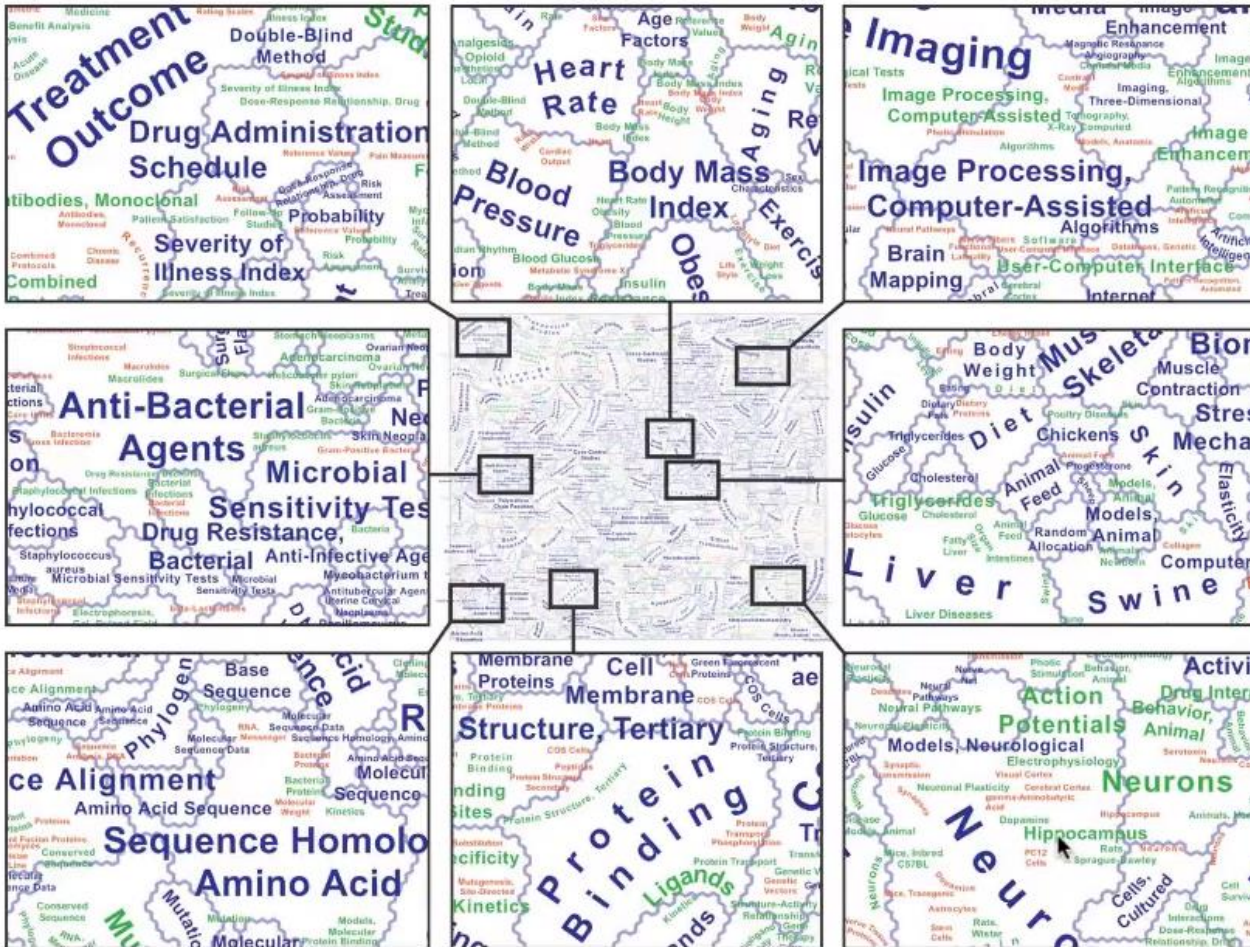
- Used to recognize patterns in data link in medical analysis.
- Any field regarding visualisation.

# SELF-ORGANISING MAPS

- The Kohonen layer is a 2-D plane for displaying the results.
- All nodes in the input layer are fully connected to nodes in every Kohonen layer.
- By adjusting the weights in the learning process, data with similar features will be sent to the similar location.



Araujo, Ernesto & R. Silva, Cassiano & J. B. S. Sampaio, Daniel. (2008). Video Target Tracking by using Competitive Neural Networks. WSEAS Transactions on Signal Processing. 4.



Skupin A, Biberstine JR, Bo"nner K (2013) Visualizing the Topical Structure of the Medical Sciences: A Self-Organizing Map Approach. PLoS ONE 8(3): e58779. doi:10.1371/journal.pone.0058779

## Examples of Using Self Organising Map

2 Million Medical Papers

Text Pre-Processing

Training SOM with 75,000 Neurons

Post-Processing

Final Result and Production

# Training the SOM

1. Initialising NN weights.

2. Randomly select an input.

3. Select the winning neuron using Euclidean distance.

4. Update Neuron weights.

5. Go back to step 2 until finishing the training.

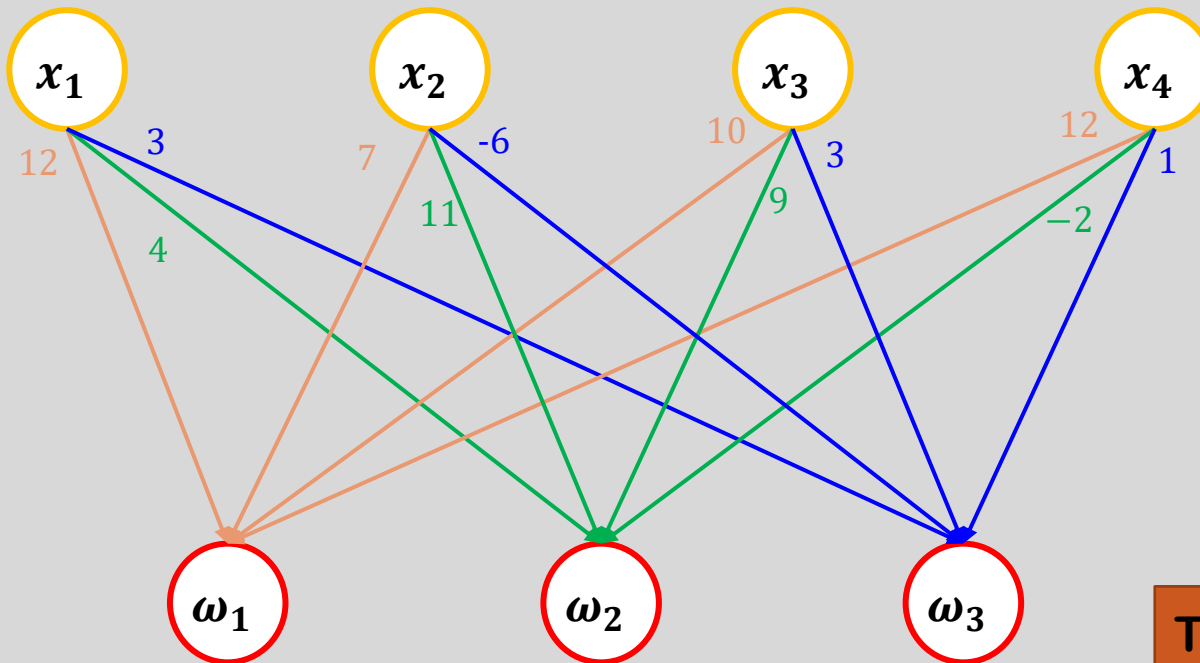


# Example of Training the SOM

$x_1$	$x_2$	$x_3$	$x_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
0.6	1.7	2.0	1.8
2.2	1.6	1.3	0.3
$\vdots$	$\vdots$	$\vdots$	$\vdots$

$x_i$

$$D(\omega_j, x_i) = \sqrt{\sum_{k=1}^n (\omega_{jk} - x_{ik})^2}$$



$$D_1(\omega_1, x_i) = \sqrt{\sum_{k=1}^n (\omega_{1k} - x_{ik})^2} = \sqrt{(12 - 0.6)^2 + (7 - 1.7)^2 + (10 - 2)^2 + (12 - 1.8)^2} \approx 18.38$$

$$D_2(\omega_2, x_i) = \sqrt{\sum_{k=1}^n (\omega_{2k} - x_{ik})^2} = \sqrt{(4 - 0.6)^2 + (11 - 1.7)^2 + (9 - 2)^2 + (-2 - 1.8)^2} \approx 12.71$$

$$D_3(\omega_3, x_i) = \sqrt{\sum_{k=1}^n (\omega_{3k} - x_{ik})^2} = \sqrt{(3 - 0.6)^2 + (-6 - 1.7)^2 + (3 - 2)^2 + (1 - 1.8)^2} \approx 8.17$$

The neuron with the minimum distance is the winning neuron. Its weight is going to be used in the weight update in the next step.



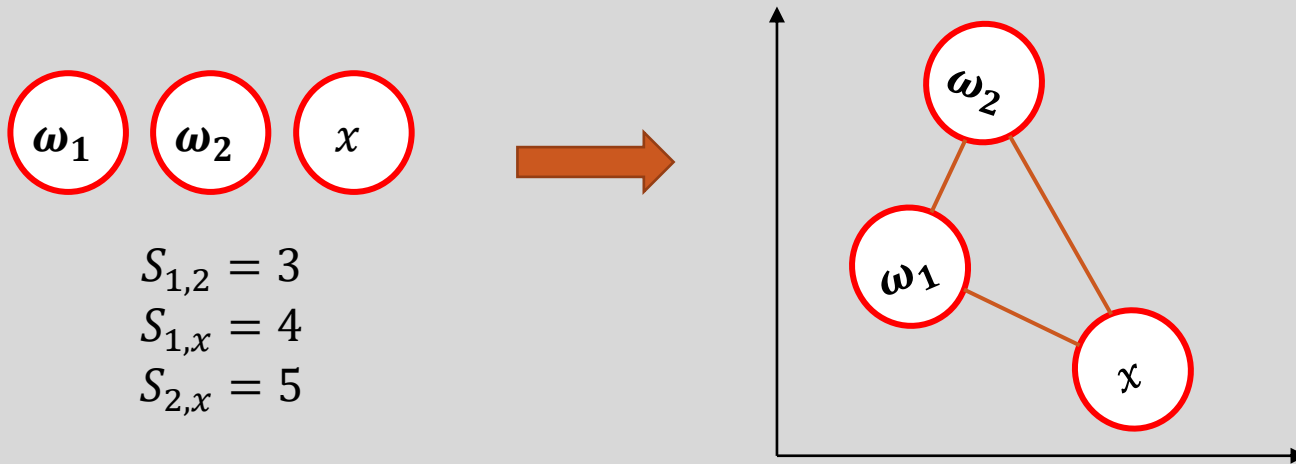
# Example of Training the SOM

- **Weight update:**  $\Delta\omega_{j,i} = \eta(t) \times T_{j,I(x)}(t) \times (x_i - \omega_{j,i})$
- **Definitions**
  - $t$ : the epoch (the number of iterations)
  - $i$  and  $j$ : the indices neurons, where  $i \neq j$
  - $I(x)$ : the winning neuron
  - $\eta(t)$ : the **learning rate**, where  $\eta(t) = \eta_0 \cdot e^{-\left(\frac{t}{\tau_\eta}\right)}$
  - $T_{j,I(x)}(t)$ : the **topological neighbourhood**, where  $T_{j,I(x)}(t) = e^{-\left(\frac{S_{j,I(x)}^2}{2\sigma(t)^2}\right)}$
  - $S_{j,i} = \|\omega_j - \omega_i\|$ : the **Lateral distance between neurons**.
  - $\sigma(t) = \sigma_0 \cdot e^{-\left(\frac{t}{\tau_0}\right)}$ : the **neighbourhood size**.
- **Hyperparameters**
  - $\eta_0$ ,  $\tau_\eta$ ,  $\sigma$ , and  $\tau_0$

$S_{j,I(x)}$  of the winning neuron would be 0 and thus the winning neuron will have the maximum value of  $T_{j,I(x)}(t) = 1$ .

# How Does the 2-D Visualisation Result Update?

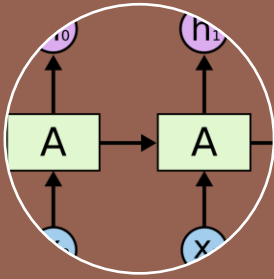
- Since the distance between neurons is calculated by  $S_{j,i} = \|\omega_j - \omega_i\|$ , we can find out the geometric location of all neurons.





# RECURRENT NEURAL NETWORK (RNN)

# Recurrent Neural Network (RNN)



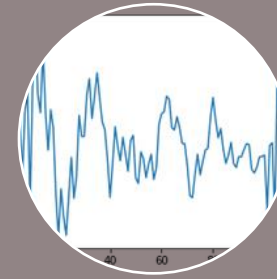
## Structure

- The hidden layers saves its output to be used for future prediction.
- It saves the previous output and uses it as part of the input in the next round.



## Strong Point

- Suitable to deal with the time-series data because of its naturality.



## Application

- Text to speech conversion models.
- Word prediction.
- Natural language processing.

# Let's Learn RNN from Examples



## Challenge

- Using the minimum living cost on dinning to survive.



## Regular Value Sales

- Hamberger
- Passta
- Dumpling

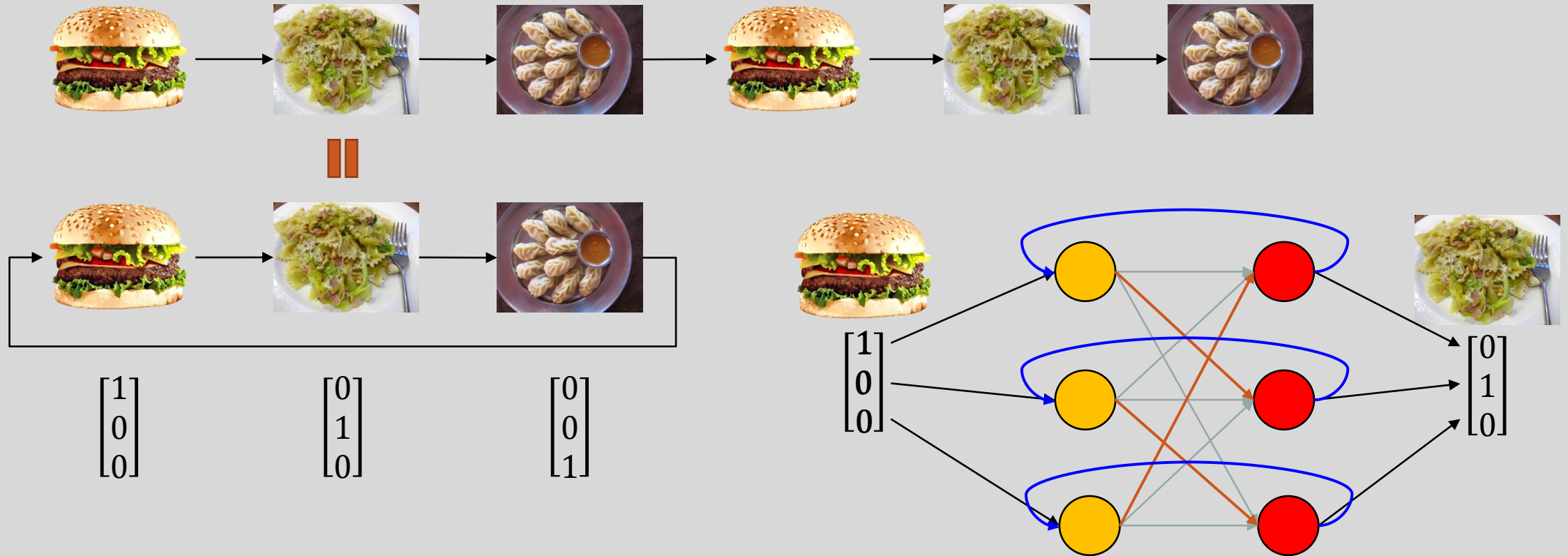


## Condition

- To prevent getting tired of specific food, the challenger repeatedly taking meals on the list in the same sequence.



# Let's Learn RNN from Examples



# Let's Learn RNN from Examples

## Advanced Challenge!

- To further reduce the living cost, the challenger decides to flip a coin everyday to decide whether to purchase new meal next day!
- If the head comes out from flipping the coin, the challenger will save half of the meal for next day and will not go for purchasing.
- Otherwise, the challenger purchases the meal following the same sequence given earlier.

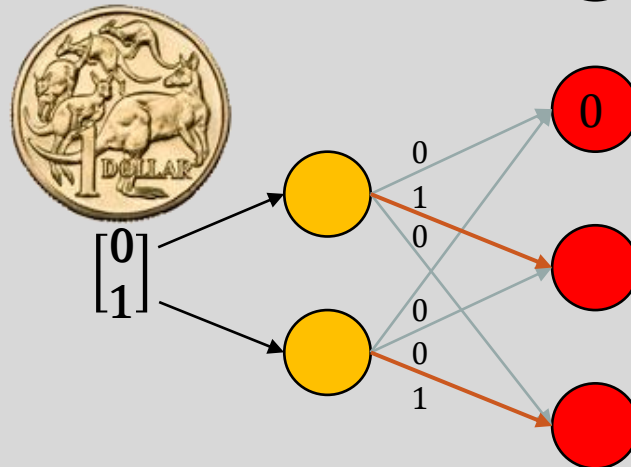
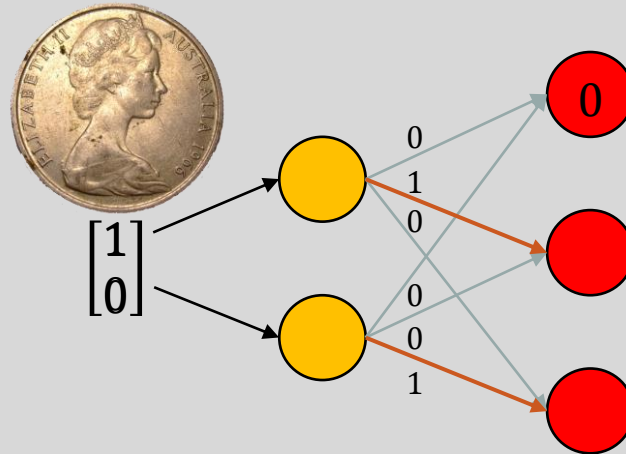


Purchase  
new meal  
tomorrow.

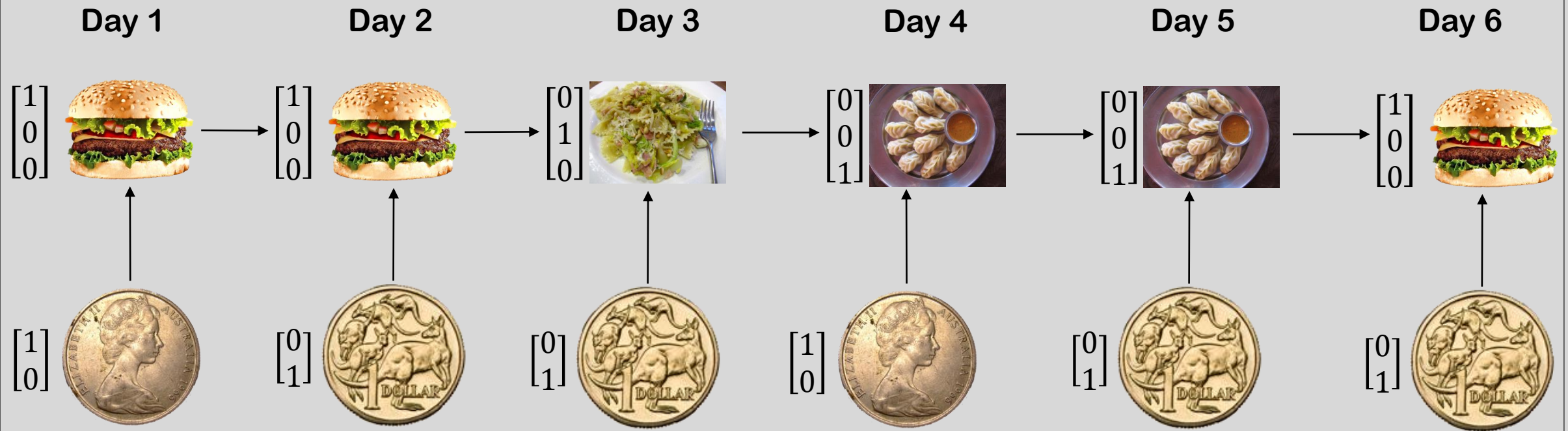


No new  
meal  
tomorrow!

# Let's Learn RNN from Examples

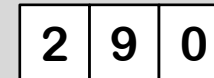
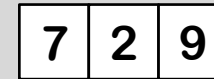
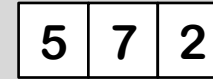
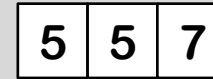
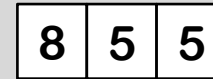
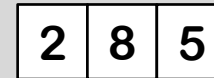
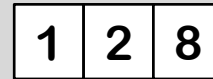
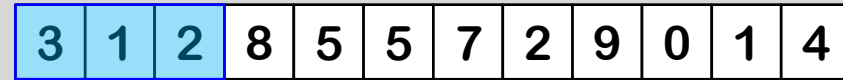


# Let's Learn RNN from Examples



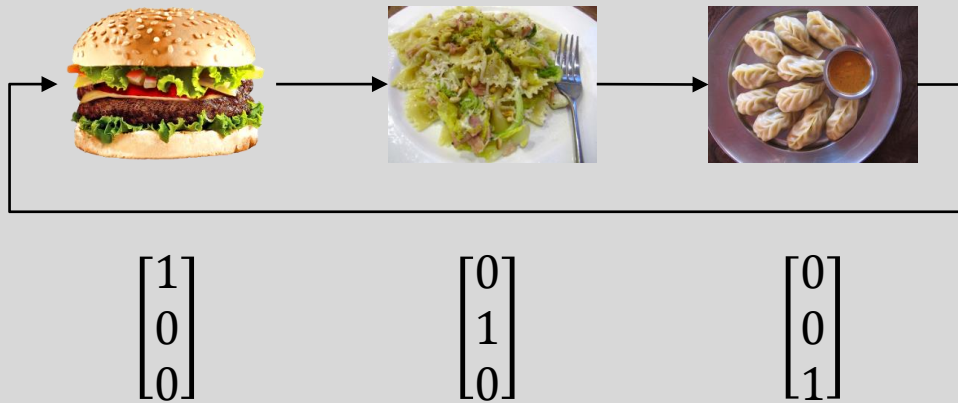
# Background Knowledge: Sliding Window

- The observation length is limited and thus the sliding window is commonly used in the time-series data processing.



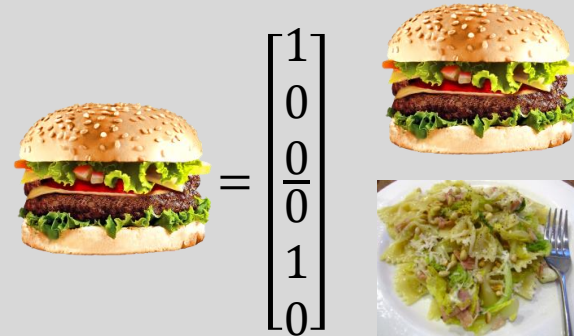


# Let's Learn RNN from Examples



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Food list



Same Dish

New Dish

- Stack up the food list into a matrix using the concept of sliding window.
- Stack up the coin flipping matrix.



$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Coin flipping





$$= \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

# Let's Learn RNN from Examples


- By adding both matrix together, we can get a clear instruction on whether to have a new dish next day.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Food list

Same Dish




New Dish

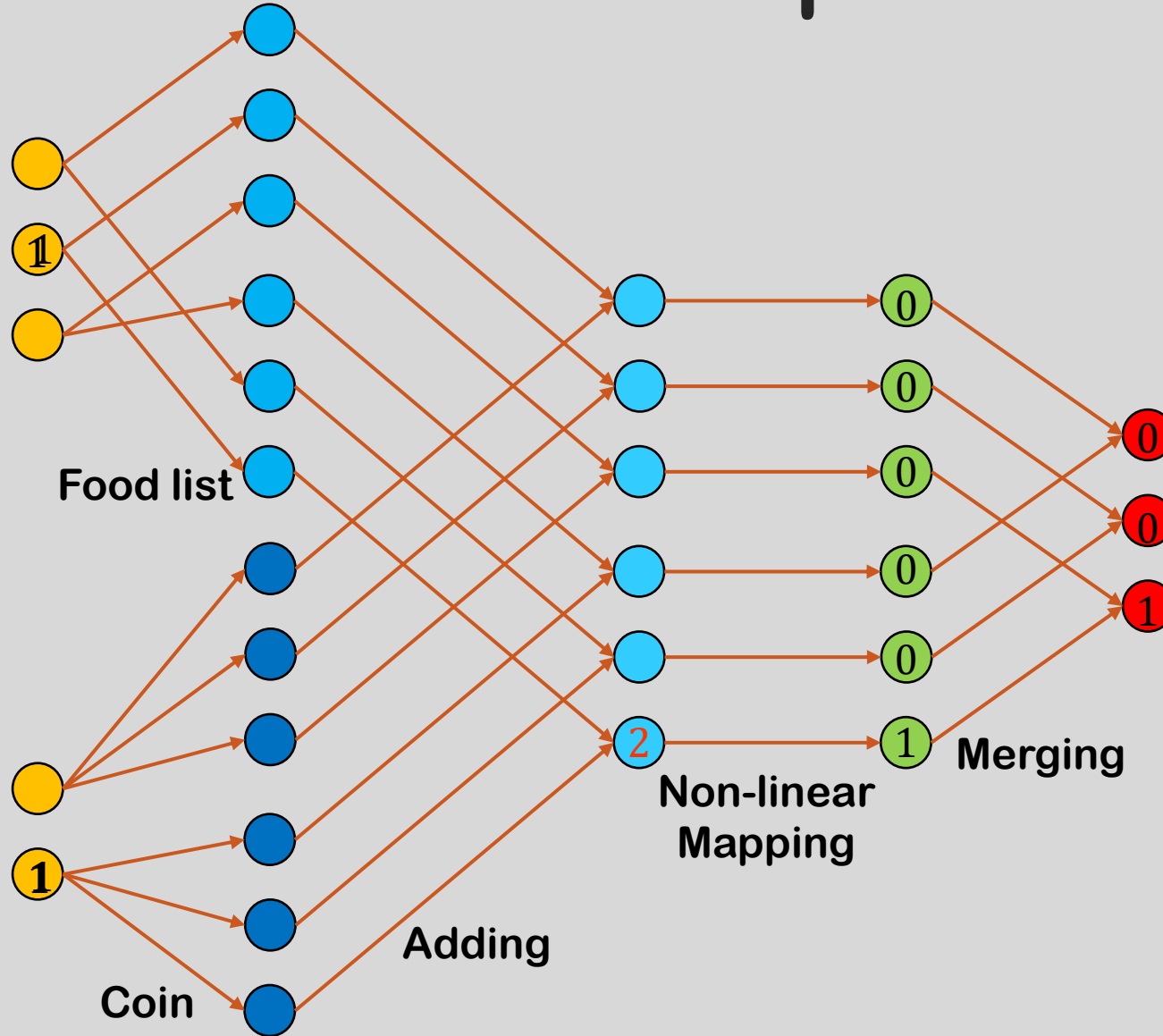
$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \hline 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Coin flipping



$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\text{Non-linear Mapping}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$


# Let's Learn RNN from Examples


$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

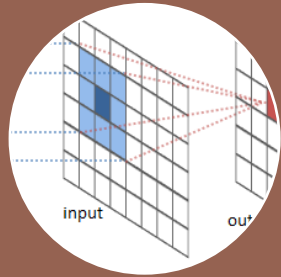
$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$





# CONVOLUTION NEURAL NETWORK (CNN)

# Convolution Neural Network (CNN)



## Structure

- The input features are taken in batches like a filter.
- It allows the network to remember a big chunk of image in parts.



## Applications

- Almost all applications related to image are somehow connected to CNN.
- Used in signal and image processing.





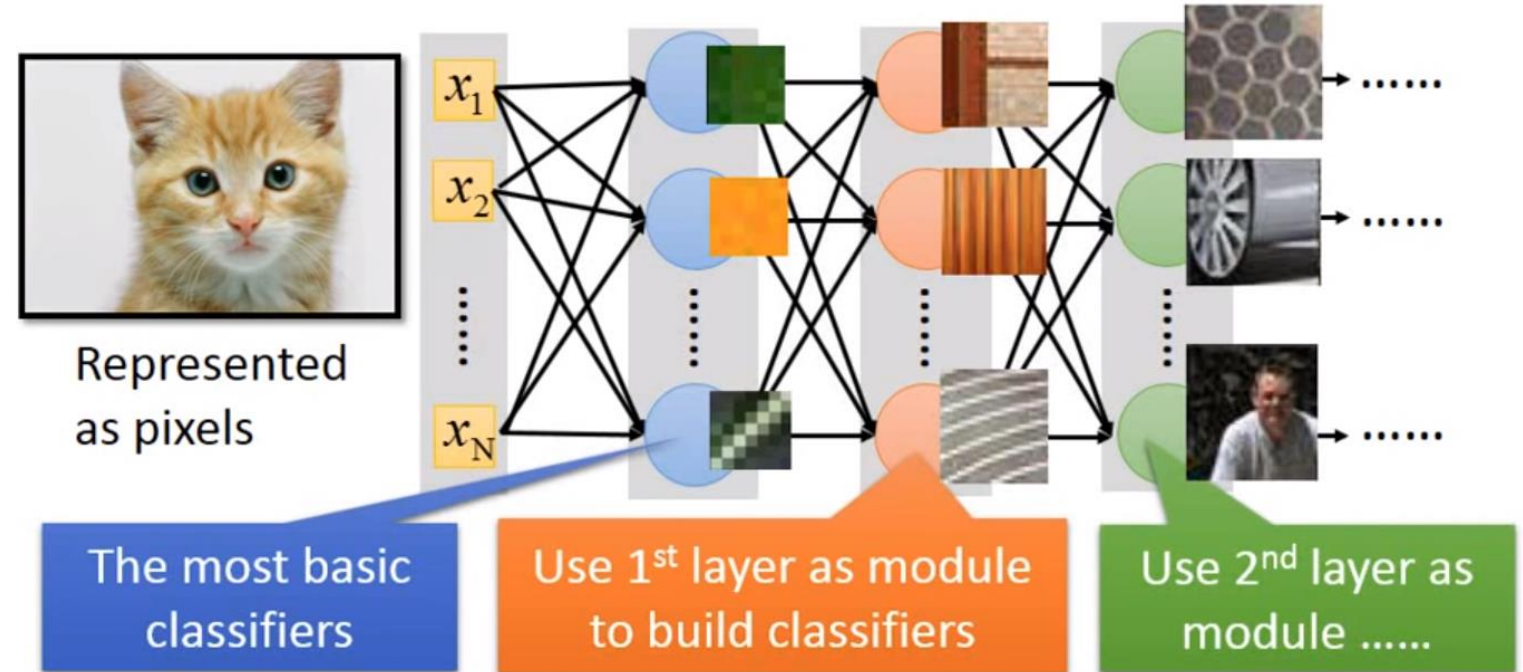
# Why CNN is Developed?

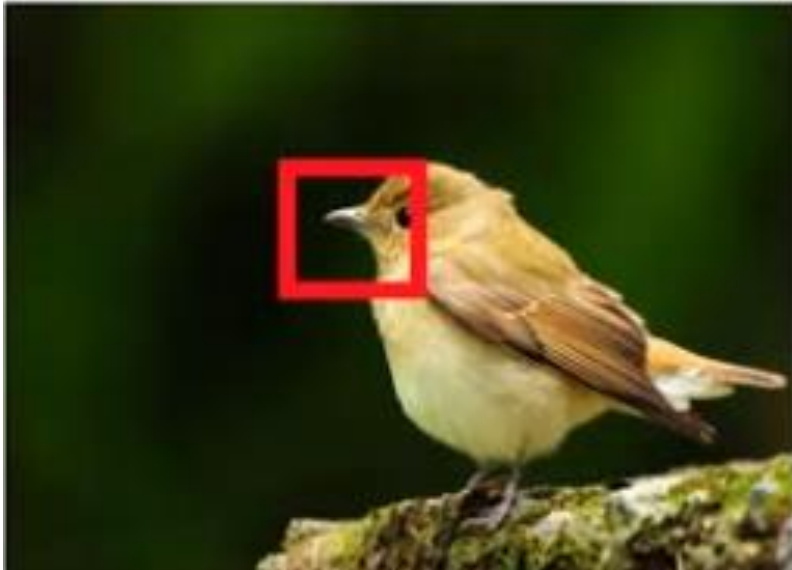
- Other NNs can also process image data. However, using a typical NN to deal with image data would cause the dramatic increase of coefficients.
  - For example, a 100x100 colour image would require 100x100x3 to represent the data. If the first layer contains 1000 neurons, we will need to deal with 30,000,000 parameters in the first layer. This number doesn't include the later layers but it is already infeasible.
- CNN uses prior knowledge to reduce the dimension of parameters before creating the network.



# CNN Structure Overview

- Each neuron in the network is expected to be a simple classifier.
- Along with the layers, a single neuron is more specialised in recognising particular information.





## CNN Characteristics

- Insensitive to rotation and shifting.
- Insensitive to scaling.



# CNN Structure



Convolution

## Property 1

- Some patterns are much smaller than the whole image. (Localisation)

## Property 2

- The same patterns appear in different regions.

Max Pooling

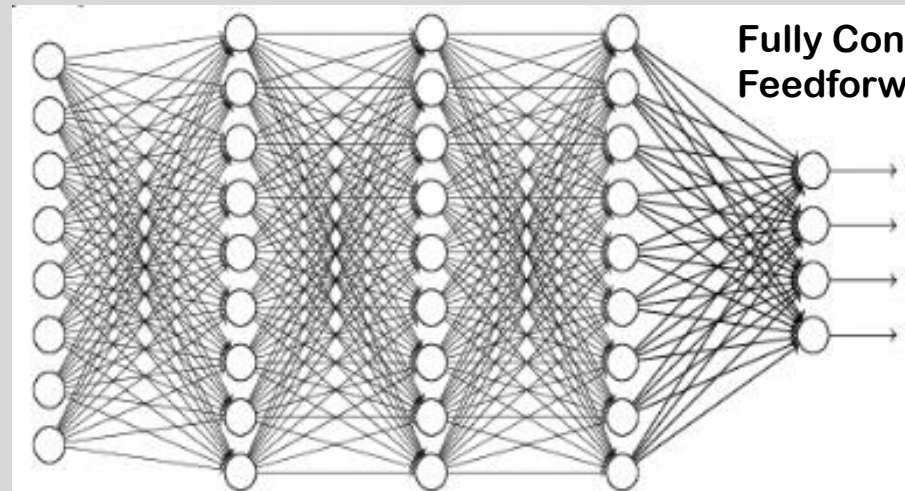
## Property 3

- Subsampling the pixels will not change the object.

Convolution

Max Pooling

Flatten

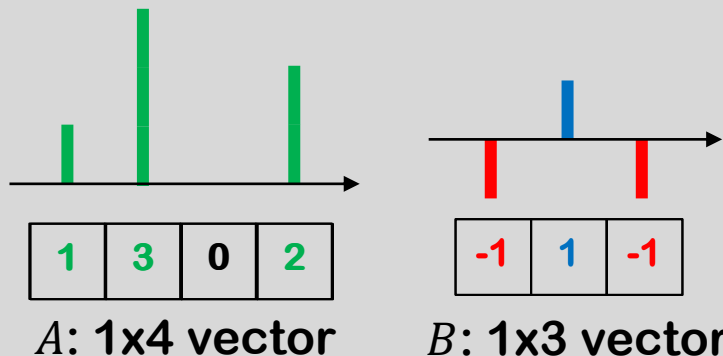


Fully Connected  
Feedforward Network

Output: Cat/Dog

# Background Knowledge

## 1-D data convolution



$A * B$ :

$$\updownarrow \updownarrow \updownarrow \updownarrow \updownarrow \updownarrow \updownarrow = 0$$

$$\begin{array}{cccc} & 1 & 3 & 0 & 2 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ -1 & 1 & -1 & & \end{array} = [1 \times (-1)] = -1$$

$$\begin{array}{cccc} & 1 & 3 & 0 & 2 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ -1 & 1 & -1 & & \end{array} = [1 \times 1] + [3 \times (-1)] = -2$$

$$\begin{array}{cccc} 1 & 3 & 0 & 2 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ -1 & 1 & -1 & \end{array} = [1 \times (-1)] + [3 \times 1] = 2$$

$$\begin{array}{cccc} 1 & 3 & 0 & 2 \\ & \updownarrow & \updownarrow & \updownarrow \\ & -1 & 1 & -1 \end{array} = [3 \times (-1)] + [2 \times (-1)] = -5$$

$$\begin{array}{cccc} 1 & 3 & 0 & 2 \\ & \updownarrow & \updownarrow & \updownarrow \\ & -1 & 1 & -1 \end{array} = [2 \times 1] = 2$$

$$\begin{array}{cccc} 1 & 3 & 0 & 2 \\ & \updownarrow & \updownarrow & \updownarrow \\ & -1 & 1 & -1 \end{array} = [2 \times (-1)] = -2$$

$$\therefore A * B: \begin{array}{cccccc} -1 & -2 & 2 & -5 & 2 & -2 \end{array}$$



# Background Knowledge

- 2-D matrix convolution

$$A * B: \begin{pmatrix} -1 & -1 & 1 & 0 & 0 & 0 \\ -2 & 0 & \dots \end{pmatrix}$$

1	0	0	0
1	0	0	1
0	1	1	0
1	0	0	1

A: 4x4 matrix

Stride = 1

-1	-1	1
-1	1	-1
1	-1	-1

B: 3x3 matrix

-1	-1	1	0	0	0
-2	0	0	-1	-1	1
0	-1	-4	-1	2	-1
0	-3	0	0	-3	0
-1	2	-1	-3	0	-1
1	-1	-1	1	-1	-1

A \* B: 6x6 matrix

# CNN - Convolution

0	1	0	0	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

6x6 image

-1	-1	1
-1	1	-1
1	-1	-1

3x3 Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

3x3 Filter 2

⋮

-1	-1	-2	-3
-1	-4	-1	3
-3	0	0	-3
3	-1	-3	-1

**Property 1**

Small Pattern

Each filter detects a  
small pattern (3x3)

# CNN - Convolution

0	1	0	0	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

6x6 image

-1	-1	1
-1	1	-1
1	-1	-1

3x3 Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

3x3 Filter 2

⋮

-1	-1	-2	-3
-1	-4	-1	3
-3	0	0	-3
3	-1	-3	-1

**Property 1**

Small Pattern

Each filter detects a  
small pattern (3x3)

**Property 2**

Different location

# CNN - Convolution

0	1	0	0	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

6x6 image

-1	-1	1
-1	1	-1
1	-1	-1

3x3 Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

3x3 Filter 2

⋮

Each filter detects a small pattern (3x3)

**Property 1**

Small Pattern

Feature Map

3	-3	-2	1
1	-2	0	-1
1	-2	-2	1
-1	-1	-1	-1

**Property 2**

Different location

- The convolution process keeps going until all filters are used.
- The collection of the convolution results is called the Feature Map.

## CNN - Convolution

0	1	0	0	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

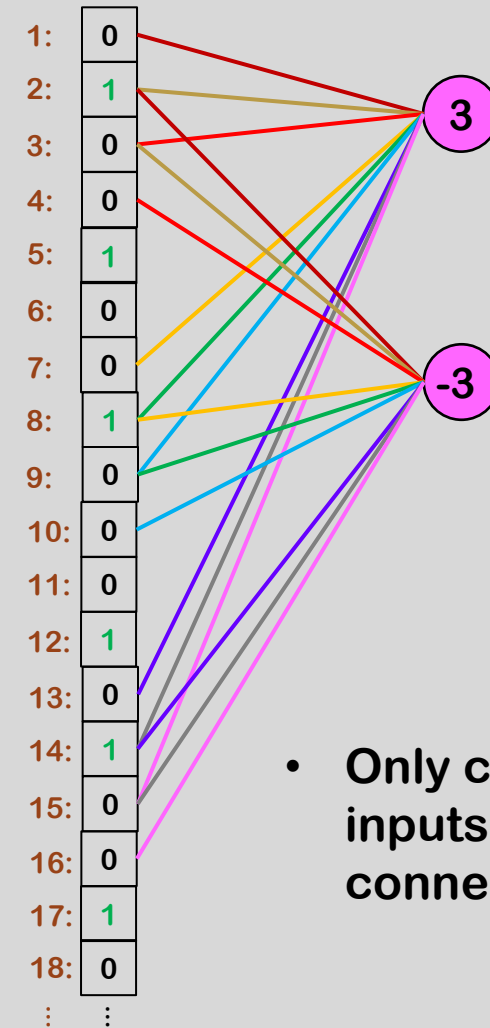
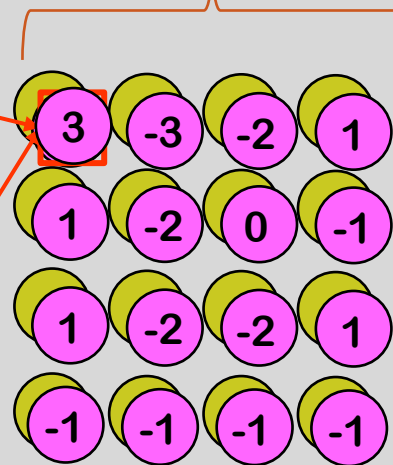
6x6 image

-1	1	-1
-1	1	-1
-1	1	-1

3x3 Filter 2

- In fact, the feature map can be obtained by the not fully connected NN structure.

Feature Map



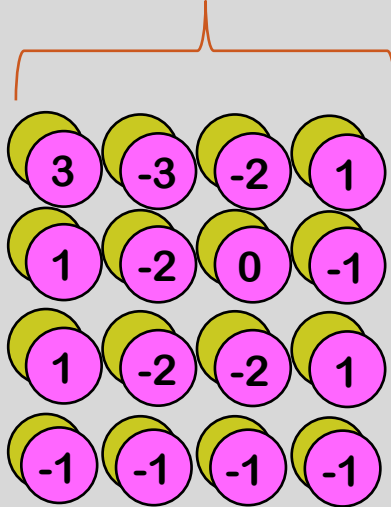
- Only connect to 9 inputs but not fully connected.



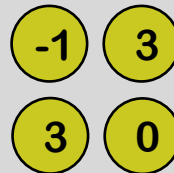
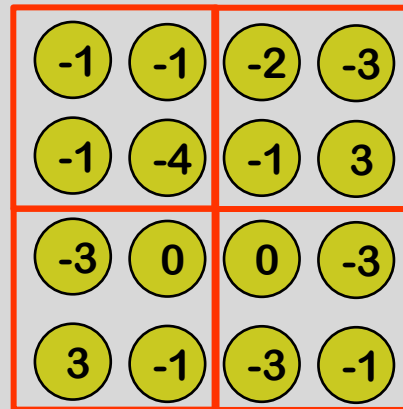
## Max Pooling

# CNN – Max Pooling

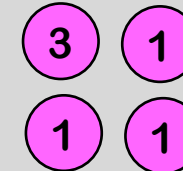
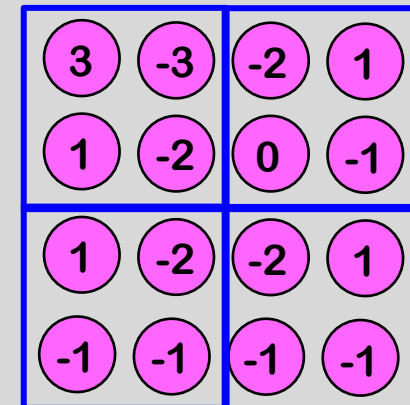
Feature Map



From Filter 1



From Filter 2



- Keep either the maximum value or the mean value to achieve down-sampling.

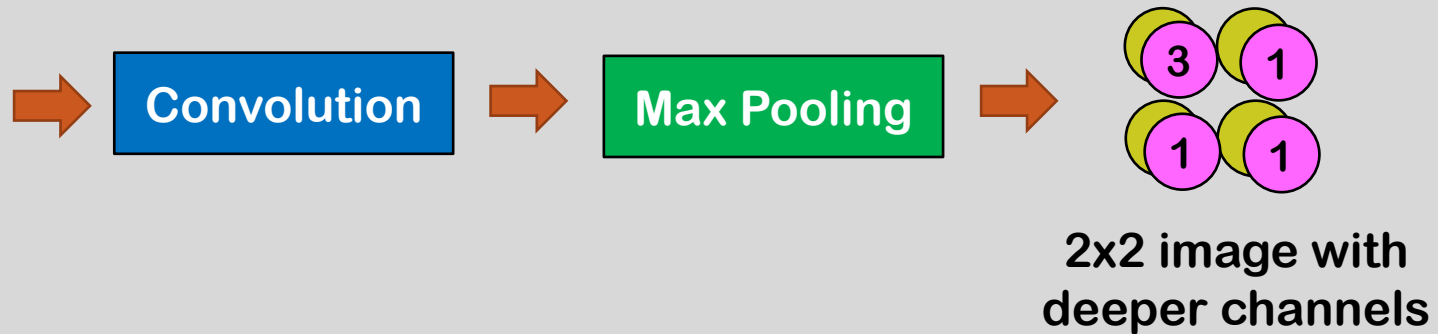
## Max Pooling

# CNN – Max Pooling

- Every time when going through the Convolution plus Max Pooling, the dimension of the image is reduced and multiple feature maps are generated.

0	1	0	0	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

6x6 image

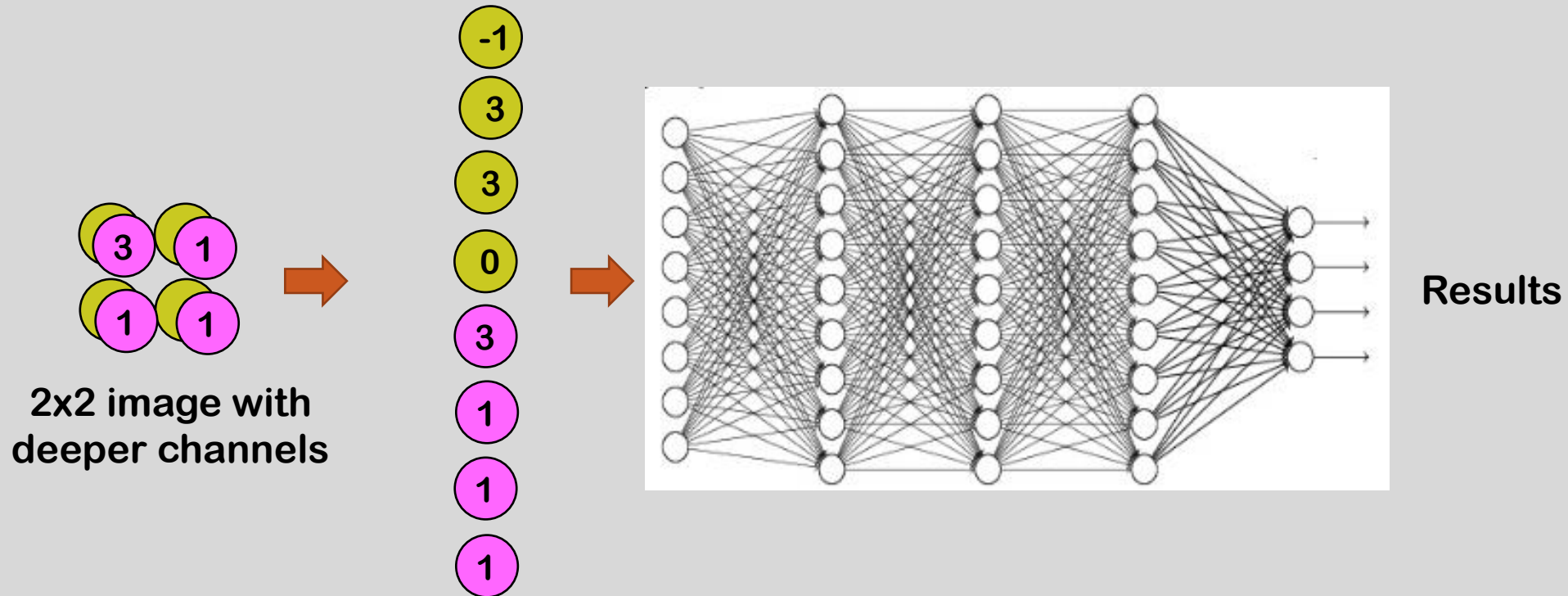


- The feature maps is considered as composed of a voxel (3-D cubes).

## Flatten

# CNN – Flatten and Fully Connected Network

- After straightening the feature map into a 1-D vector, it is pushed into the fully connected network for output.





## Advanced Predictive Model

- Perceptron
- Feed Forward Neural Network (FFNN)
- Radial Basis Function NN (RBF)
- Koonen Self Organizing Map
- Recurrent Neural Network (RNN)
- Convolution Neural Network (CNN)

# Summary

