# Artificial Intelligence

**Fiona Yan Liu**

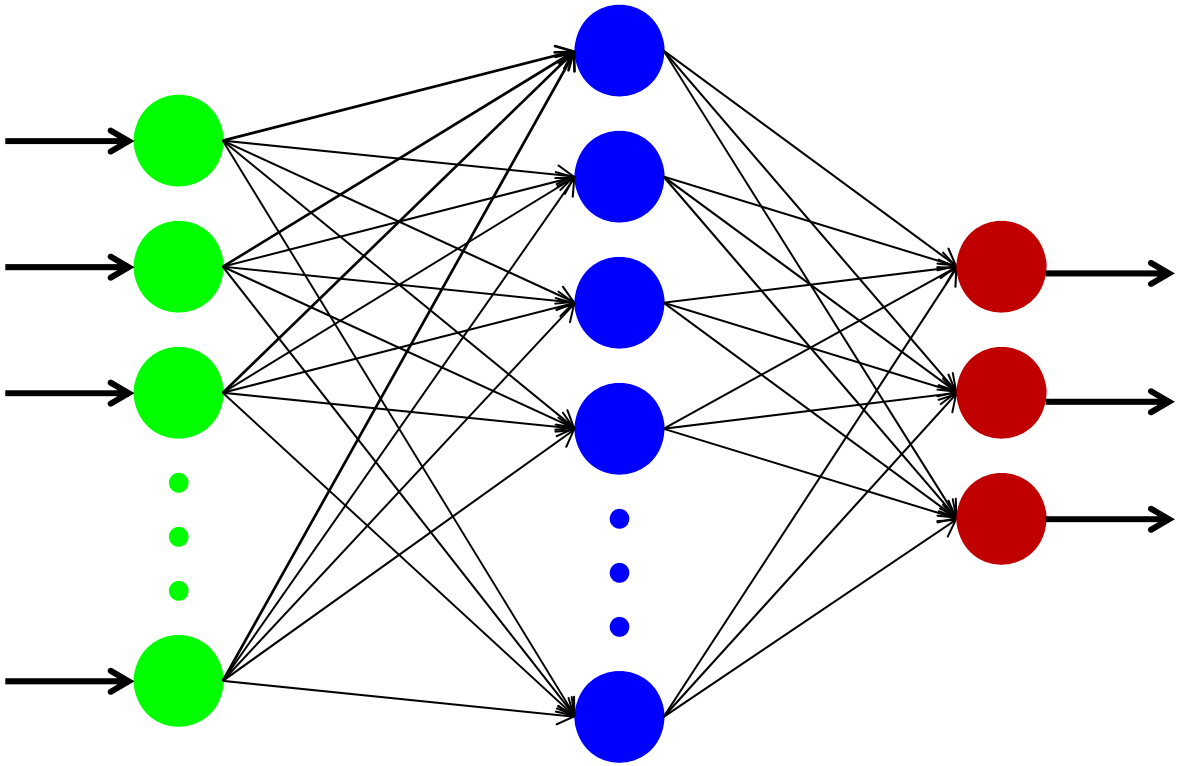
*Department of Computing*

*The Hong Kong Polytechnic University*

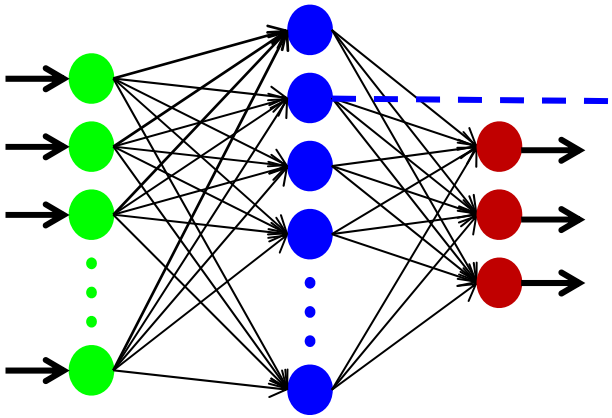
# Final Exam

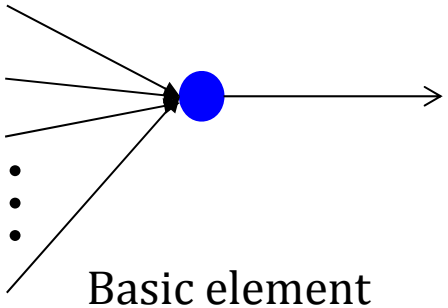
- Examination
  - Sat. Dec. 12 12:30 – 14:30
  - P306
  - Close book
  - Calculator is permitted
- Office hour
  - Sat. Dec. 12 9:00 – 11:00
  - Fri. Dec. 11 15:00 – 17:00
  - Tue. Dec. 8 10:00 – 12:00
- Form of questions
  - True or False 20%
  - Answer the questions and calculation 80%
- Cover all lectures
  - No matlab
  - No EEG

# A Typical Artificial Neural Networks



# Basic Element



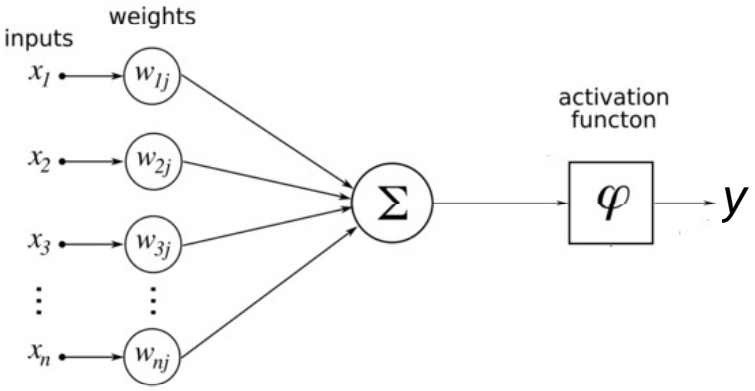


Basic element
artificial neuron

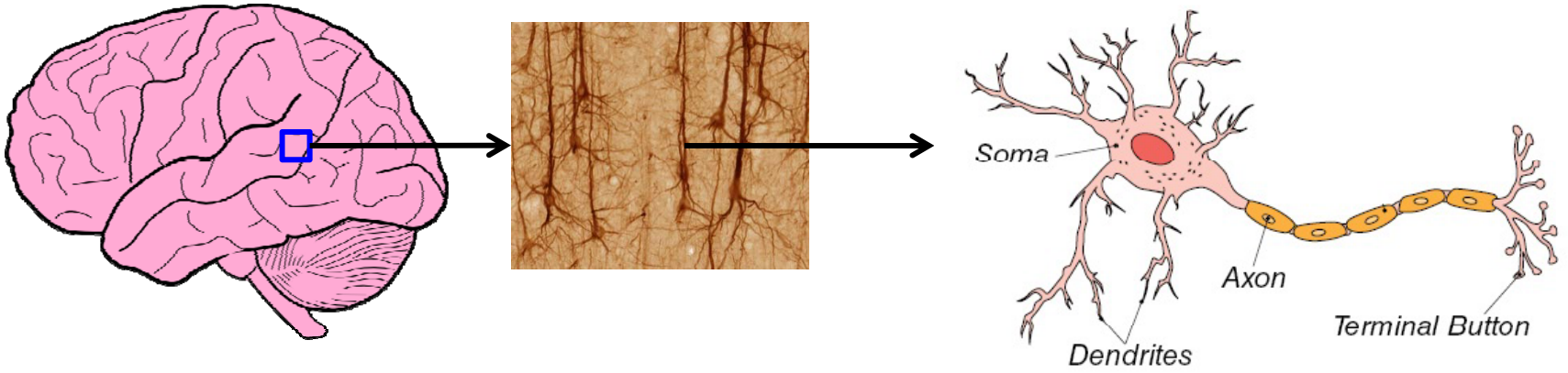
An artificial neuron transforms weighted input nonlinearly:

$$y = \varphi(\mathbf{w}^T \mathbf{x})$$

$\varphi$ is called activation function



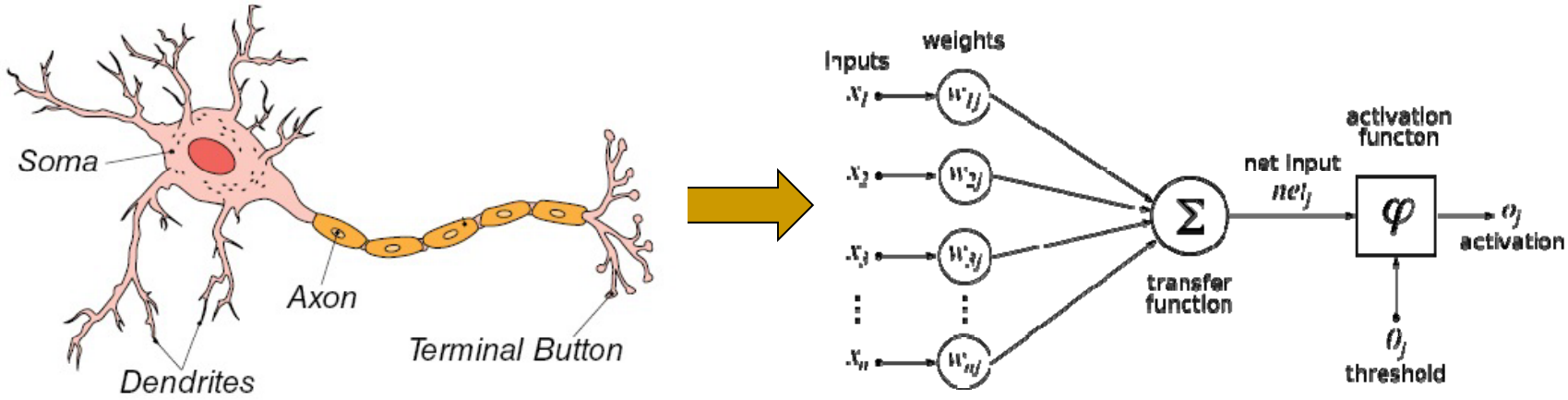
# Neuron




Brain Cerebral cortex Neuron

- Neuron is the structural constituent of the brain
- Each neuron is a cell that uses biochemical reactions to receive, process and transmit information
- The brain is a collection of about 10 billion interconnected neurons, with approximately 60 trillion connections

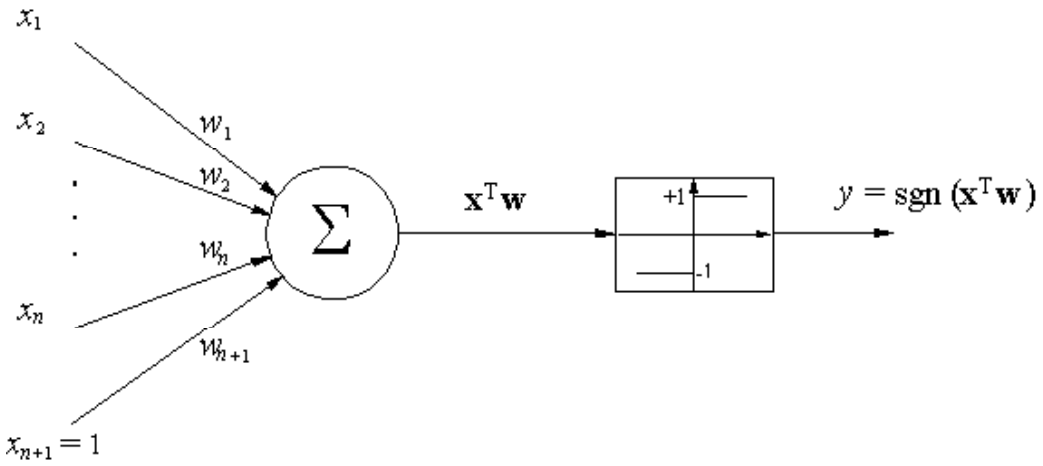
# Artificial Neuron




Biological neuron Artificial neuron

- Neurons are specialized for information transmitting through generating spiking sequences
- Dendrites receive inputs from other neurons through synaptic connection
- At the synapses between the dendrite and axons, electrical signals are modulated in various amounts
- Axon carries neuronal outputs to other neuron cells

# Perceptron

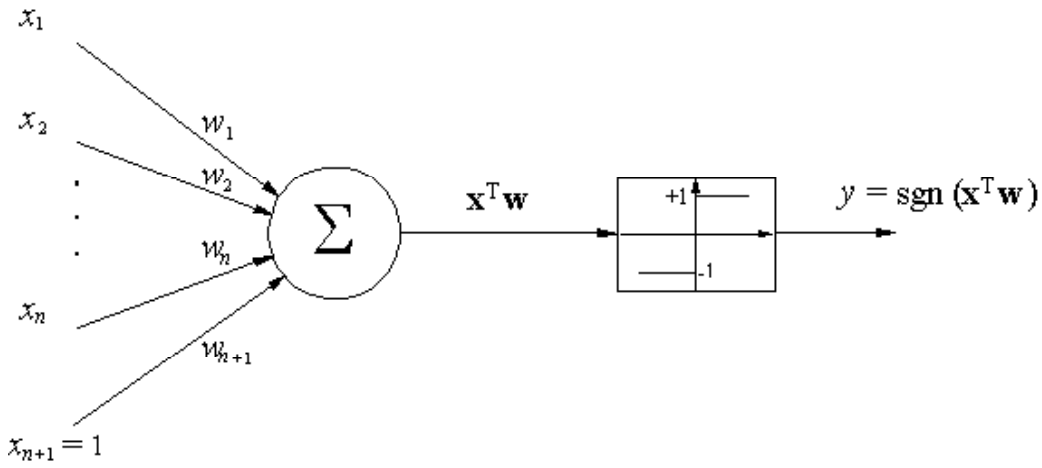



- Two-layer ANN, with threshold activation function
- Used for classification problem

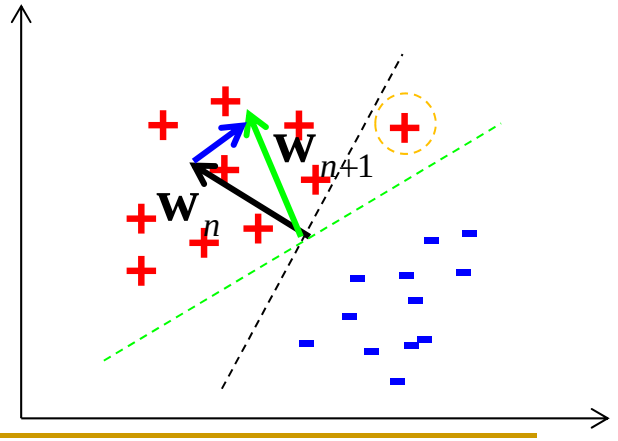
Assume $\{\mathbf{x}_i\}_{i=1}^N$ are $N$ data vectors, $\{y_i\}_{i=1}^N$ are their labels, $y_i \in \{-1, +1\}$, perceptron is to find a weight $\mathbf{w}$ satisfying:

$$y_i = \text{sgn}(\mathbf{x}_i^T \mathbf{w}) \qquad (1)$$

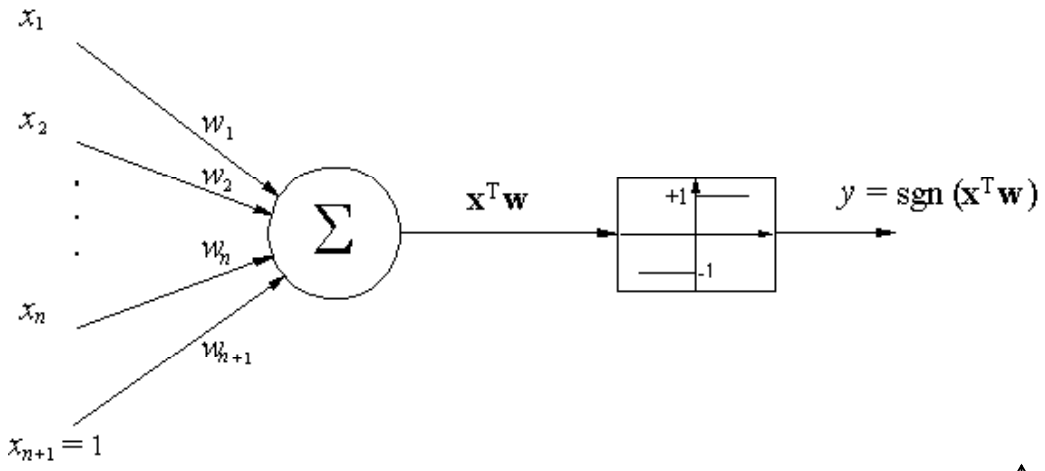
# Perceptron Learning Algorithm



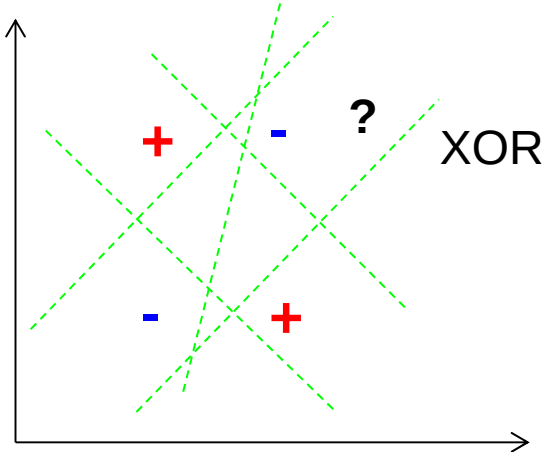

1. Initialize a random $\mathbf{w}_0$;
2. For each $(\mathbf{x}_i, y_i)$, $\eta$ is the learning rate:
   2a: calculate the estimated label $\tilde{y}_i = \text{sgn}(\mathbf{w}_n^T \mathbf{x}_i)$
   2b: update the weight $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta(y_i - \tilde{y}_i)\mathbf{x}_i$
3. The algorithm stops until $\mathbf{w}$ converges.



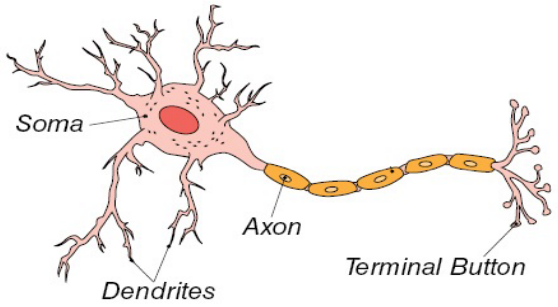
# Perceptron: Limitation

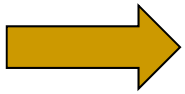


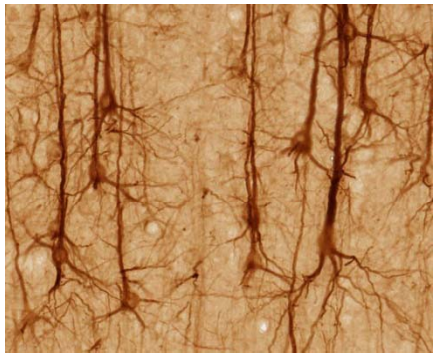

- The perceptron algorithm is guaranteed to converge if the data is linearly separable
- The algorithm does not converge for linearly inseparable cases

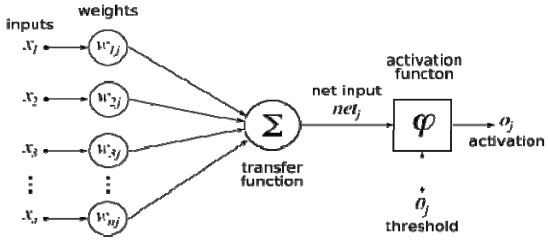


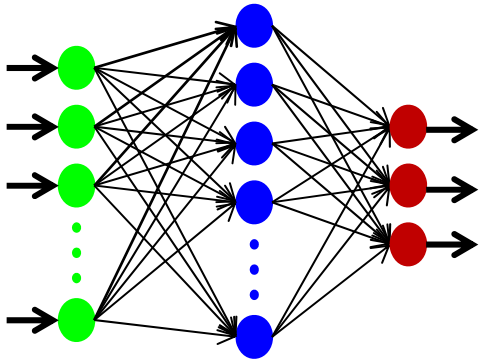
# Artificial Neural Networks



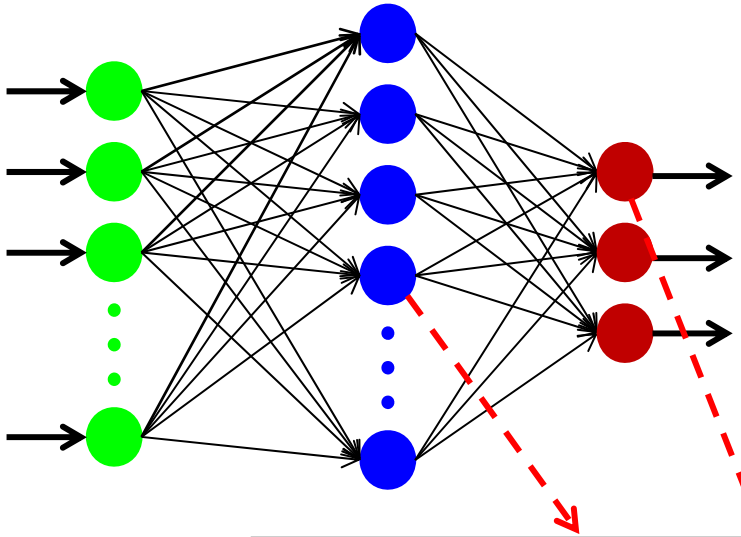




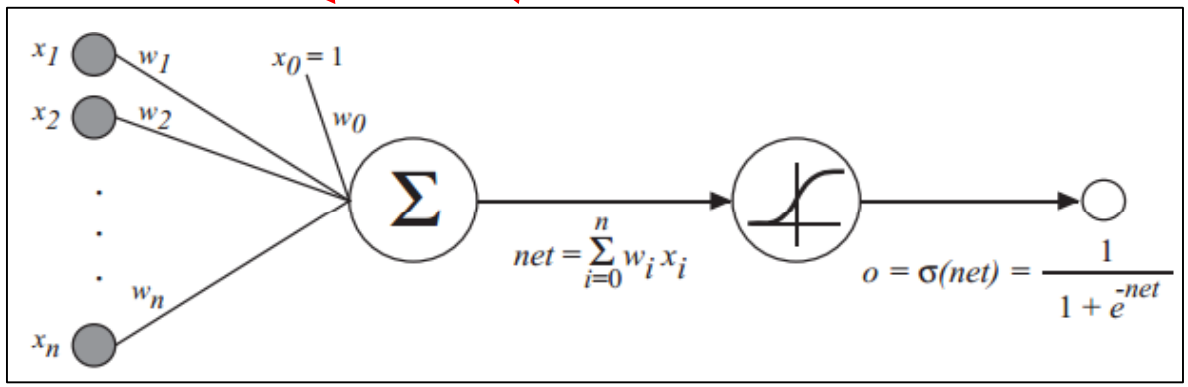



- ANNs are biologically inspired computer programs designed to simulate the way in which the human brain processes information
- A massively parallel distributed processor made up of simple processing

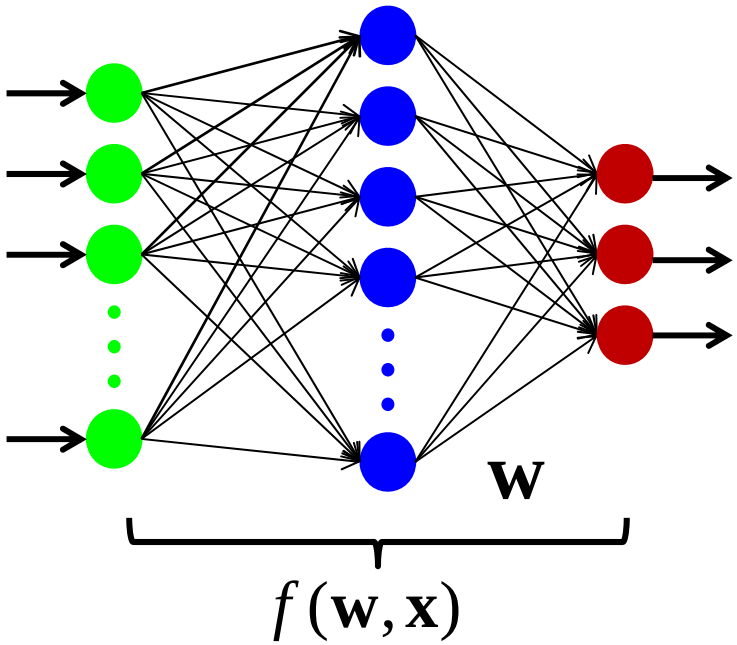
# Multilayer Perceptron (MLP)



- One or several hidden layers
- Continuous activation function, sigmoid
- Learning with a teacher signal
- Powerful ability for classification and function approximation



# MLP Learning: Back Propagation

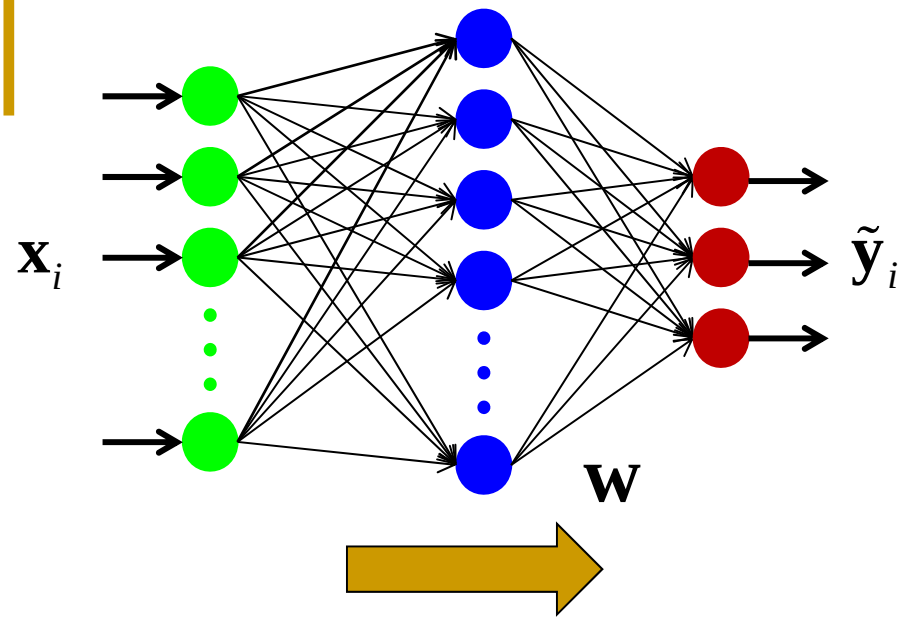


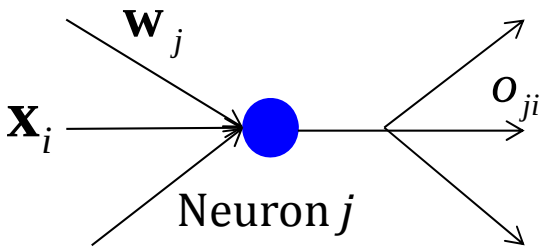

- Back propagation searches weights **w** to minimize the total error of the network over the set of training examples
- Gradient descent based method
- Backpropagation consists of two repeated process: forward process and error propagation process

$\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ is the training set, back propagation tries to minimize

$$\mathbf{w}^* = \min_{\mathbf{w}} \frac{1}{2N} \sum_{i=1}^{N} \| \mathbf{y}_i - f(\mathbf{w}, \mathbf{x}_i) \|^2$$

# MLP learning: Forward Process



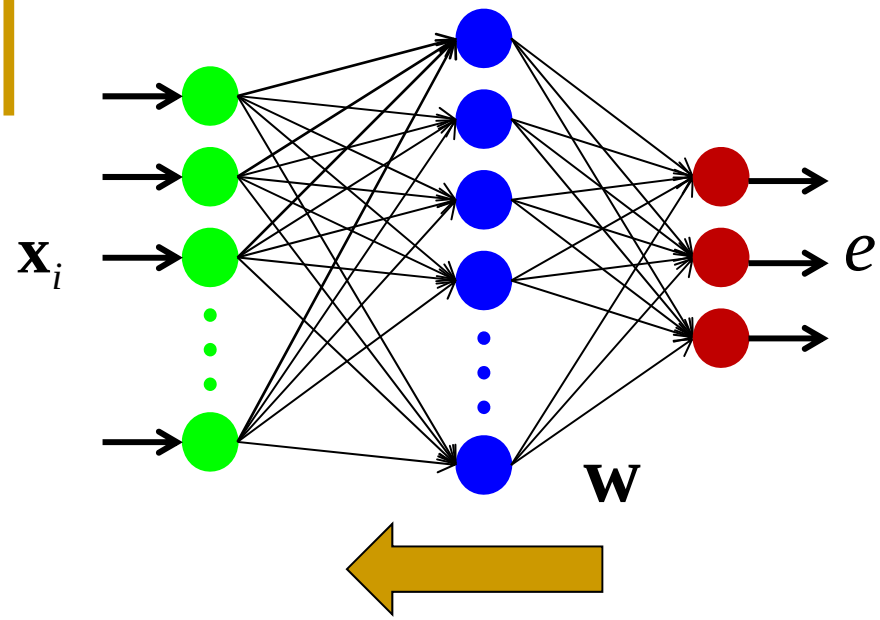

Fix the weights $\mathbf{w}$, compute the output for any given sample $\mathbf{x}_i$:

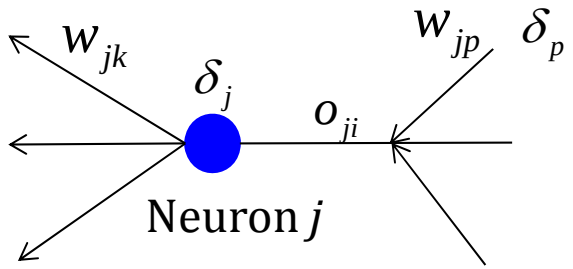
$$\tilde{\mathbf{y}}_i = f(\mathbf{w}, \mathbf{x}_i)$$

For neuron $j$, its associated weight is $\mathbf{w}_j$, the input signal to $j$ is $\mathbf{x}_i$, then the output is calculated as:

$$o_{ji} = \sigma(\mathbf{w}_j^T \mathbf{x}_i) = \frac{1}{1 + \exp(-\mathbf{w}_j^T \mathbf{x}_i)}$$

# MLP learning: Backward Process




In this step, the network error is used for updating the weights.

$$e = \frac{1}{2N}\sum_{i=1}^{N} \| \mathbf{y}_i - \tilde{\mathbf{y}}_i \|^2$$

The error is propagated backward from the output layer through the network layer by layer. Weights **w** are updated according to the propagated error signal:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta \Delta \mathbf{w}_j$$

where $\eta$ is the learning rate, $\Delta \mathbf{w}_j = \{\Delta w_{jk}\}$

$$\Delta w_{jk} = o_{ji}(1-o_{ji})\delta_j x_{ik}$$

where $x_{ik}$ is the $k$-th element of $\mathbf{x}_i$, $\delta_j$ is:

$$\delta_j = \sum_p w_{jp}\delta_p$$

For final layer:

$$\delta_p = (y_{ip} - \tilde{y}_{ip}) y_{ip}(1-y_{ip})$$