Neural Network

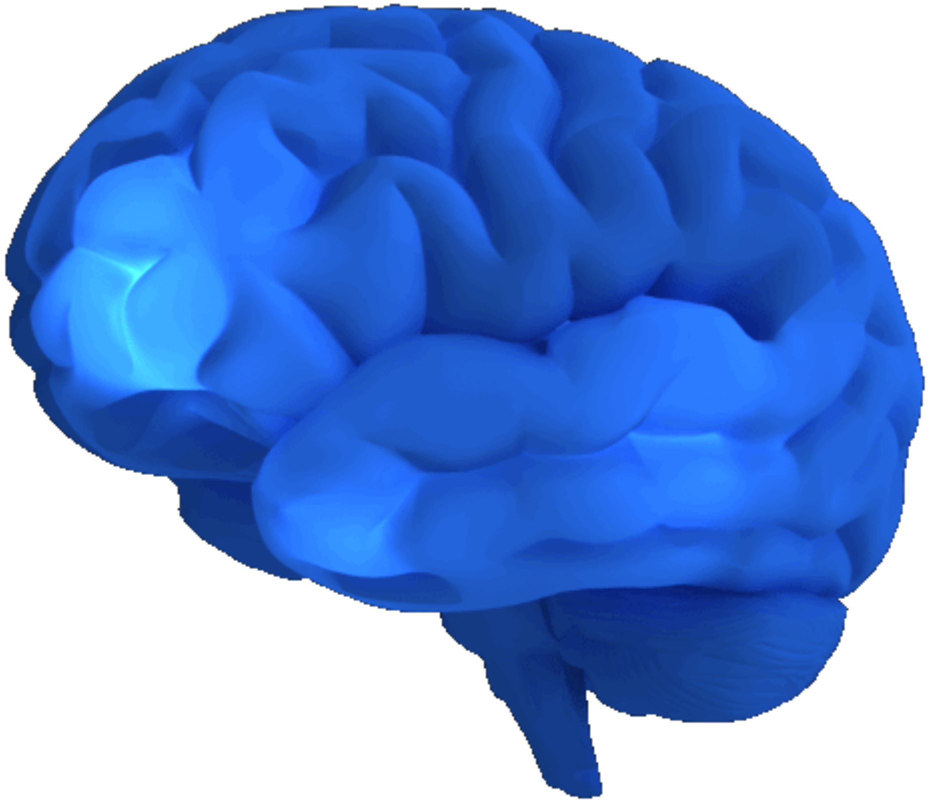
Anahita Zarei, Ph.D.

Overview

- Reading: Section 1.1.2 and e-chapter 7 from Learning from Data

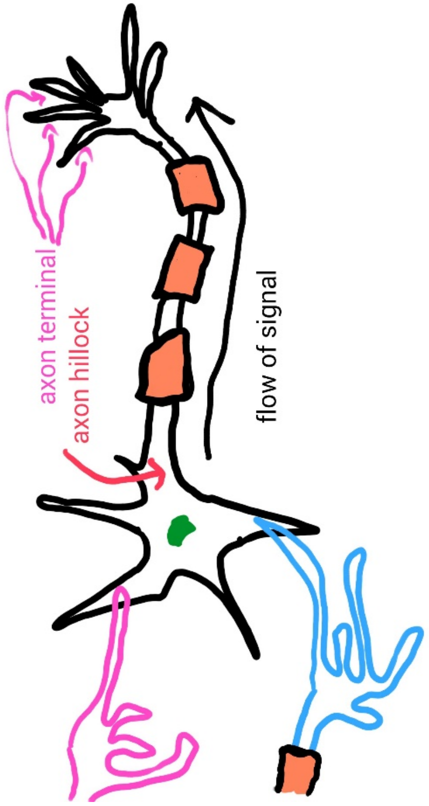
Why Neural Network?

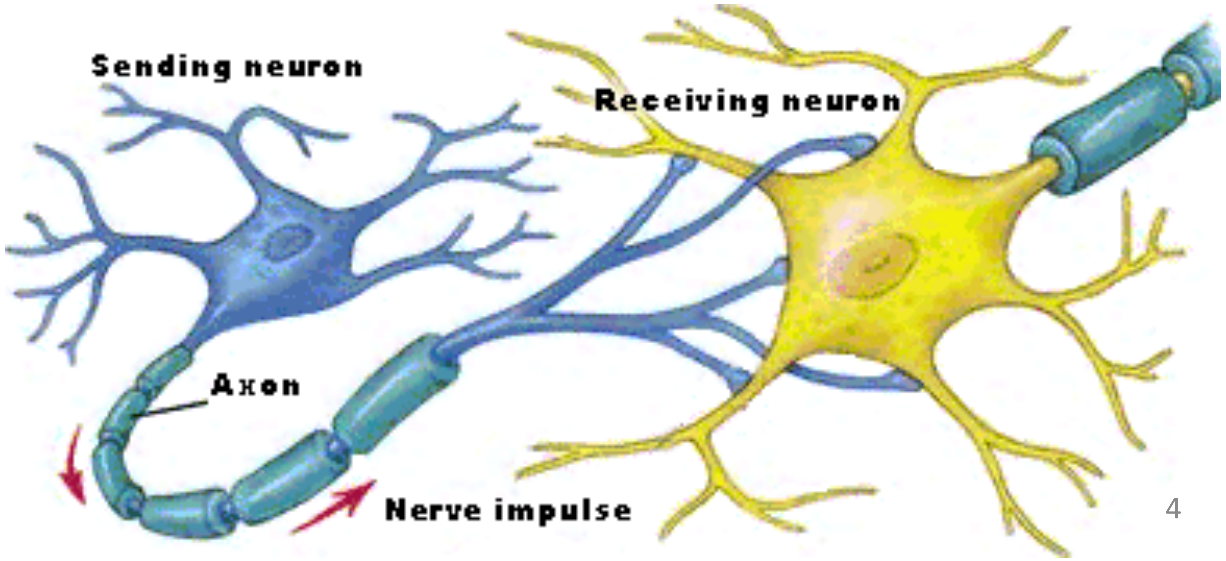
- It’s hip!
- It can learn complex functions from examples based on generalized training data in supervised learning.
- It can learn by extracting patterns and finding the underlying structure of the data in unsupervised learning.



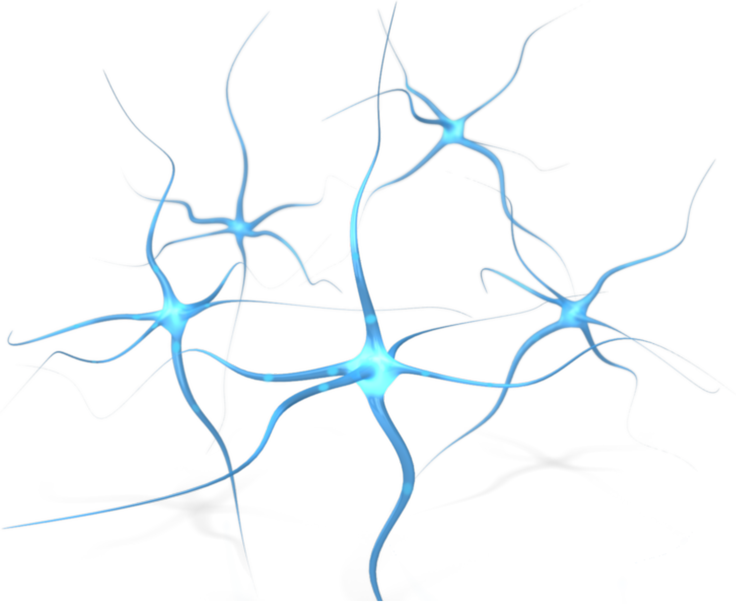
Human Neural Network

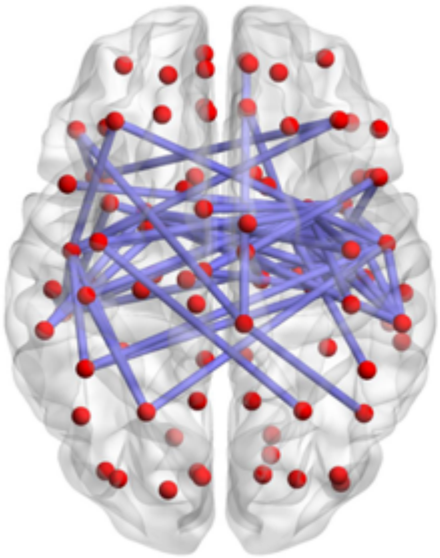
- Neuron is a fundamental cellular unit of the brain's nervous system.
- The biological neuron consists of four main parts:
    - Dendrites: resemble roots of a tree and act as input channels that receive impulses through synapses of other neurons.
    - Cell Body: processes (integrates) the signals received by the dendrites. If the combined input signal is strong enough the neuron "fires" .
    - Axon: resembles tree trunk. It conducts electrical impulses and transmit information to neighboring neurons.
    - Synapse: are gaps between neurons where neurons communicate with another. This junction is filled with neurotransmitter fluid.





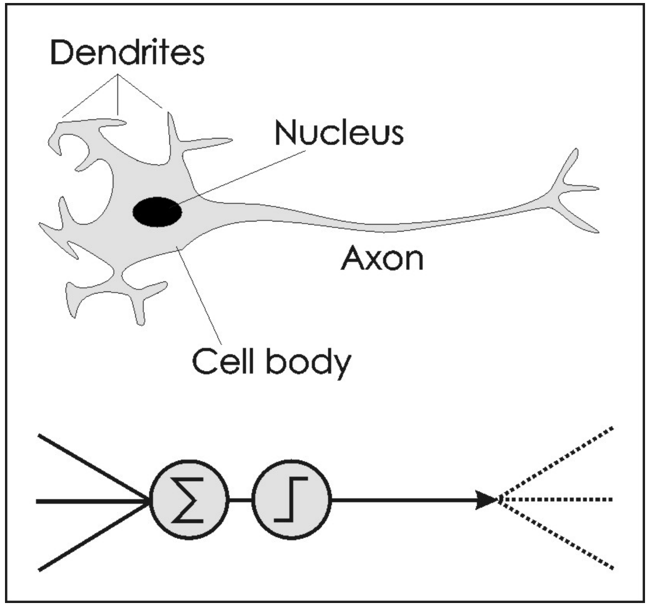
Human Neural Network

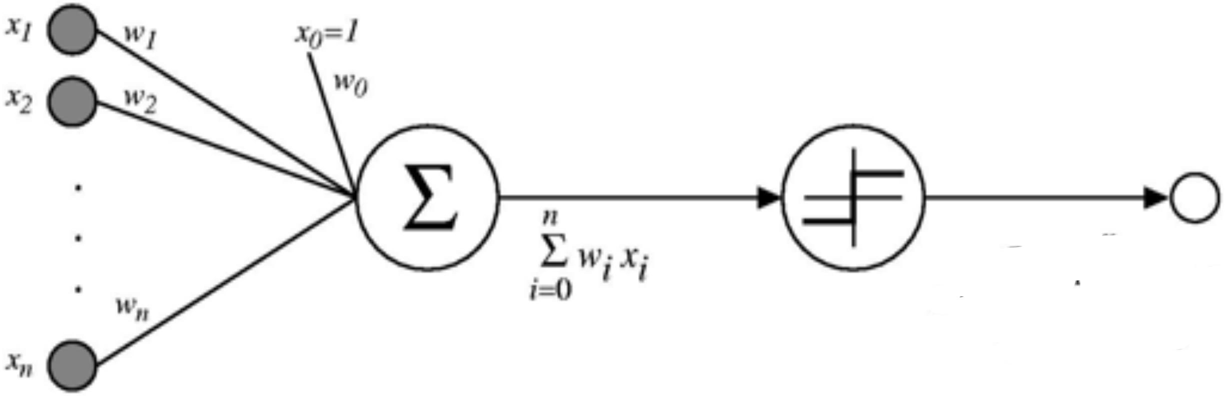




Artificial Neuron VS. Biological Neuron

Artificial Neuron	Biological Neuron
Lines that connect the input features to the summation processing element.	Dendrites
Processing element that has two parts: summation and the nonlinearity that decides if there's an action potential or not.	Cell Body
The output of a neuron that is used by other neurons	Axon





Single Neuron: Perceptron
Example: Approve or Deny Credit

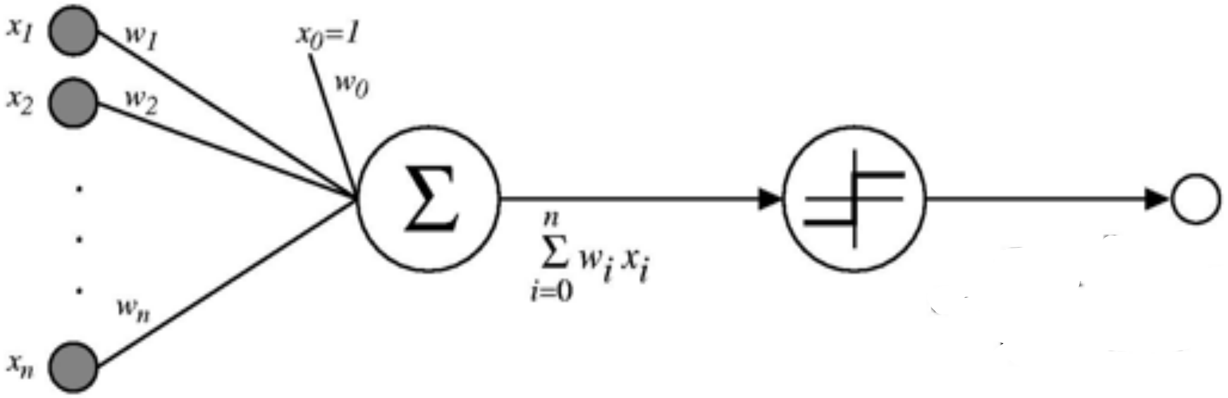
This guy is my hero!

Single Neuron: Perceptron
Example: Approve or Deny Credit

For input $\mathbf{x} = (x_1, \cdots, x_d)$

Approve credit if $\sum_{i=1}^{d} w_i x_i >$ threshold

Deny credit if $\sum_{i=1}^{d} w_i x_i <$ threshold



$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

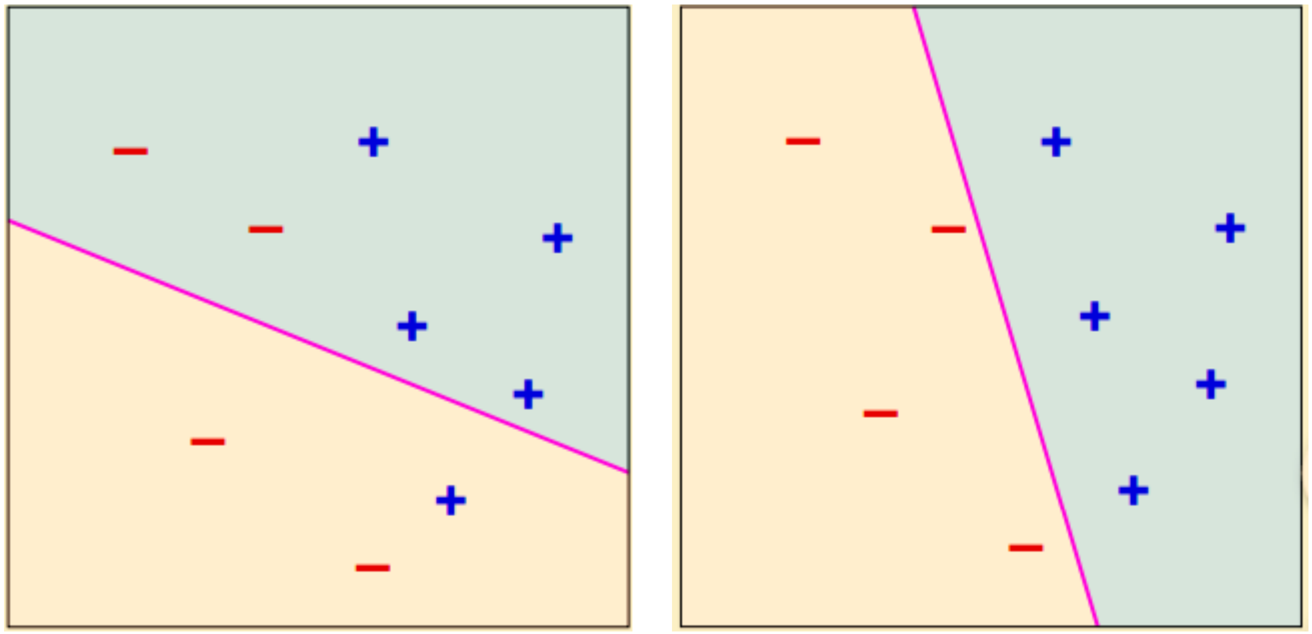
$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i \, x_i\right) + \quad w_0 \quad\right)$$

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=0}^{d} w_i \, x_i\right)$$

Perceptron Decision Boundary

If vector $\vec{X}$ is a row vector and vector $\vec{w}$ is a column vector then $h(\vec{X})$ can be expressed in terms of the dot products of these two vectors as follows:

$$h(\vec{X}) = sign(\vec{X}\vec{w})$$



Perceptron Learning Algorithm (PLA)

- PLA Pseudo Code:

a) Choose a misclassified point $(\overrightarrow{X_n}, y_n)$ from the following training set: $(\overrightarrow{X_1}, y_1) \ldots (\overrightarrow{X_N}, y_N)$ where vector

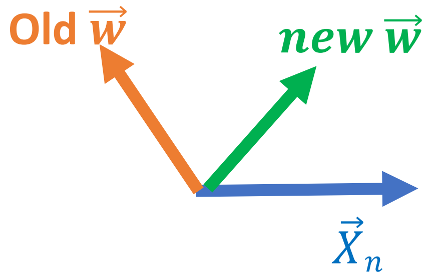
$$\vec{X} = (x_1, \ldots, x_d)$$

b) Update the weight vector with the following rule

$$\boldsymbol{new\ \overrightarrow{w} = old\ \overrightarrow{w} + y_n \overrightarrow{X_n}}$$

c) Repeat the above process until all points are correctly classified.

Rational Behind PLA Algorithm

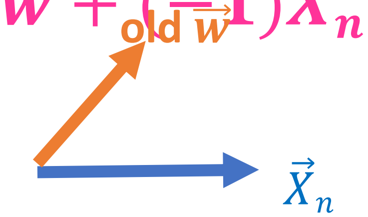


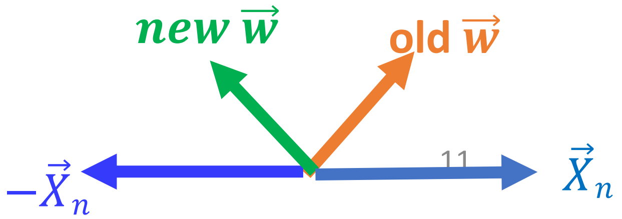
- Case 1: If the weight and X vector obtuse angle then the dot product will give you a negative value.
- $y_n = +1 \quad h(\vec{X}_n) = -1$

$$\boldsymbol{new\ \vec{w} = old\ \vec{w} + y_n\overrightarrow{X_n}}$$
$$\boldsymbol{new\ \vec{w} = old\ \vec{w} + (+1)\overrightarrow{X_n}}$$

- Case 2: If the weight and X vector acute angle then the dot product will give you a positive value.
- $y_n = -1 \quad h(\vec{X}_n) = +1$

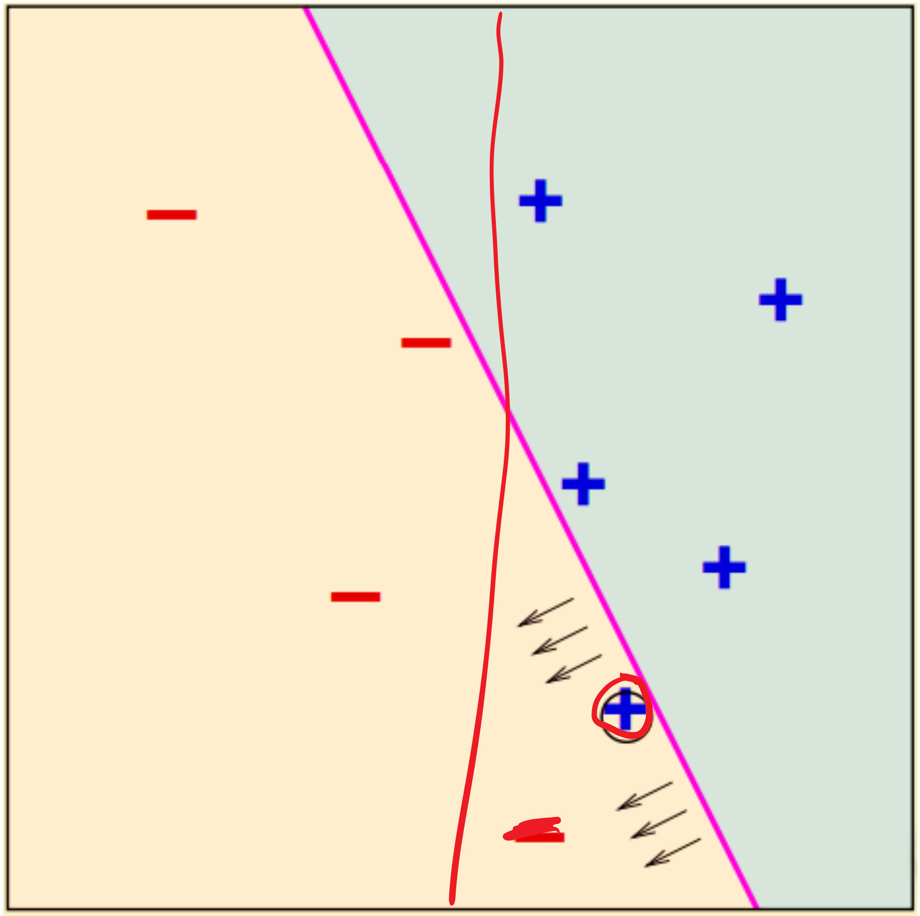
$$\boldsymbol{new\ \vec{w} = old\ \vec{w} + y_n\overrightarrow{X_n}}$$
$$\boldsymbol{new\ \vec{w} = old\ \vec{w} + (-1)\overrightarrow{X_n}}$$





PLA- cont.

- The update rule moves the boundary in the direction of classifying point X correctly, as showed in the figure.
- PLA considers only one training example at a time. In this process, it may misclassify some of the previously correctly classified points.
- However, it's proved that there's a guarantee that PLA converges to the correct boundary decision.
- Does this mean that this hypothesis will also be successful in classifying new data points that were not in the training set?

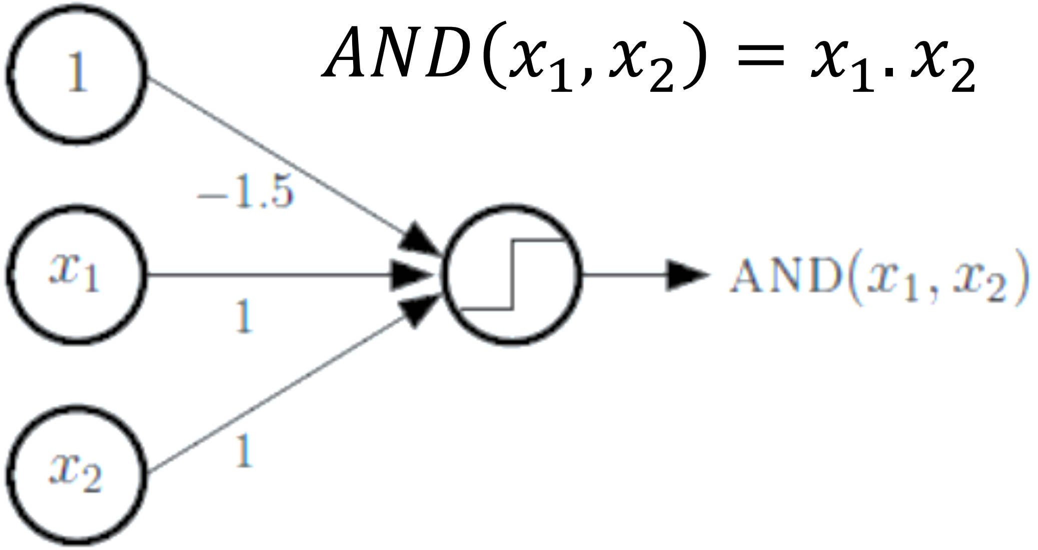


Example: Perceptron Implementation of an AND operator

- Recall the boundary equation:
- If $w_1x_1+w_2x_2$ > Threshold, System fires (y=1)
- If $w_1x_1+w_2x_2$ < Threshold, System doesn't fire (y=-1)
- If $w_1 = w_2 = 1$, what threshold value implements the AND operator?
  - -1-1 <Threshold => -2 < Threshold
  - 1+-1<Threshold => 0 < Threshold
  - 1+1 > Threshold => 2 >Threshold
- Any value between 0 and 2 would work for Threshold!
- For example if Threshold =1.5 then $\mathbf{w_0 = -1.5}$
- $AND(x_1, x_2) = sign(x_1 + x_2 - 1.5)$.

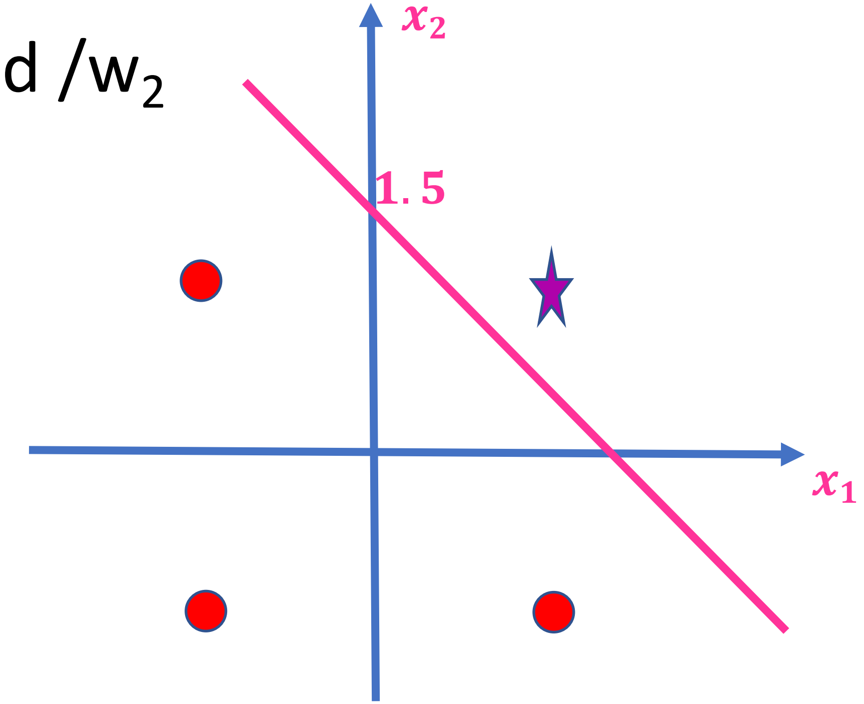
x_1	x_2	$y = AND(x_1, x_2)=x_1. x_2$
false (-1)	false (-1)	false (-1)
false (-1)	true (1)	false (-1)
true (1)	false (-1)	false (-1)
true (1)	true (1)	true (1)

$$AND(x_1, x_2) = x_1 . x_2$$



Example – cont.
Boundary Decision

- $w_1 x_1 + w_2 x_2$ = Threshold, Dividing Line
- $x_2 = \frac{-w_1}{w_2} x_1 + \frac{Threshold}{w_2}$
- Slope = $-w_1/w_2$
- y-intercept = Threshold $/w_2$
- $x_2 = -x_1 + 1.5$



x_1	x_2	y = AND(x_1, x_2)
false (-1)	false (-1)	false (-1)
false (-1)	true (1)	false (-1)
true (1)	false (-1)	false (-1)
true (1)	true (1)	true (1)

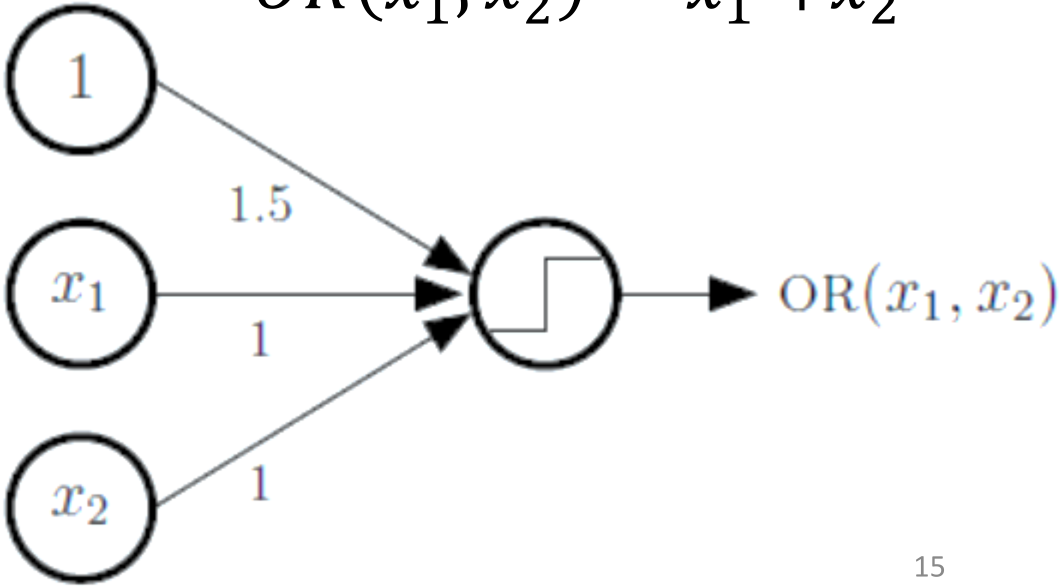
$$AND(x_1, x_2) = x_1 . x_2$$

Example: Perceptron Implementation of an OR operator

x_1	x_2	$y = OR(x_1, x_2) = x_1+x_2$
false (-1)	false (-1)	false (-1)
false (-1)	true (1)	true (1)
true (1)	false (-1)	true (1)
true (1)	true (1)	true (1)

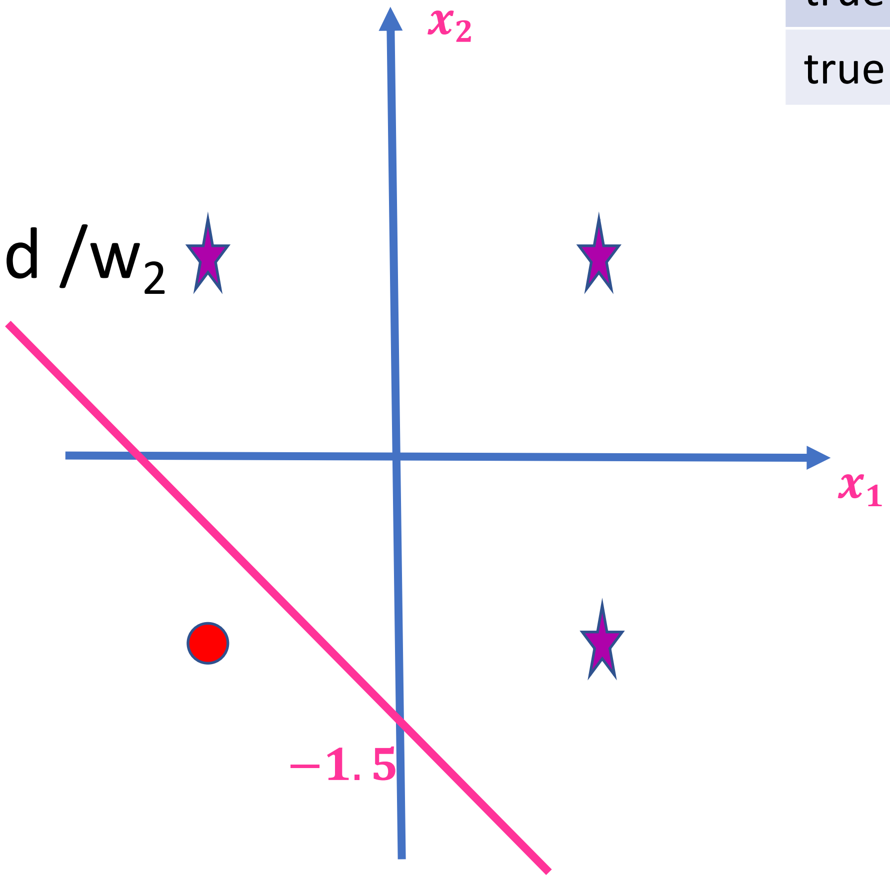
- Recall the boundary equation:
- $w_1x_1+w_2x_2$ > Threshold System fires (y=1)
- $w_1x_1+w_2x_2$ < Threshold System doesn't fire (y=-1)
- If $w_1 = w_2 = 1$, what threshold value implements the OR operator?
  - -1-1<Threshold => -2 < Threshold
  - 1-1 >Threshold => 0 > Threshold
  - 1+1 > Threshold => 2>Threshold
- Any value between 0 and -2 would work for Threshold!
- For example if Threshold=-1.5 then $\mathbf{w_0 = 1.5}$
- $OR(x_1, x_2) = sign(x_1 + x_2 + 1.5)$.

$$OR(x_1, x_2) = x_1 + x_2$$



Example – cont.
Boundary Decision

- $w_1x_1+w_2x_2$ = Threshold, Dividing Line
- $x_2 = \frac{-w_1}{w_2}x_1 + \frac{T}{w_2}$
- Slope = $-w_{1/}w_2$
- y-intercept = Threshold /w_2
- $x_2 = -x_1 - 1.5$

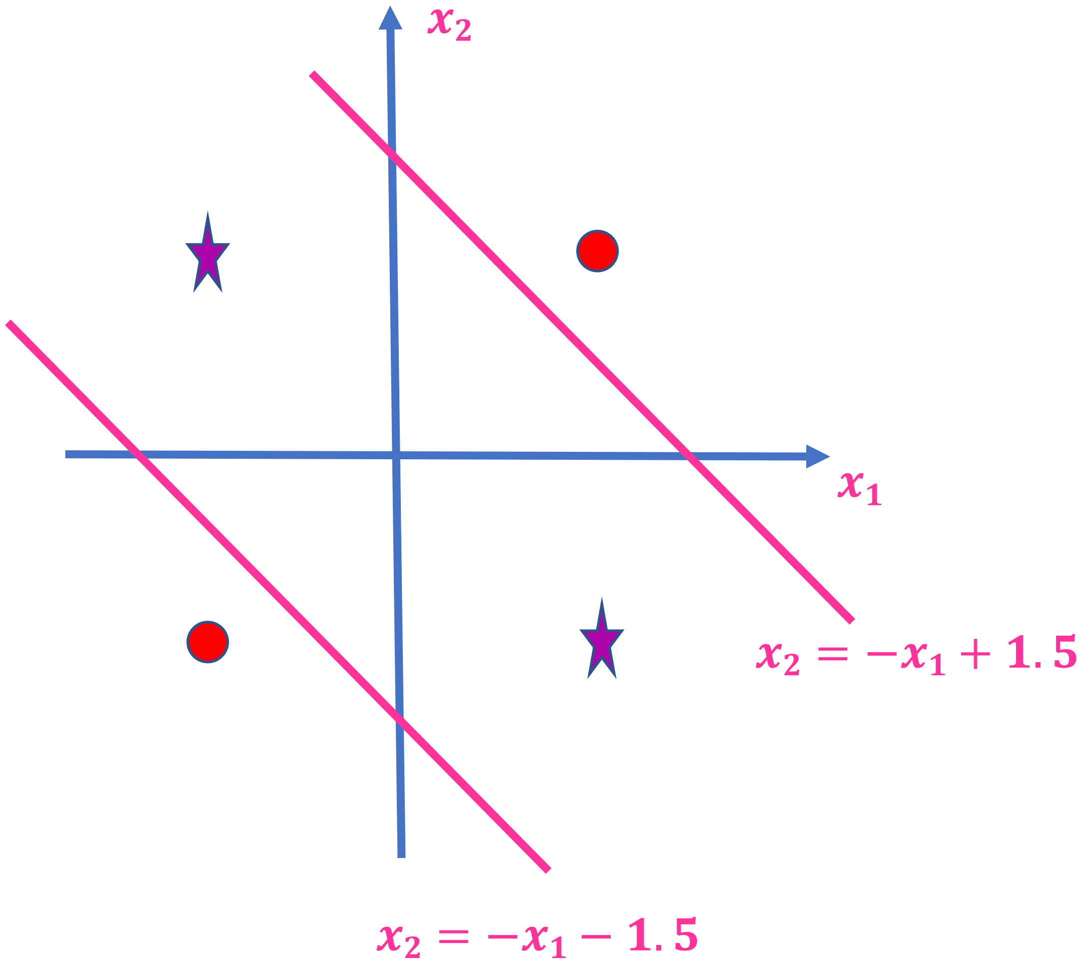


x_1	x_2	y = OR(x_1, x_2)
false (-1)	false (-1)	false (-1)
false (-1)	true (1)	true (1)
true (1)	false (-1)	true (1)
true (1)	true (1)	true (1)

$$OR(x_1, x_2) = x_1 + x_2$$

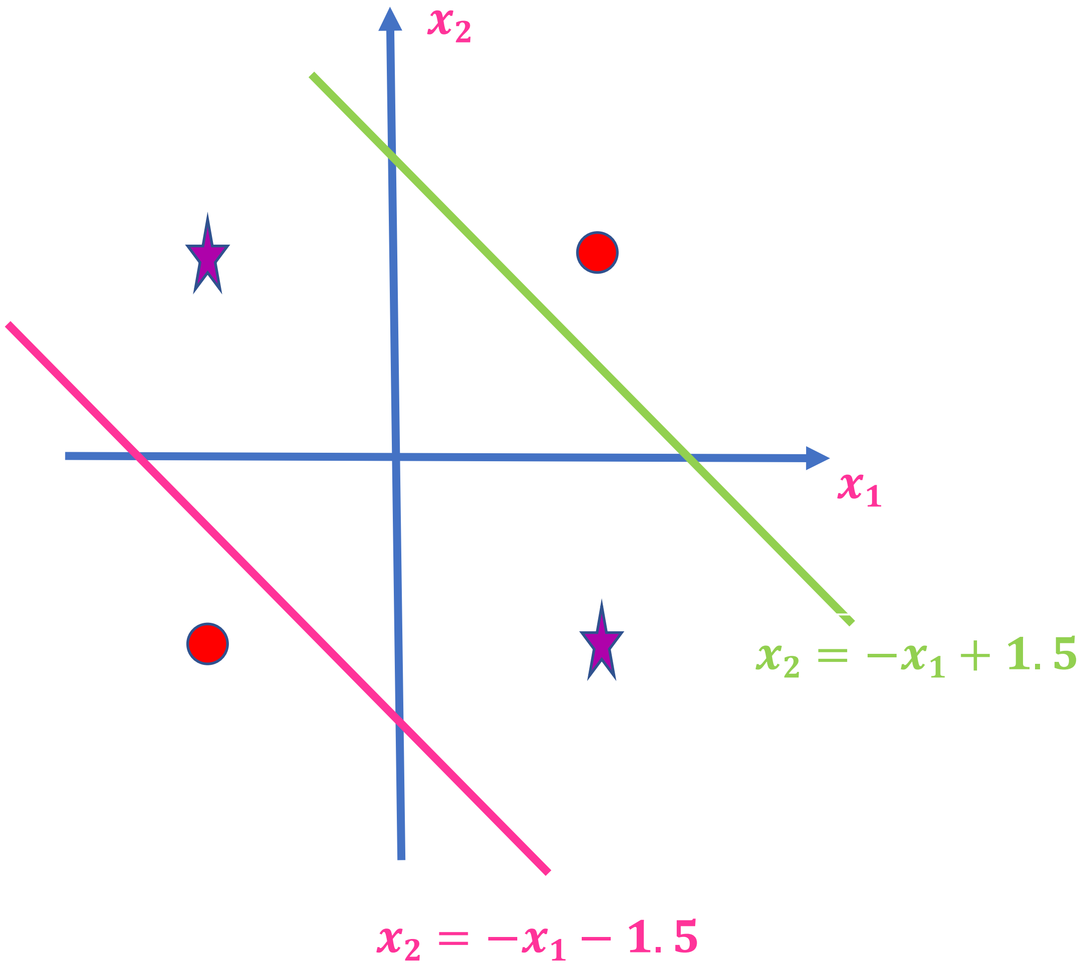
Example: Perceptron Implementation of an XOR operator?

x_1	x_2	$y = XOR(x_1, x_2)$
false (-1)	false (-1)	false (-1)
false (-1)	true (1)	true (1)
true (1)	false (-1)	true (1)
true (1)	true (1)	false (-1)



Example – cont.

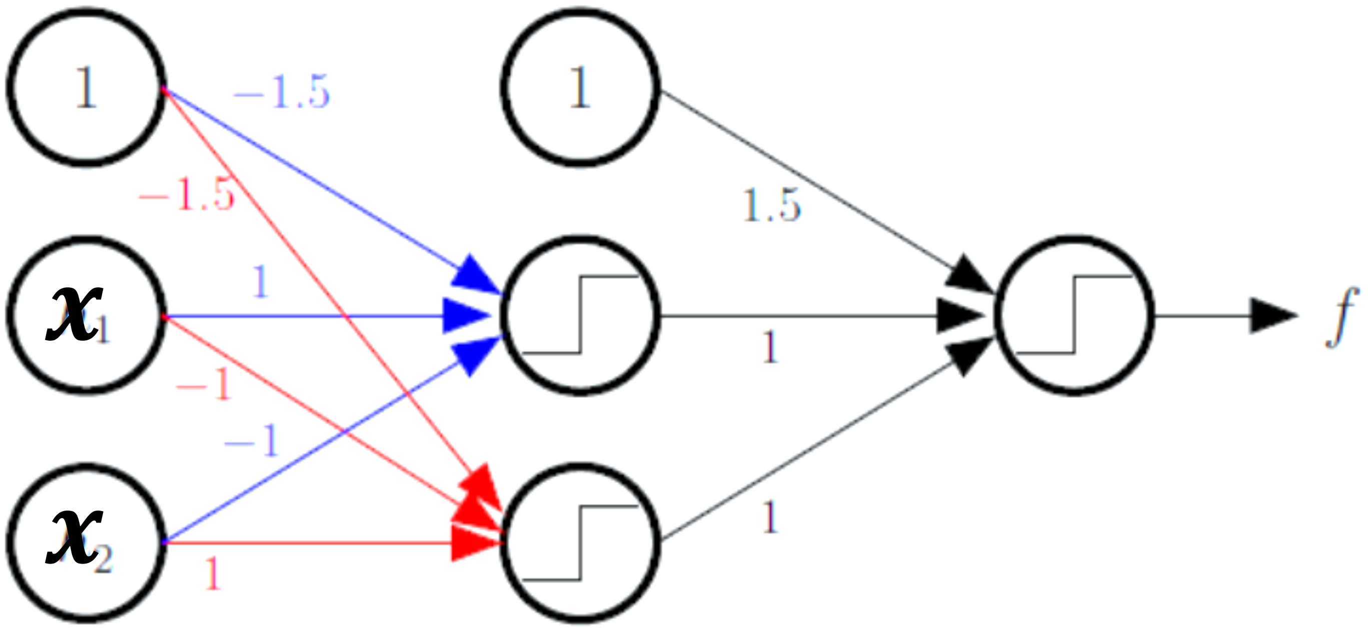
- These 2 lines will classify the points correctly.
- E.g. let $(x_1, x_2) = (-1, +1)$
- $\boldsymbol{x_1 + x_2 > -1.5}$ => (-1,+1) is above this line.
- $\boldsymbol{x_1 + x_2 < +1.5}$ => (-1,+1) is below this line.
- Intersection of the two regions will be the area between the 2 lines.



Combining Perceptrons to Achieve XOR

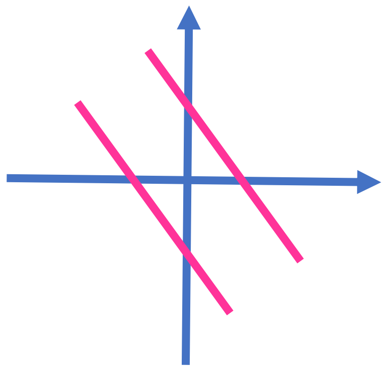
x_1	x_2	$y = XOR(x_1, x_2)$
false (-1)	false (-1)	false (-1)
false (-1)	true (1)	true (1)
true (1)	false (-1)	true (1)
true (1)	true (1)	false (-1)

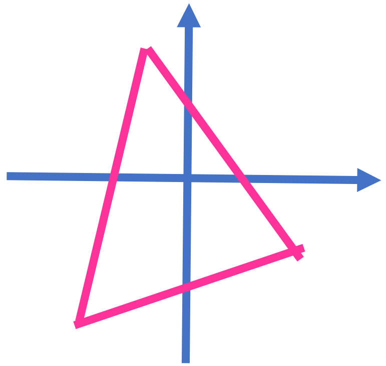
$$XOR(x_1, x_2) = x_1\overline{x_2} + \overline{x_1}x_2$$

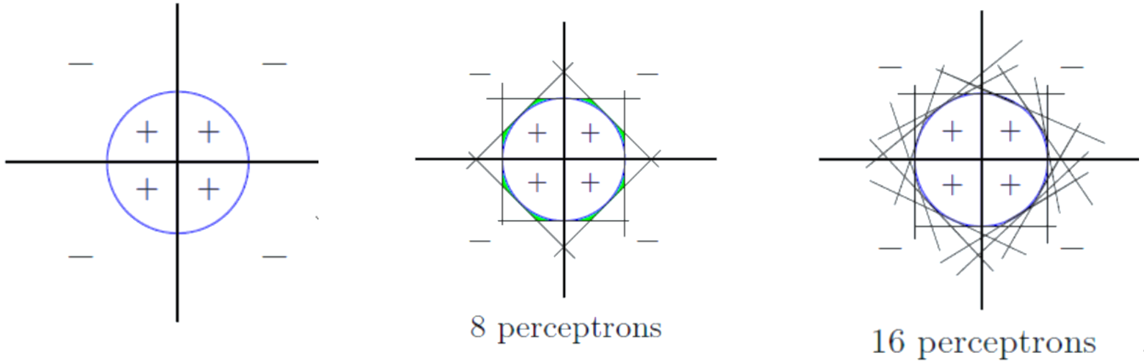


What can Multiple Layer Perceptron (MLP) learn?

- Use of 2 processing elements (excluding bias) in the hidden layer provides double division of the plane.
- Use of 3 hidden neurons (excluding bias) in the middle layer subdivides the plane with three lines, producing a triangular closed region.
- Use of additional neurons allows us to generate virtually any type of enclosed area from polygon to an approximation of a circle.

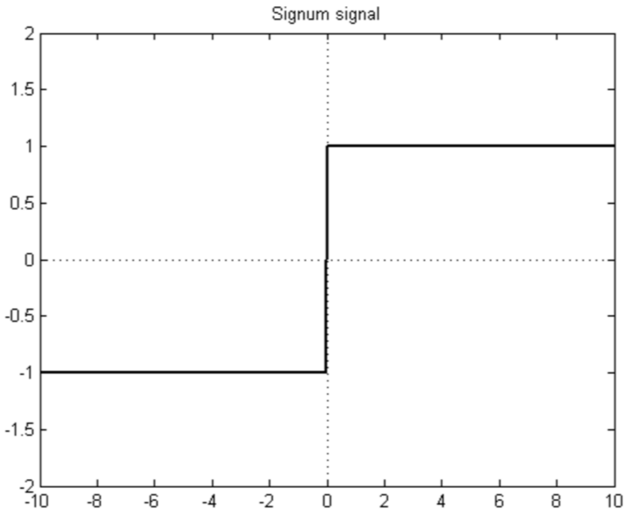


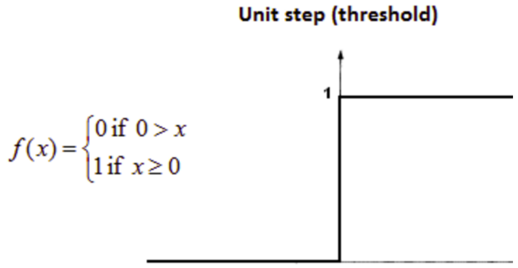


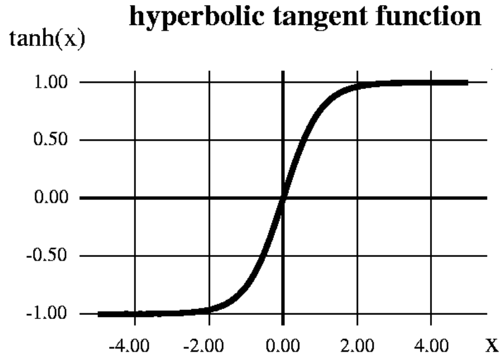


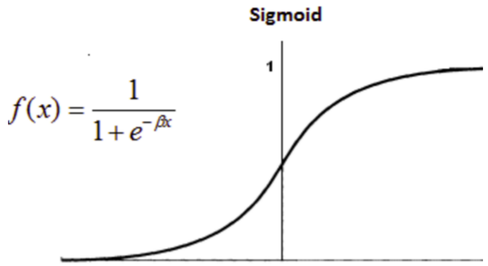
Soft Threshold Activation Function

- Using hard threshold activation functions such as step, or sign are ok for a single perceptron, however it’ll be a difficult combinatorial optimization problem.
- A smooth, differentiable approximation to step or sign will allow us to use analytic methods such as GD rather than purely combinatorial methods, to find the optimal weights.
- We therefore approximate, sign function by using the tanh function. (step can be approximated by sigmoid)
- Note that tanh is in between linear and the hard threshold: nearly linear for $x \approx 0$ and nearly ± 1 for $|x|$ large.





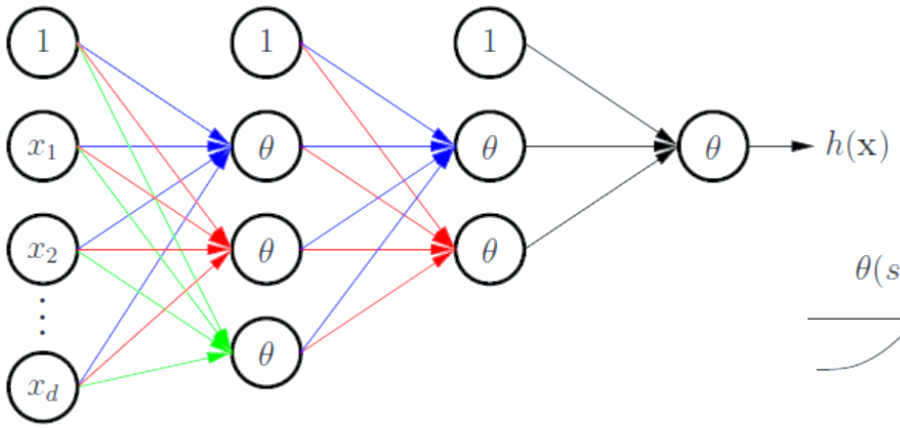


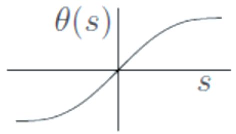


$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Neural Network

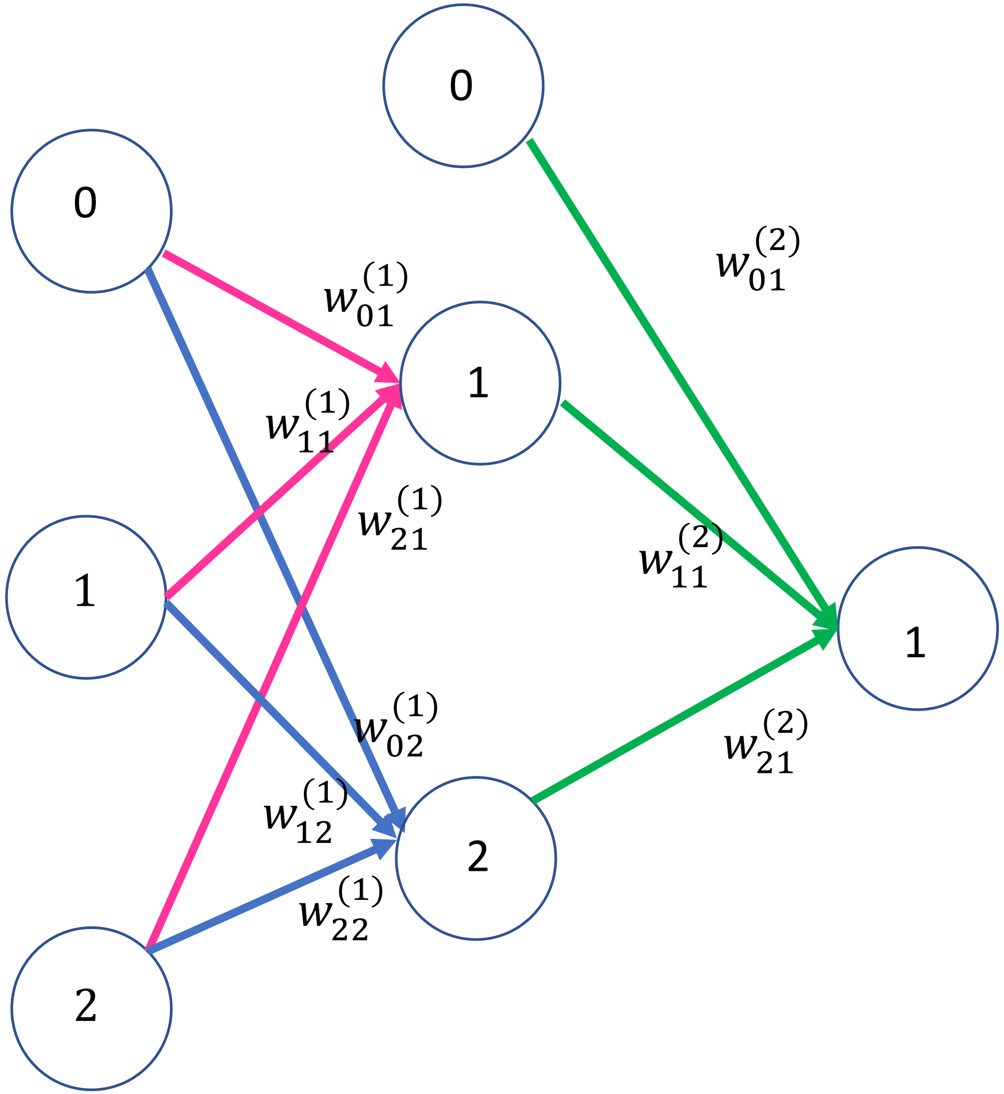
- The neurons are organized into a sequence of layers:
  - Input Layer: It presents data to the network. (It's not a computing layer, because the nodes have no input weights and activation function).
  - Hidden Layer: It's the intermediate layer and has no connections to the outside world.
  - Output Layer: It's the last layer that presents the output response to a given input.
- A ***feedforward*** neural network is fully connected ; there are no lateral connections between neurons in a given layer and none back to previous layers.





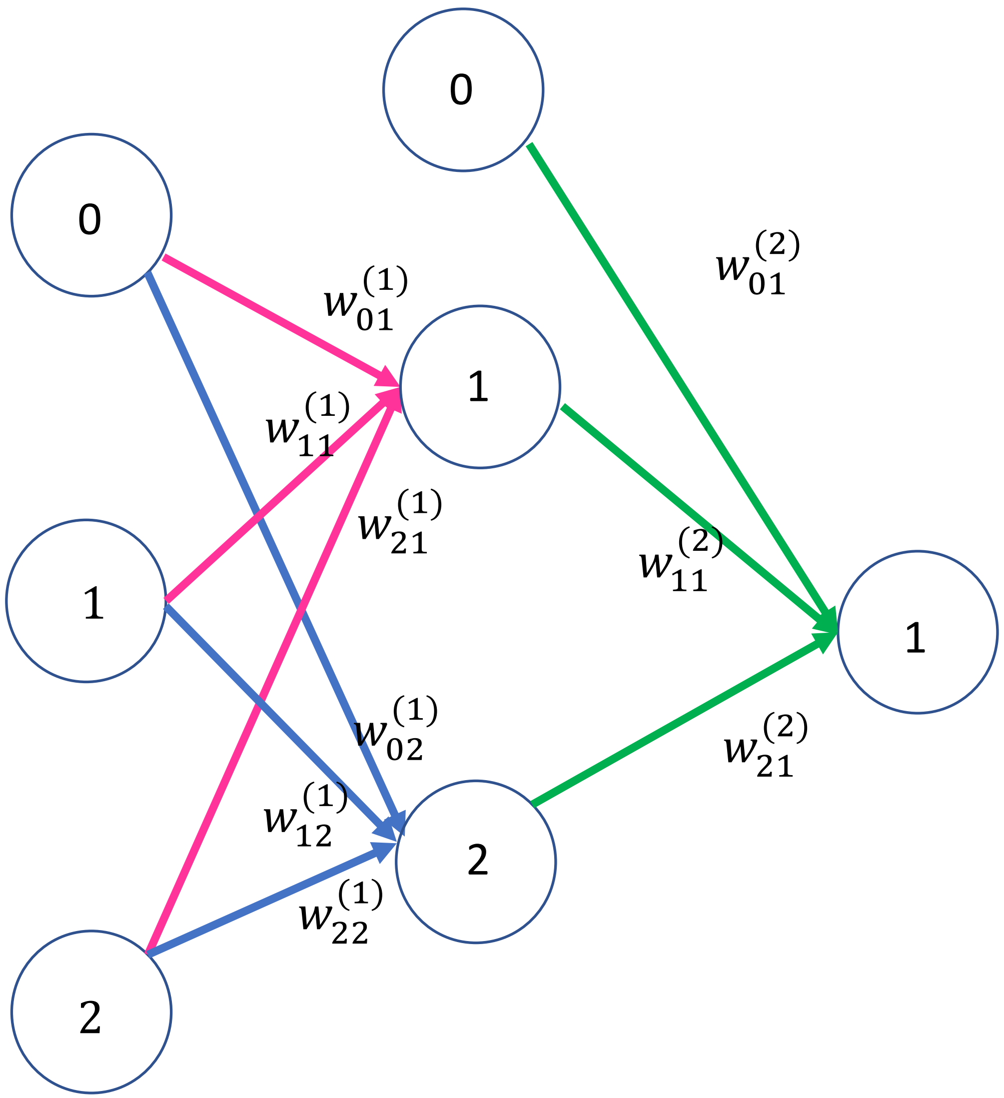
Weights

- Each of the connections between neurons has an adjustable weight.
- Each is designated by symbol $w_{ij}^{(l)}$ where i, j, and l denote starting neuron, ending neuron, and layer number, respectively.



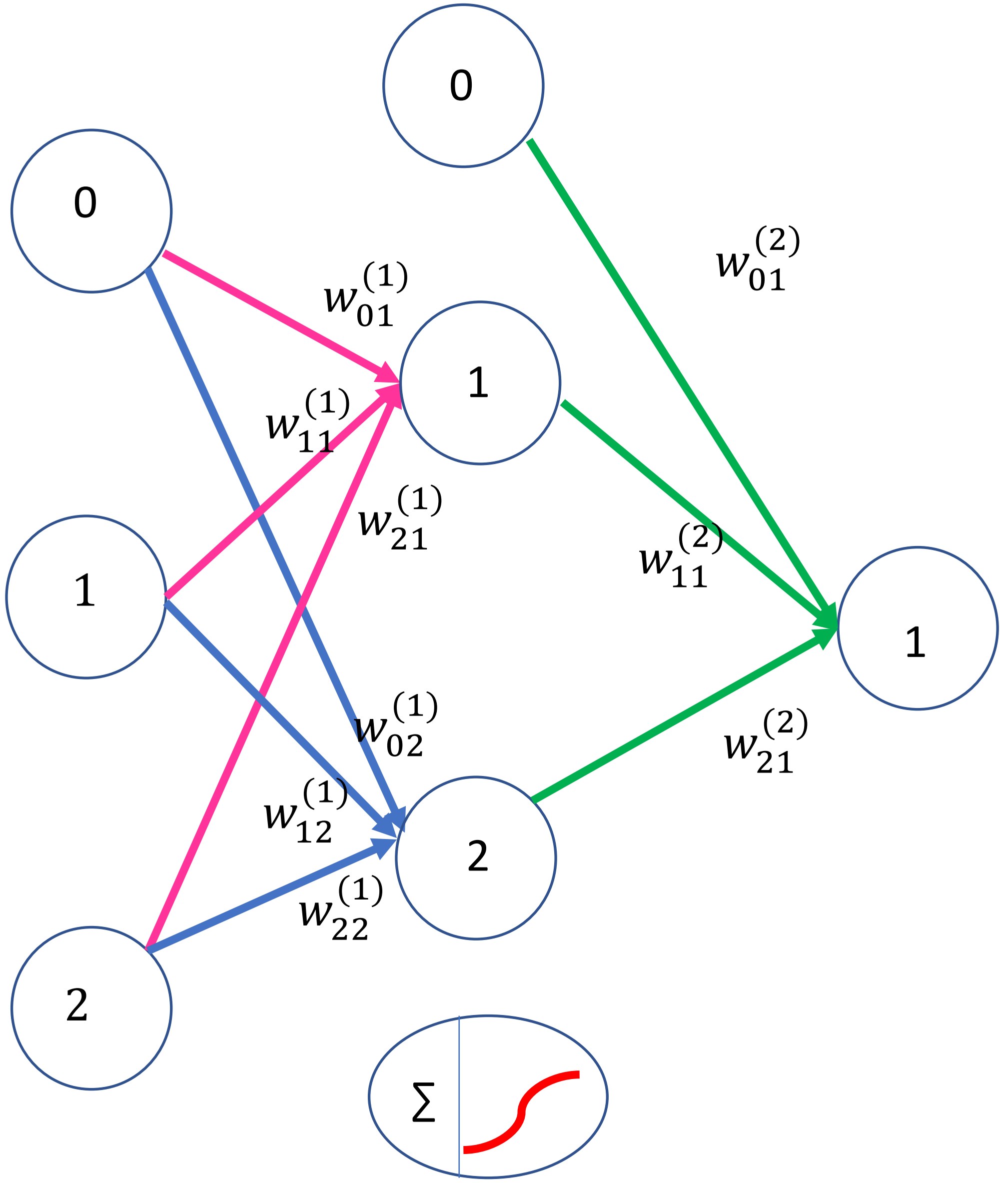
Forward Pass

- When x_1 is applied to neuron 1 in the input layer, the output goes to the hidden layer, passing through weights $w_{11}^{(1)}$, and $w_{12}^{(1)}$.
- The input x_2 behaves in the similar manner, sending signals to hidden nodes through the appropriate weights.



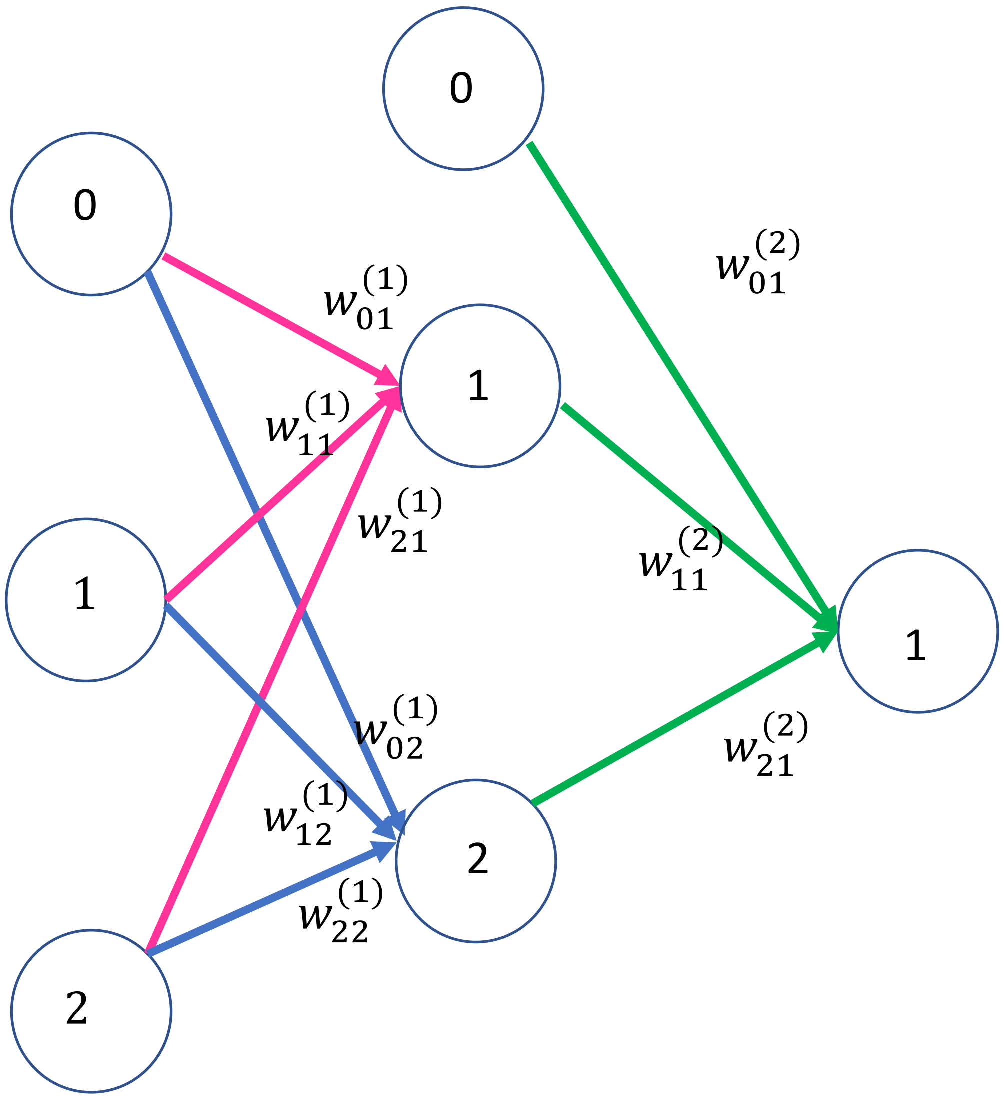
Forward Pass – cont.

- Now consider neuron 1 in the hidden layer:
- It has 3 inputs from the input layer that have been modified by the connection weights $w_{01}^{(1)}, w_{11}^{(1)}, w_{21}^{(1)}$
- The first part of this neuron simply sums up these two weighted inputs.
- The summation then passes through the second part (activation function).
- The output of this activation function is then sent to the output neuron.
- Neuron 2 in the hidden layer behaves in a similar manner.



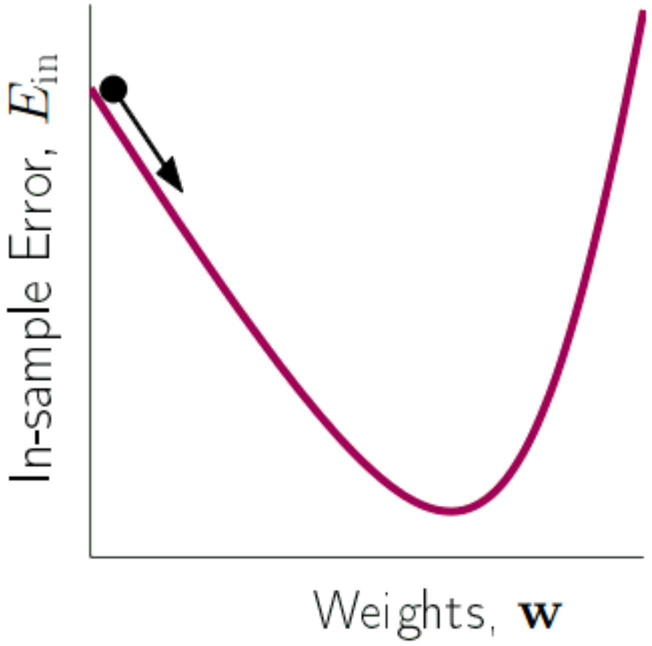
Forward Pass – cont.

- Output neuron collects the weighted inputs from hidden neurons, sum them, and pass the sums through the activation functions to produce the output.
- We use linear activation in the output neuron for regression, and a sigmoidal for classification problems.



How to learn the weights?

- The objective is to find a set of weights that minimizes the error.
- Recall the update rule in GD:
  $$new\ \vec{w} = old\ \vec{w} - \eta \nabla E(\vec{w})$$
- Therefore we need to find the derivative of error with respect to all w's.



How to learn the weights?

$$e = (h(x) - y)^2$$

$$\theta(s^{(2)})$$

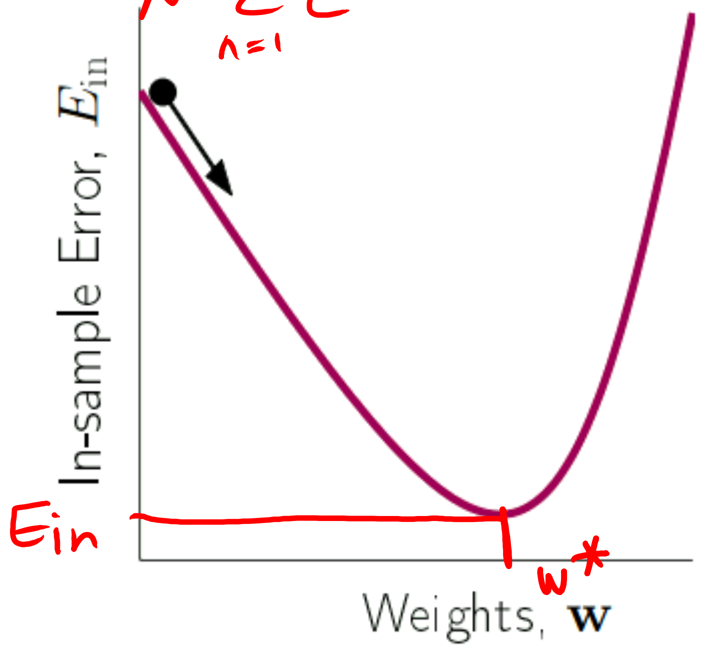
$$E_{in} = \frac{1}{N}\sum_{n=1}^{N} e$$

- The objective is to find a set of weights that minimizes the error.
- Recall the update rule in GD:

$$\textbf{new } \vec{w} = \textbf{old } \vec{w} - \eta \nabla E(\vec{w})$$

- Therefore we need to find the derivative of error with respect to all w's.

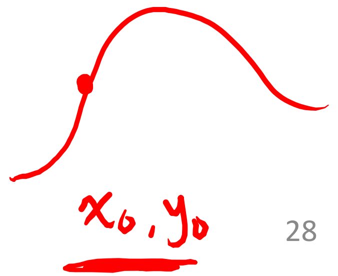
$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \frac{\partial E}{\partial w_1} \end{bmatrix}$$

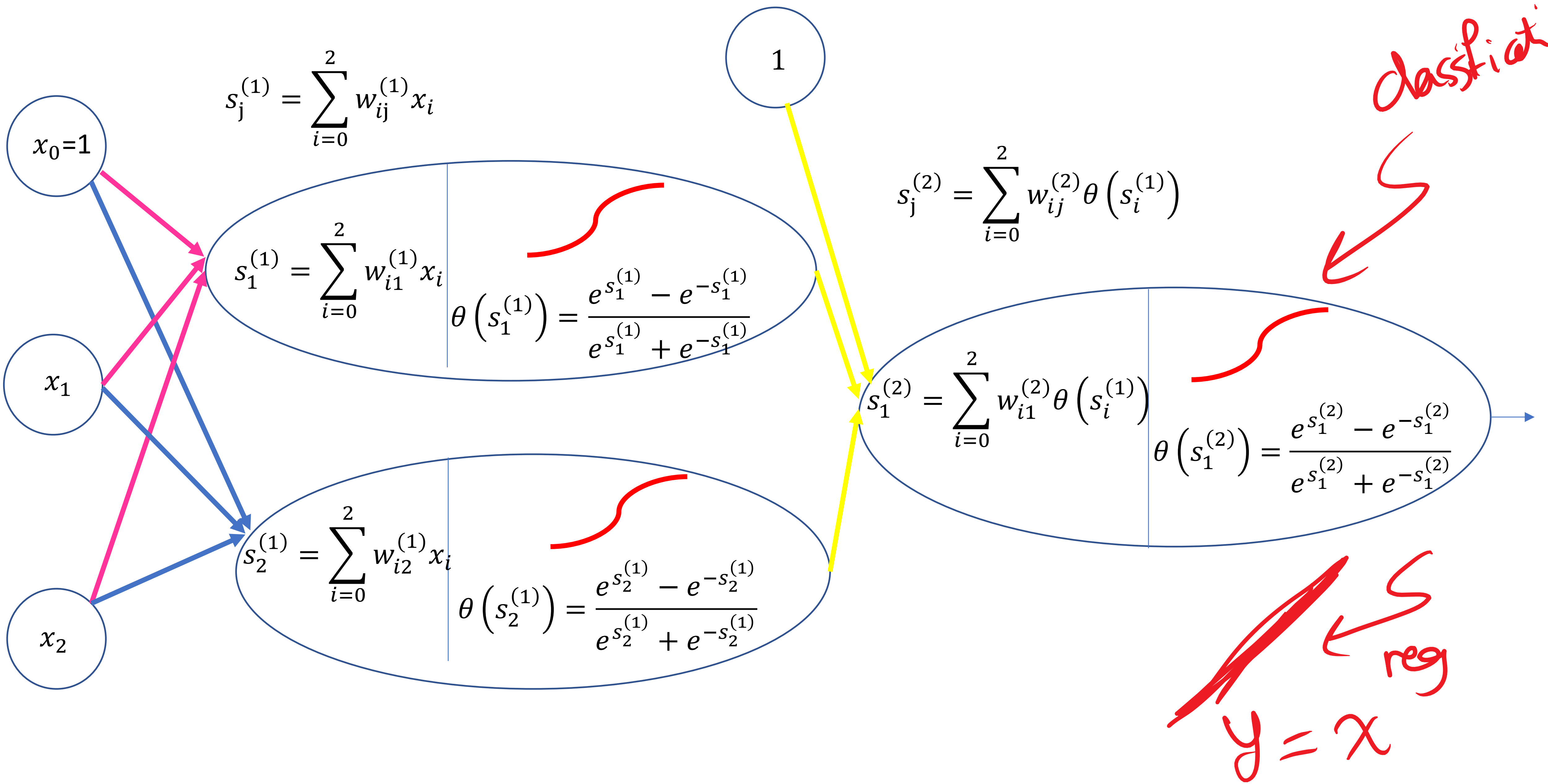


$$\Sigma$$

$$f(x,y)$$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$





x_0=1
x_1
x_2
1
$s_j^{(1)} = \sum_{i=0}^{2} w_{ij}^{(1)} x_i$
$s_1^{(1)} = \sum_{i=0}^{2} w_{i1}^{(1)} x_i$
$\theta\left(s_1^{(1)}\right) = \frac{e^{s_1^{(1)}} - e^{-s_1^{(1)}}}{e^{s_1^{(1)}} + e^{-s_1^{(1)}}}$
$s_2^{(1)} = \sum_{i=0}^{2} w_{i2}^{(1)} x_i$
$\theta\left(s_2^{(1)}\right) = \frac{e^{s_2^{(1)}} - e^{-s_2^{(1)}}}{e^{s_2^{(1)}} + e^{-s_2^{(1)}}}$
$s_j^{(2)} = \sum_{i=0}^{2} w_{ij}^{(2)} \theta\left(s_i^{(1)}\right)$
$s_1^{(2)} = \sum_{i=0}^{2} w_{i1}^{(2)} \theta\left(s_i^{(1)}\right)$
$\theta\left(s_1^{(2)}\right) = \frac{e^{s_1^{(2)}} - e^{-s_1^{(2)}}}{e^{s_1^{(2)}} + e^{-s_1^{(2)}}}$
classificat
reg
$y = x$

Exercise

Based on this figure determine the following quantities:

$$\frac{\partial s_1^{(1)}}{\partial w_{21}^{(1)}}, \frac{\partial s_j^{(1)}}{\partial w_{ij}^{(1)}}, \frac{\partial s_1^{(2)}}{\partial w_{21}^{(2)}}, \frac{\partial s_j^{(2)}}{\partial w_{ij}^{(2)}}, \frac{d\theta(s)}{d(s)}$$

$$\boldsymbol{\frac{d\theta(s)}{d(s)} = 1 - \theta(s)^2}$$

Solution

$$s_1^{(1)} = \sum_{i=0}^{2} w_{i1}^{(1)} x_i = w_{01}^{(1)}(1) + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2$$

$$\frac{\partial s_1^{(1)}}{\partial w_{21}^{(1)}} = x_2$$

$$\boldsymbol{\frac{\partial s_j^{(1)}}{\partial w_{ij}^{(1)}} = x_i}$$

$$s_1^{(2)} = w_{01}^{(2)}(1) + w_{11}^{(2)} \theta\left(s_1^{(1)}\right) + w_{21}^{(2)} \theta\left(s_2^{(1)}\right)$$

$$\frac{\partial s_1^{(2)}}{\partial w_{21}^{(2)}} = \theta\left(s_2^{(1)}\right) \Rightarrow \boldsymbol{\frac{\partial s_j^{(2)}}{\partial w_{ij}^{(2)}} = \theta\left(s_i^{(1)}\right)}$$

Proof: $\theta'(s) = 1 - \theta^2(s)$

$$\tanh(x) = \frac{\overbrace{e^{x} - e^{-x}}^{u}}{\underbrace{e^{x} + e^{-x}}_{v}} \qquad \left(\frac{u}{v}\right)' = \frac{u'v - v'u}{v^2}$$

$$\frac{\overbrace{e^{x} - (-e^{-x})}^{u'}\underbrace{(e^{x} + e^{-x})}_{v} - \underbrace{(e^{x} - e^{-x})}_{v'}\underbrace{(e^{x} - e^{-x})}_{u}}{(e^{x} + e^{-x})^2}$$

$$\frac{(e^{x} + e^{-x})^2 - (e^{x} - e^{-x})^2}{(e^{x} + e^{-x})^2} = 1 - \left(\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}\right)^2 = 1 - \theta^2(x)$$

Example (Forward Pass)

Assume the following weight values are provided. Draw a network that corresponds to these weight matrices and determine the error if the data point is (x=2,y=1).

$$i \xrightarrow{w_{ij}} j$$

$$W^{(1)} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}; \quad W^{(2)} = \begin{bmatrix} 0.2 \\ 1 \\ -3 \end{bmatrix}$$

Solution

- $\overrightarrow{x^T} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- $s^1 = w^T x^T = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$
- $\theta(s^1) = tanh \begin{bmatrix} 0.7 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.60 \\ 0.76 \end{bmatrix}$
- $s^2 = \begin{bmatrix} 0.2 & 1 & -3 \end{bmatrix} \begin{bmatrix} 1 \\ 0.6 \\ 0.76 \end{bmatrix} = -1.48$
- $e = (\tanh(-1.48) - 1)^2 = 3.62$