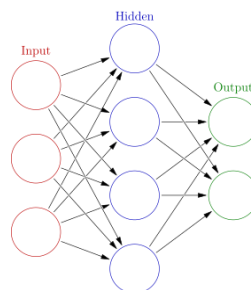


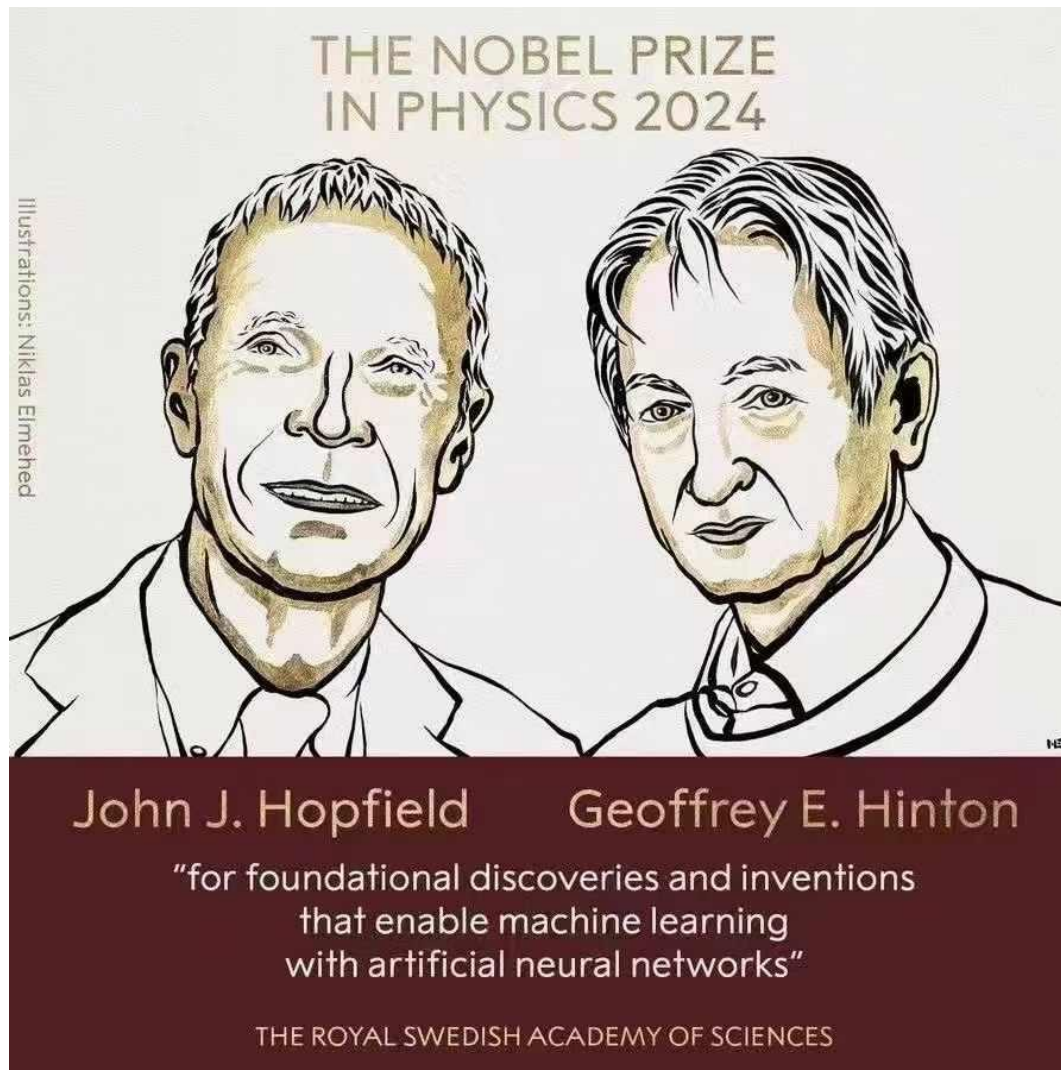
# DATA SCIENCE

## COMP2200/6200

### I0 – Artificial Neural Networks



# Big News!



ABC News: [Scientists John Hopfield and Geoffrey Hinton win Nobel Prize in Physics for developing methods that are 'the foundation' of artificial intelligence](#)






## ❖ Artificial Neural Networks

- Basic Concepts
- Multi-Layer Perceptron

## ❖ Back Propagation

- Gradient Descent Method
- Error Back Propagation

## ❖ Practical

- ❖ 1940s-1950s: 
  - Hebb learning rule, Perceptrons (single layer networks)
- ❖ 1969: 
  - Minsky demonstrated the limitation of Perceptrons
- ❖ 1974:
  - Paul Werbos from Harvard Uni invented the BP algorithm
- ❖ 1980s: 
  - Hopfield Networks, re-invention of BP
- ❖ 2000s: emergence of SVM 
- ❖ 2010s: deep learning (big data + cloud computing) 

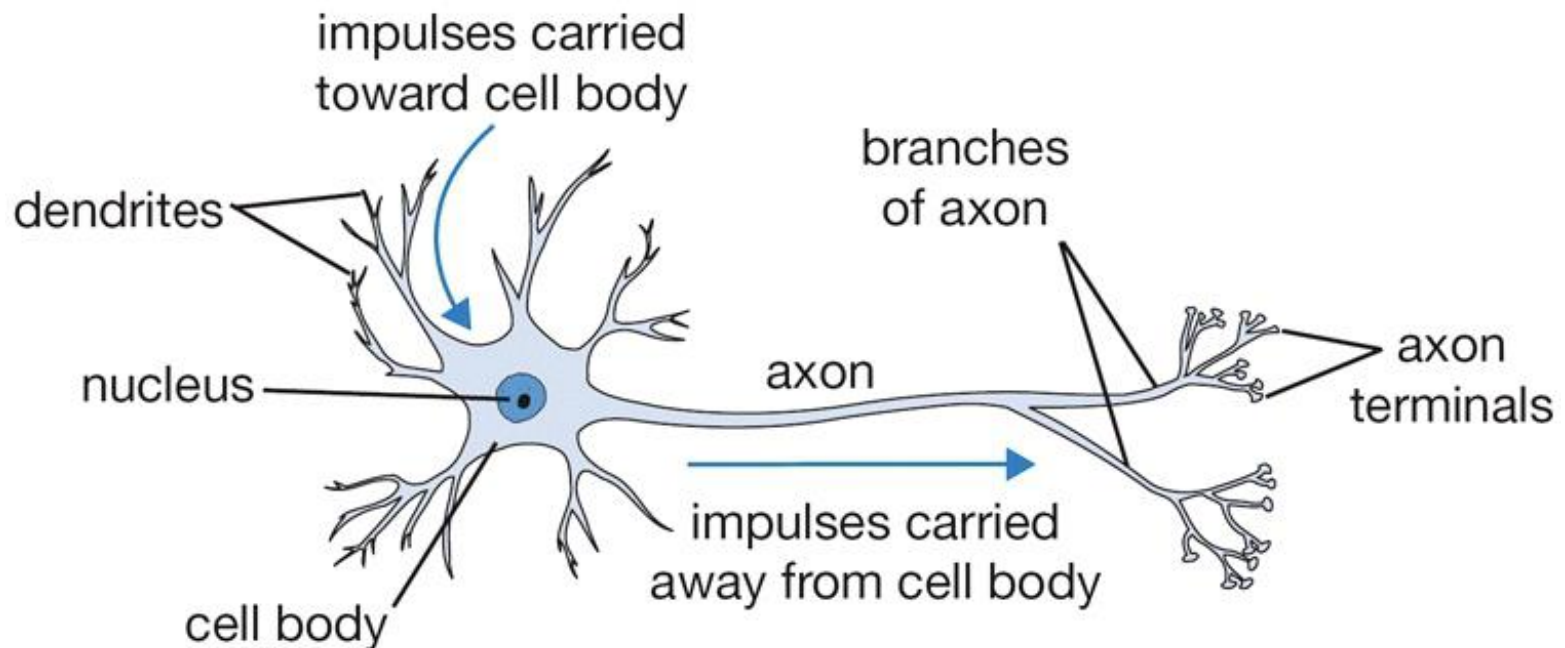
## ❖ Fathers of the Deep Learning Revolution



- Geoffrey Hinton ([University of Toronto](#), and Google)
  - **Backpropagation (1986)**; Boltzmann machines (1983)
- Yoshua Bengio ([University of Montreal](#))
  - Generative adversarial networks (2010, with Ian Goodfellow)
- Yann LeCun ([New York University](#), and Meta)
  - Convolutional neural networks (1980s)

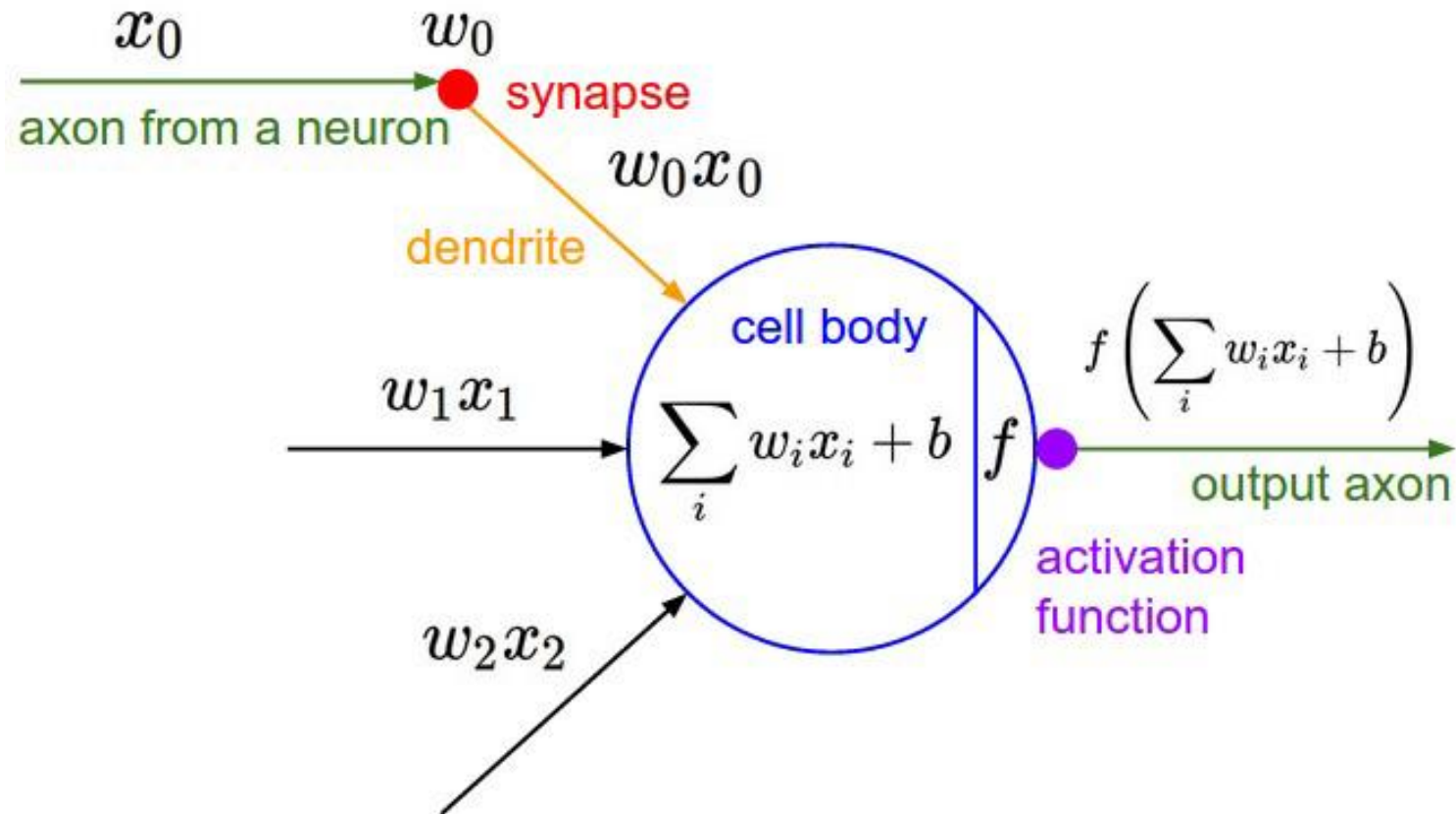
# Biological Neuron

- ❖ Human brain: 100 billion nerve cells called **neurons**
- ❖ Neurons are connected by other cells **axons**



## ❖ Computational model

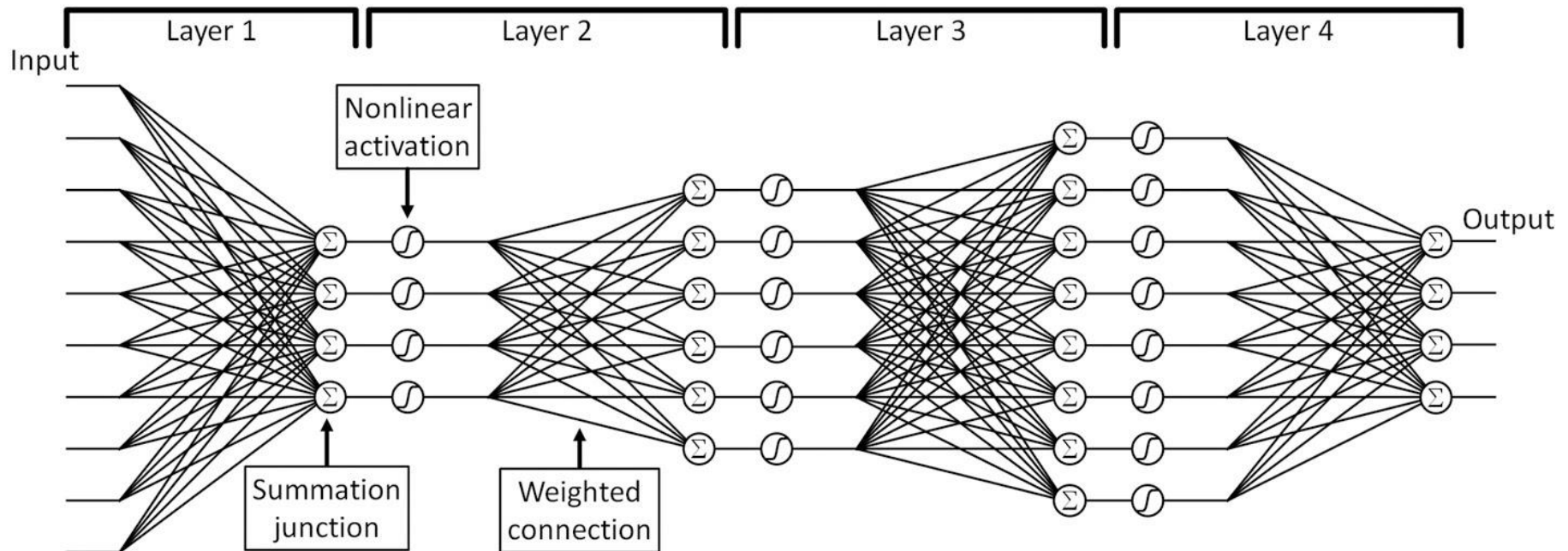
- Activation function on weighted sum of inputs





# Artificial Neural Network

- ❖ Stack neurons up to form a neural network
  - Many different network topologies
  - Many different types of activation function



©Numeric Insight, Inc.



## ❖ Input layer

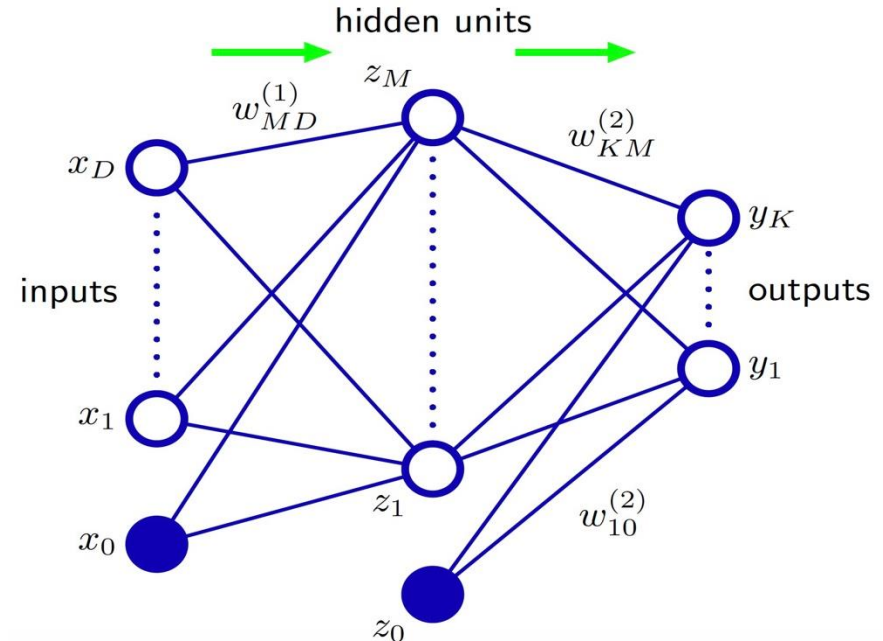
- Determined by input features, usually one unit for one feature

## ❖ Output layer

- Determined by output results
- E.g., one unit for regression, multiple units for classification

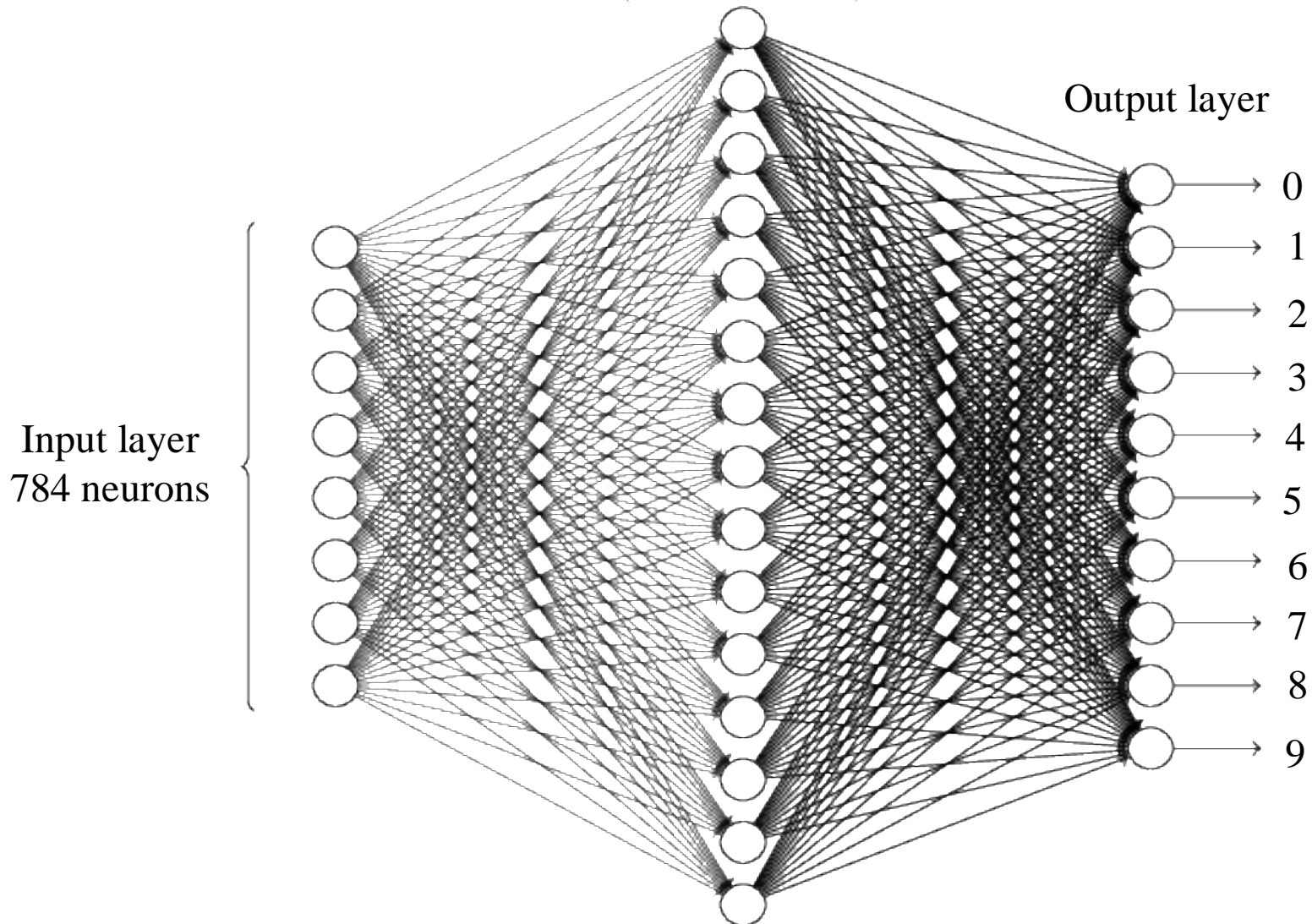
## ❖ Hidden layer(s)

- Determined by users
- # of layers
- # of units in each layer
- How to connect units
- Which activation function to use



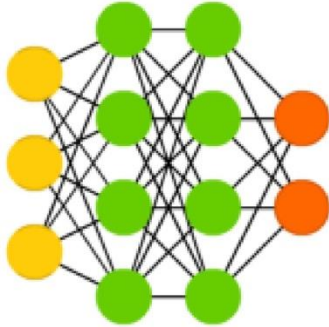
# MNIST Example

Hidden layer  $n = 15$  neurons

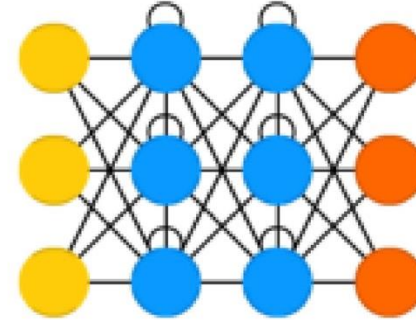


# Different NN Types

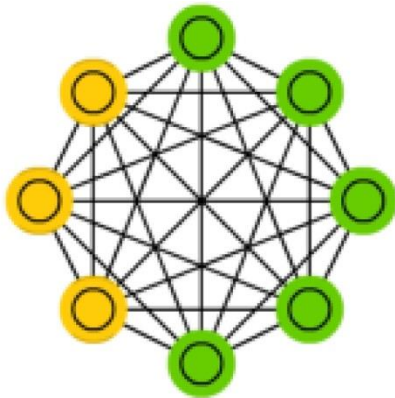
Feed-Forward



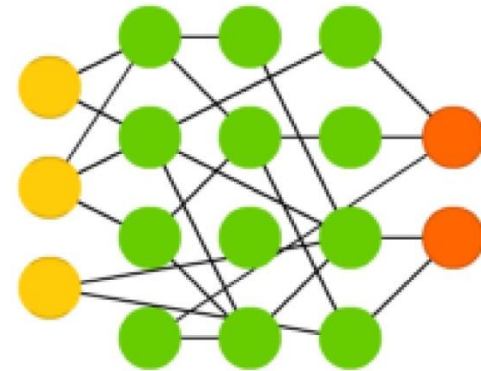
Recurrent



Boltzmann  
Machine



Extreme Learning  
Machine

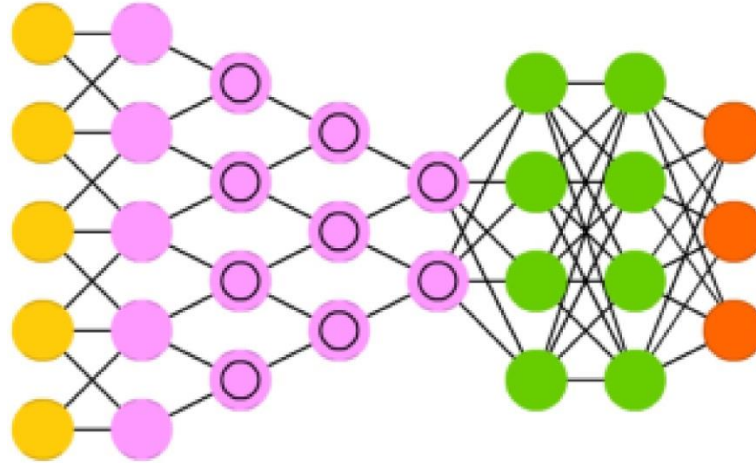


# Different NN Types (Cont'd)

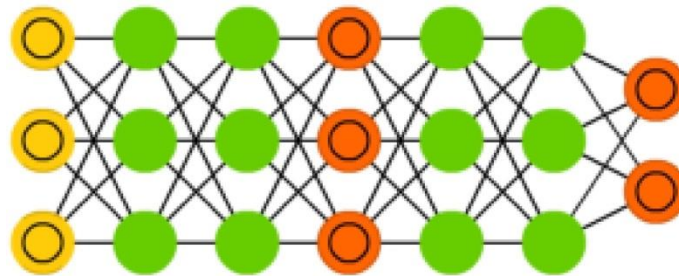


MACQUARIE  
University

Deep Convolutional  
Network



Generative  
Adversarial  
Network





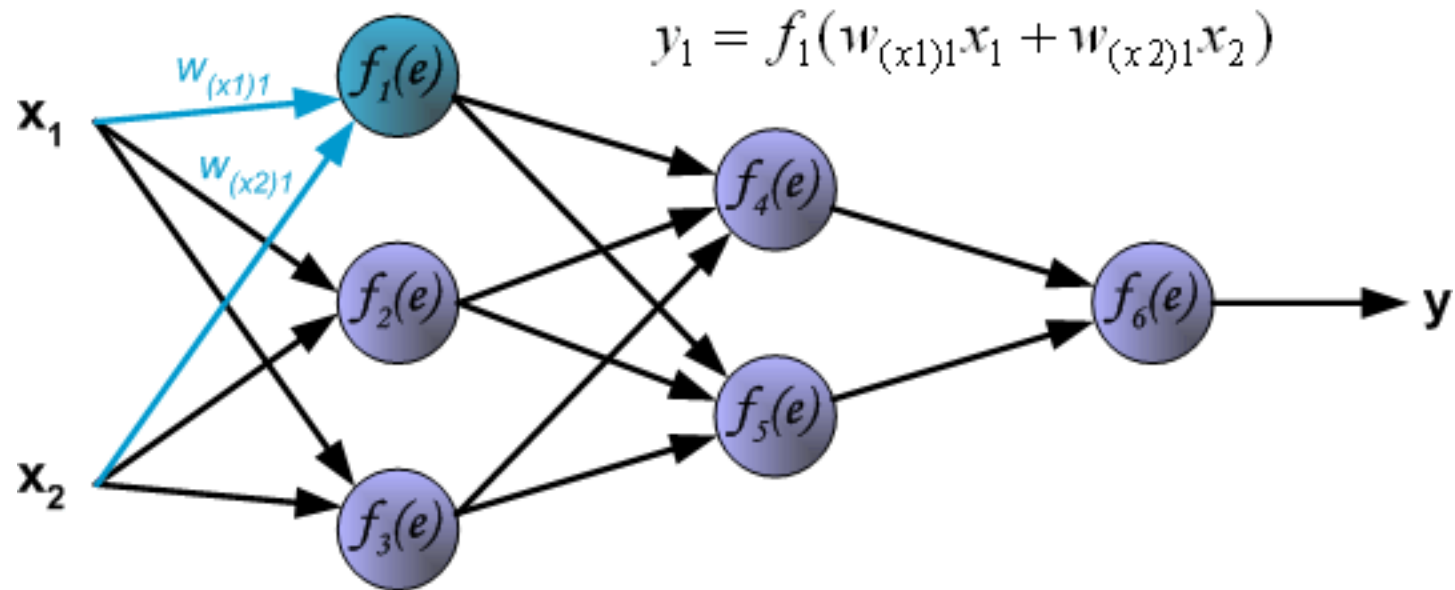
## ❖ Feed-forward network

- Information moves only from input layer directly through any hidden layers to the output layer without cycles/loops
- The first and the simplest type

## ❖ Typical examples

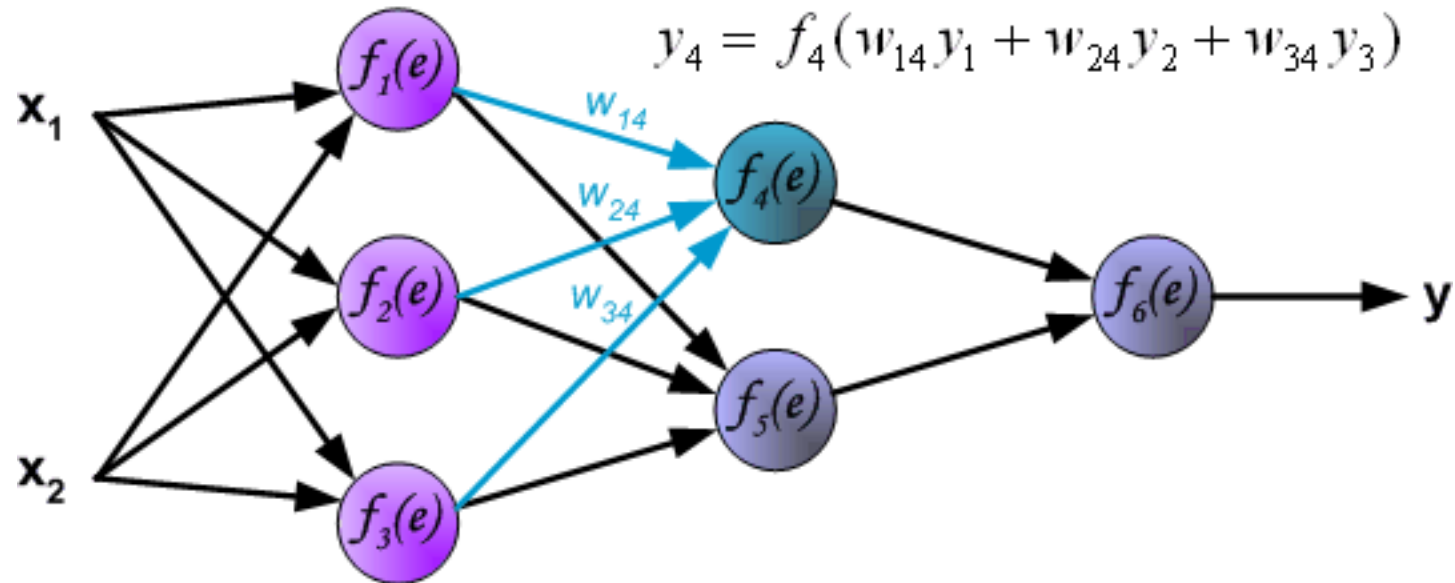
- Perceptron
  - No hidden layer
- Multi-layer perceptron (MLP)
  - One or multiple hidden layers
- Convolutional neural network (CNN)
  - Deep learning
- ...

# Feed-Forward NN



- This illustrates how signal is propagating through the network.
- Symbols  $w_{(xm)n}$  represent the weights of connections between network input  $x_m$  and neuron  $n$  in input layer.
- Symbols  $y_n$  represents output signal of neuron  $n$ .

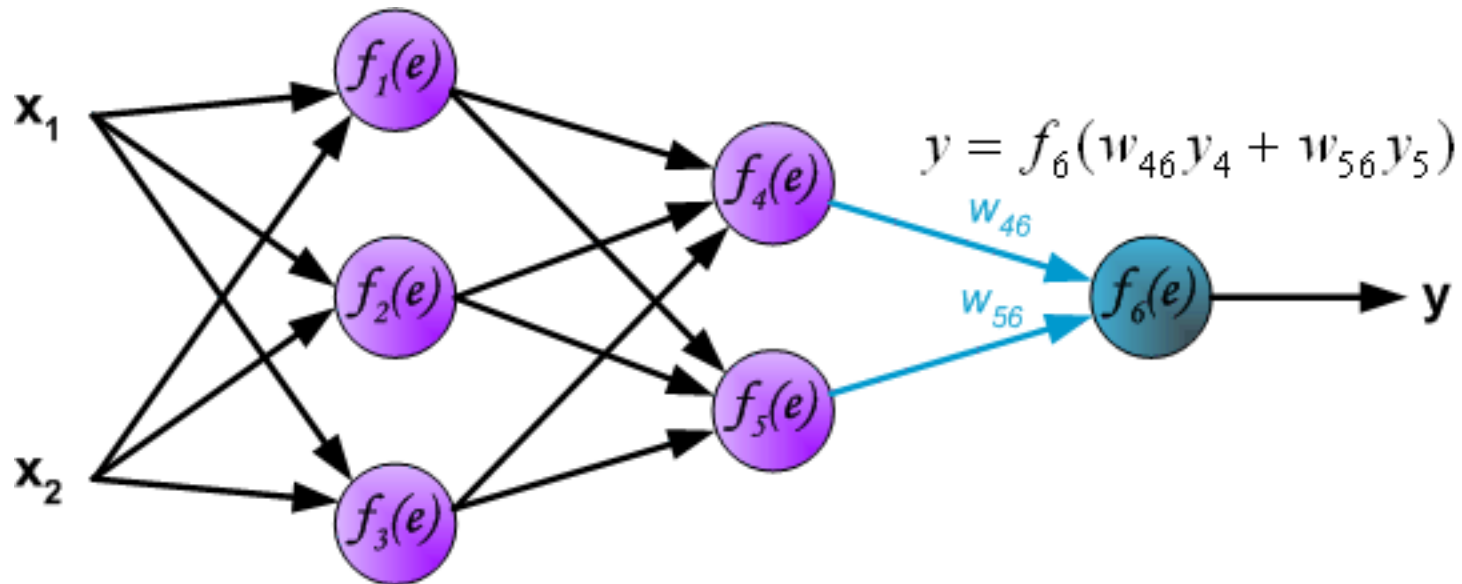
# Feed-Forward NN (Cont'd)



- This illustrates how signal is propagating through the network.
- Symbols  $w_{(xm)n}$  represent the weights of connections between network input  $x_m$  and neuron  $n$  in input layer.
- Symbols  $y_n$  represents output signal of neuron  $n$ .



# Feed-Forward NN (Cont'd)

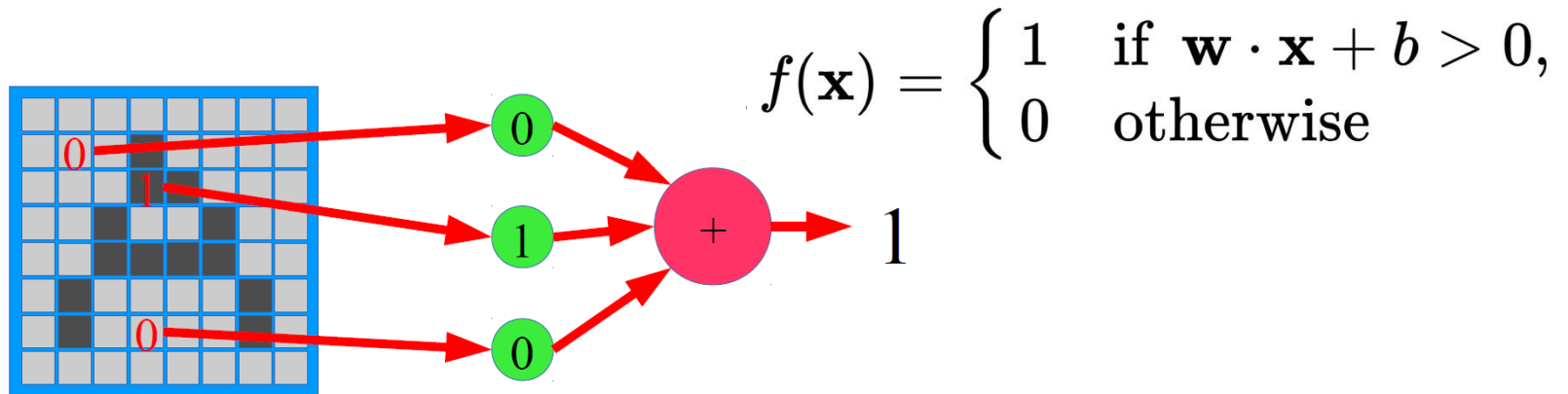


- This illustrates how signal is propagating through the network.
- Symbols  $w_{(xm)n}$  represent the weights of connections between network input  $x_m$  and neuron  $n$  in input layer.
- Symbols  $y_n$  represents output signal of neuron  $n$ .

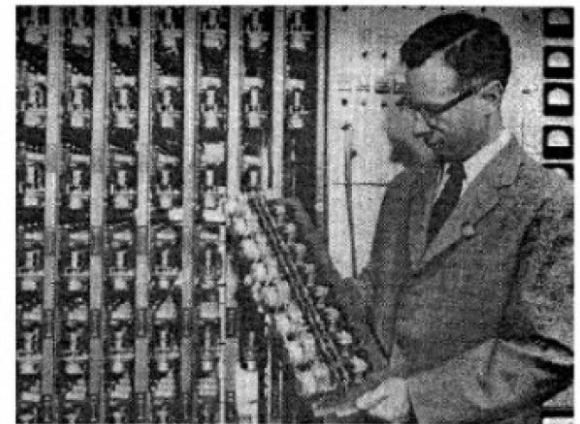
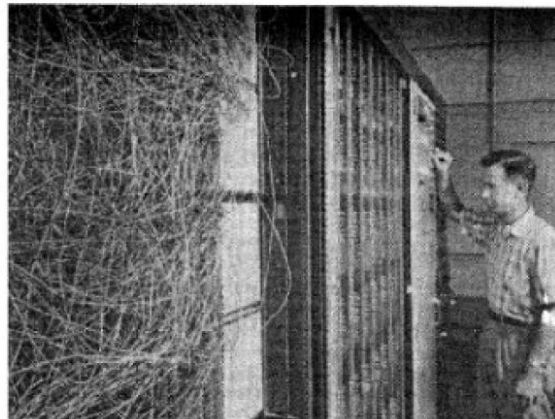
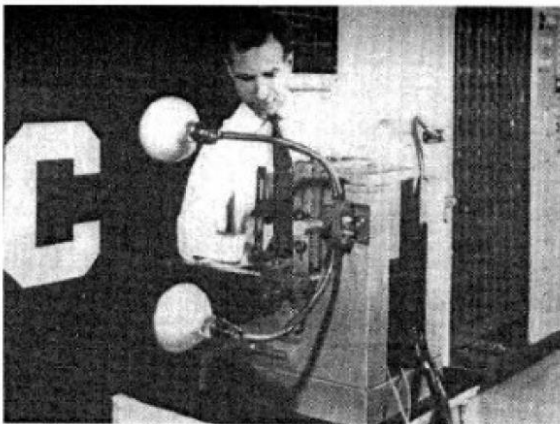
# Perceptron



MACQUARIE  
University

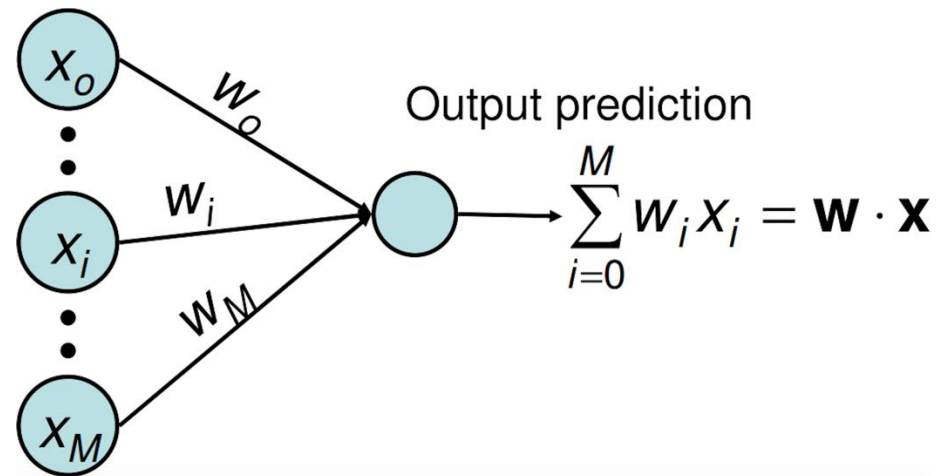


■ Perceptron (Cornell University, 1957)



# Revisit Linear Regression

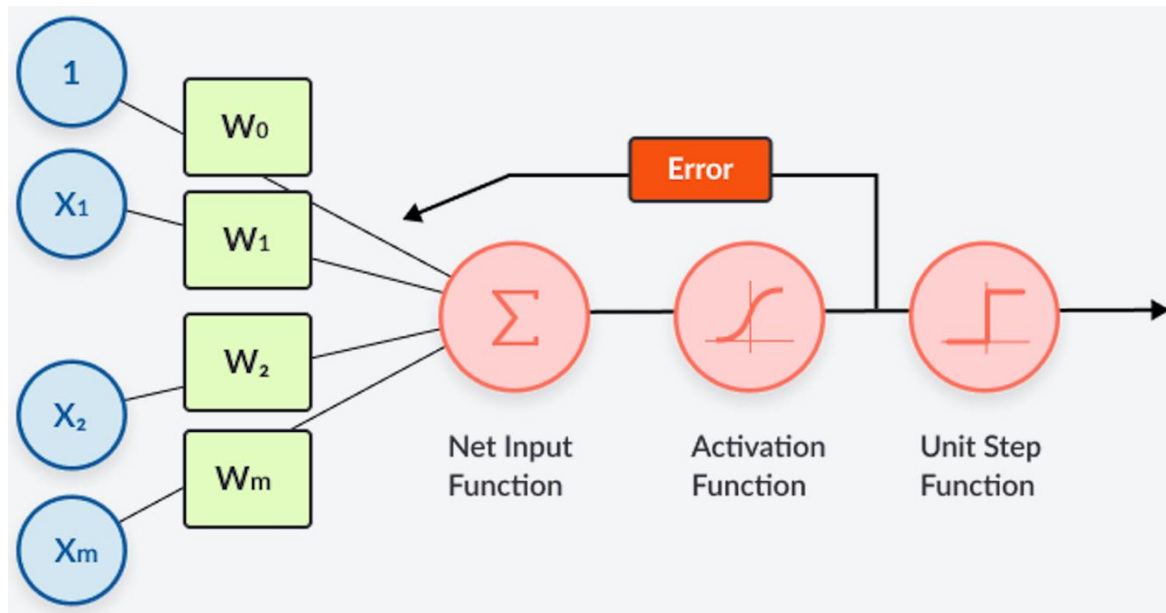
- ❖ Linear regression:  $f(\mathbf{x}) = \mathbf{w}\mathbf{x}$



- ❖ Activation function?
  - Identify function  $f(x) = x$
- ❖ Is it a good idea to use **linear functions** for activation?
  - No! No matter how many layers you have, the final activation function will be equivalent to a linear function of inputs

# Revisit Logistic Regression

❖ Logistic regression:  $f(x) = \sigma(wx)$



❖ Activation function? **Logistic sigmoid function**

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

# Logistic Sigmoid Function

- ❖ ‘Sigmoid’ means S-shaped

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

- ❖ Squashing the real axis down to  $[0, 1]$

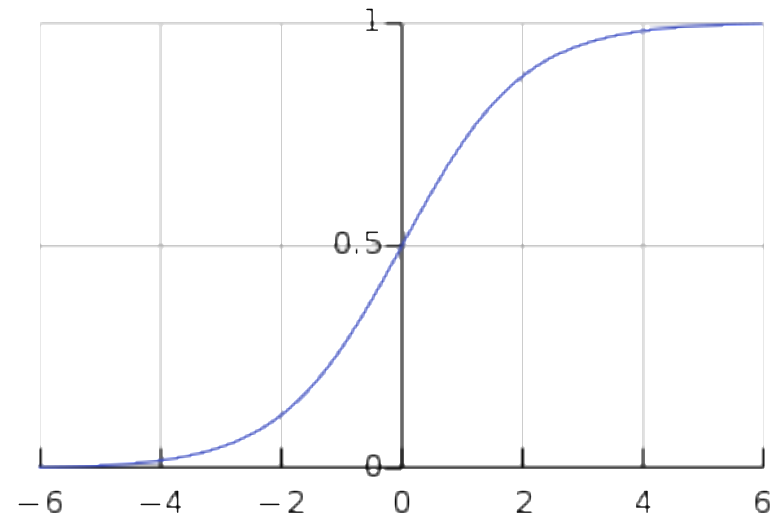
- ❖ Continuous and differentiable

- ❖ Symmetry property

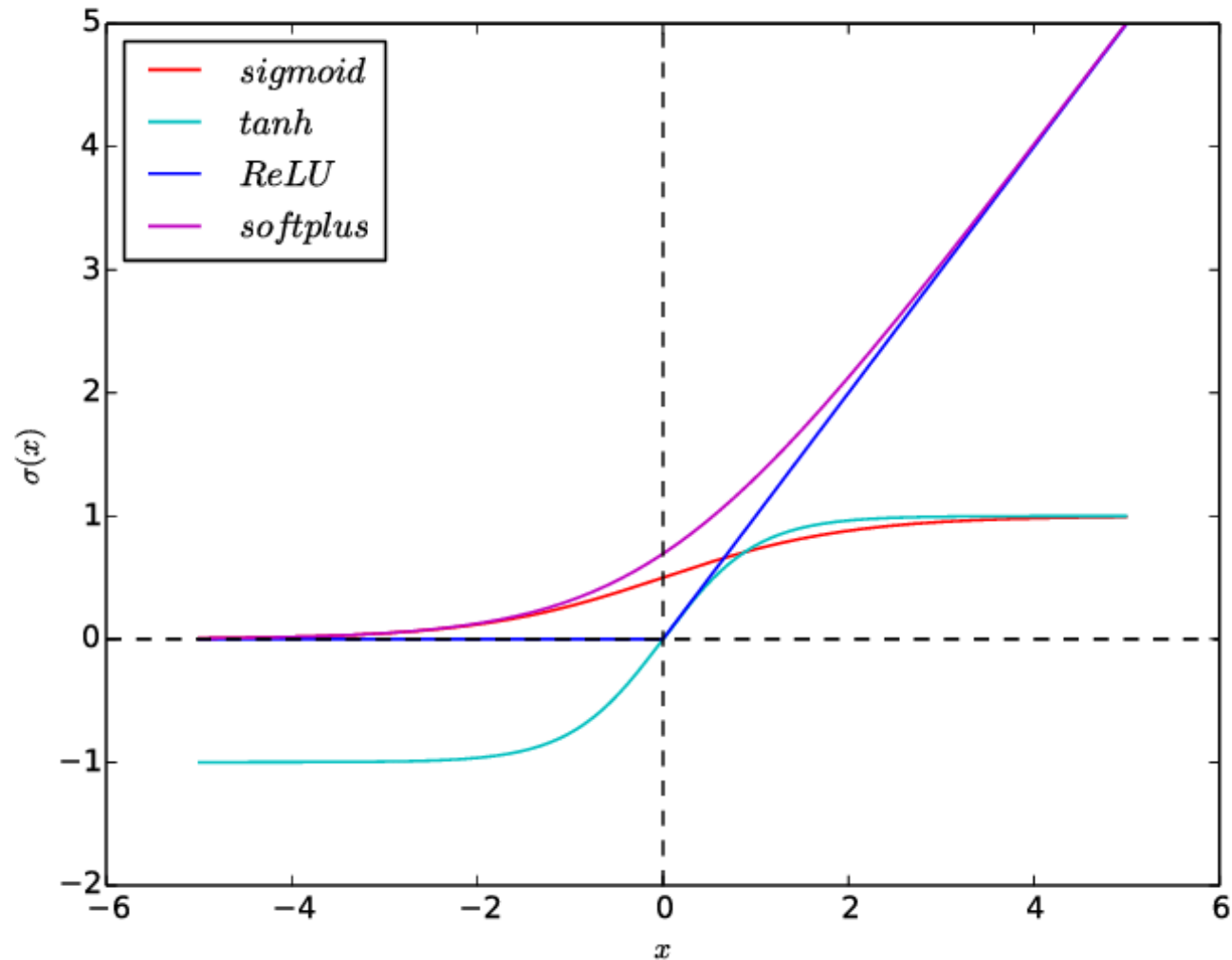
$$\sigma(-x) = 1 - \sigma(x)$$

- ❖ Convenient derivative

$$\sigma'(x) = \sigma(1 - \sigma)$$



# More Activation Functions

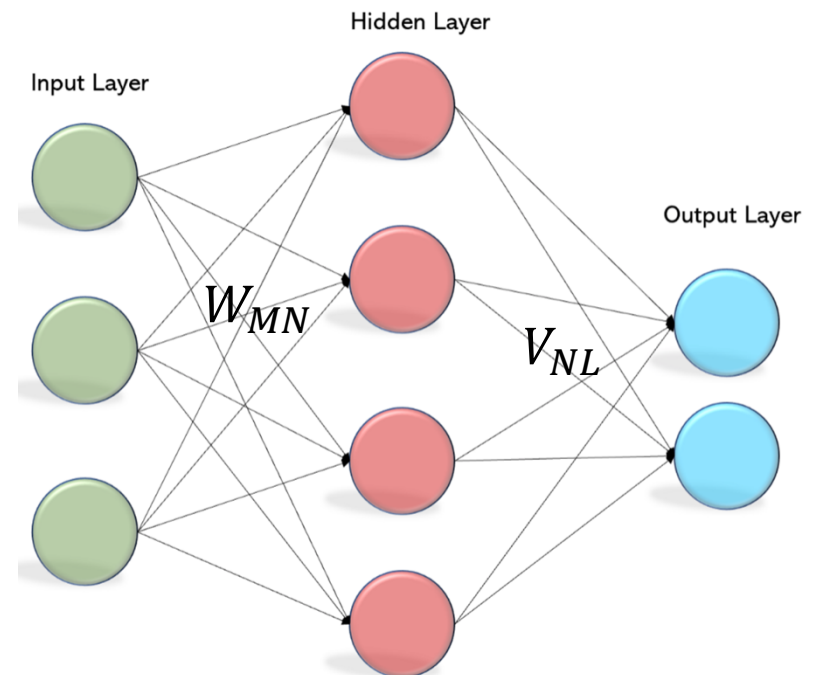


# Multi-Layer Perceptron

- ❖ One or more hidden layers
- ❖ Here we consider **one hidden layer**

- ❖ Consider following setting

- # Input feature:  $M$
- # Output dimension:  $L$
- # Hidden units:  $N$
- Weights:  $W_{MN}$  and  $V_{NL}$
- Activation function  $h$
- Output function  $g$
- Note: bias terms included for denotation convenience



- ❖ Model:  $\mathbf{y} = \mathbf{f}(\mathbf{x}|\mathbf{W}, \mathbf{V}) = \mathbf{g}(\mathbf{h}\mathbf{V}_{NL}) = \mathbf{g}(\mathbf{h}(\mathbf{x}\mathbf{W}_{MN})\mathbf{V}_{NL})$



$$\begin{aligned}
 \diamond \mathbf{y} = \mathbf{f}(\mathbf{x}|W, V) &= \begin{bmatrix} y_1 \\ \vdots \\ y_k \\ \vdots \\ y_L \end{bmatrix} = \begin{bmatrix} g(V_1 \mathbf{h}) \\ \vdots \\ g(V_k \mathbf{h}) \\ \vdots \\ g(V_L \mathbf{h}) \end{bmatrix} = \begin{bmatrix} g(\sum_{j=1}^N v_{j1} h_j) \\ \vdots \\ g(\sum_{j=1}^N v_{jk} h_j) \\ \vdots \\ g(\sum_{j=1}^N v_{jL} h_j) \end{bmatrix} \\
 \diamond &= \begin{bmatrix} g(\sum_{j=1}^N v_{j1} h(W_j \mathbf{x})) \\ \vdots \\ g(\sum_{j=1}^N v_{jk} h(W_j \mathbf{x})) \\ \vdots \\ g(\sum_{j=1}^N v_{jL} h(W_j \mathbf{x})) \end{bmatrix} = \begin{bmatrix} g(\sum_{j=1}^N v_{j1} h(\sum_{i=1}^M w_{ij} x_i)) \\ \vdots \\ g(\sum_{j=1}^N v_{jk} h(\sum_{i=1}^M w_{ij} x_i)) \\ \vdots \\ g(\sum_{j=1}^N v_{jL} h(\sum_{i=1}^M w_{ij} x_i)) \end{bmatrix}
 \end{aligned}$$

## ❖ Squared error

- The observed target  $t$

$$E(\mathbf{y}) = \|\mathbf{y} - \mathbf{t}\| = \frac{1}{2} \sum_{k=1}^L (y_k - t_k)^2$$

- Substituting  $W, V$

$$E(W, V) = \frac{1}{2} \sum_{k=1}^L \left( g\left(\sum_{j=1}^N v_{jk} h\left(\sum_{i=1}^M w_{ij} x_i\right)\right) - t_k \right)^2$$

## ❖ Squared sum of errors for a data set of size $n$

$$E_n(W, V) = \frac{1}{2} \sum_{r=1}^n \sum_{k=1}^L \left( g\left(\sum_{j=1}^N v_{jk} h\left(\sum_{i=1}^M w_{ij} x_i^{(r)}\right)\right) - t_k^{(r)} \right)^2$$

## ❖ Can also use other error functions, e.g., cross-entropy

- ❖ What to learn?
  - **Model parameters:**  $W, V$
  - # of model parameters:  $M * N + N * L$
- ❖ Network topology is a hyperparameter
  - Not a model parameter
  - Determine the number of model parameters
  - Control model complexity
  - Note that activation functions also affect model complexity
- ❖ How to learn?
  - **Minimize** cost function w.r.t. a data set to obtain  $W, V$
  - An optimization problem



## ❖ Artificial Neural Networks

- Basic Concepts
- Multi-Layer Perceptron

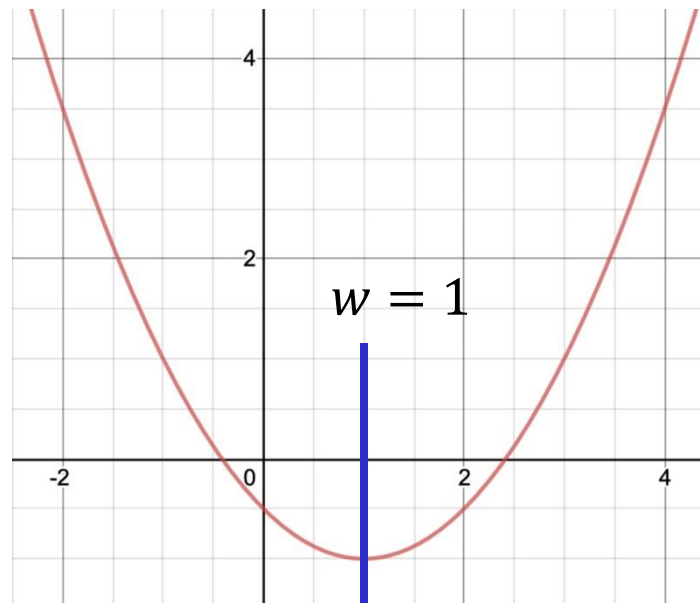
## ❖ Back Propagation

- Gradient Descent Method
- Error Back Propagation

- ❖ Question: which  $w$  minimize the following function?

$$f(w) = \frac{1}{2}(w - 1)^2 - 1$$

- ❖ Option I: observe its curve (middle/high school)



# Gradient Descent (Cont'd)



## ❖ Option 2: analytical method with derivative (calculus)

$$f'(w) = \frac{1}{2} \cdot 2 \cdot (w - 1) \cdot 1 - 0 = w - 1$$

- To find stationary points, let

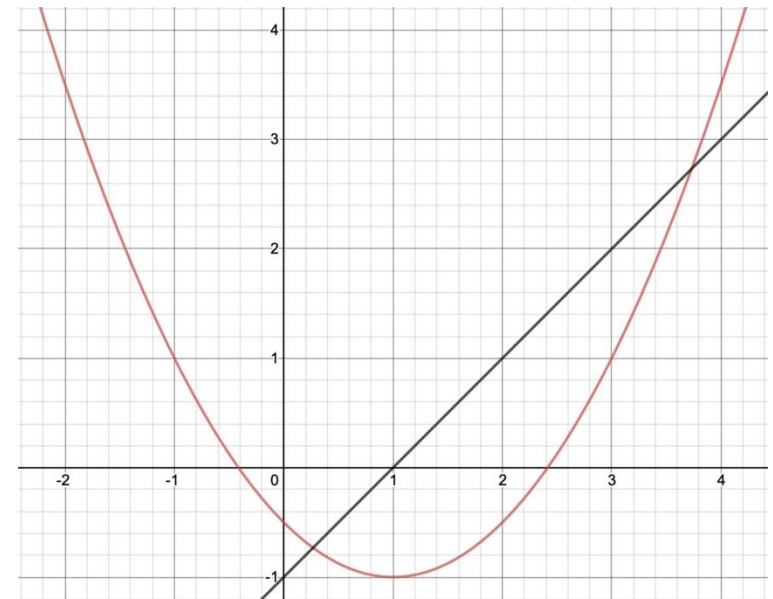
$$f'(w) = 0$$

$$f'(w) = w - 1 = 0$$

$$\Rightarrow w^* = 1$$

$$f''(w) = (w - 1)' = 1 > 0$$

- $w^* = 1$  is the value minimizing the function
- **Linear regression** adopt this method to compute weights



## ❖ Option 3: Numerical method

- Randomly initialize the value ( $w^{(0)}$ ), and optimize it gradually
- Needs an update rule (still make use of derivative)

$$w^{(\tau+1)} \leftarrow w^{(\tau)} - \eta f'(w) \Big|_{w^{(\tau)}} = w^{(\tau)} - \eta(w^{(\tau)} - 1)$$

- $\eta$  is step size (or learning rate)

- Stops after the result is good enough
  - E.g., difference between two iterations is small enough

## ❖ Unlike Option 2, we don't need to solve $f'(w) = 0$

- When a function is more complicated, solving the equation system analytically is really challenging
- So, this method is more general with an approximate result



# Gradient Descent (Cont'd)

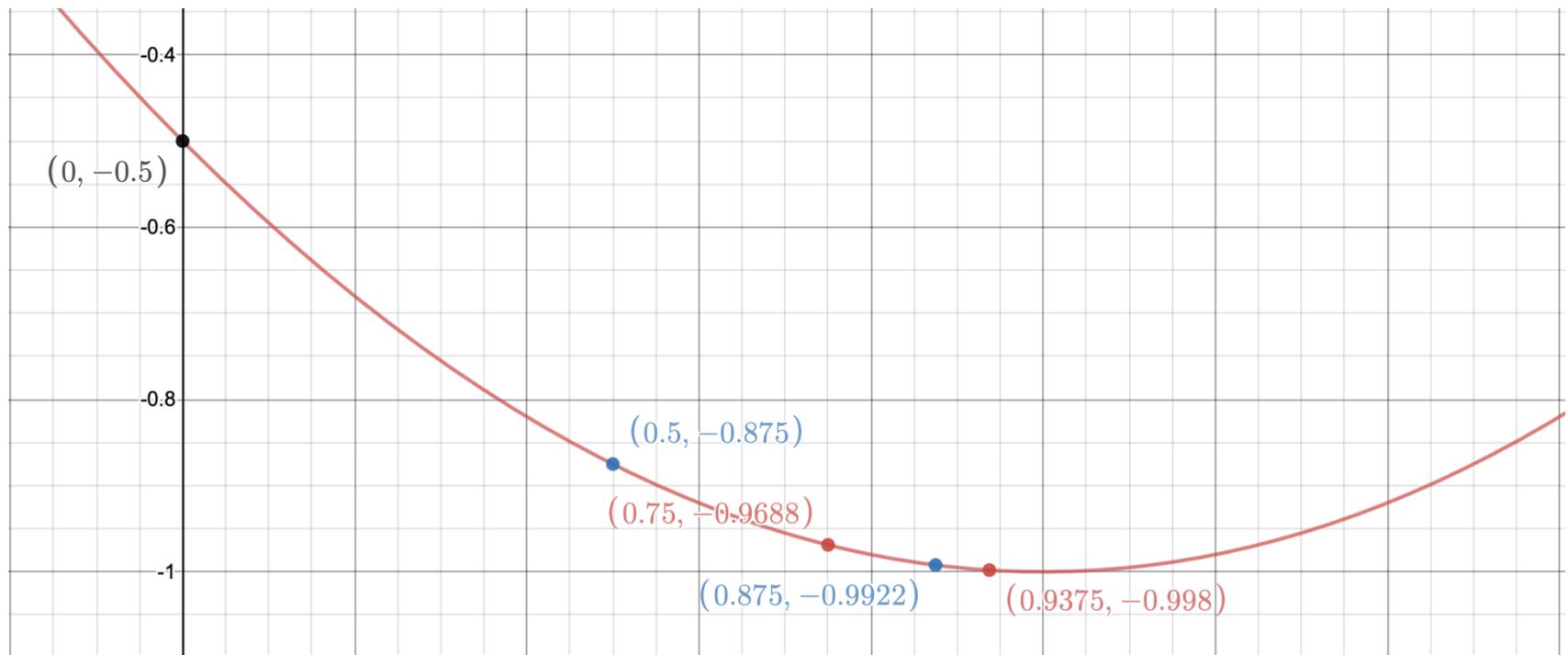
## ❖ Option 3: Numerical method

- Show an example when  $w^{(0)} = 0$ , and  $\eta = \frac{1}{2}$

$\tau$	$w$	$f'(w)$	$f(w)$	$ f^{(\tau+1)}(w) - f^{(\tau)}(w) $
0	0	-1	$-\frac{1}{2}$	
1	$0 - \frac{1}{2}(-1) = \frac{1}{2}$	$-\frac{1}{2}$	$-\frac{7}{8}$	0.375
2	$\frac{1}{2} - \frac{1}{2}\left(-\frac{1}{2}\right) = \frac{3}{4}$	$-\frac{1}{4}$	$-\frac{31}{32}$	0.093
3	$\frac{3}{4} - \frac{1}{2}\left(-\frac{1}{4}\right) = \frac{7}{8}$	$-\frac{1}{8}$	$-\frac{127}{128}$	0.0234
4	$\frac{7}{8} - \frac{1}{2}\left(-\frac{1}{8}\right) = \frac{15}{16}$	$-\frac{1}{16}$	$-\frac{511}{512}$	0.0058
...	...	...	...	...

## ❖ Option 3: Numerical method

- $w^* \leftarrow w^{(4)} = \frac{15}{16}$  is a good approximate to the true value 1

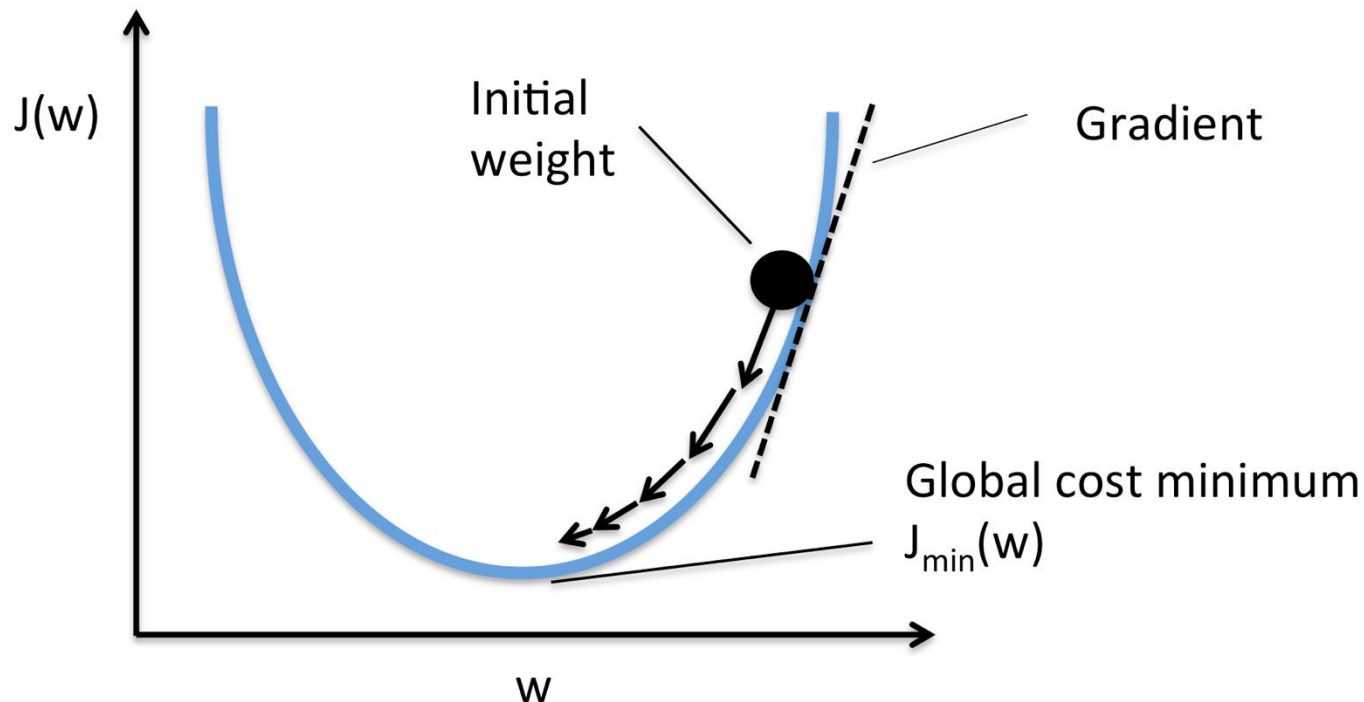


# Gradient Descent (Cont'd)



- ❖ General rule for a cost function  $J(w)$

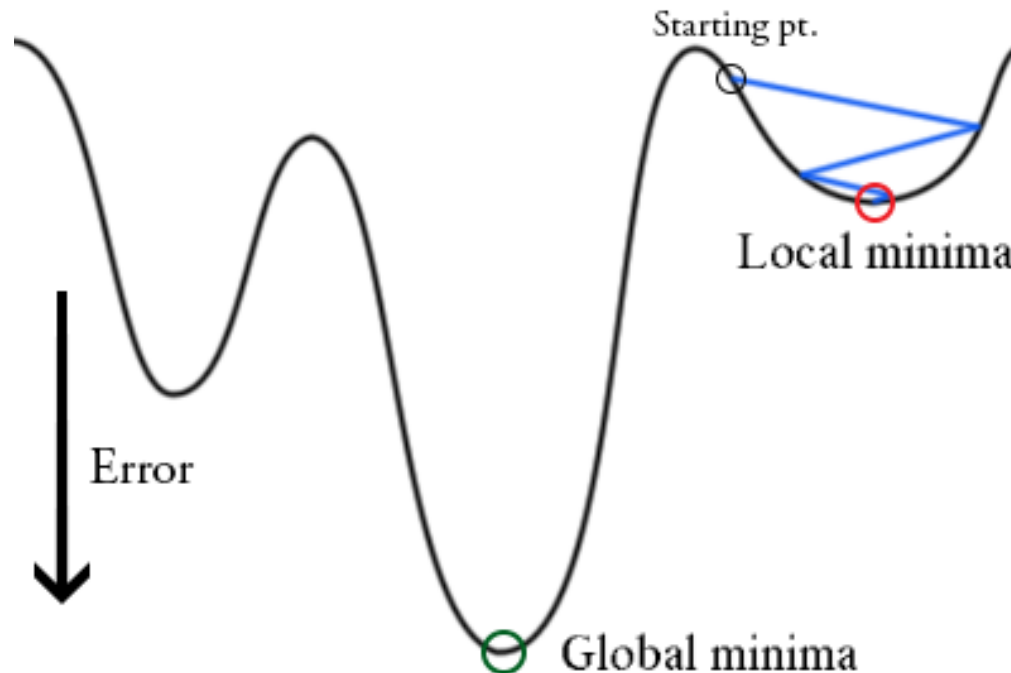
$$w^{(\tau+1)} \leftarrow w^{(\tau)} - \eta \cdot J'(w) \Big|_{w^{(\tau)}}$$



# Gradient Descent (Cont'd)



- ❖ What if the function is **non-convex**?
  - Can get stuck in local minima



- ❖ Extend to multiple variable function
  - This is a much more common case in machine learning
- ❖ Beyond 'derivative', we need 'gradient'
- ❖ For a function  $f(\mathbf{w}) = f(w_1, \dots, w_i, \dots, w_n)$
- ❖ Gradient of  $f(\mathbf{w})$

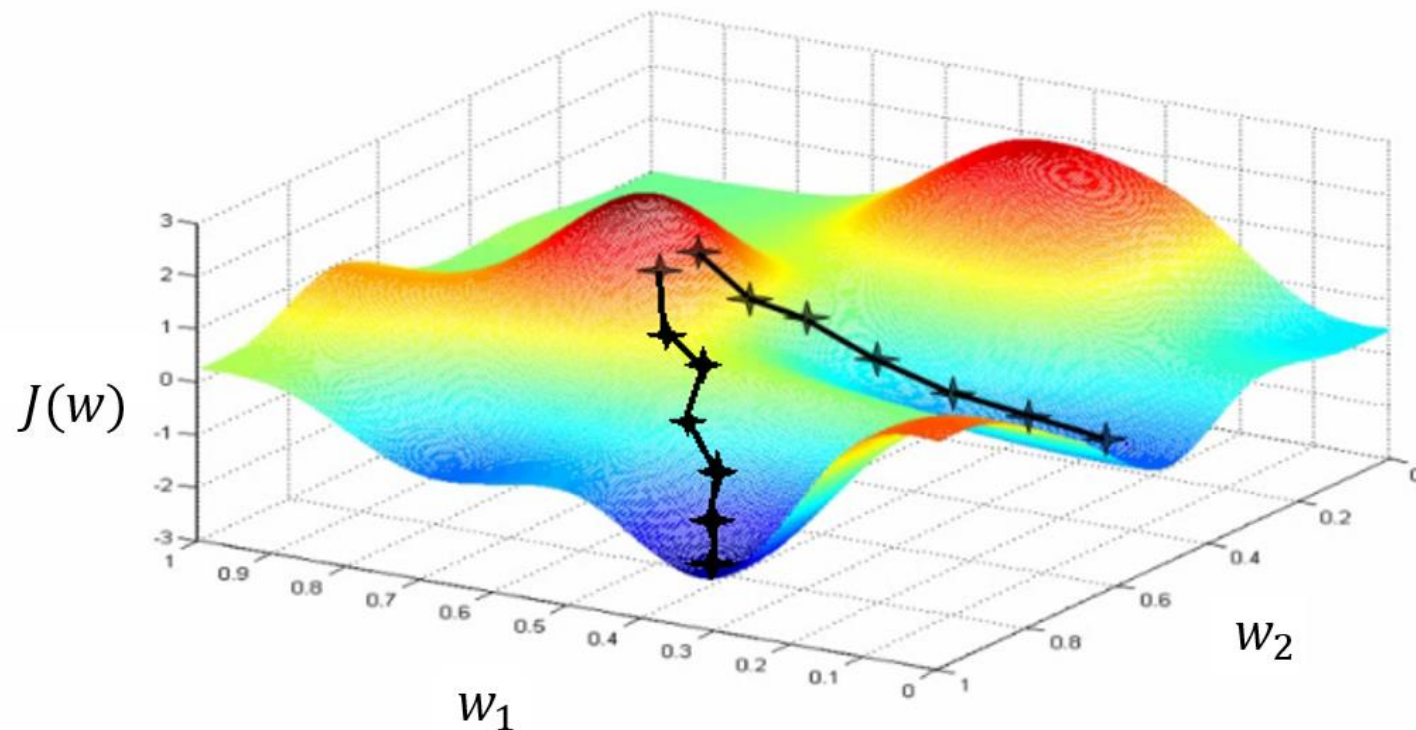
$$\nabla f(\mathbf{w}) = \left( \frac{\partial f(\mathbf{w})}{\partial w_1}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_i}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_n} \right)$$

- $\frac{\partial f(\mathbf{w})}{\partial w_i}$  is partial derivative w.r.t.  $w_i$
- **Direction**: direction of steepest increase
- **Magnitude**: rate of change in that direction

# Gradient Descent (Cont'd)

- ❖ Update rule for a cost function  $J(\mathbf{w})$

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \eta \cdot \nabla J(\mathbf{w}) \Big|_{\mathbf{w}^{(\tau)}}$$



## ❖ Derivative for function composition

- Actually very common when we compute derivative

- Let's revisit the function  $f(w) = \frac{1}{2}(w - 1)^2 - 1$

- Regard it as a function composition of  $h$  and  $g$

- $h(x) = \frac{1}{2}x^2 - 1, h'(x) = x$

- $g(x) = x - 1, g'(x) = 1$

- $f(w) = h \circ g(w) = h(g(w)) = \frac{1}{2}[g(w)]^2 - 1 = \frac{1}{2}(w - 1)^2 - 1$

- Chain rule for function composition derivative

- $h \circ g(w)' = h'(x)|_{g(w)} \cdot g'(w) = x|_{g(w)} \cdot 1 = g(w) = w - 1$

## ❖ General chain rule: $h \circ g(w)' = h'(x)|_{g(w)} \cdot g'(w)$



- ❖ Squared sum of errors for a data set of size  $n$

$$E_n(W, V) = \frac{1}{2} \sum_{r=1}^n \sum_{k=1}^L \left( g\left(\sum_{j=1}^N v_{jk} h\left(\sum_{i=1}^M w_{ij} x_i^{(r)}\right)\right) - t_k^{(r)} \right)^2$$

- ❖ Observations

- Training the model is to minimize the cost function
- Hard to use the analytical method due to the complexity
- We need to use gradient descent method
- Just need to compute the gradient w.r.t. to  $W$  and  $V$

- ❖ Weight update rule, let  $\theta = (W, V)$

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta \cdot \nabla E_n(\theta) \Big|_{\theta^{(\tau)}}$$

$$E_n(W, V) = \frac{1}{2} \sum_{r=1}^n \sum_{k=1}^L \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right)^2$$

❖ Consider weights between hidden and output layers

$$\begin{aligned} \frac{\partial E_n(W, V)}{\partial v_{jk}} &= \sum_{r=1}^n \frac{\partial}{\partial v_{jk}} \sum_{k=1}^L \frac{1}{2} \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right)^2 \\ &= \sum_{r=1}^n \frac{\partial}{\partial v_{jk}} \frac{1}{2} \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right)^2 \\ &= \sum_{r=1}^n \left( g_k - t_k^{(r)} \right) \frac{\partial}{\partial v_{jk}} \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right) \\ &= \sum_{r=1}^n \left( g_k - t_k^{(r)} \right) \frac{\partial}{\partial v_{jk}} g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) \\ &= \sum_{r=1}^n \left( g_k - t_k^{(r)} \right) g' \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) \frac{\partial}{\partial v_{jk}} \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \\ &= \sum_{r=1}^n \left( g_k - t_k^{(r)} \right) g'_k \frac{\partial}{\partial v_{jk}} v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) = \sum_{r=1}^n \left( g_k - t_k^{(r)} \right) g'_k h_j \end{aligned}$$

- The partial derivative w.r.t.  $v_{jk}$  is  $\sum_{r=1}^n \left( g_k - t_k^{(r)} \right) g'_k h_j$

## ❖ Consider weights between input and hidden layers

$$\begin{aligned}
 \frac{\partial E_n(W, V)}{\partial w_{ij}} &= \sum_{r=1}^n \frac{\partial}{\partial w_{ij}} \sum_{k=1}^L \frac{1}{2} \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right)^2 \\
 &= \sum_{r=1}^n \sum_{k=1}^L \frac{\partial}{\partial w_{ij}} \frac{1}{2} \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right)^2 \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) \frac{\partial}{\partial w_{ij}} \left( g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) - t_k^{(r)} \right) \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) \frac{\partial}{\partial w_{ij}} g \left( \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \right) \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) g'_k \frac{\partial}{\partial w_{ij}} \sum_{j=1}^N v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) g'_k \frac{\partial}{\partial w_{ij}} v_{jk} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) g'_k v_{jk} \frac{\partial}{\partial w_{ij}} h \left( \sum_{i=1}^M w_{ij} x_i^{(r)} \right) \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) g'_k v_{jk} h'_j \frac{\partial}{\partial w_{ij}} \sum_{i=1}^M w_{ij} x_i^{(r)} \\
 &= \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) g'_k v_{jk} h'_j \frac{\partial}{\partial w_{ij}} w_{ij} x_i^{(r)} = \sum_{r=1}^n \sum_{k=1}^L (g_k - t_k^{(r)}) g'_k v_{jk} h'_j x_i^{(r)}
 \end{aligned}$$

## ❖ Partial derivatives (can be used for weight update)

- $\frac{\partial E_n(W,V)}{\partial v_{jk}} = \sum_{r=1}^n \left( g_k - t_k^{(r)} \right) g'_k h_j$
- $\frac{\partial E_n(W,V)}{\partial w_{ij}} = \sum_{r=1}^n \left( \sum_{k=1}^L \left( g_k - t_k^{(r)} \right) g'_k v_{jk} \right) h'_j x_i^{(r)}$

## ❖ Observations (consider a single data instance update)

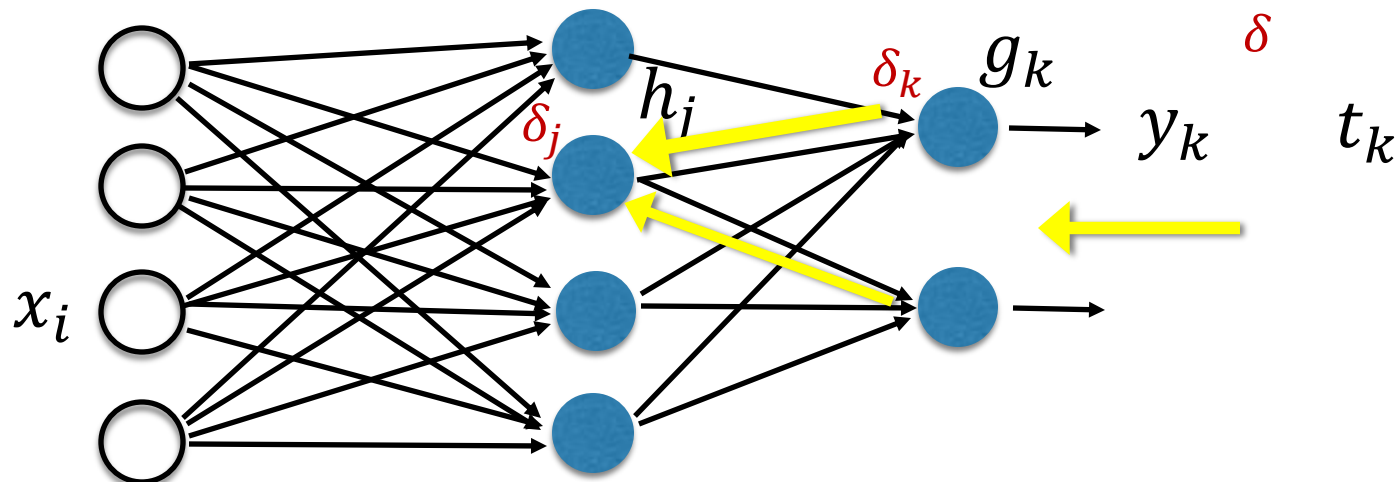
- Let error signal  $(g_k - t_k) \equiv \delta$
- For the output layer:  $\frac{\partial E_n(W,V)}{\partial v_{jk}} = \delta_k h_j, \delta_k \equiv \delta g'_k$
- For hidden layers:  $\frac{\partial E_n(W,V)}{\partial w_{ij}} = \delta_j x_i, \delta_j \equiv \left( \sum_{k=1}^L \delta_k v_{jk} \right) h'_j$
- Error signal  $\delta_j$  is calculated from  $\delta_k$ , i.e., **back propagation!**

# Error Back Propagation (Cont'd)



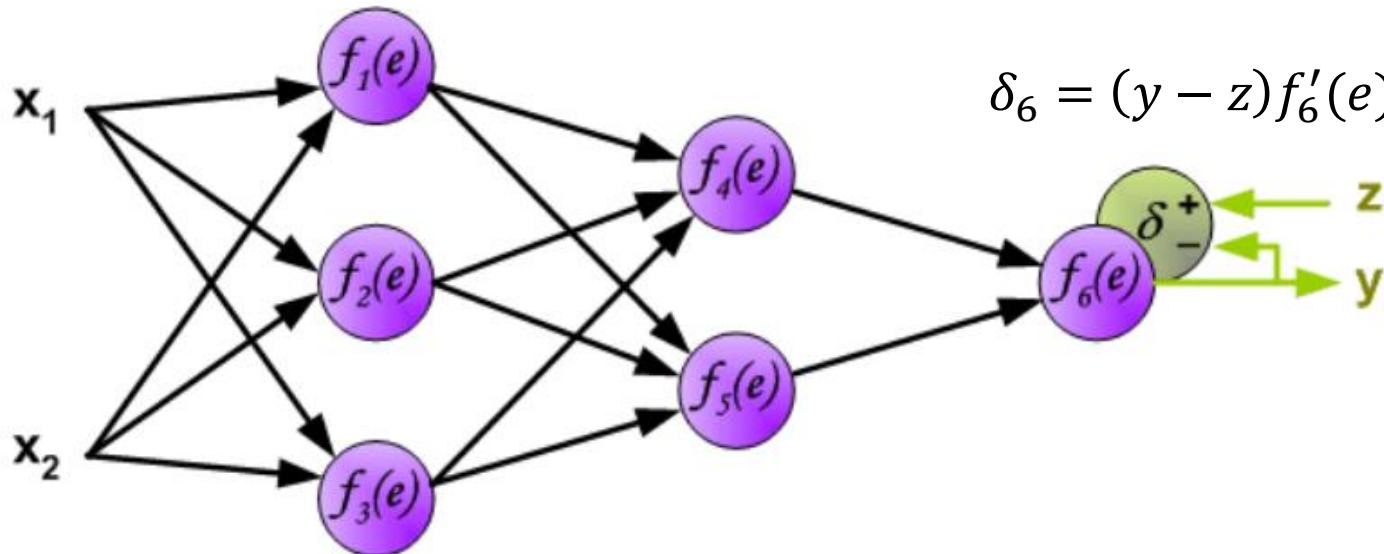
## ❖ Back propagation of error signals

- $\delta = (g_k - t_k)$ . Note: this error will be different for other cost functions, e.g., cross-entropy.
- $\delta_k = \delta g'_k$
- $\delta_j = (\sum_{k=1}^L \delta_k v_{jk}) h'_j$

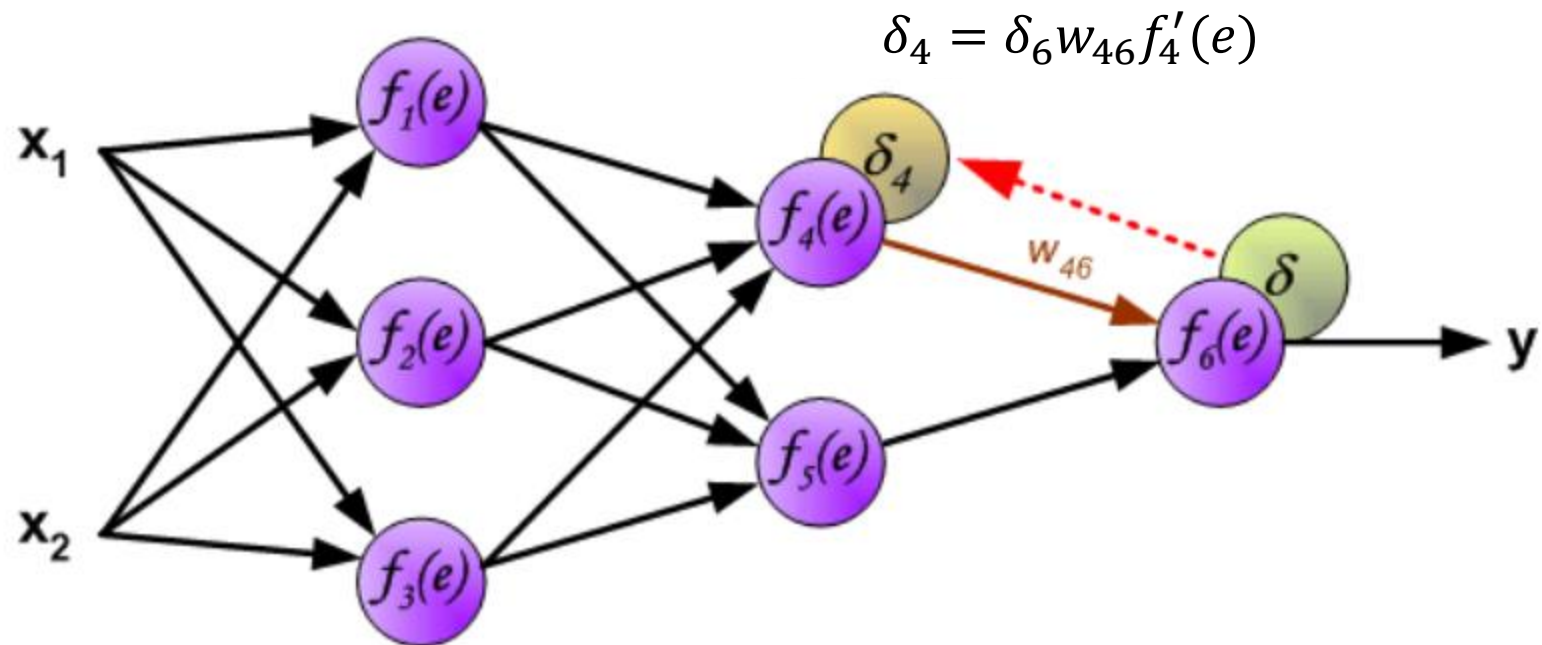


# BP Example

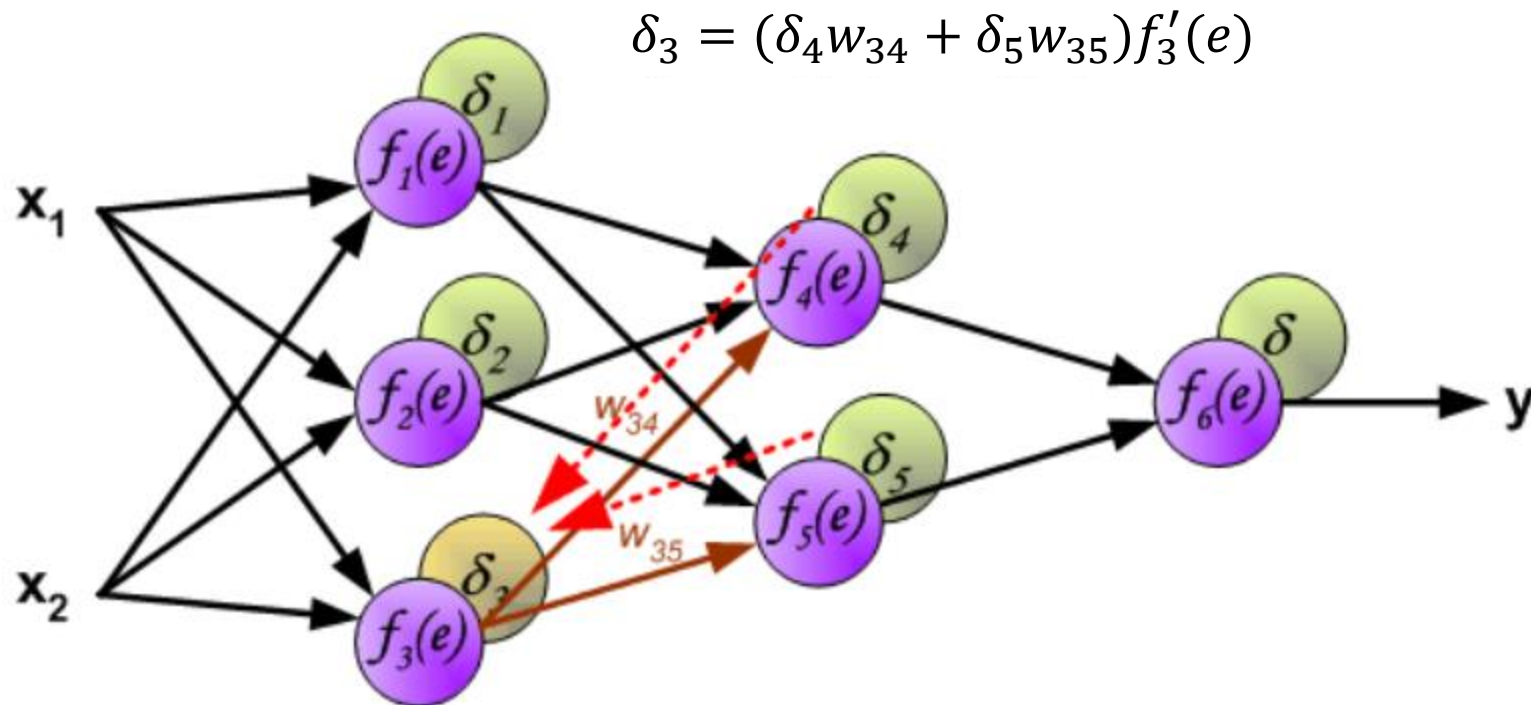
- The output signal of the network  $y$  is compared with the desired output value (the target).
- $f'_6(e)$  is the derivative for the function  $f_6(e)$ .
- Then, we can calculate the error signal  $\delta$  at the node  $N_6$ .



# BP Example (Cont'd)



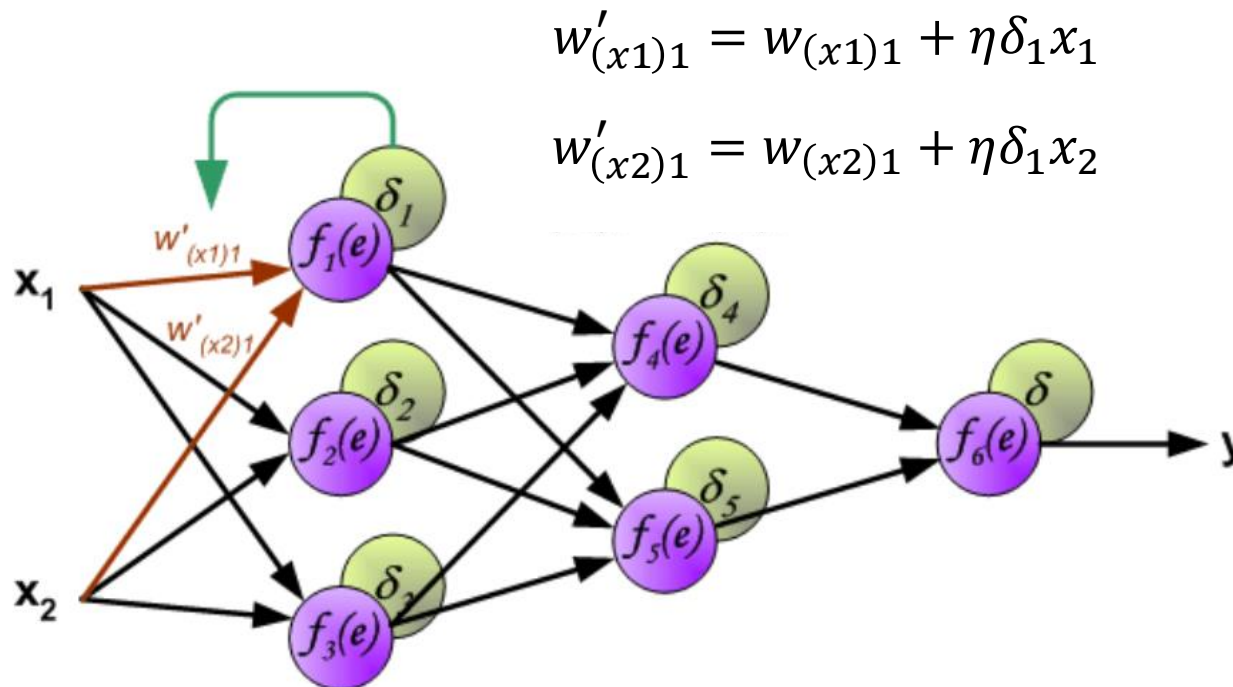
# BP Example (Cont'd)





# BP Example (Cont'd)

- ❖ After the error signal for each neuron is computed, the weights coefficients can be modified according to the update rule, with the learning rate  $\eta$ .



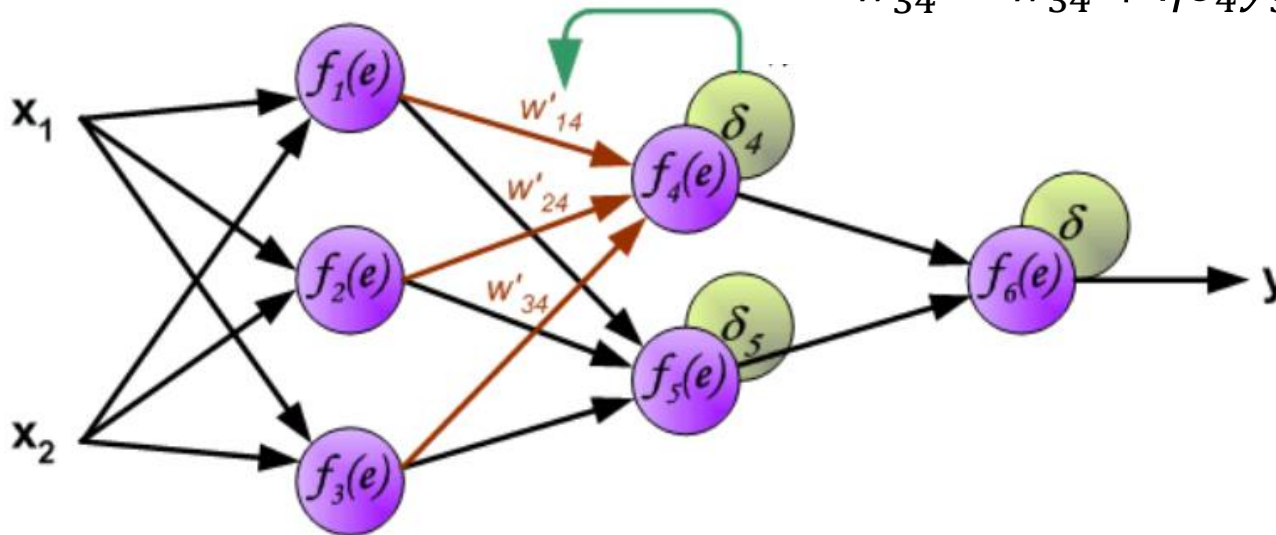
# BP Example (Cont'd)



$$w'_{14} = w_{14} + \eta \delta_4 y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 y_2$$

$$w'_{34} = w_{34} + \eta \delta_4 y_3$$

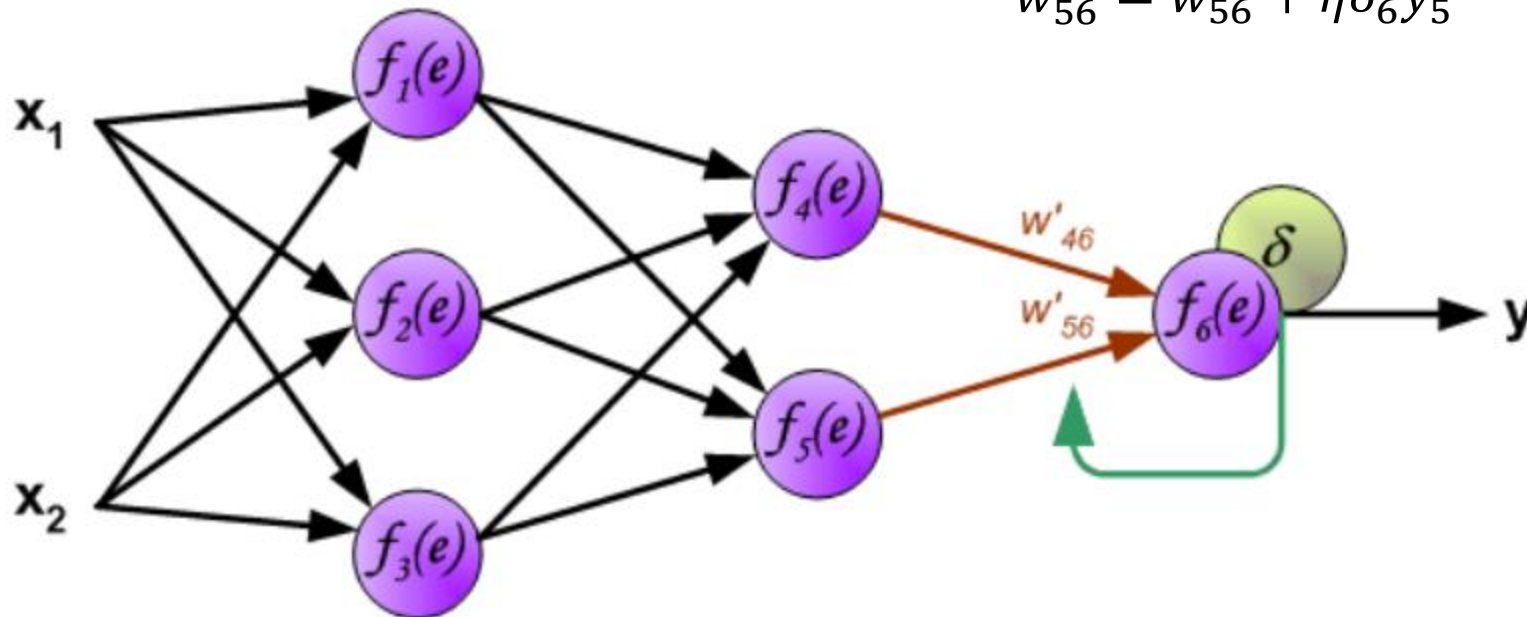


# BP Example (Cont'd)



$$w'_{46} = w_{46} + \eta \delta_6 y_4$$

$$w'_{56} = w_{56} + \eta \delta_6 y_5$$



## ❖ Two steps

- Forward: calculate the activation for each unit
- Backward: calculate **error signal** and update weights

## ❖ Intensive computation

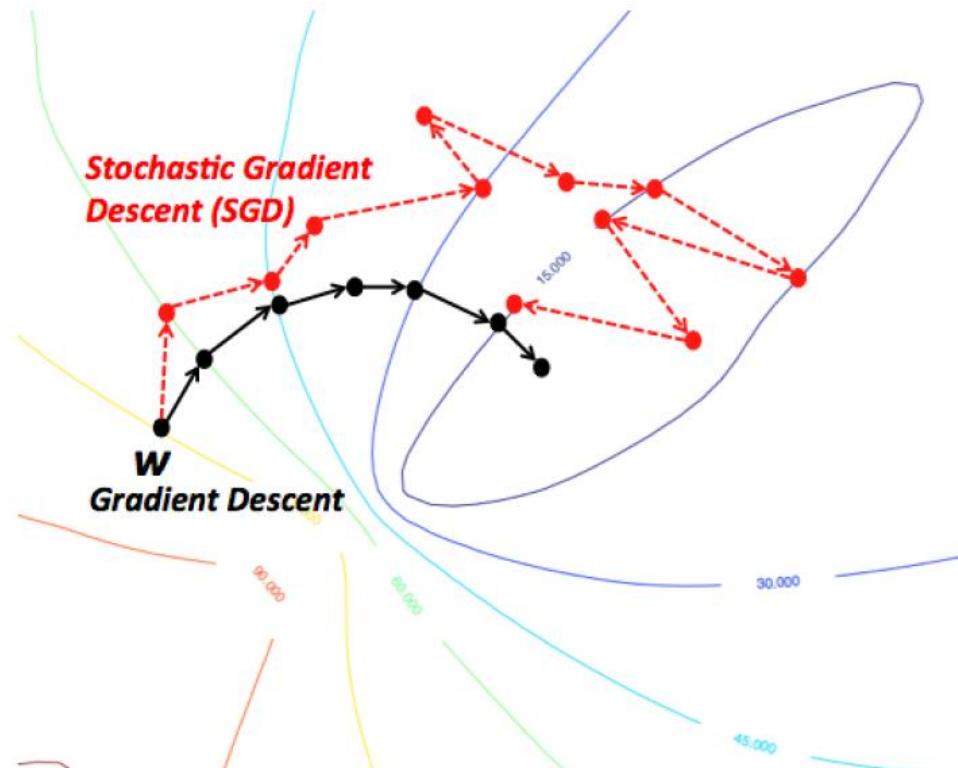
- Appropriate activation function to ease derivative calculation
  - $\sigma'(x) = \sigma(1 - \sigma)$
- Reuse common terms in partial derivative calculation
- Use **stochastic gradient descent (SGD)**
  - Evaluate the error overall whole dataset is expensive
  - Evaluate the error on a single data instance (more updates)

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \cdot \nabla E_i(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}^{(\tau)}}$$

# Stochastic Gradient Descent



MACQUARIE  
University



- ❖ Universal approximation theorem
  - MLP with a single hidden layer with a finite number of neurons can approximate any continuous functions
- ❖ Overfitting can happen when data is relatively small
  - Neural networks are prone to overfitting due to its ability
  - Countermeasures
    - Early stopping: stop when training error keep decreasing but validation error starts increasing
    - Regularization: add a penalty term to the cost function, e.g., squared sum of the weights, to avoid relatively high model complexity
    - Use fewer hidden units

## ❖ Pros

- The model is powerful with high accuracy and can be applied to complex non-linear problems.
- Can work with large-scale datasets (big data).
- Prediction is fast.
- No need for feature engineering in deep learning models.
- Now many deep learning frameworks, e.g., [PyTorch](#).

## ❖ Cons

- Model training is computation intensive, e.g., GPUs are often required for deep learning models.
- Data intensive, otherwise, overfitting is likely to occur
- Poor interpretability, i.e., difficult in explaining the prediction

## ❖ Artificial Neural Networks

- Basic Concepts
- Multi-Layer Perceptron

## ❖ Back Propagation

- Gradient Descent Method
- Error Back Propagation

## ❖ Practical



❖ `class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)`

- [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html#sklearn.neural\\_network.MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier)

- ❖ **hidden\_layer\_sizes:** The  $i^{\text{th}}$  element represents the number of neurons in the  $i^{\text{th}}$  hidden layer.
- ❖ **activation:** {'identity', 'logistic', 'tanh', 'relu'}
  - Activation function for the hidden layer
  - 'identity', useful to implement linear bottleneck,  $f(x) = x$ .
  - 'logistic', the logistic sigmoid function,  $f(x) = \frac{1}{1+e^{-x}}$ .
  - 'tanh', hyperbolic tan function,  $f(x) = \tanh(x)$ .
  - 'relu', rectified linear unit function,  $f(x) = \max(0, x)$ .

## ❖ **solver**: {'lbfgs', 'sgd', 'adam'}

- The solver for weight optimization.
- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba.
- The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

## ❖ **alpha:** Strength of the L2 regularization term.

- Alpha is a parameter for regularization term, a.k.a. penalty term, that combats overfitting by constraining the size of the weights.

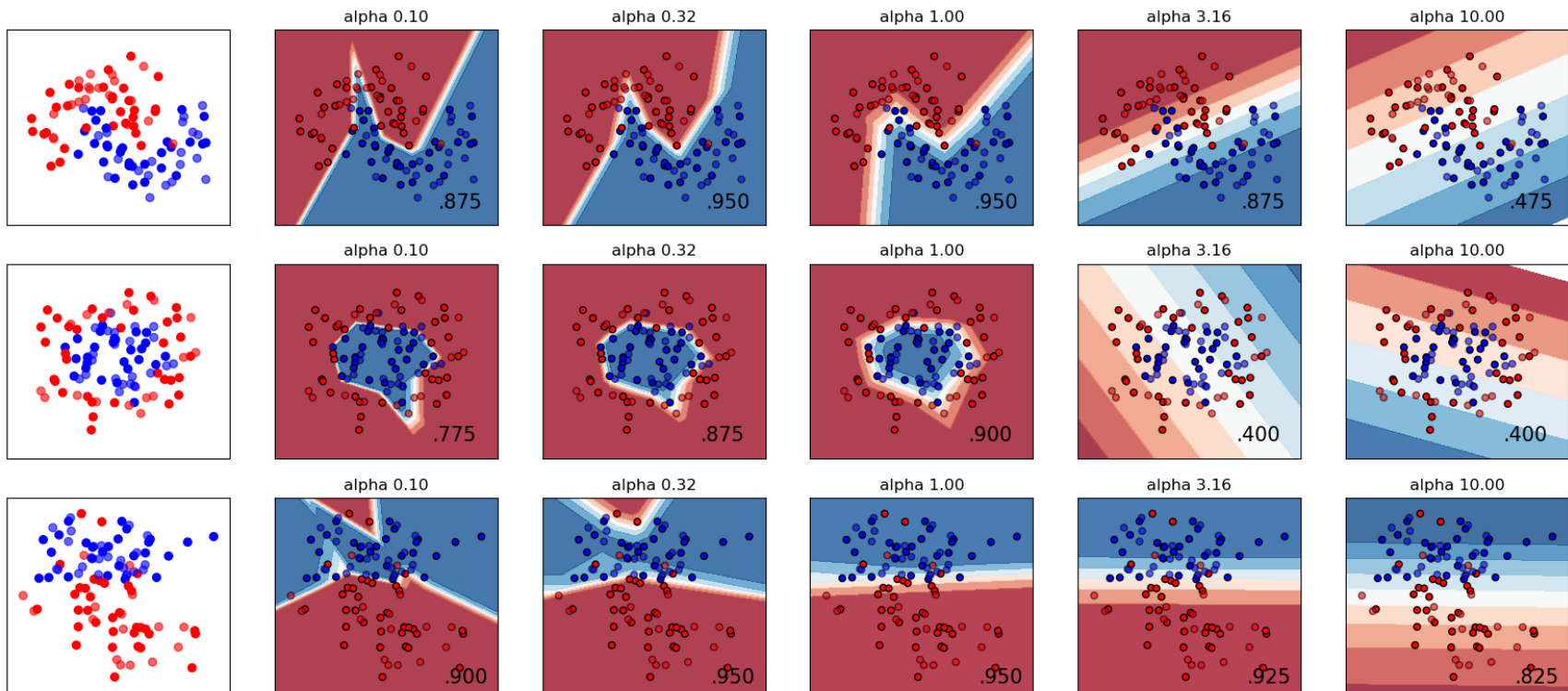
$$E(\mathbf{w}) + \alpha ||\mathbf{w}||_2^2$$

- The L2 regularization term is divided by the sample size when added to the loss.
- Increasing alpha may mitigate overfitting by encouraging smaller weights, resulting in a smooth decision boundary.
- Decreasing alpha may mitigate underfitting by encouraging larger weights, resulting in a complicated decision boundary.

# MLPClassifier (Cont'd)



❖ **alpha**: Strength of the L2 regularization term.



- ❖ **batch\_size**: Size of minibatches for stochastic optimizers.
- ❖ **learning\_rate**: {'constant', 'invscaling', 'adaptive'}
  - Learning rate schedule for weight updates.
- ❖ **max\_iter**: Maximum number of iterations. a solver iterates until convergence or this number of iterations.
- ❖ **tol**: Tolerance for the optimization. Used to consider whether convergence is reached and training stops.
- ❖ **early\_stopping**: Whether to use early stopping to terminate training when validation score is not improving. Only effective when solver='sgd' or 'adam'.

- ❖ **n\_features\_in\_**: Number of features seen during fit().
- ❖ **n\_outputs\_**: Number of outputs.
- ❖ **coefs\_**: *list of shape (n\_layers - 1,)*
  - The  $i^{\text{th}}$  element in the list represents the weight matrix corresponding to layer  $i$ .
- ❖ **intercepts\_**: *list of shape (n\_layers - 1,)*
  - The  $i^{\text{th}}$  element in the list represents the bias vector corresponding to layer  $i + 1$ .
- ❖ **loss\_curve\_**: *list of shape (n\_iter\_,)*
  - The  $i^{\text{th}}$  element in the list represents the loss at the  $i^{\text{th}}$  iteration. Not for the solver 'lbfgs'

# MLPClassifier Demo

---



MACQUARIE  
University

- ❖ [https://colab.research.google.com/drive/1uF3er2jbwYhQe59S9KKb9Gdzt\\_va9pbY?usp=sharing](https://colab.research.google.com/drive/1uF3er2jbwYhQe59S9KKb9Gdzt_va9pbY?usp=sharing)



- ❖ Biological neuron and artificial neural networks
- ❖ Three basic layers
- ❖ Different types of neural networks
- ❖ Relationships to other ML models
- ❖ Feed forward neural networks and MLP
- ❖ Different activation functions and cost functions
- ❖ Gradient descent and derivative chain rule
- ❖ Error back propagation
- ❖ Overfitting issue and countermeasures