



UNIVERSIDADE
CATÓLICA
PORTUGUESA

BRAGA

Deep Learning

Session 3

The Perceptron

Applied Data Science

2024/2025

Perceptron

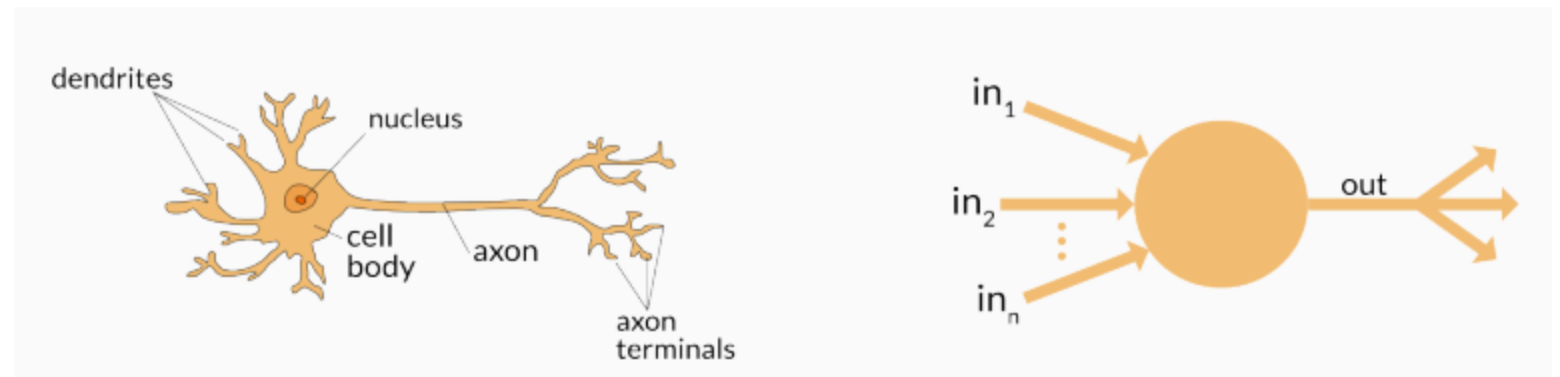
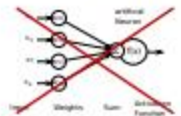


1958 Perceptron



1969
Perceptrons
book

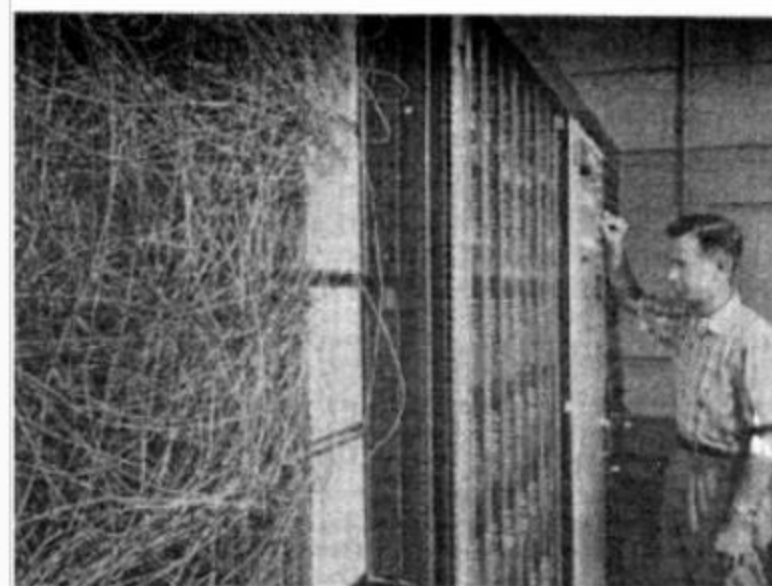
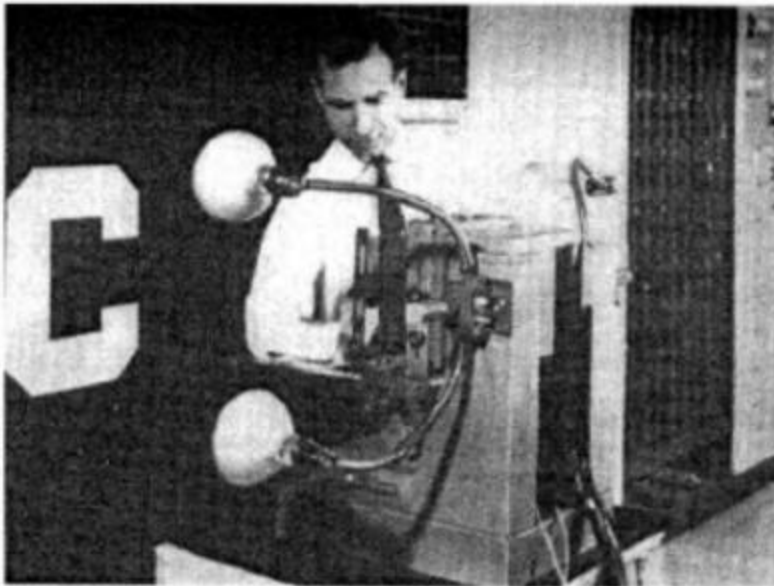
Perceptron criticized



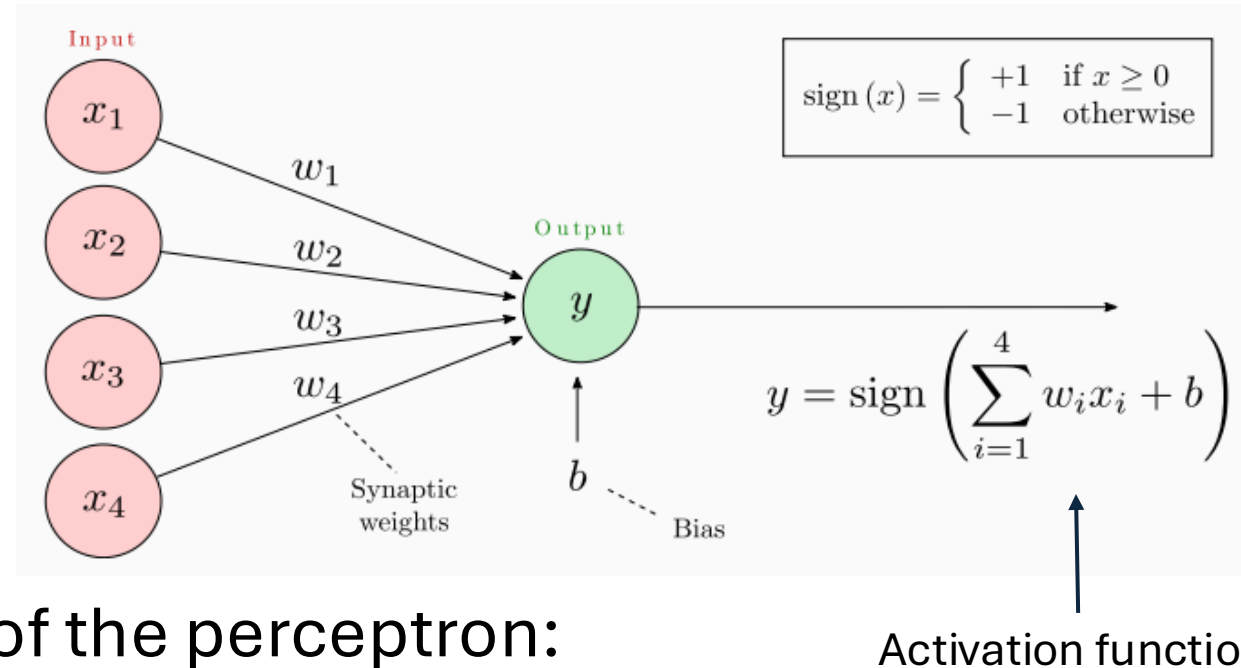
Perceptron

- First binary classifier based on supervised learning;
- Foundation of modern artificial neural networks;

Perceptron (Frank Rosenblatt, 1958)

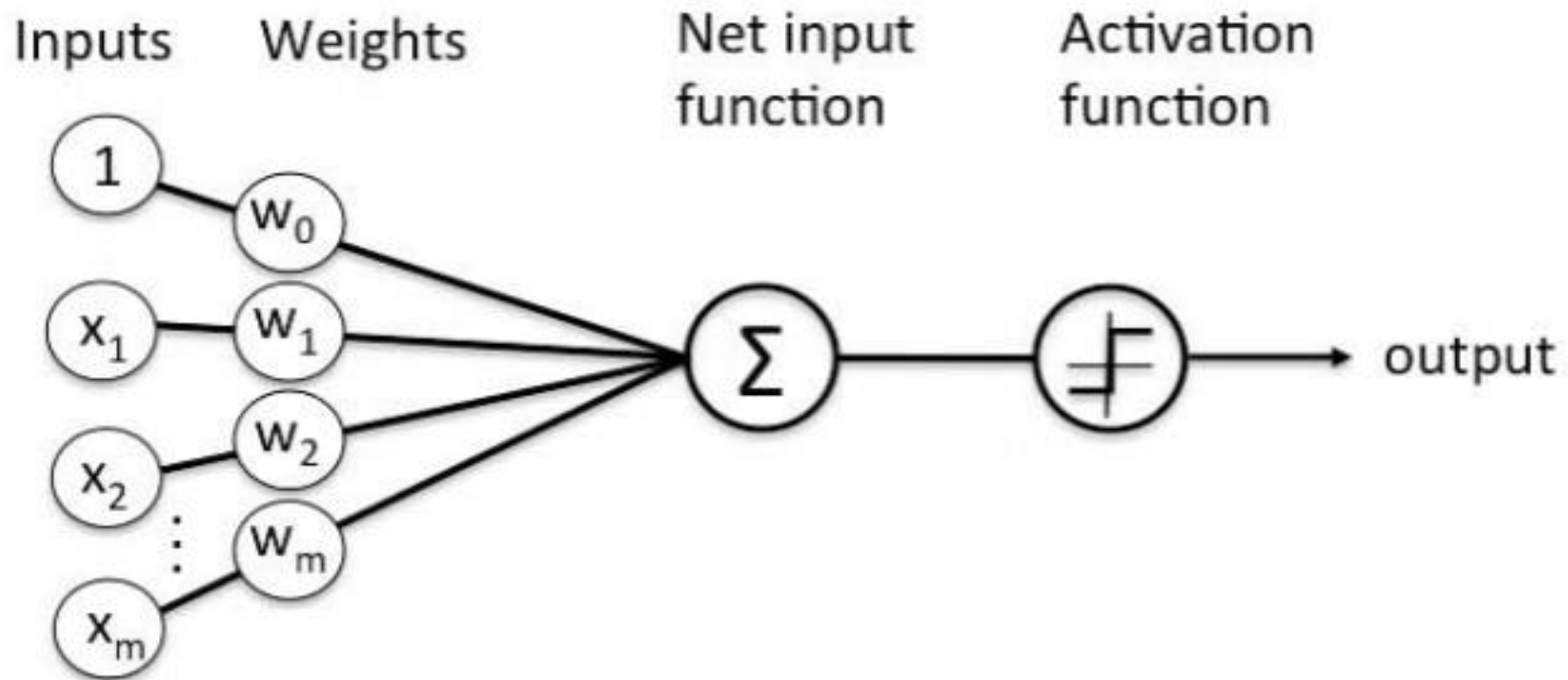


Representation of the Perceptron

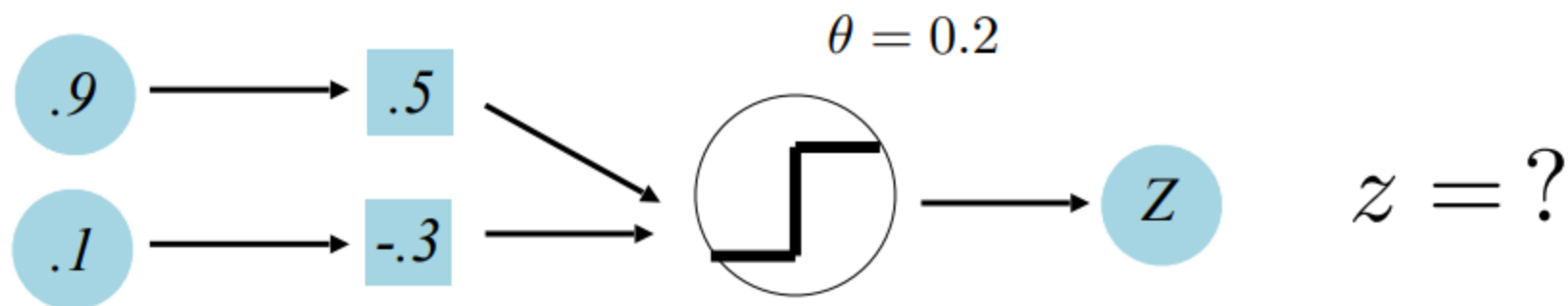


- Parameters of the perceptron:
 - w_k : weights
 - b : bias
- Training \rightarrow adjusting the weights and bias.

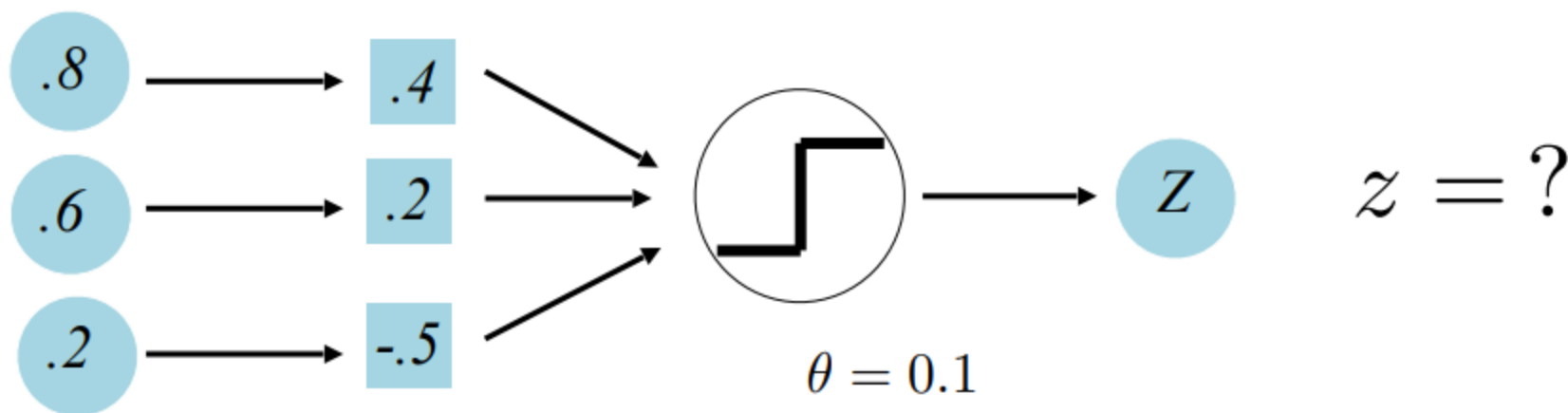
Alternative Representation of the Perceptron



Perceptron Examples

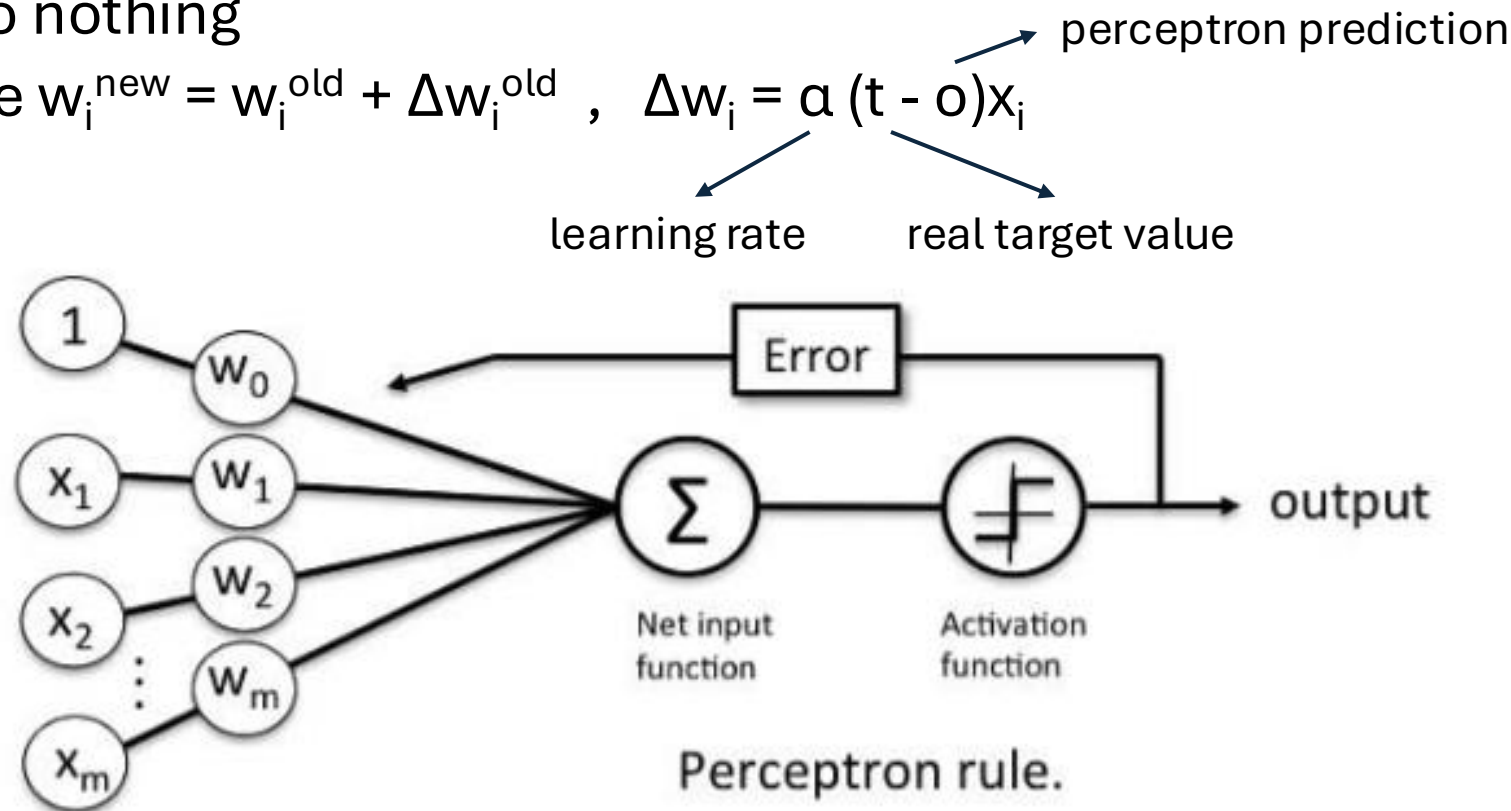


$$z = \begin{cases} 1, & \text{if } w \cdot x > \theta \\ 0, & \text{if } w \cdot x \leq \theta \end{cases}$$



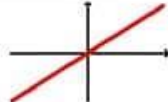
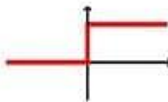
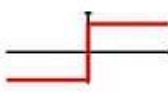




Perceptron Learning Rule

- Suppose that x is a feature vector, y is the correct class label, and y' is the predicted class label computed using the current weights.
 - If $y' = y$, do nothing
 - Otherwise $w_i^{\text{new}} = w_i^{\text{old}} + \Delta w_i^{\text{old}}$, $\Delta w_i = \alpha (t - o)x_i$



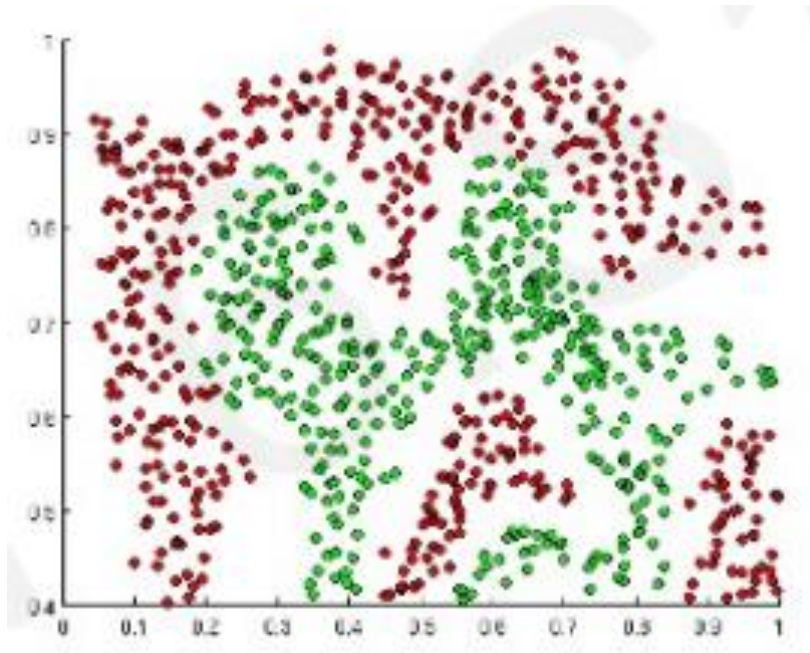
Activation Functions

- The purpose of activation functions is to **introduce non-linearities** into the network.

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -1/2 \\ z + 1/2 & -1/2 \leq z \leq 1/2 \\ 1 & z \geq 1/2 \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Activation Functions

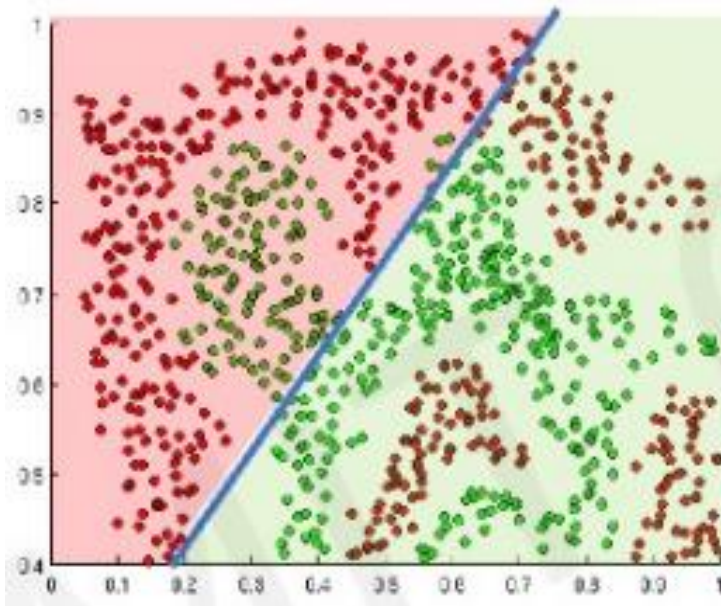
- The purpose of activation functions is to **introduce non-linearities** into the network.



What if we wanted to build a neural network to distinguish green vs red points?

Activation Functions

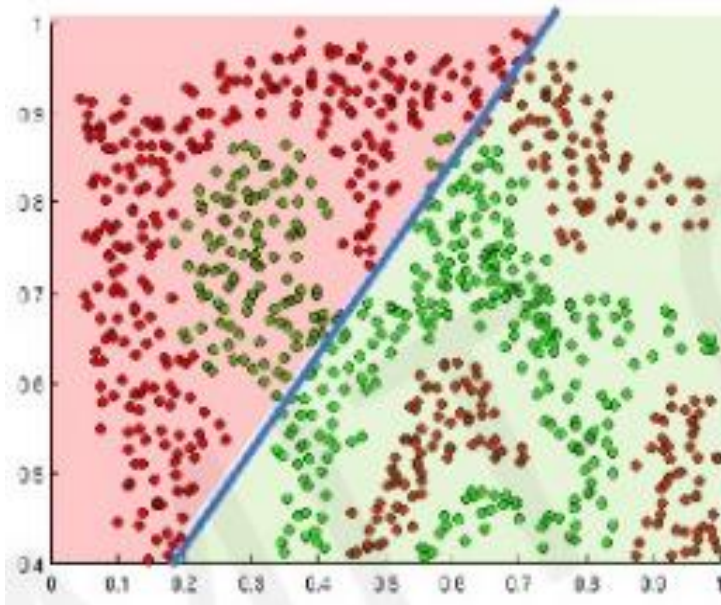
- The purpose of activation functions is to **introduce non-linearities** into the network.



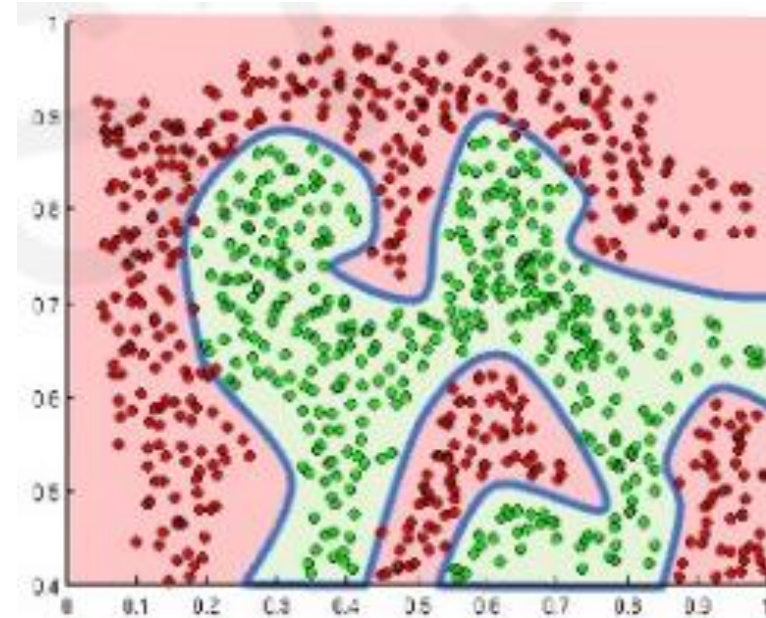
Linear activations produce linear decisions
no matter the network size.

Activation Functions

- The purpose of activation functions is to **introduce non-linearities** into the network.



Linear activations produce linear decisions
no matter the network size.



Non-linearities allow us to approximate
arbitrarily complex functions.

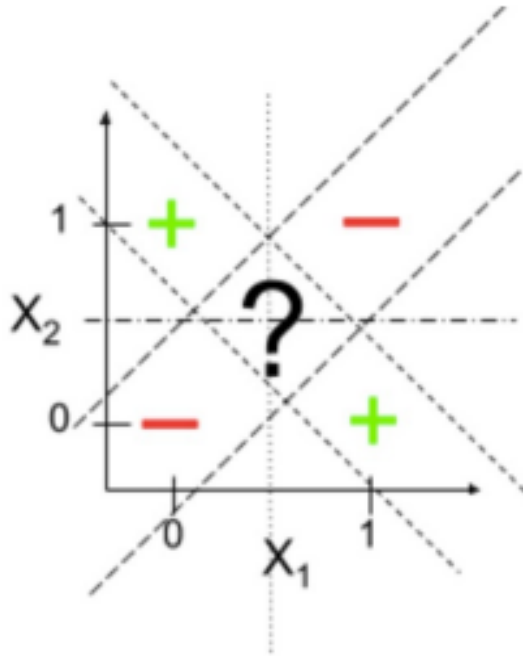
Perceptron Limitation: XOR Problem

- XOR = "Exclusive Or"
 - Input: two binary values x_1 and x_2
 - Output:
 - 1, when exactly one input equals 1
 - 0, otherwise

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	?
0	1	?
1	0	?
1	1	?

Perceptron Limitation: XOR Problem

- Cannot solve XOR problem and so separate 1s from 0s with a perceptron (linear function):



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Multilayer Perceptron



1958 Perceptron

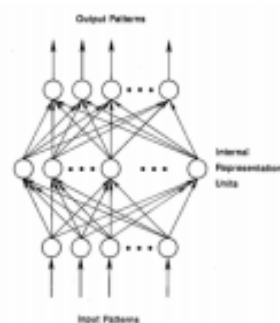


1969
Perceptrons
book

Perceptron criticized



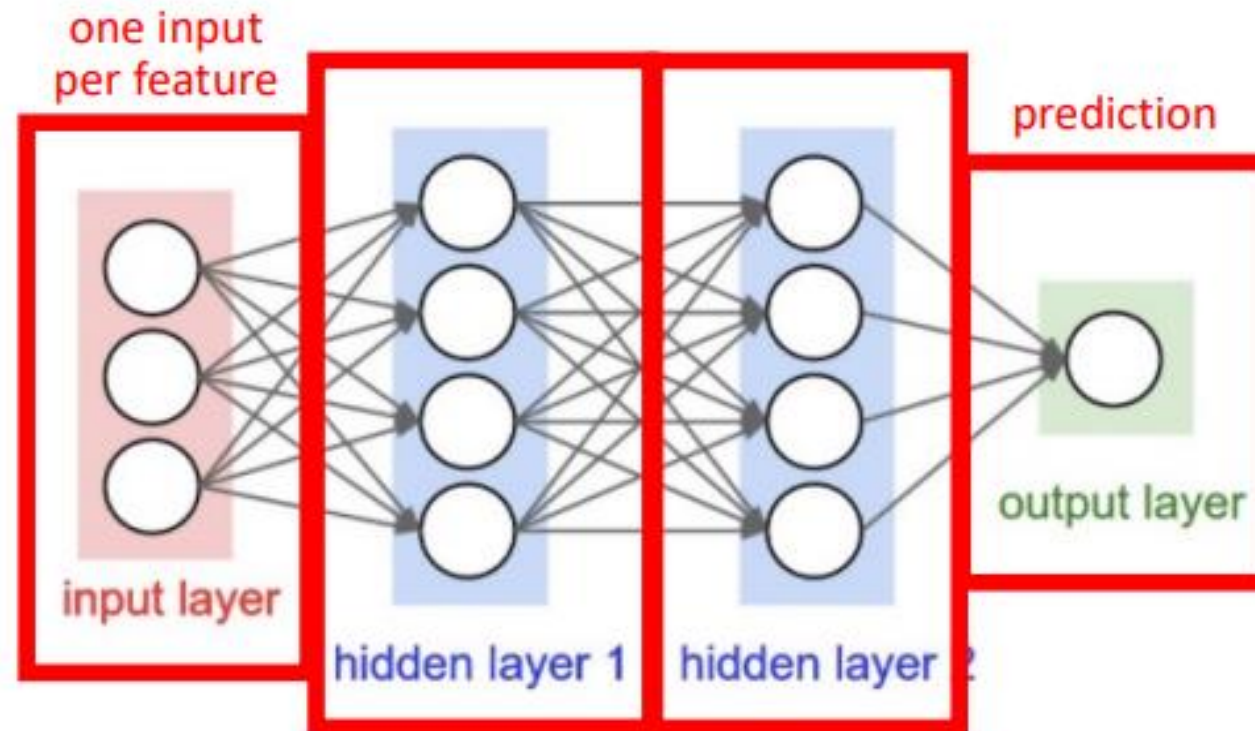
~1980
Multilayer
network



1989
Universal
Approximation
Theorem

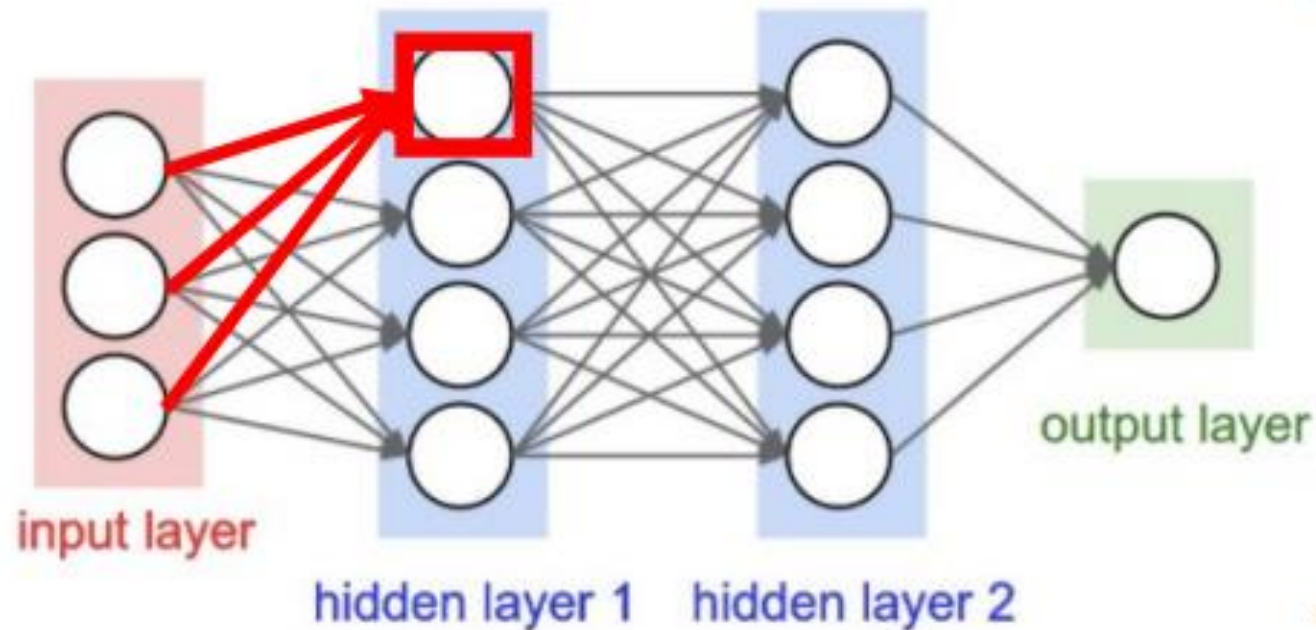
Mulilayer Perceptron

- AKA Artificial Neural Networks

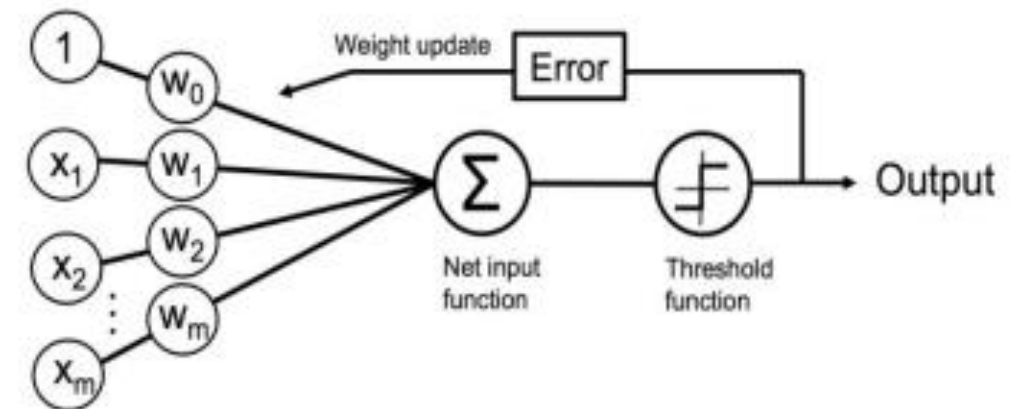


each "hidden layer" uses outputs of units (i.e., neurons) and provides them as inputs to other units (i.e., neurons)

Mulilayer Perceptron

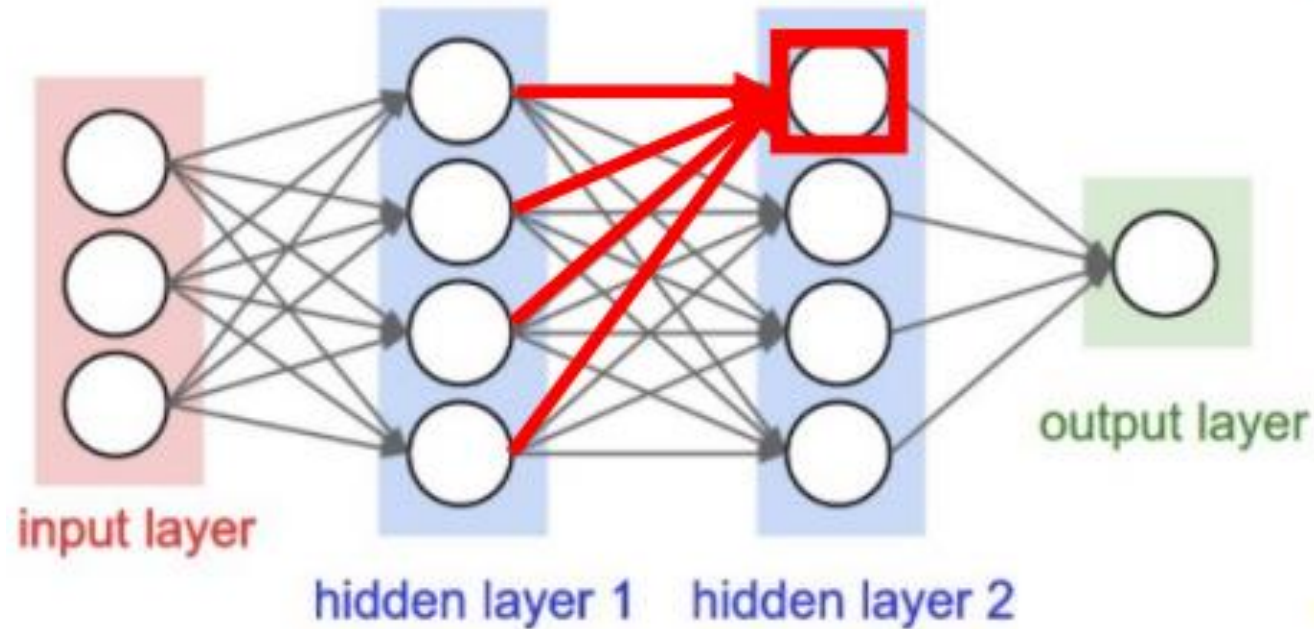


- How does this relate to a perceptron?

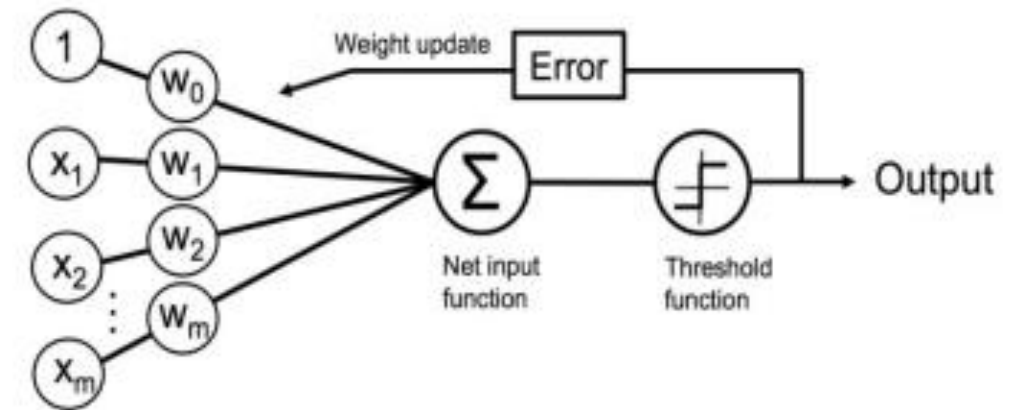


- Unit: takes as input a weighted sum and applies an activation function

Mulilayer Perceptron

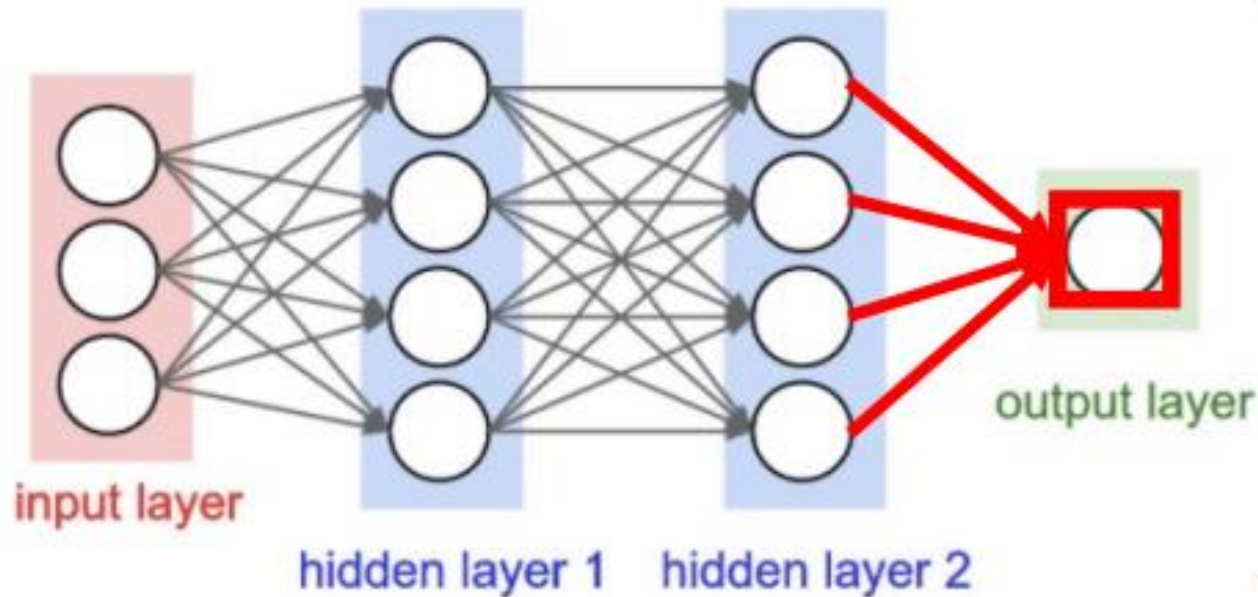


- How does this relate to a perceptron?

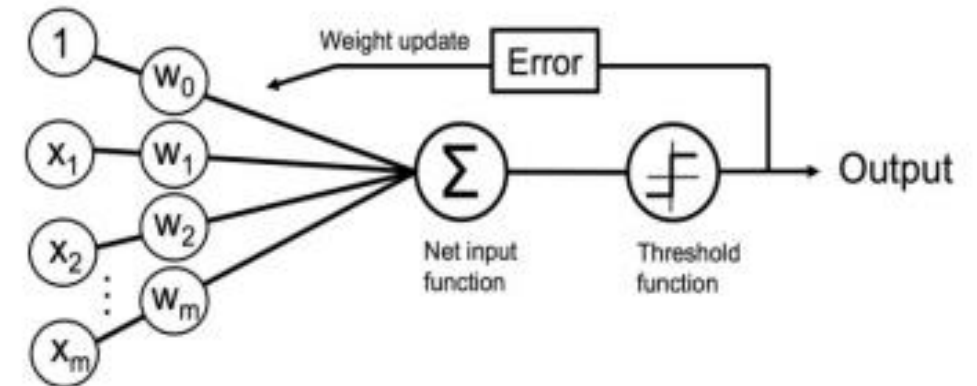


- Unit: takes as input a weighted sum and applies an activation function

Mulilayer Perceptron



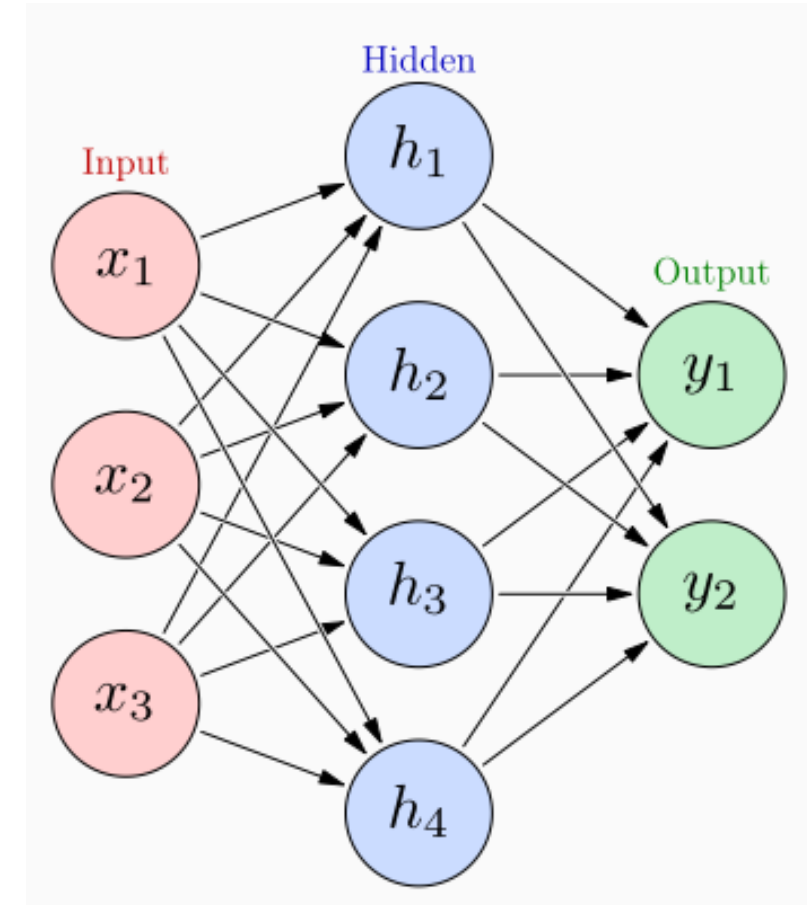
- How does this relate to a perceptron?



- Unit: takes as input a weighted sum and applies an activation function

Artificial Neural Networks

- Inter-connection of several artificial neurons (also called nodes or units);
- Each "level" in the graph is called a layer:
 - Input layer;
 - Hidden layer(s);
 - Output layer.
- Each neuron in the hidden layers acts as a classifier / feature detector;
- Feedforward neural network (no cycles):
 - First and simplest type of neural network;
 - Information moves in one direction.



Artificial Neural Networks

$$h_1 = g_1 (w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1)$$

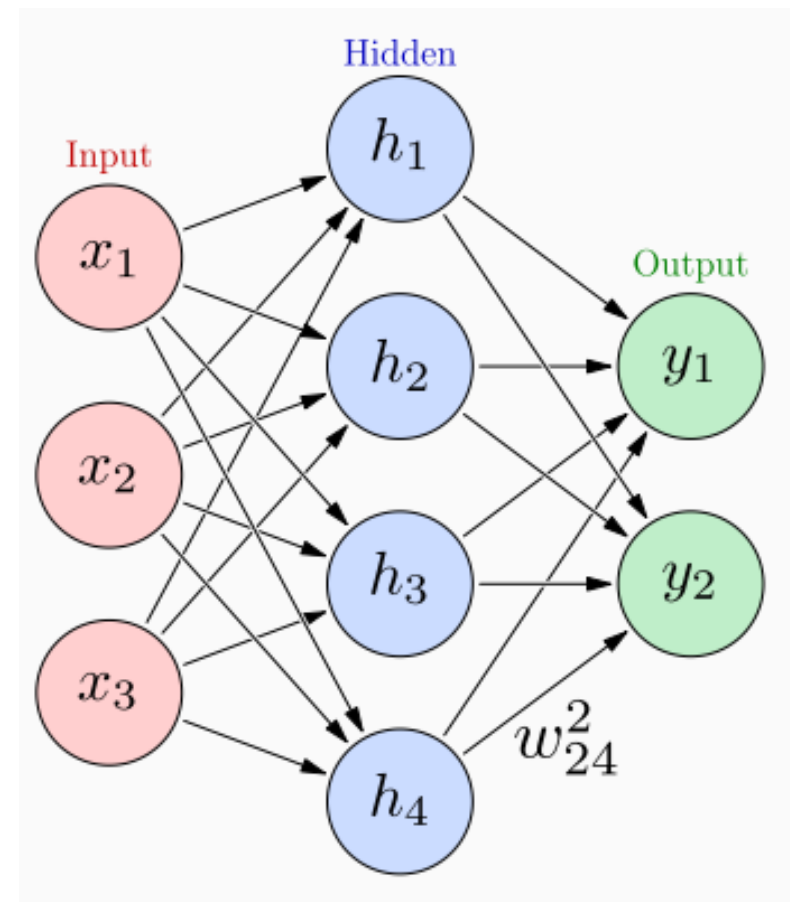
$$h_2 = g_1 (w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1)$$

$$h_3 = g_1 (w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1)$$

$$h_4 = g_1 (w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1)$$

$$y_1 = g_2 (w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2)$$

$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$



- w_{ij}^k weight between previous node j and next node i at layer k ;
- g_k is any activation function applied to each its input vector

Artificial Neural Networks

$$h_1 = g_1 (w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1)$$

$$h_2 = g_1 (w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1)$$

$$h_3 = g_1 (w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1)$$

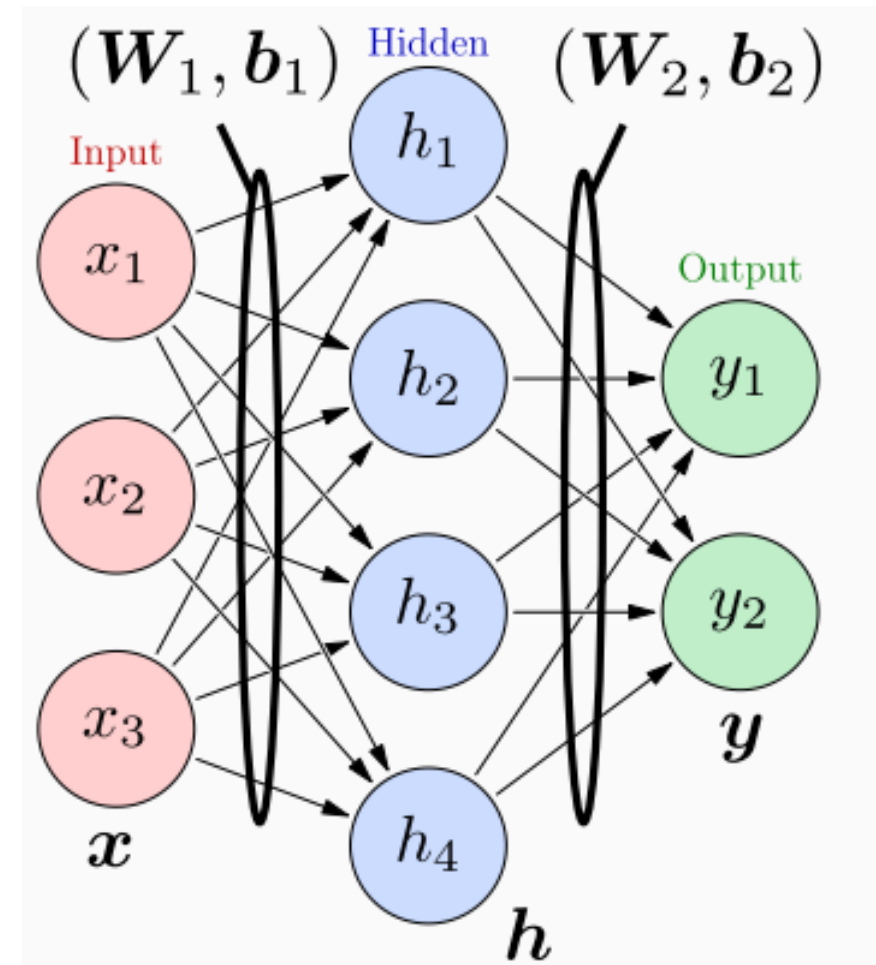
$$h_4 = g_1 (w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1)$$

$$\mathbf{h} = g_1 (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$y_1 = g_2 (w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2)$$

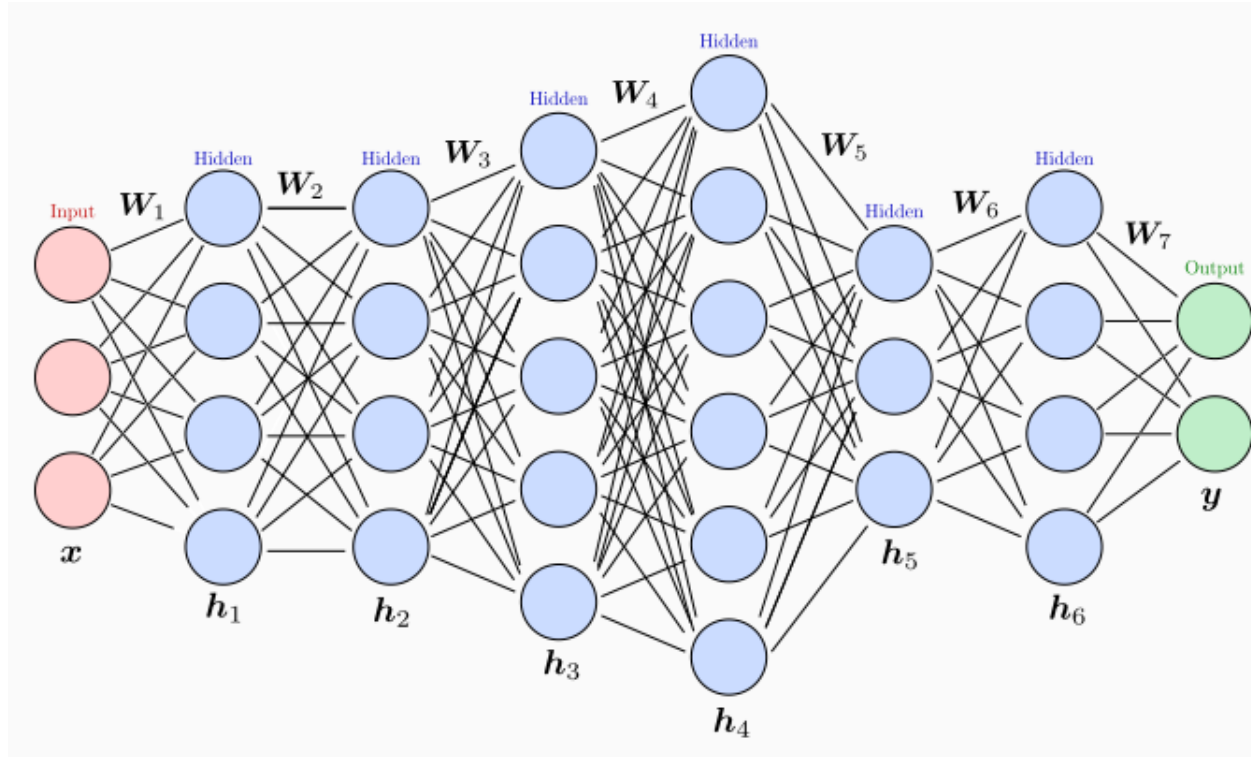
$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$

$$\mathbf{y} = g_2 (\mathbf{W}_2 \mathbf{h} + \mathbf{b}_2)$$



- The matrices \mathbf{W}_k and biases \mathbf{b}_k are learned from labeled training data.

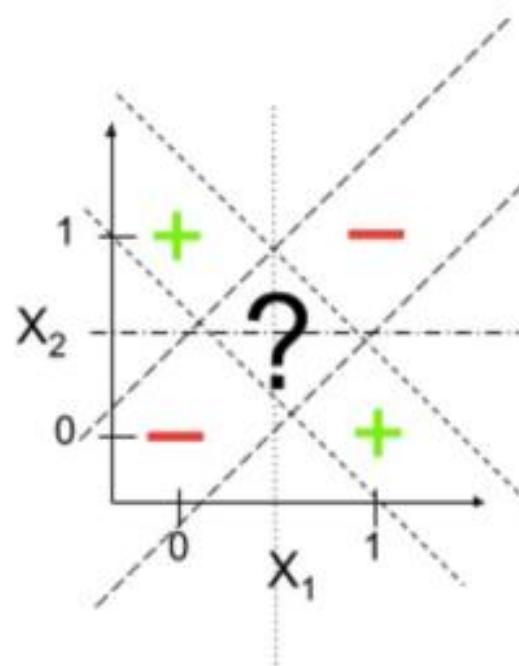
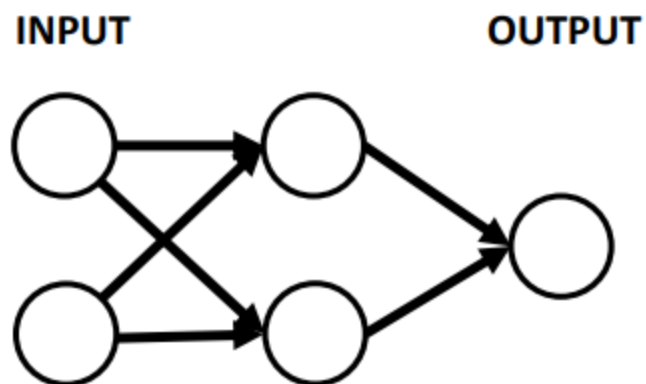
Artificial Neural Networks



- It can have 1 hidden layer only (shallow network);
- It can have more than 1 hidden layer (deep network);
- Each layer can have a different size, and hidden and output layers often have different activation functions.

Revisiting the XOR Problem

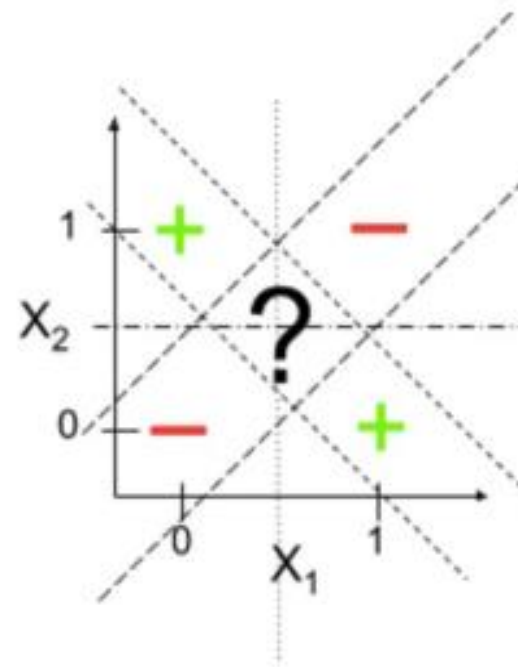
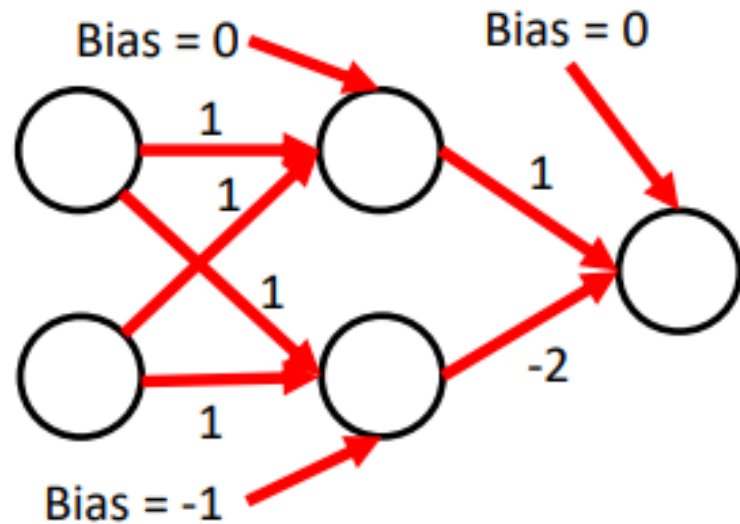
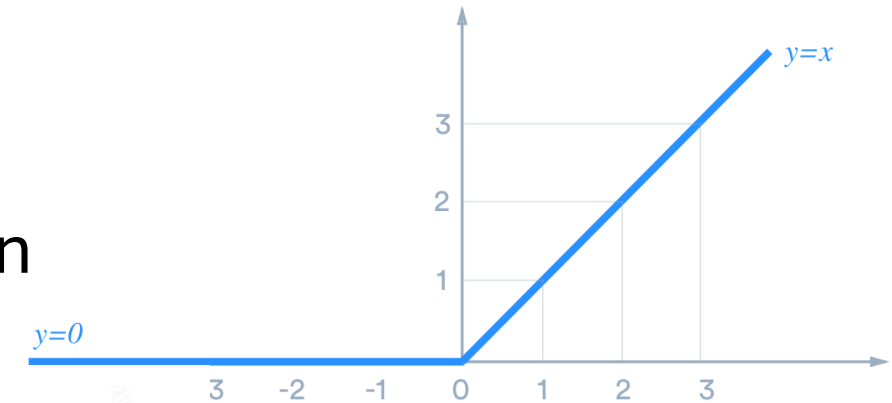
- Non-linear function: separate 1s from 0s



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting the XOR Problem

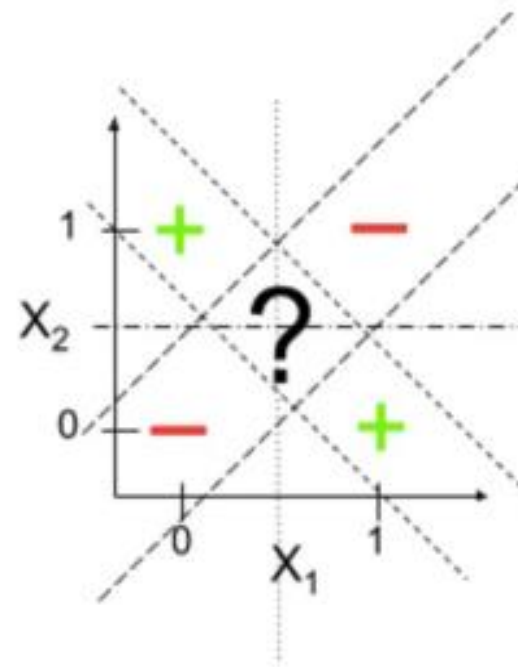
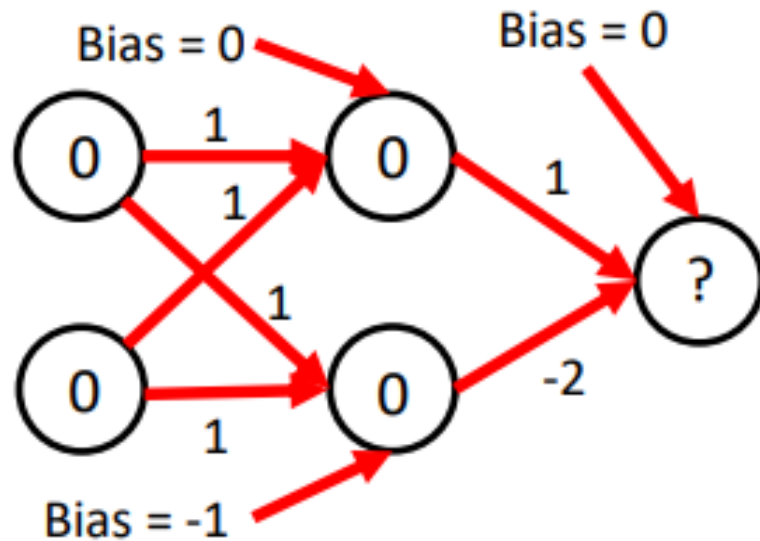
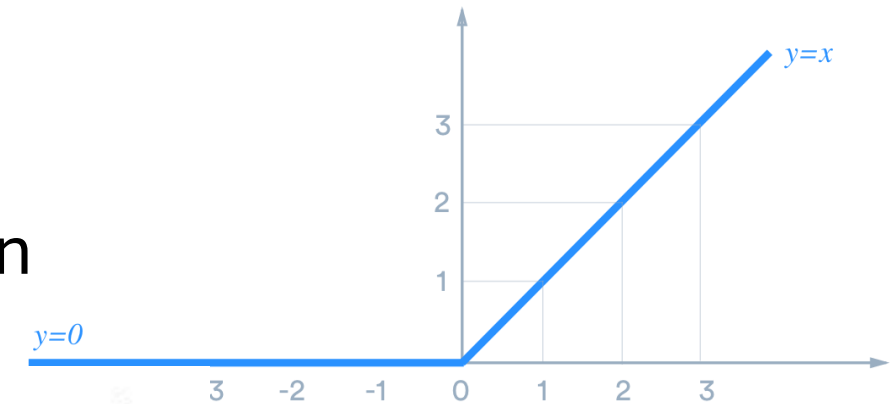
- Non-linear function: separate 1s from 0s
- Approach: use the ReLU activation function



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting the XOR Problem

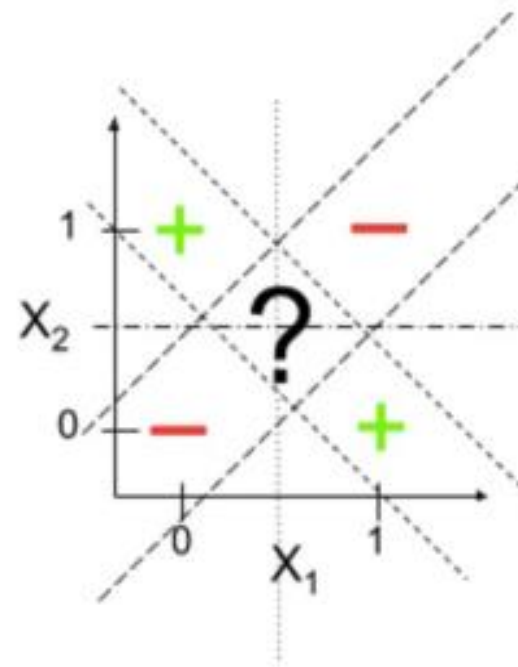
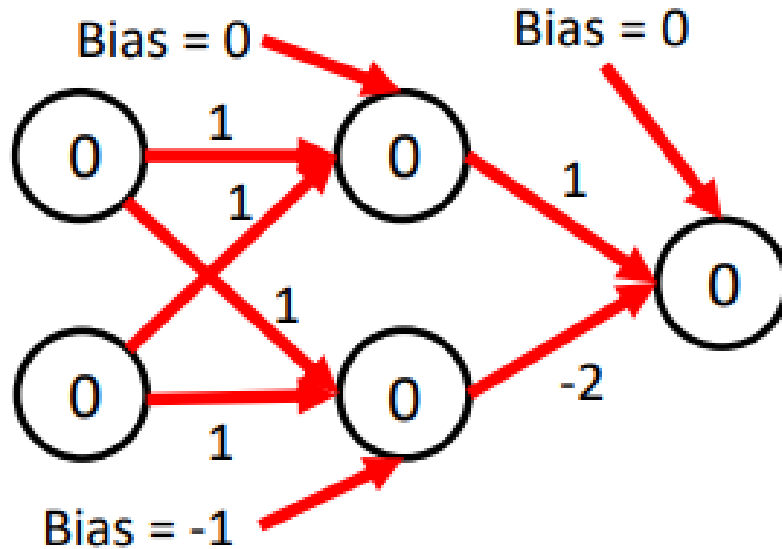
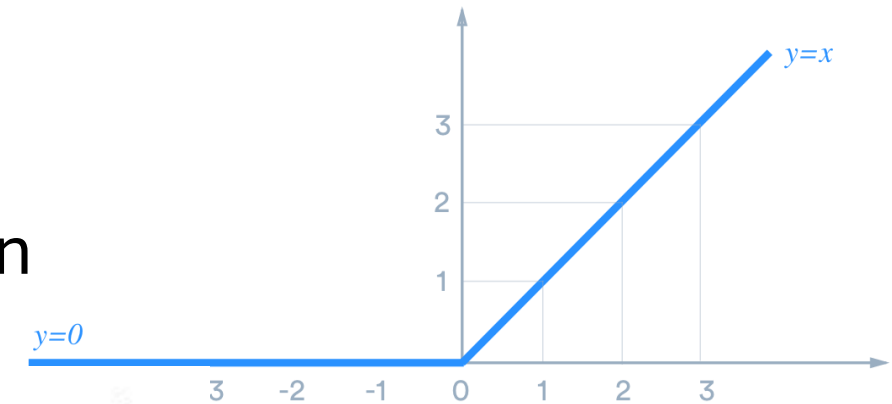
- Non-linear function: separate 1s from 0s
- Approach: use the ReLU activation function



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting the XOR Problem

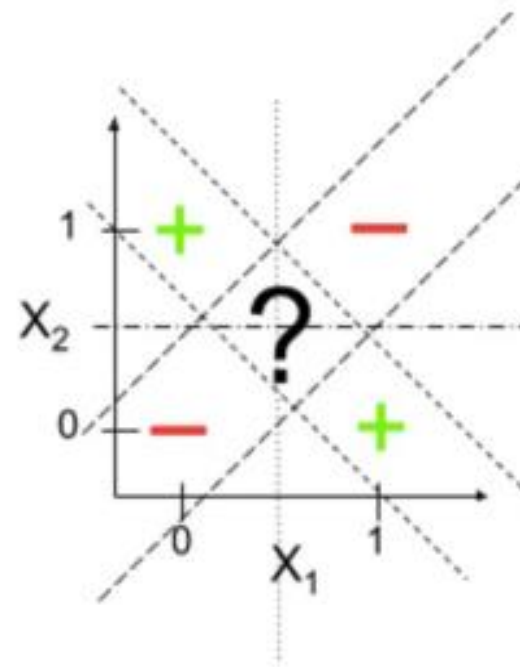
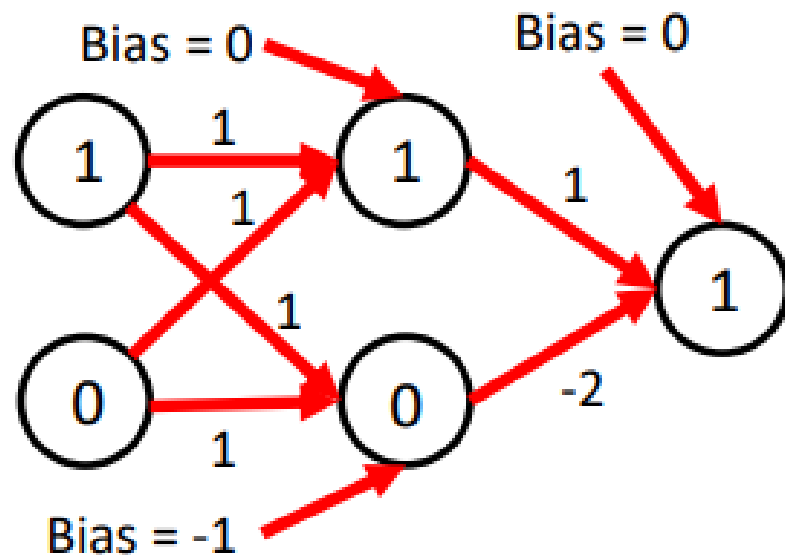
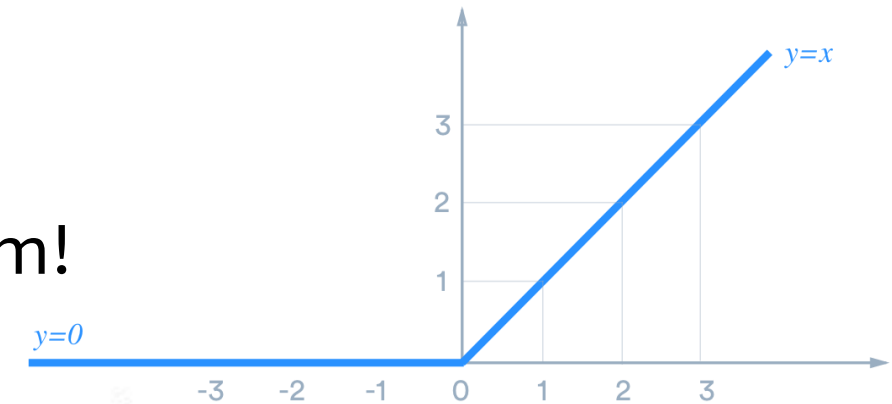
- Non-linear function: separate 1s from 0s
- Approach: use the ReLU activation function



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting the XOR Problem

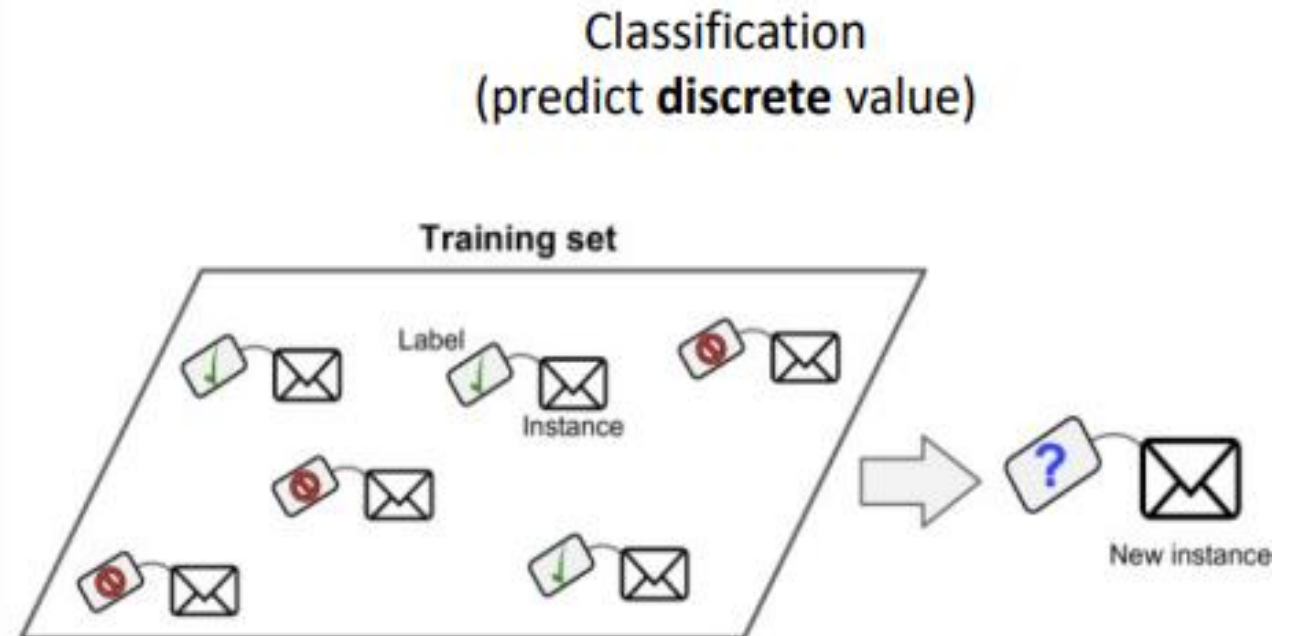
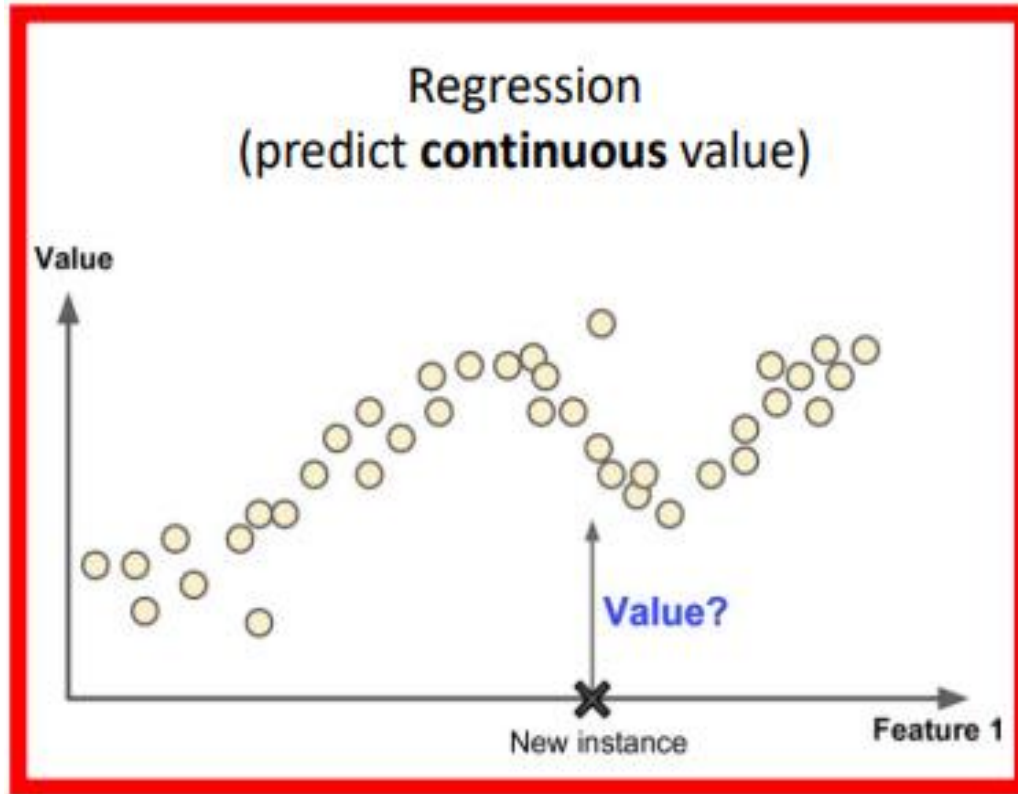
- Non-linear function: separate 1s from 0s
- Neural networks can solve the XOR problem!
And so model non-linear functions!



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

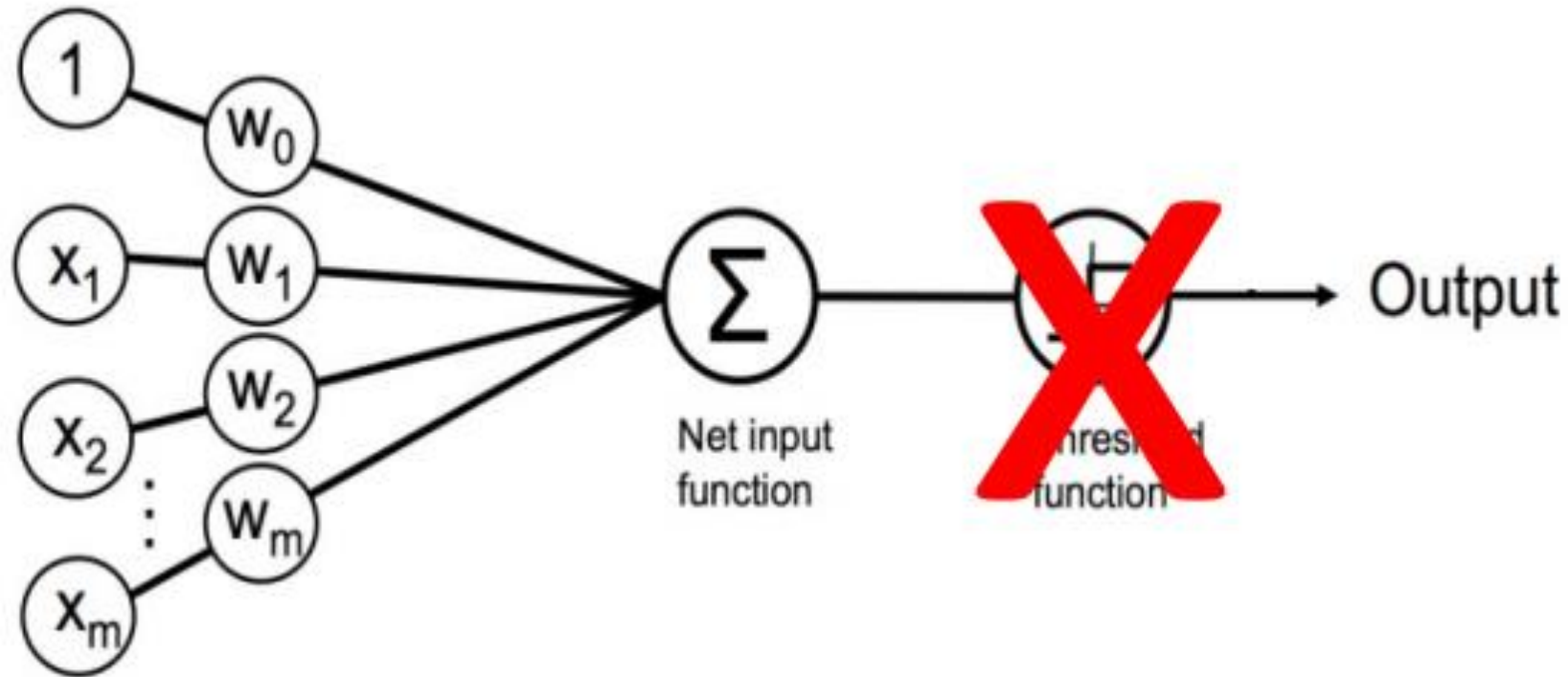
Output Activations

- Desired output driven by task



Output Activations

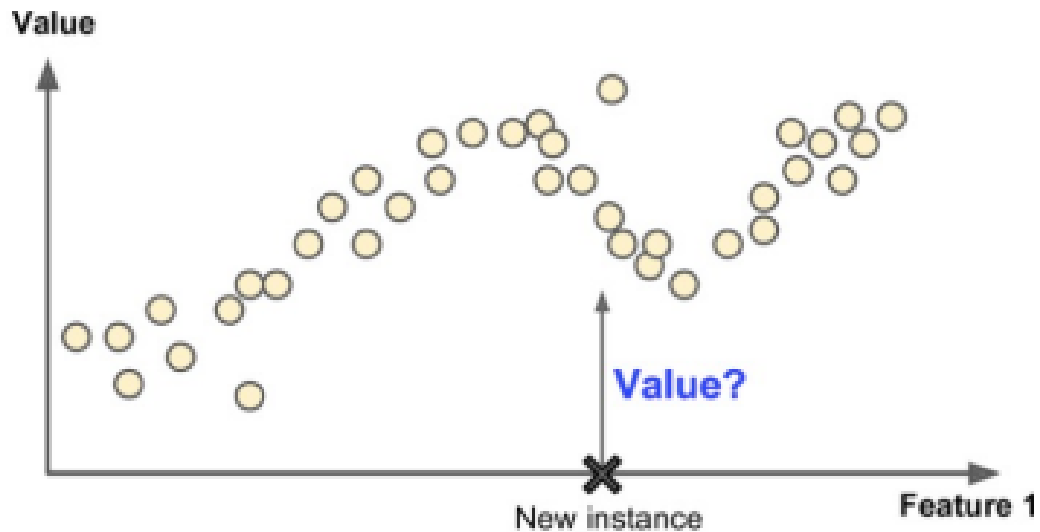
- Linear (No Activation Function)



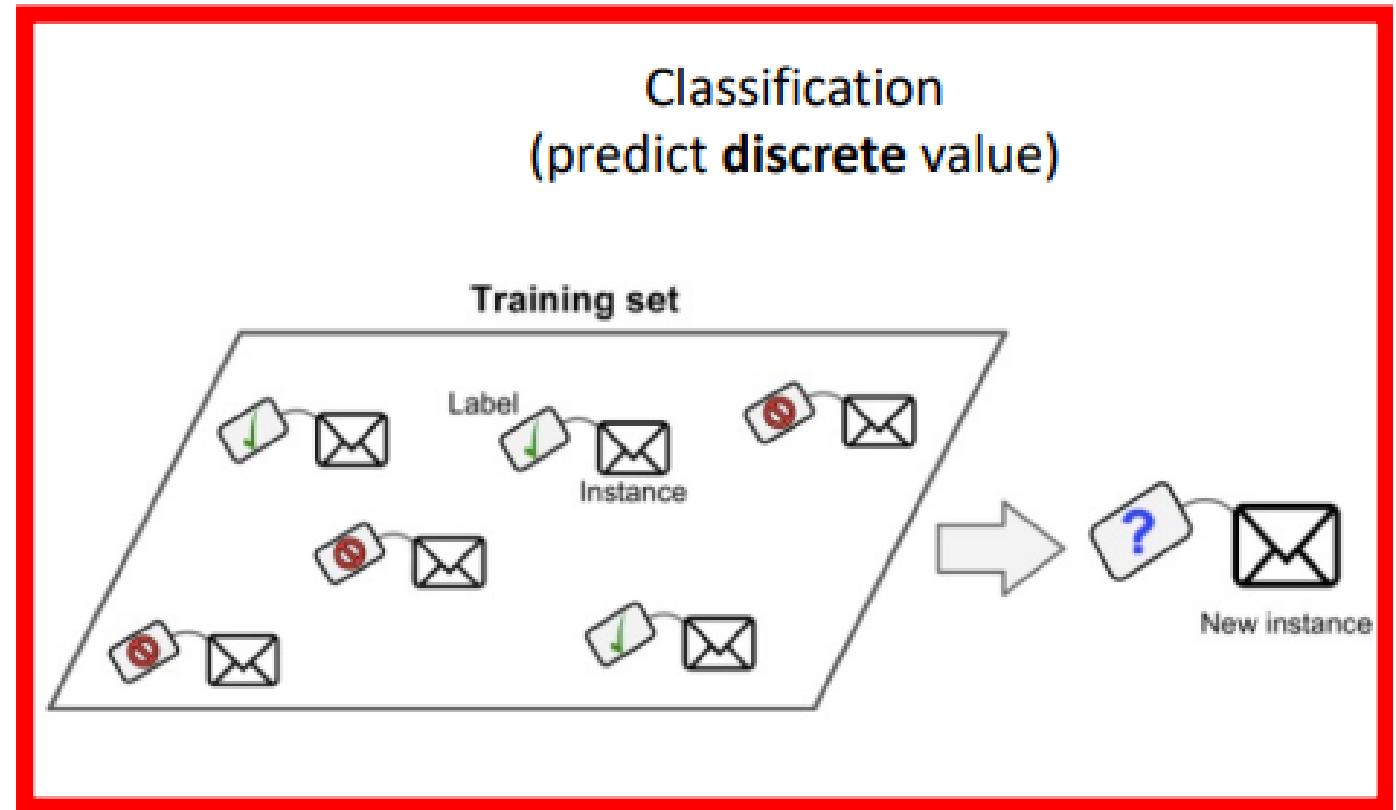
Output Activations

- Desired output driven by task

Regression
(predict **continuous** value)

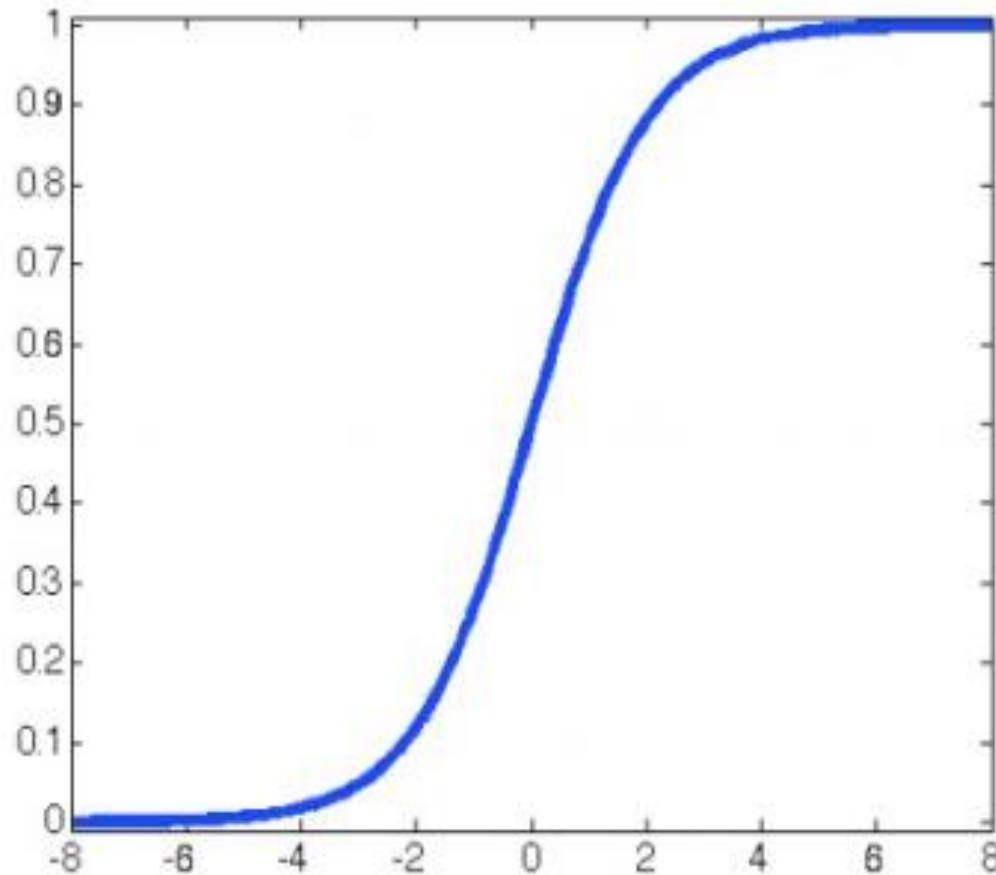


Classification
(predict **discrete** value)



Output Activations

- Sigmoid for Binary Classification



$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

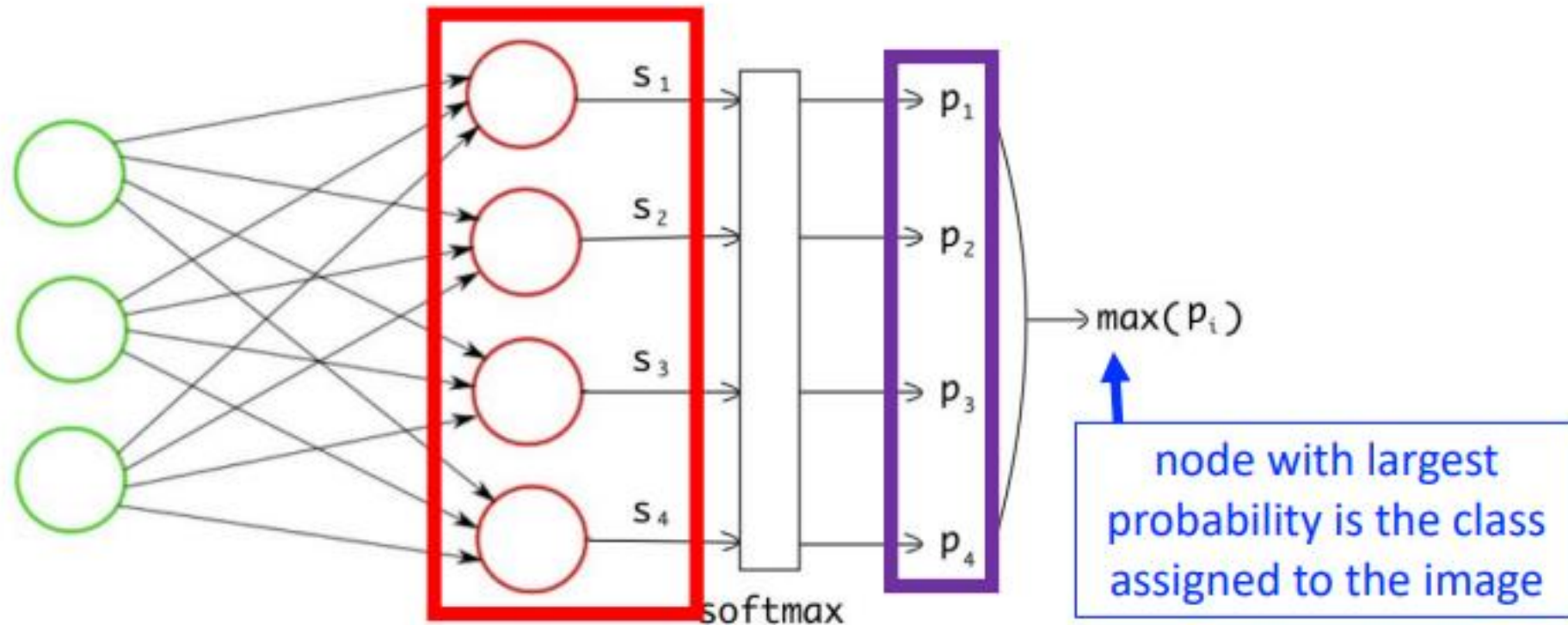
If ≥ 0.5 , output 1;

Else, outputs 0

Output Activations

- Softmax for Multiclass Classification

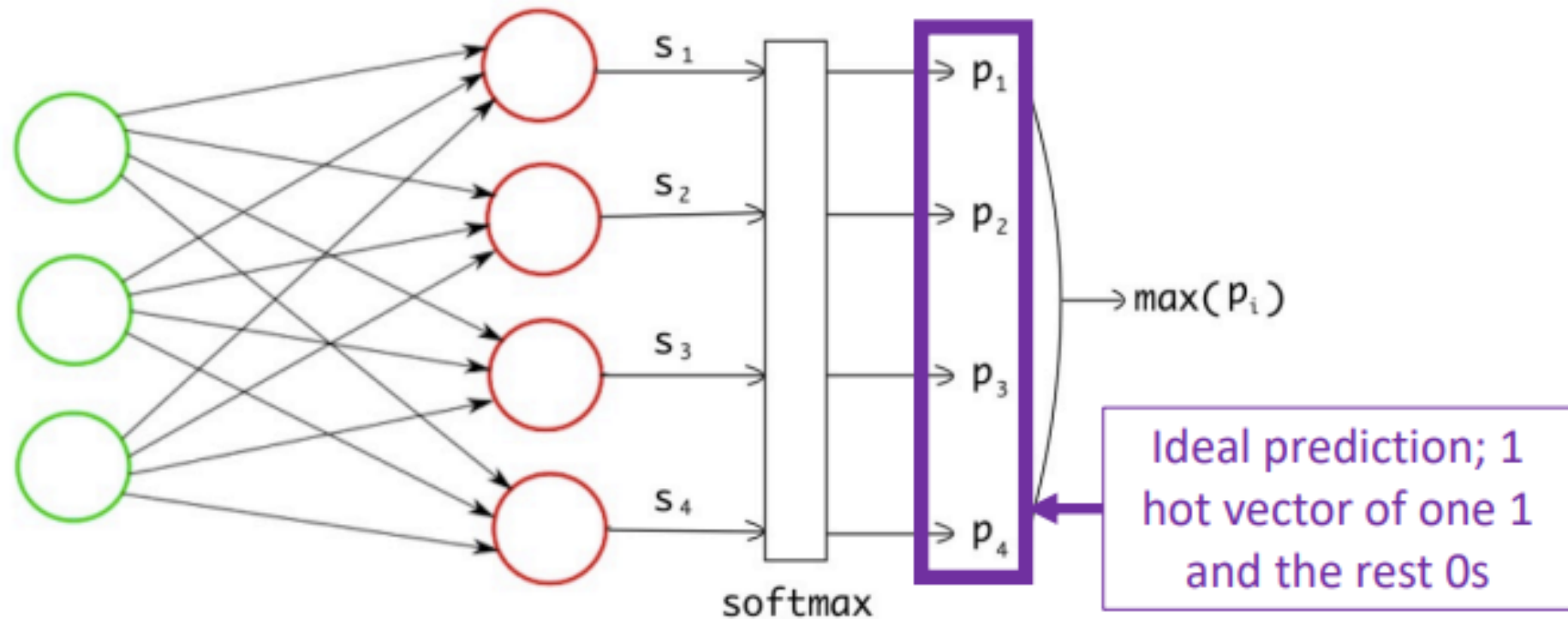
Converts vector of **scores** into a **probability distribution** that sums to 1; e.g.,



Output Activations

- Softmax for Multiclass Classification

Converts vector of **scores** into a **probability distribution** that sums to 1; e.g.,



Output Activations

- Softmax for Multiclass Classification

Converts vector of **scores** into a probability distribution that sums to 1

Get rid of negative values while preserving original order of scores; e causes negative scores to become slightly larger than 0 while positive values grow exponentially (choosing e rather than another exponent base simplifies math during training)

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

$i = 1, \dots, K$

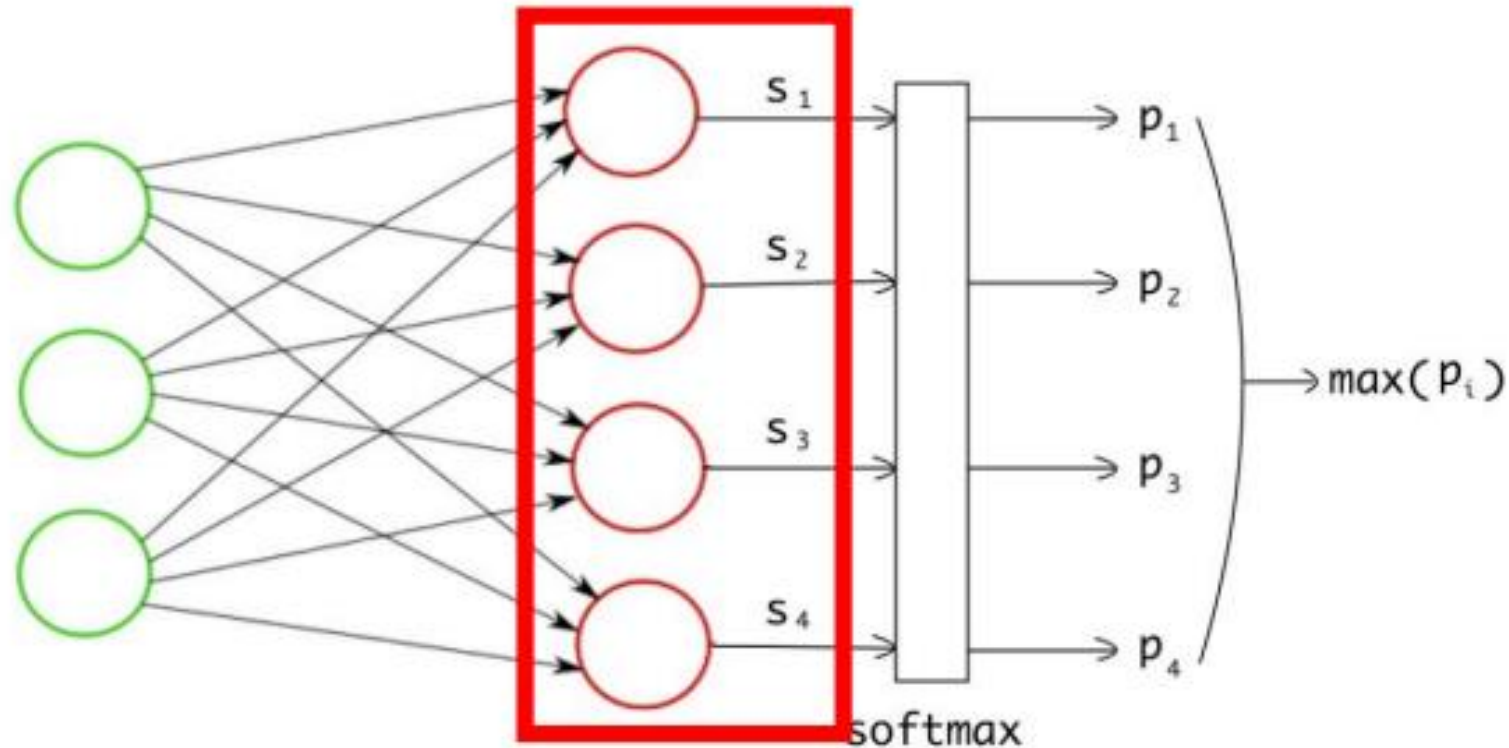
Number of classes

Want to divide each node's score by sum of all entries to make them sum to 1 (normalization)

Output Activations

- Softmax for Multiclass Classification

Converts vector of **scores** into a probability distribution that sums to 1; e.g.,



Output Activations

- Softmax for Multiclass Classification

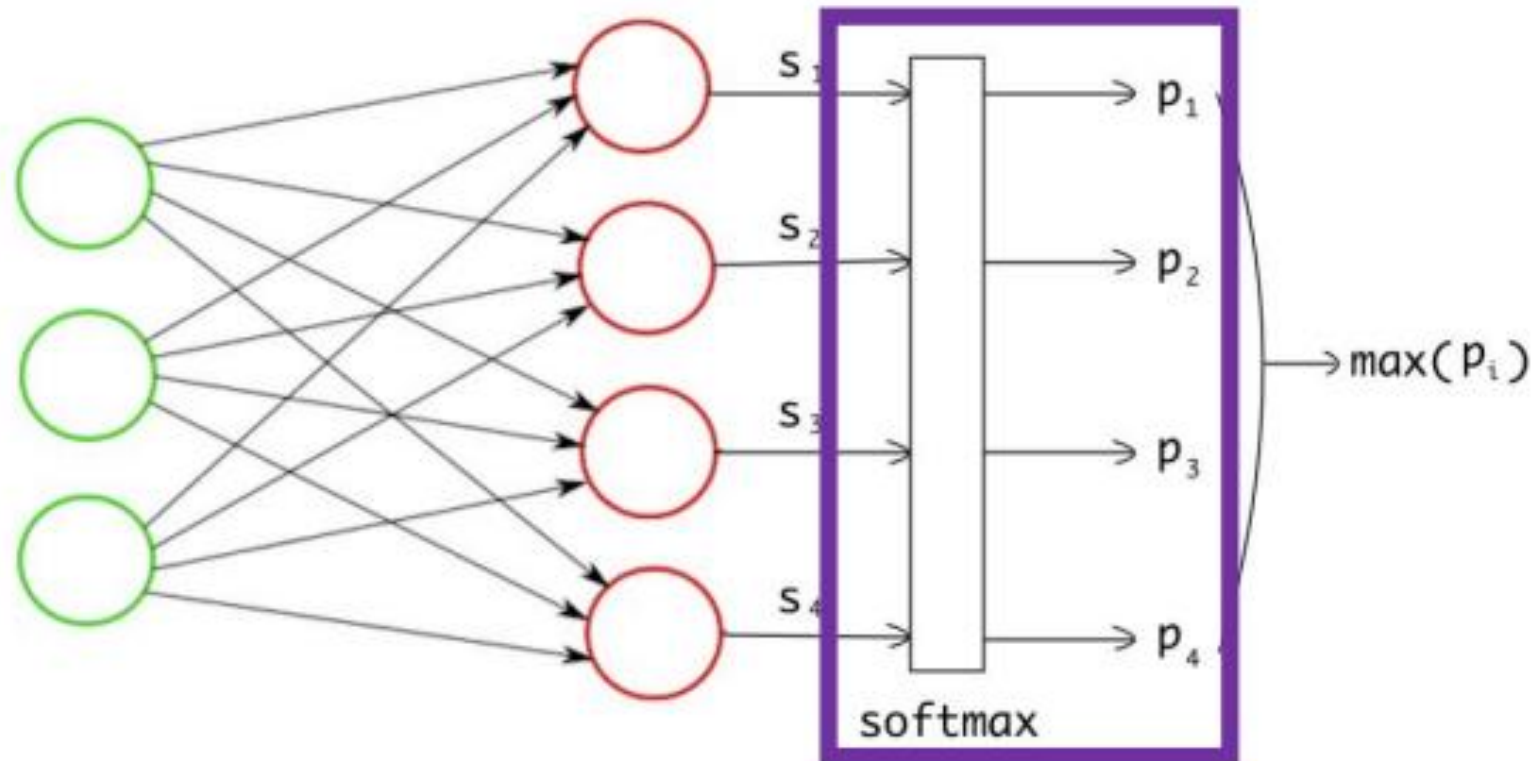
Converts vector of **scores** into a probability distribution that sums to 1; e.g.,

	Scoring Function
Dog	-3.44
Cat	1.16
Boat	-0.81
Airplane	3.91

Output Activations

- Softmax for Multiclass Classification

Converts vector of scores into a **probability distribution** that sums to 1; e.g.,



Output Activations

- Softmax for Multiclass Classification

Converts vector of scores into a **probability distribution** that sums to 1; e.g.,

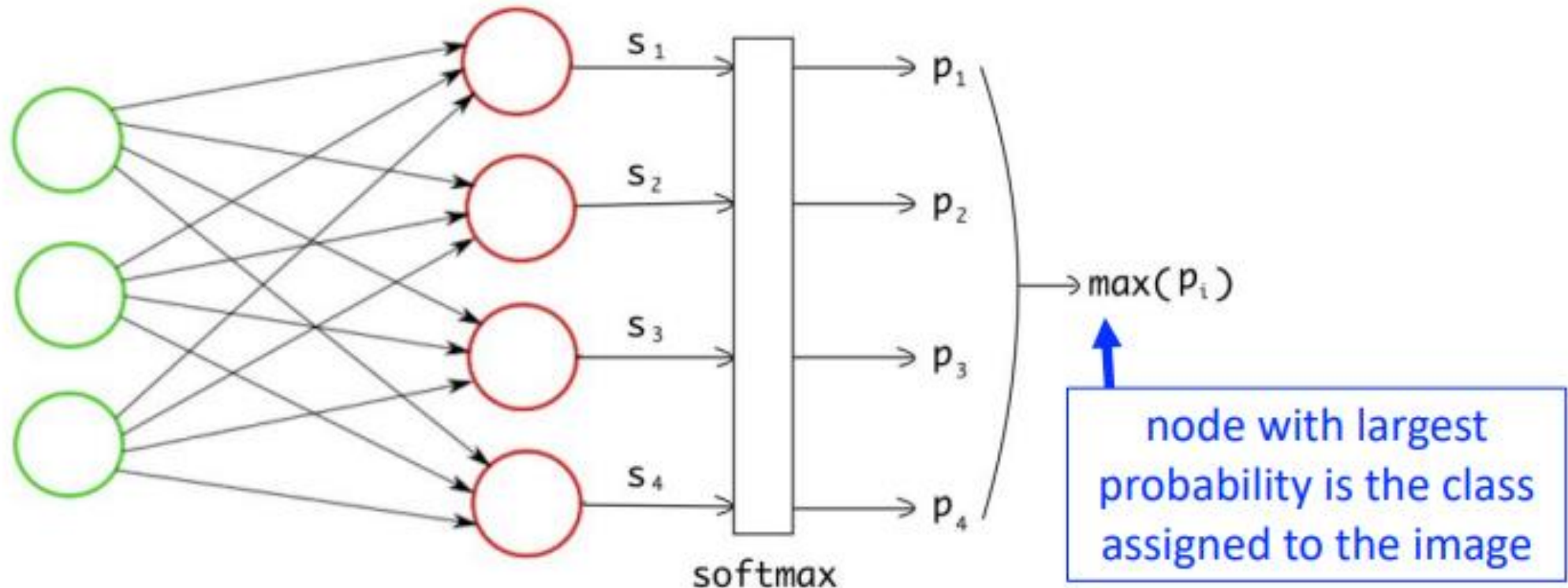
$$e^{z_i} \quad \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0596
Boat	-0.81	0.4449	0.0083
Airplane	3.91	49.8990	0.9315

Output Activations

- Softmax for Multiclass Classification

Converts vector of **scores** into a **probability distribution** that sums to 1; e.g.,



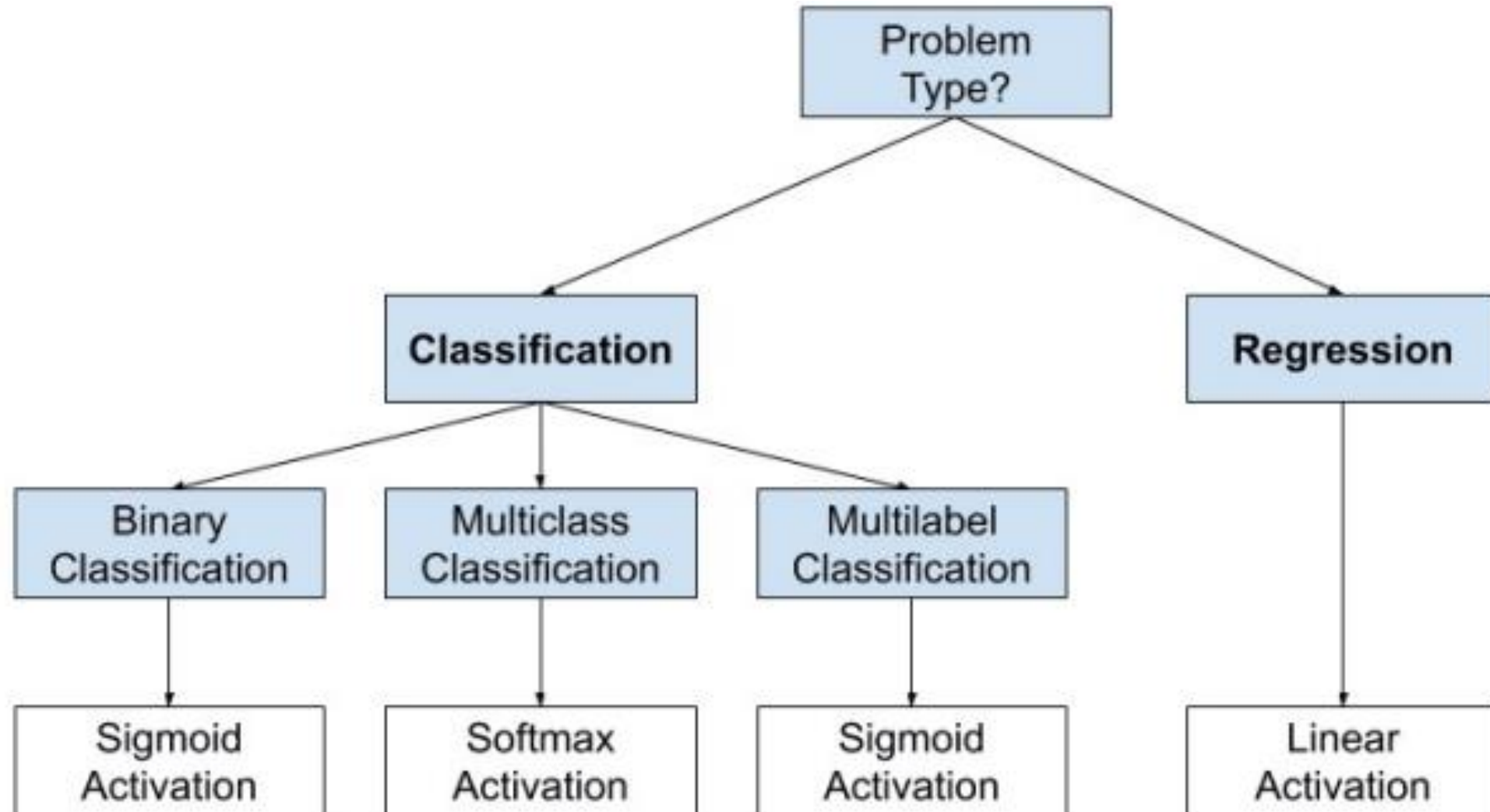
Output Activations

- Softmax for Multiclass Classification

Converts vector of scores into a probability distribution that sums to 1; e.g.,

	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0596
Boat	-0.81	0.4449	0.0083
Airplane	3.91	49.8990	0.9315

Output Activations



Example Problem

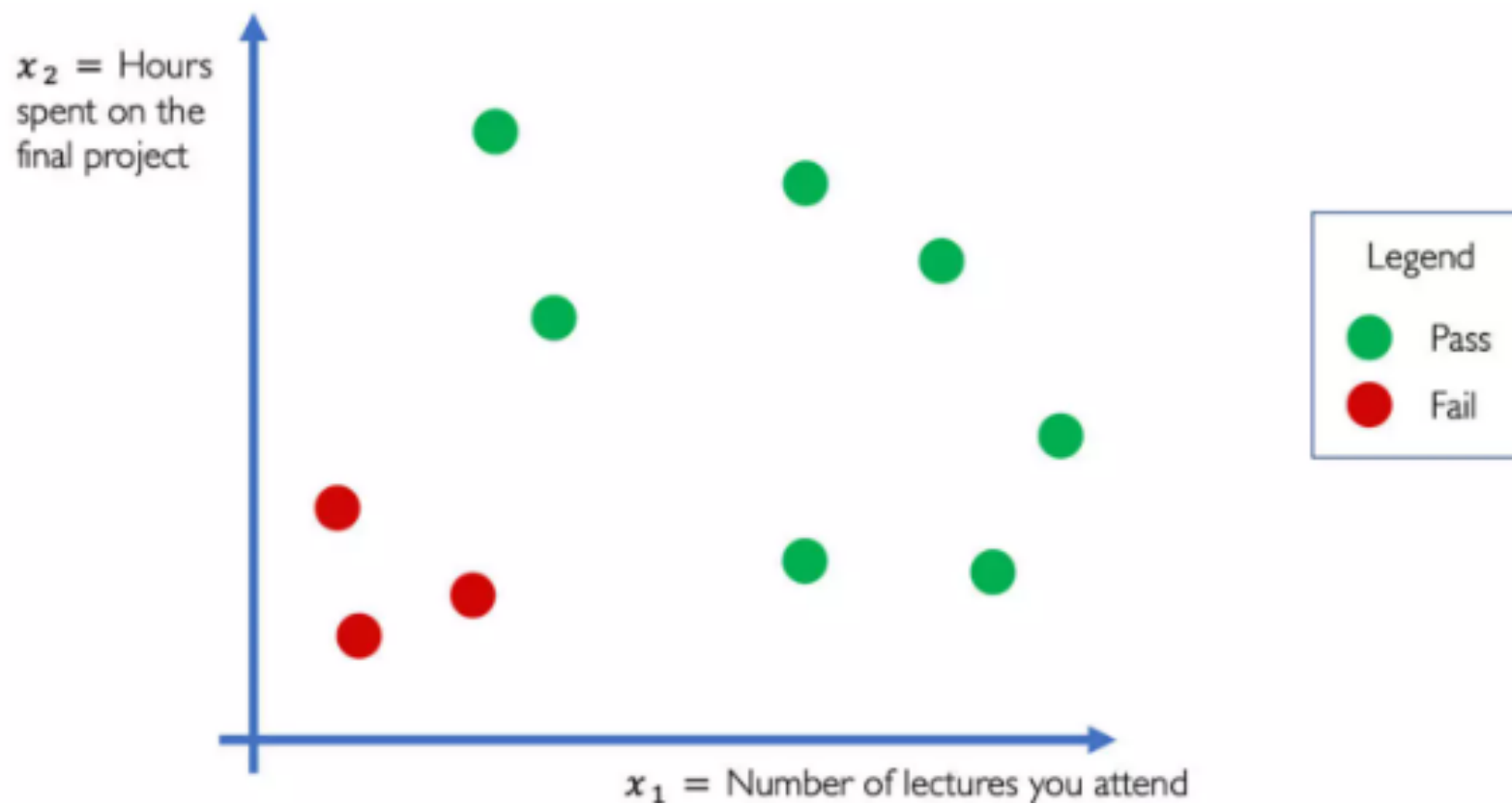
Will I pass this class?

Let's start with a simple two feature model

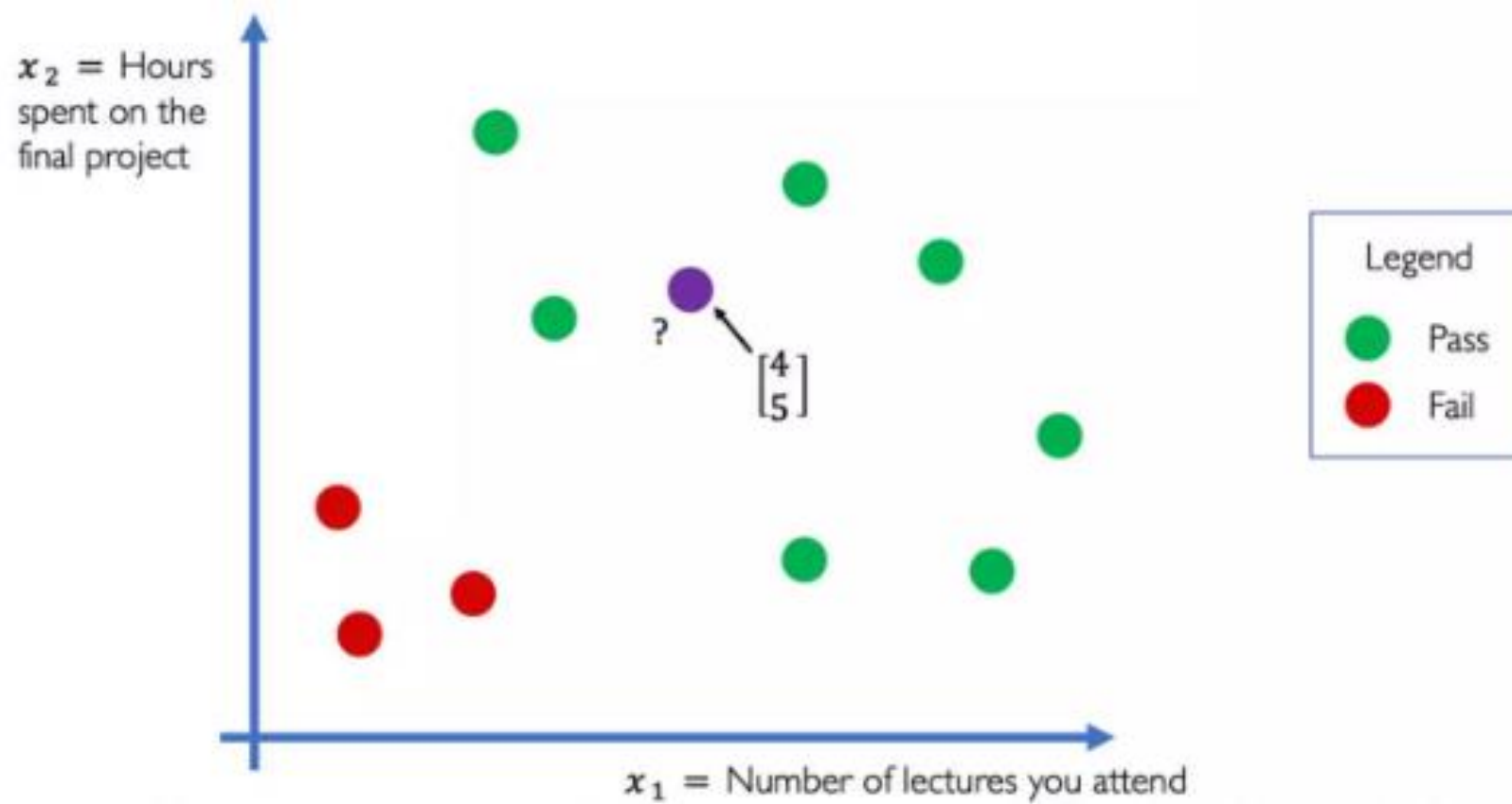
x_1 = Numbers of lectures you attend

x_2 = Hours spent on the final project

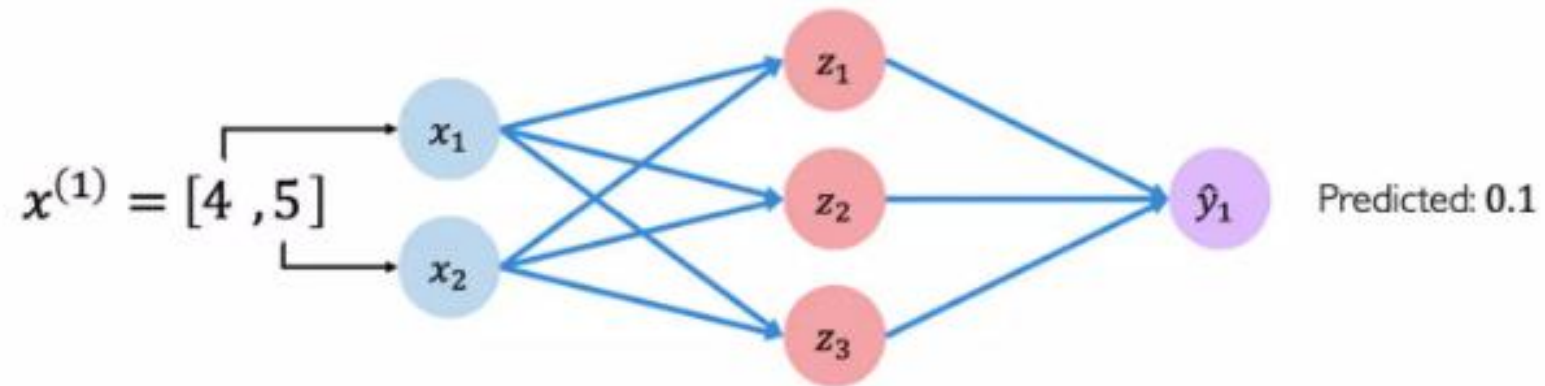
Example Problem: Will I pass this class?



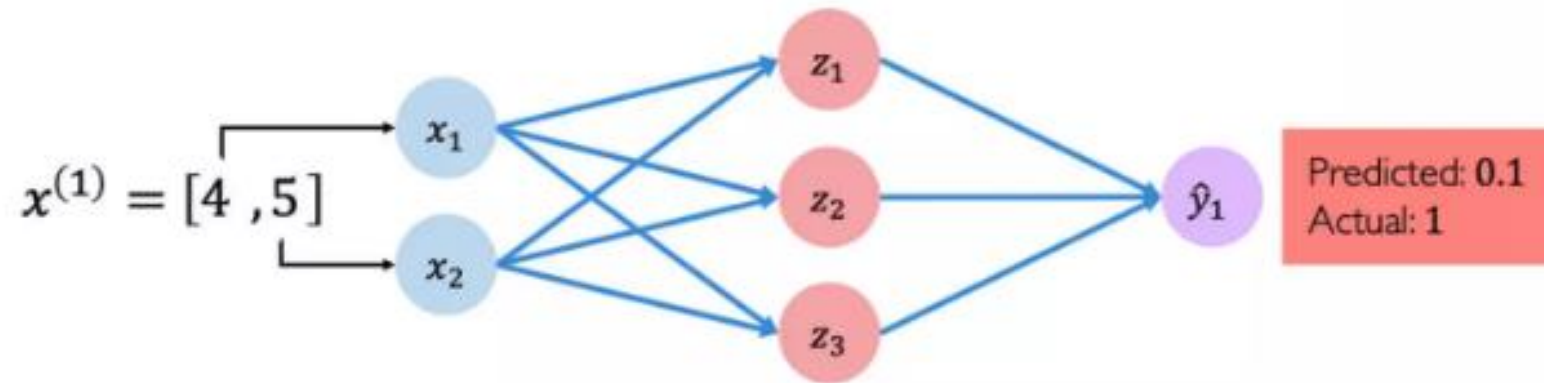
Example Problem: Will I pass this class?



Example Problem: Will I pass this class?



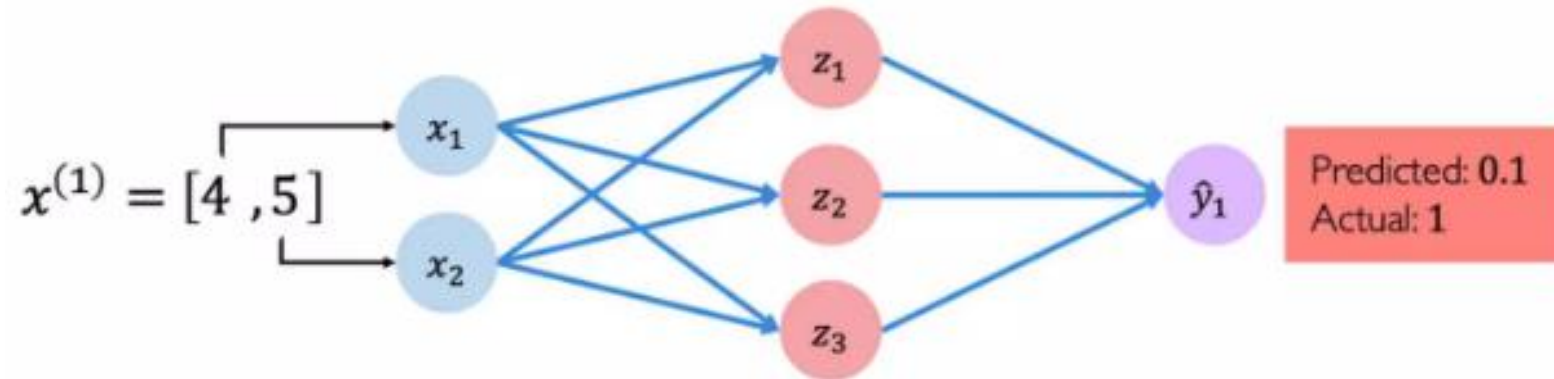
Example Problem: Will I pass this class?



Quantifying Loss



The **loss** of our network measures the cost incurred from incorrect predictions

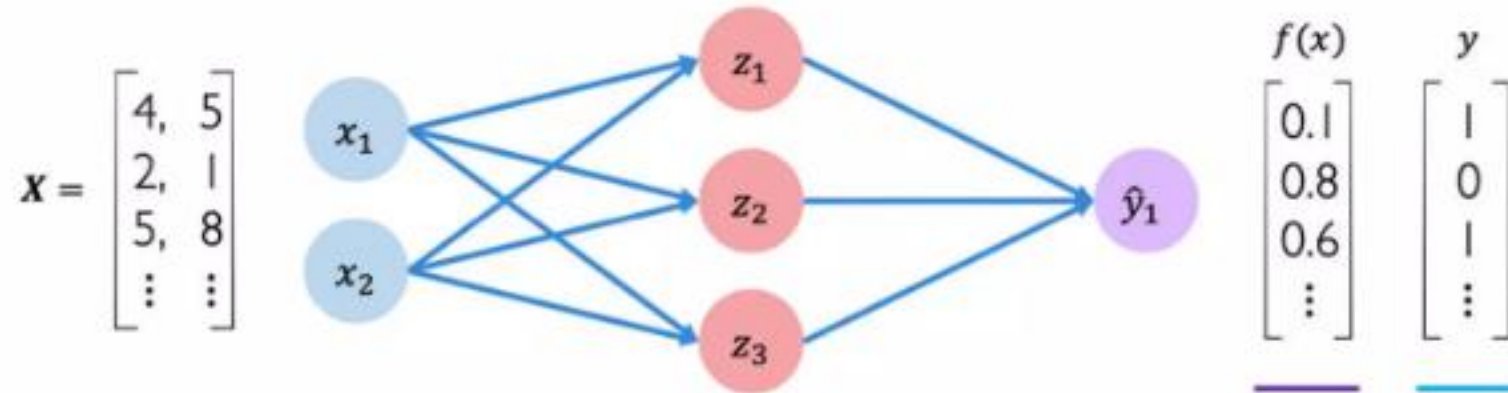


$$\mathcal{L}(\underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Empirical Loss



The **empirical loss** measures the total loss over our entire dataset



Also known as:

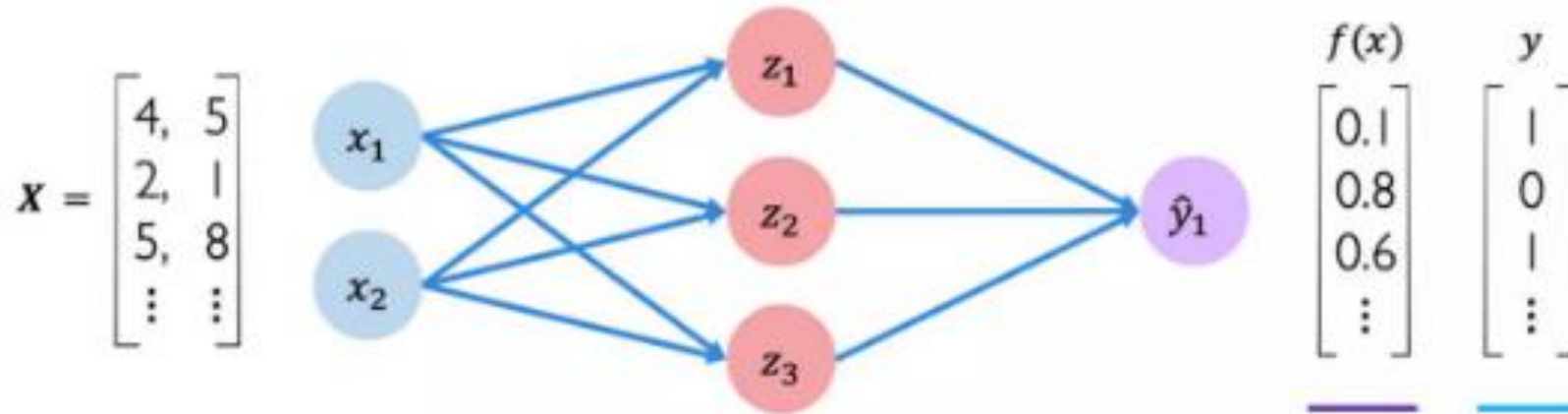
- Objective function
- Cost function
- Empirical Risk

$$J(W) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\underbrace{f(x^{(i)}; W)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Binary Cross Entropy Loss



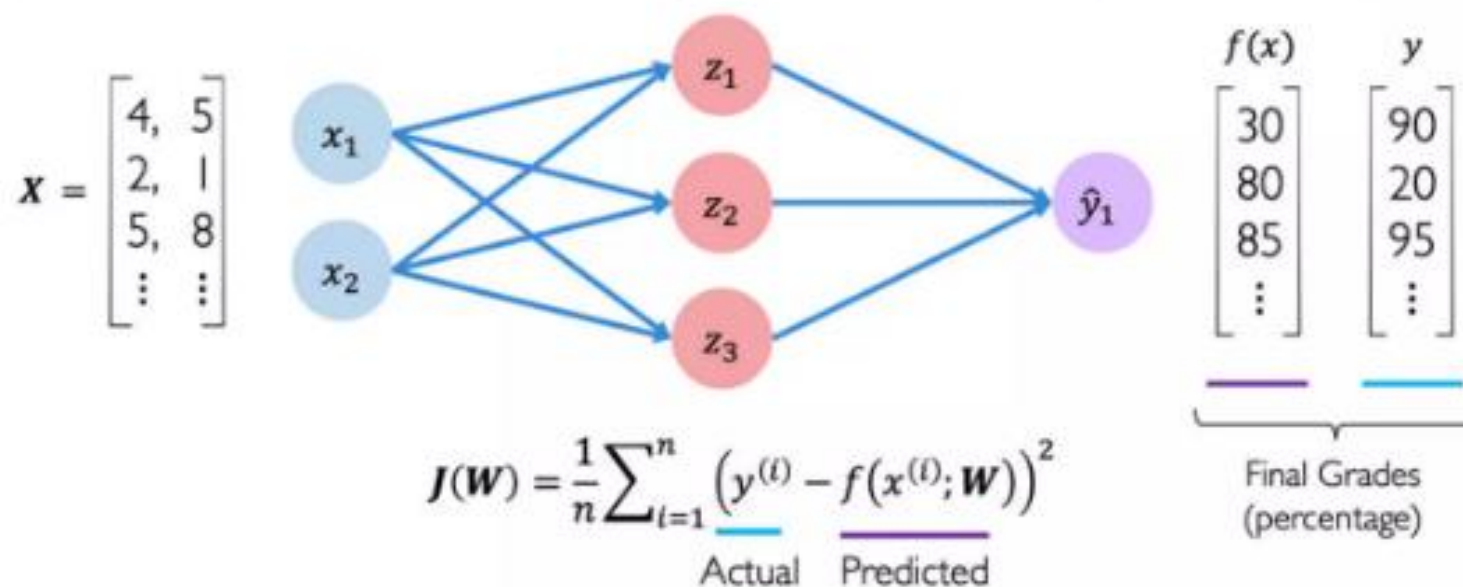
Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(W) = \frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)}}_{\text{Actual}} \log \left(\underbrace{f(x^{(i)}; W)}_{\text{Predicted}} \right) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f(x^{(i)}; W)}_{\text{Predicted}} \right)$$

Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers



The Perceptron from Scratch

- Let's implement the perceptron architecture from scratch using only numpy.
- Use the *perceptron.py* script as a starting point.