TUTORIAL 1

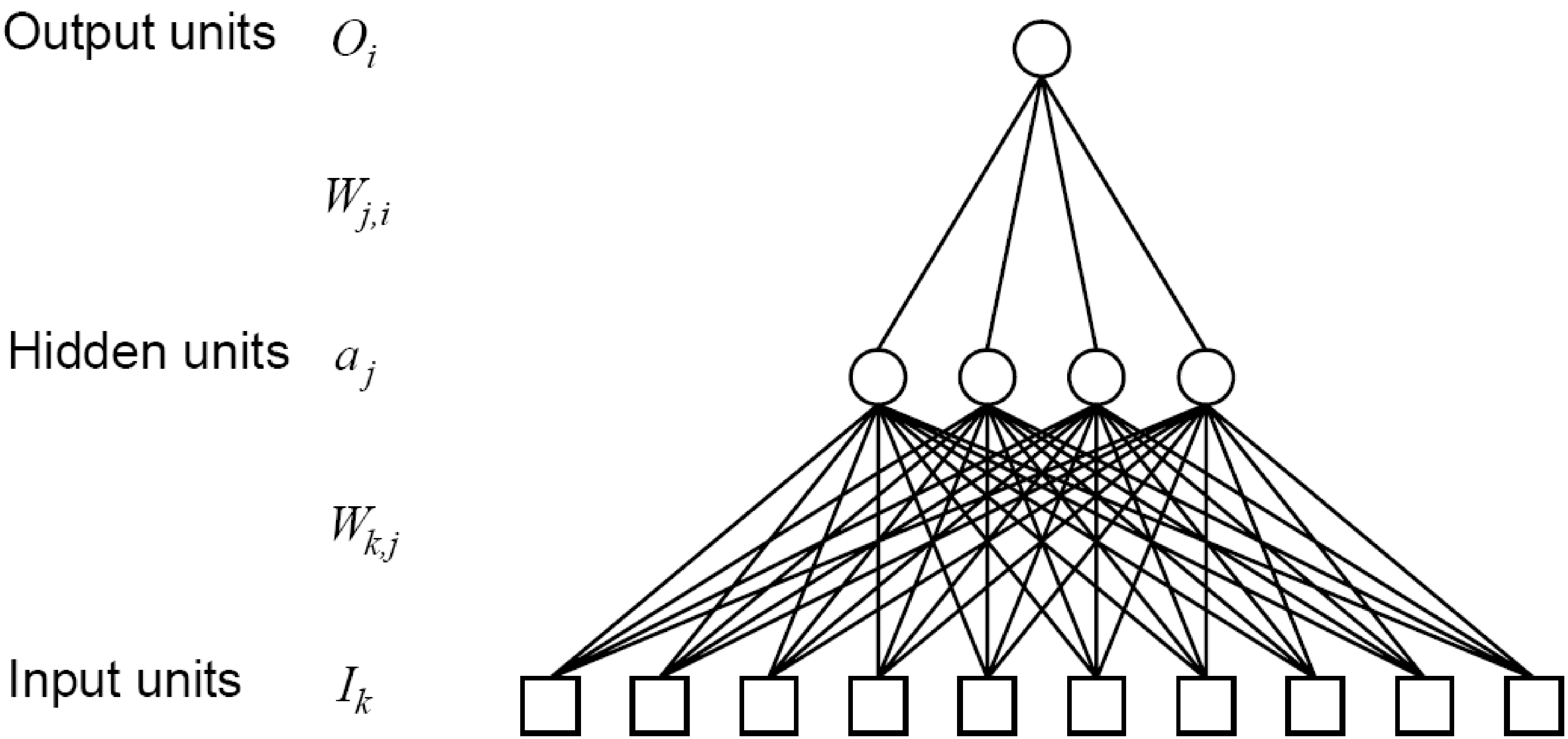
CSCI3230 (2019-2020 First Term)

By Xubin Zheng(xbzheng@cse.cuhk.edu.hk)

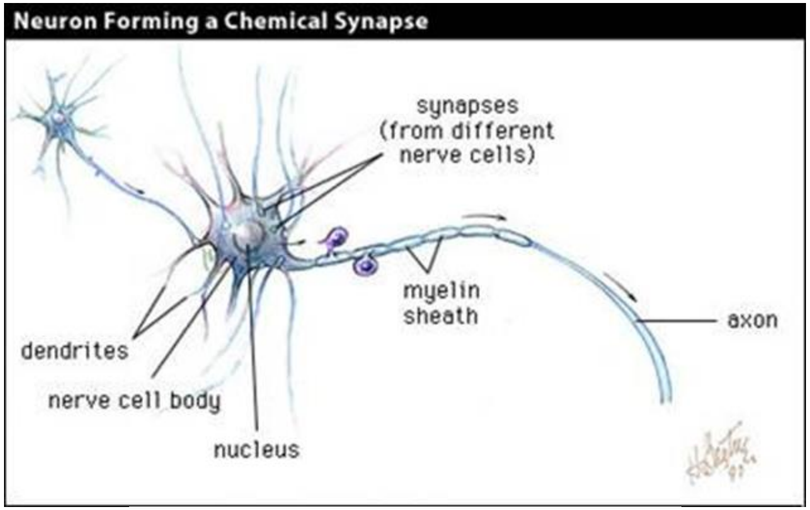
Outline

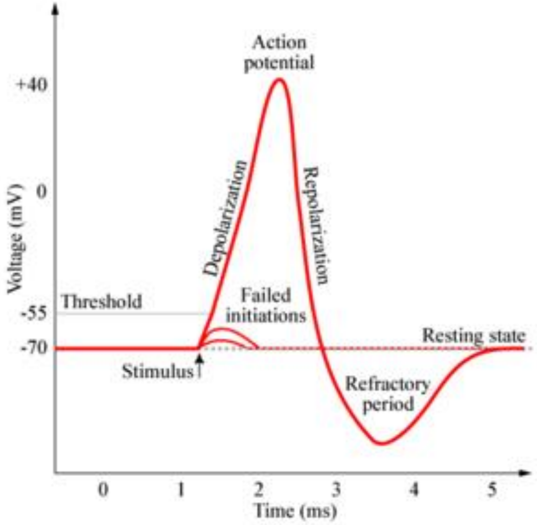
- Artificial Neural Network
  - Biological Neuron vs. Artificial Neuron
  - Gradient Descent
  - Vanishing Gradient Problem

Artificial Neural Network



Biological Neuron



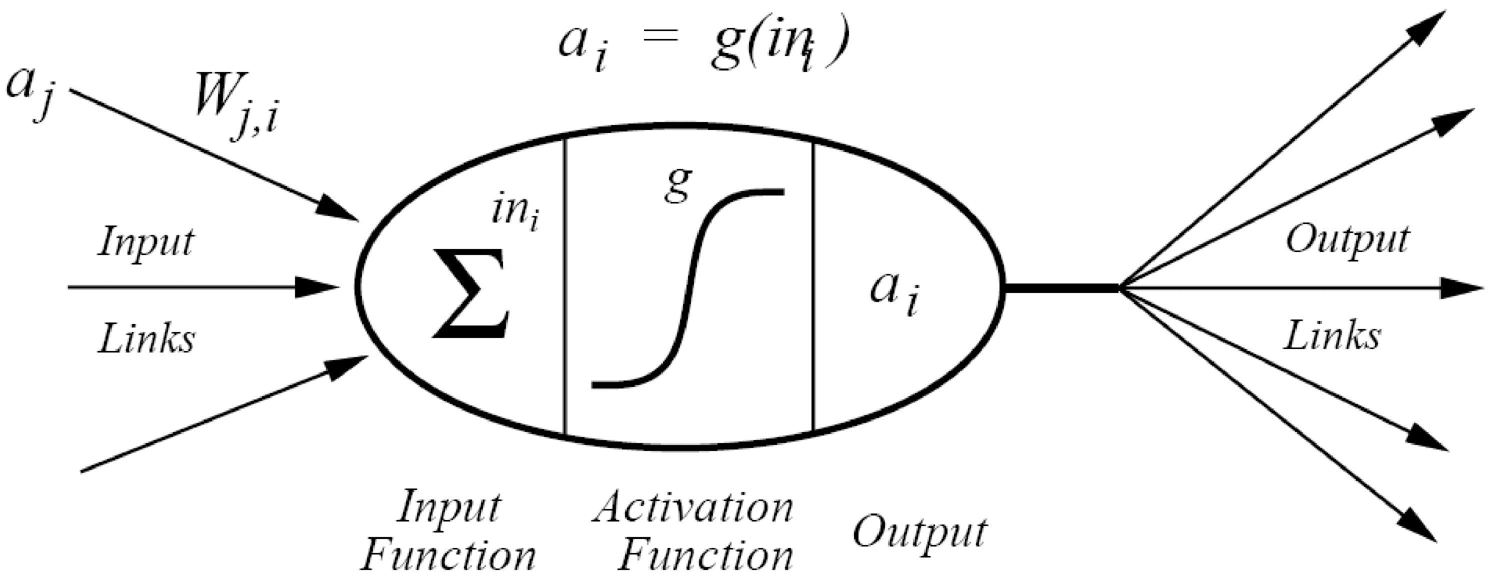


- A neuron is an electrically excitable cell that processes and transmits information through electrical and chemical signals.

- A chemical signal is transmitted via a synapse, a specialized connection with other cells.

Artificial Neuron

- An artificial neuron is a logic computing unit, conceived as a model of biological neurons.



Idea Behind ANN

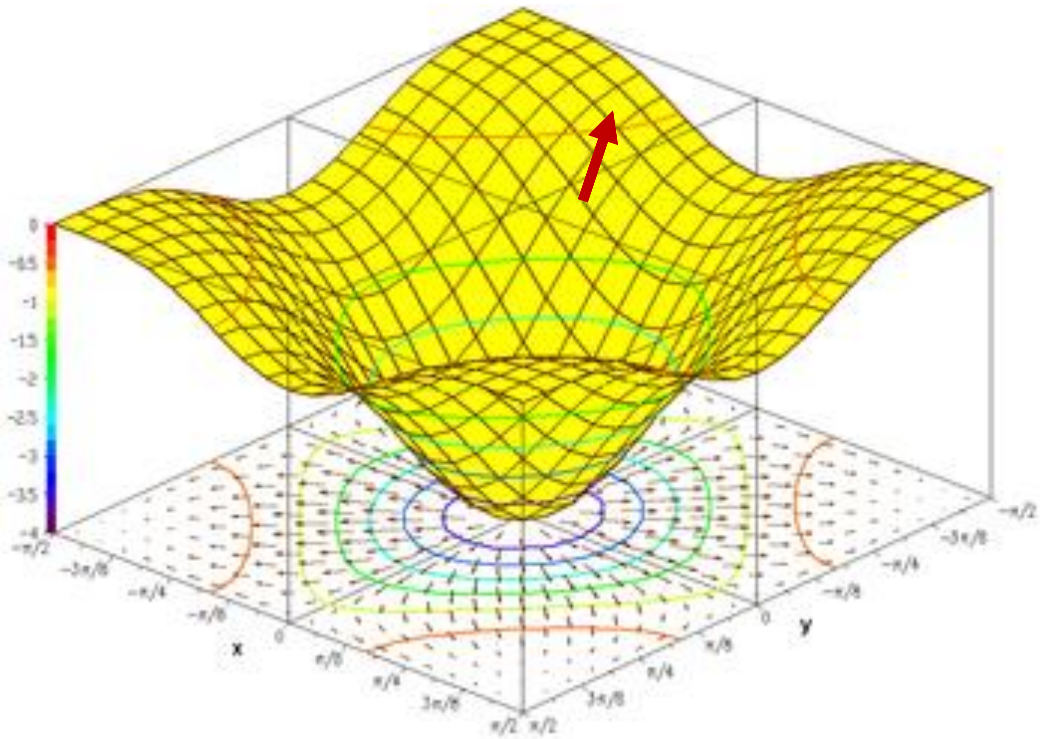
- **Gradient Descent**
- To find a *local* minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point.
- A widely-used iterative optimization technique

For a smooth function $f(\vec{x})$,

$\frac{\partial f}{\partial \vec{x}}$ is the direction that f increases most rapidly.

So $\vec{x}_{t+1} = \vec{x}_t - \eta \left. \frac{\partial f}{\partial \vec{x}} \right|_{\vec{x}=\vec{x}_t}$ until $\vec{x}$ converges

Visualization of Gradient



Gradient Descent Method

- Let E be the loss function of some model
- **Gradient**

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

- **Training Rule**

$$w_i \leftarrow w_i + \Delta w_i$$

- **with**

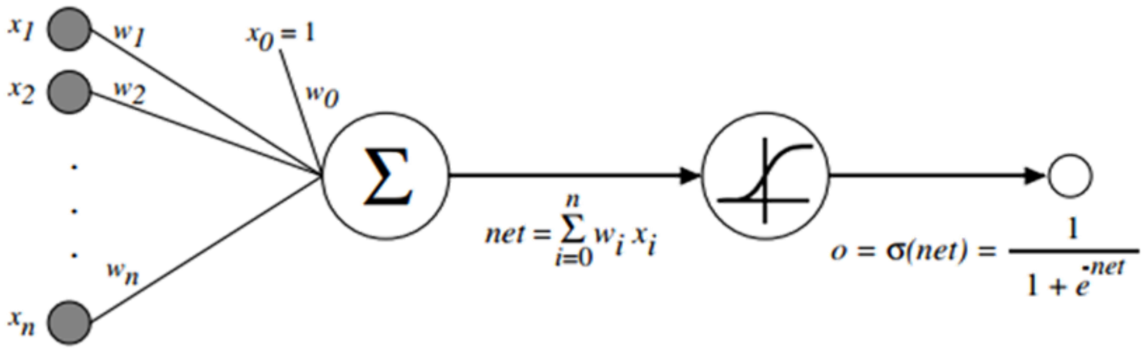
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

- **i.e.**

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Partial Derivatives and Chain Rule are the keys

Chain Rule



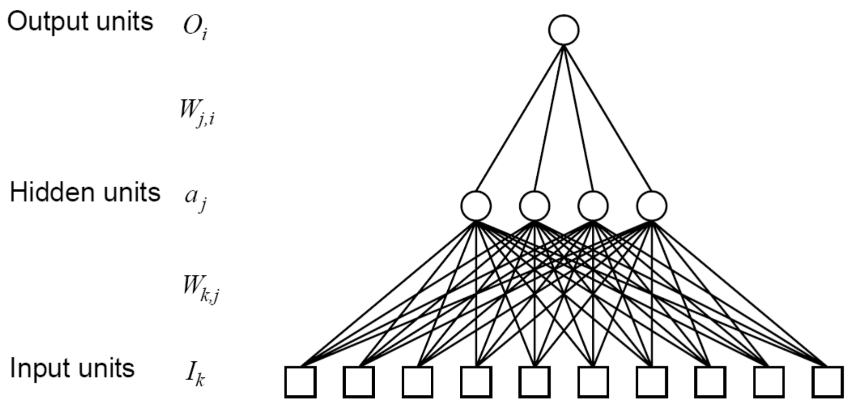
$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

Input x ➡ $net = \sum w_i x_i$ ➡ $o = \dfrac{1}{1 + e^{-net}}$ ➡ Error $E = \frac{1}{2}(t - o)^2$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial o} * \frac{\partial o}{\partial net} * \frac{\partial net}{\partial w_i}$$

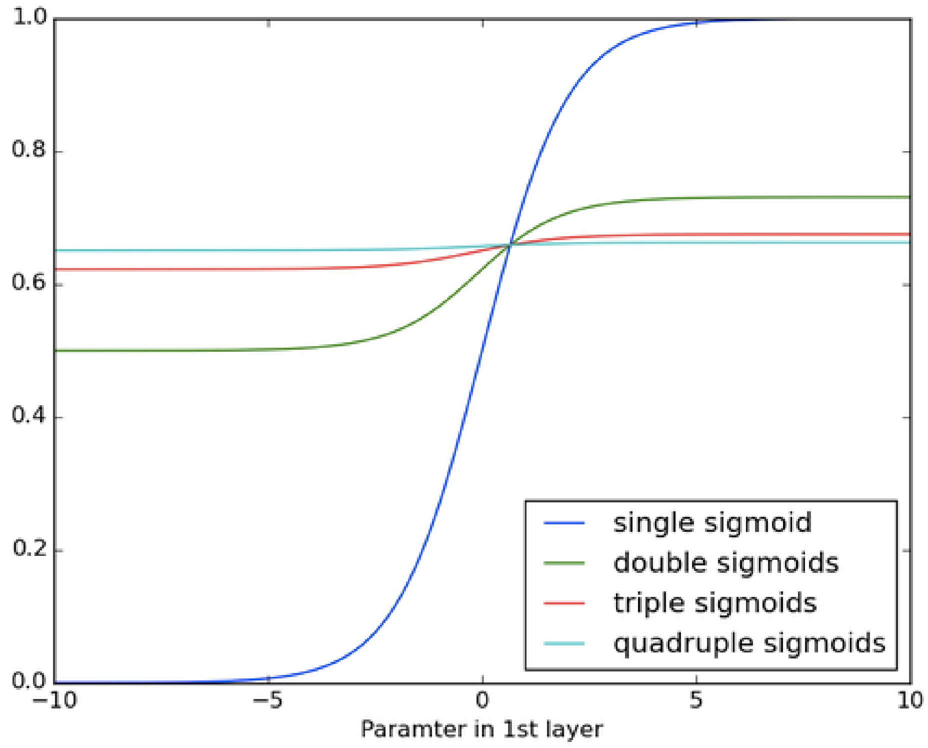
Training via Gradient Descent

function Back-Prop-Update(*network*, *examples*, α) **returns** a network with modified weights
  inputs: *network*, a multilayer network
    examples, a set of input/output pairs
    α, the learning rate
  repeat
    for each *e* **in** *examples* **do**
      /* Compute the output for this example */
      $\mathbf{O} \leftarrow$ Run-Network(*network*, $\mathbf{I^e}$)
      /* Compute the error and Δ for units in the output layer */
      $\mathbf{Err^e} \leftarrow \mathbf{T^e} - \mathbf{O}$
      /* Update the weights leading to the output layer */
      $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err^e_i \times g'(in_i)$ /* $Err^e_i \times g'(in_i) = \Delta_i$ */
      for each subsequent layer **in** network **do**
        /* Compute the error at each node */
        $\Delta_j \leftarrow g'(in_j)\Sigma_i W_{j,i}\,\Delta_i$
        /* Update the weights leading into the layer */
        $W_{k,j} \leftarrow W_{k,j} + \alpha \times I_k \times \Delta_j$
      end
    end
  until network has converged
  return network



Limitation of ANN

- **Vanishing Gradient Problem**
- Activation function becomes flatter with more layers of neurons, so the network cannot go too deep



What if

- Generally speaking, the power of neural networks increase with more layers
- Bottleneck: vanishing gradient

- What if we can overcome this problem?
- Answer: Deep Neural Network