

# Feedforward Neural Networks

**Danna Gurari**

University of Colorado Boulder

Fall 2022



<https://home.cs.colorado.edu/~DrG/Courses/NeuralNetworksAndDeepLearning/AboutCourse.html>

# Review

- Last lecture:
  - Binary classification applications
  - Evaluating classification models
  - Biological neurons: inspiration
  - Artificial neuron: Perceptron
- Assignments (Canvas):
  - Problem set 1 due earlier today (should receive grades in ~1 week)
  - Lab assignment 1 due in 1.5 weeks (quick review of expectations)
- Questions?

# Today's Topics

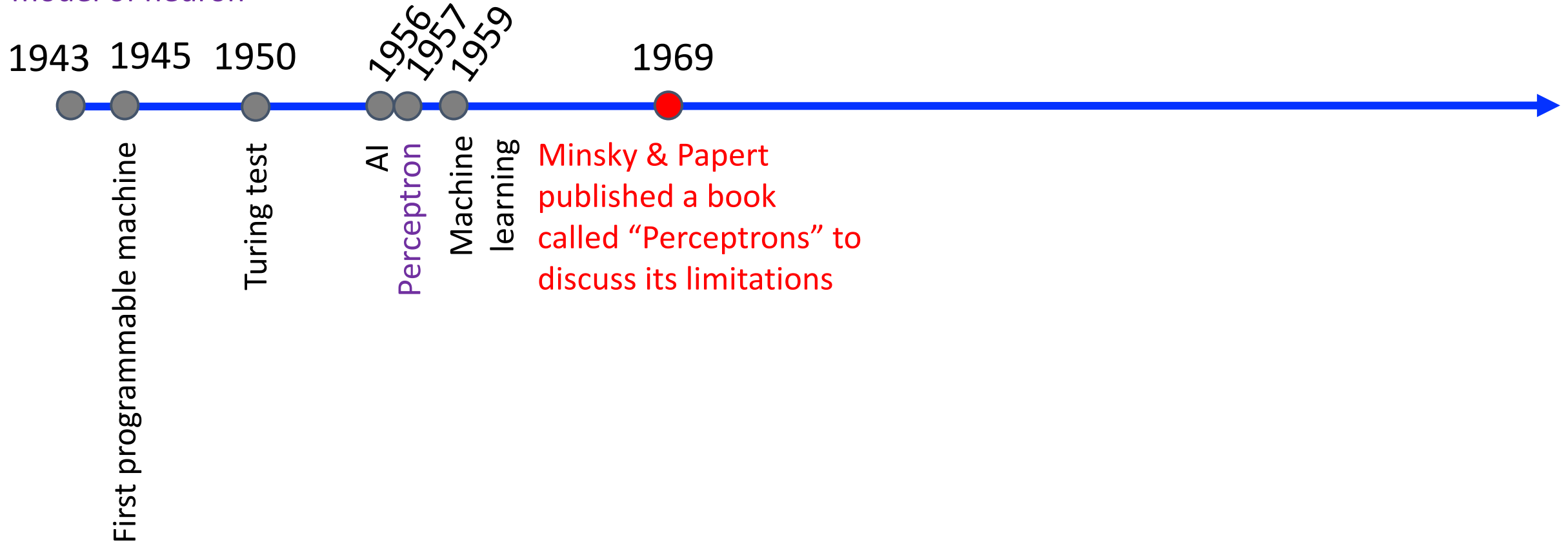
- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- Neural network architecture: activation functions
- Neural network architecture: output units
- Programming tutorial

# Today's Topics

- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- Neural network architecture: activation functions
- Neural network architecture: output units
- Programming tutorial

# Historical Context: Artificial Neurons

First mathematical  
model of neuron



# Recall: Vision for Perceptron



Frank Rosenblatt  
(Psychologist)

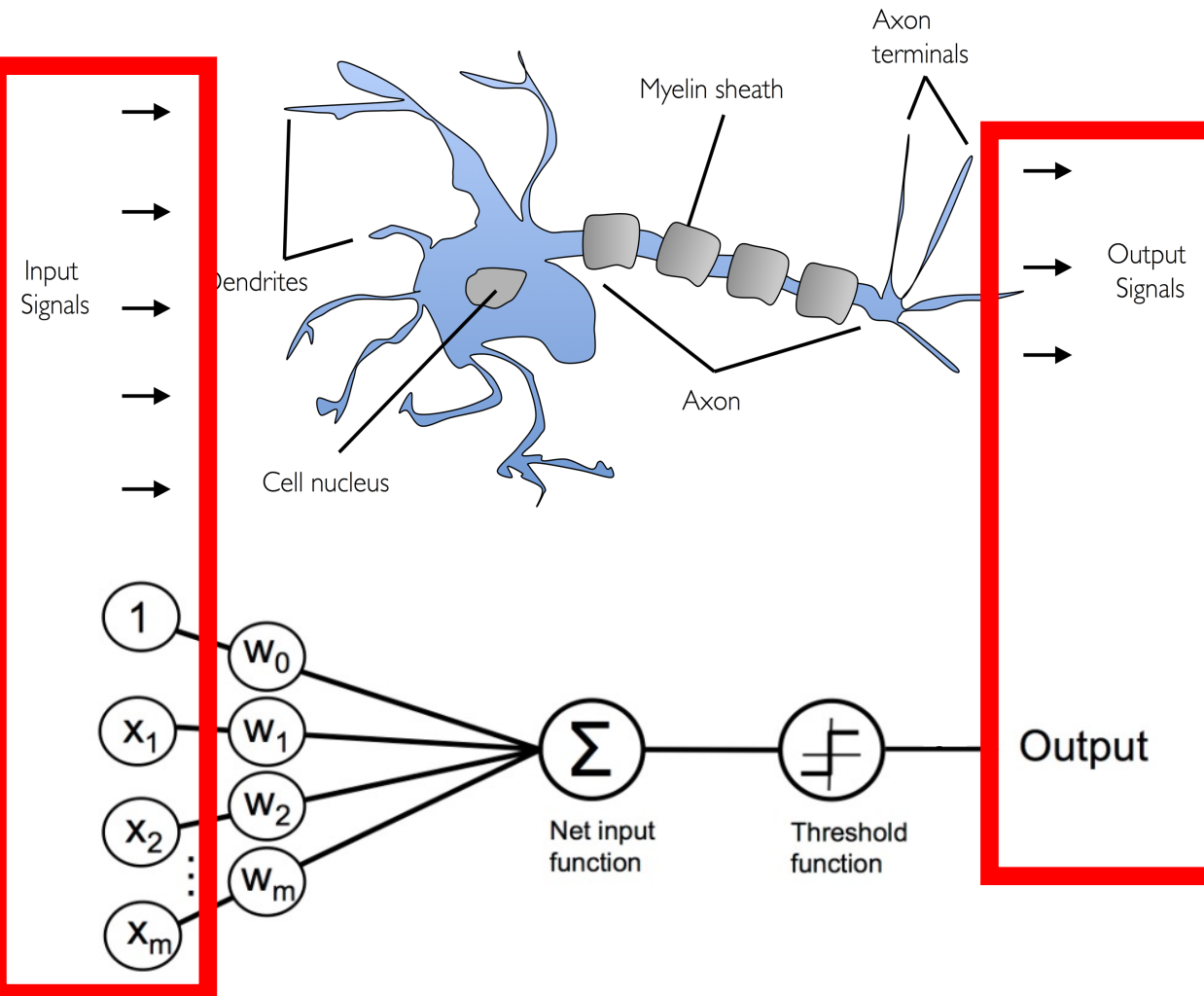
*“[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.... [It] is expected to be finished in about a year at a cost of \$100,000.”*

1958 New York Times article: <https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html>

[https://en.wikipedia.org/wiki/Frank\\_Rosenblatt](https://en.wikipedia.org/wiki/Frank_Rosenblatt)

# Recall: Perceptron

Biological Neuron:



Python Machine Learning; Raschka & Mirjalili

<https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch02/ch02.ipynb>

# 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

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

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
0	1	?
1	0	?
1	1	?

# 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
0	1	1
1	0	?
1	1	?

# 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
0	1	1
1	0	1
1	1	?

# Perceptron Limitation: XOR Problem

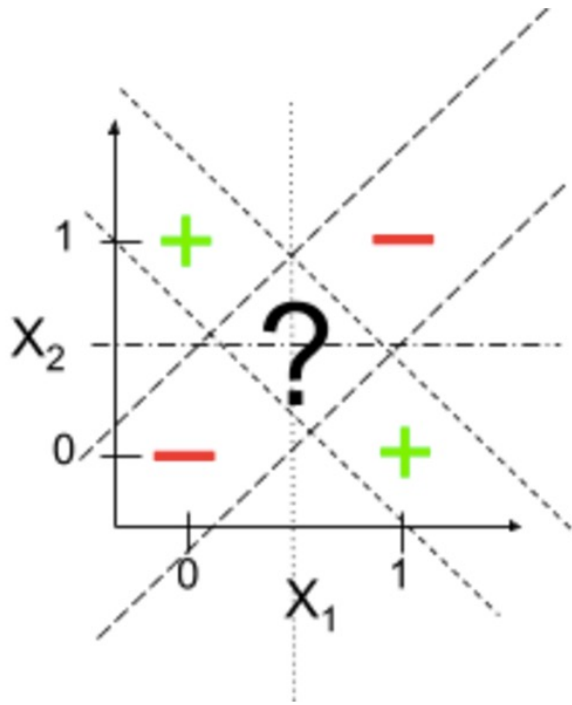
XOR = “Exclusive Or”

- 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
0	1	1
1	0	1
1	1	0

# 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

# Perceptron Limitation: XOR Problem



Frank Rosenblatt  
(Psychologist)

*“[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.... [It] is expected to be finished in*

**How can a machine be “conscious”  
when it can’t solve the XOR problem?**

new-

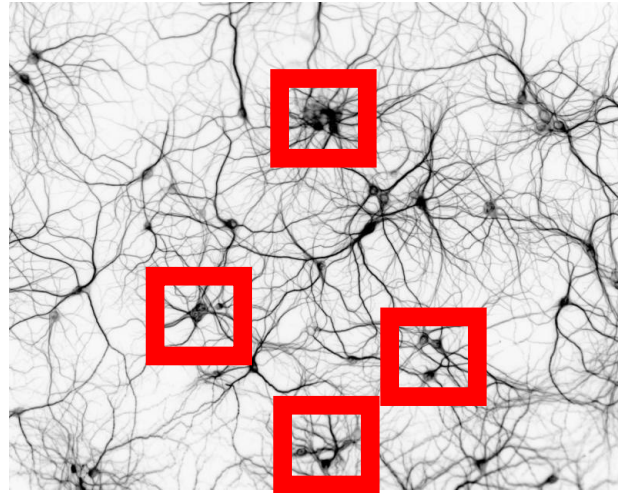
# Today's Topics

- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- Neural network architecture: activation functions
- Neural network architecture: output units
- Programming tutorial



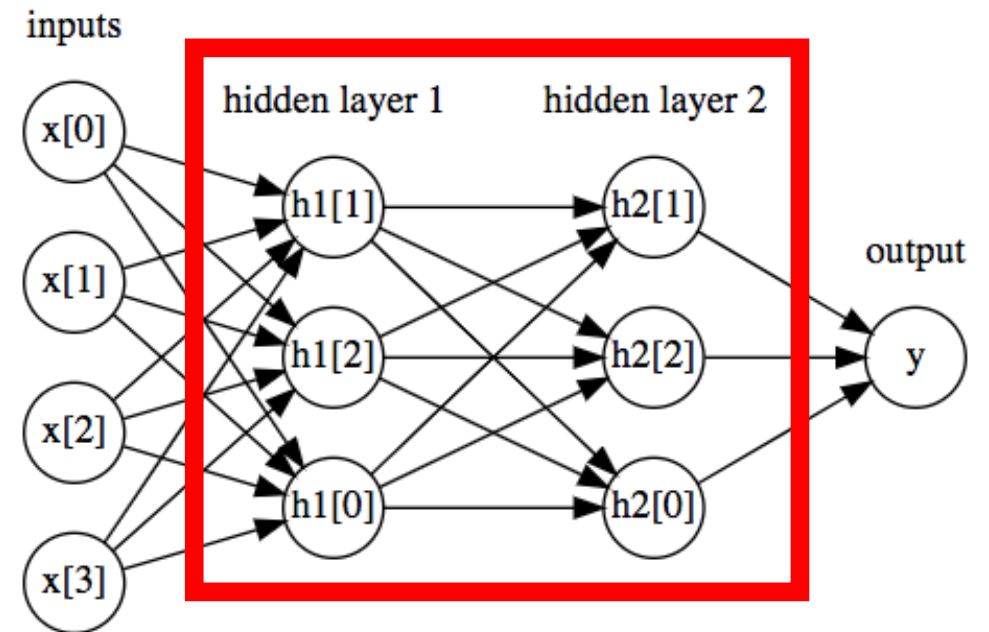
# Neural Networks: Connected Neurons

Biological Neural Network:



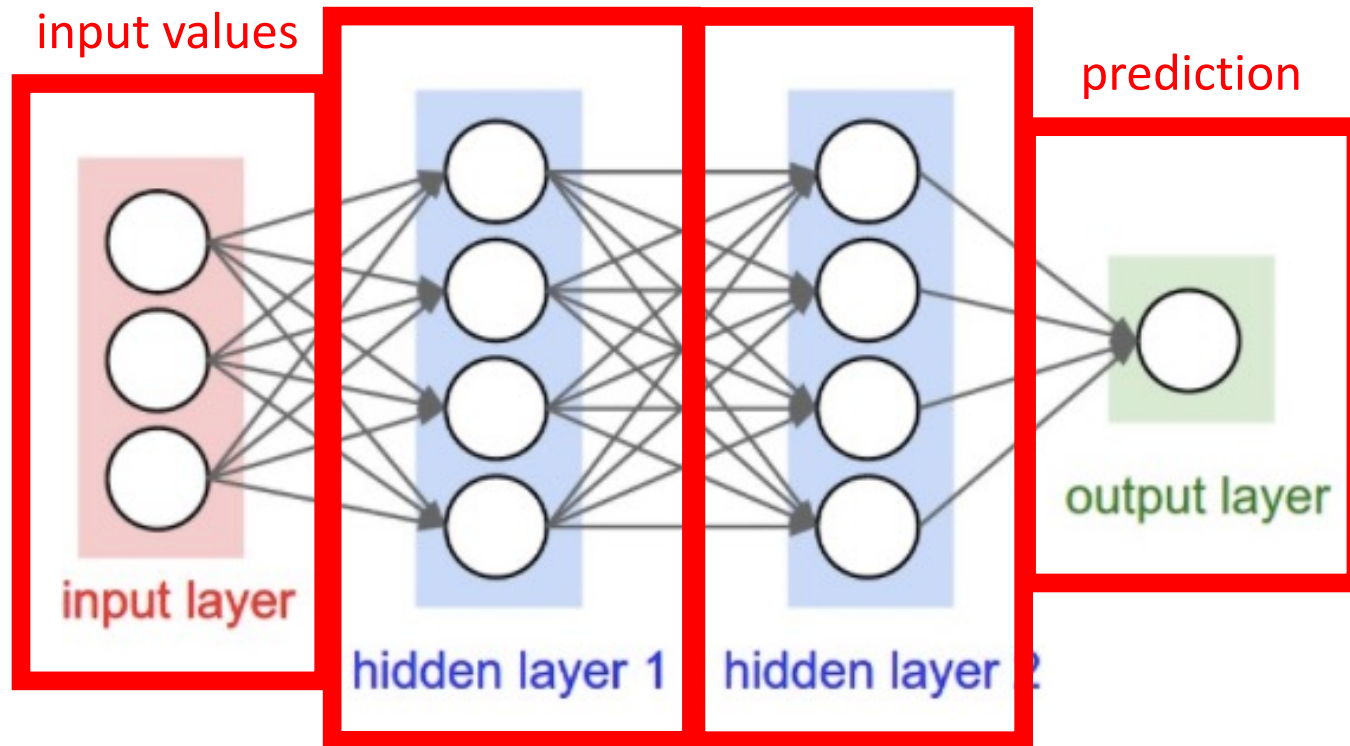
<http://www.rzagabe.com/2014/11/03/an-introduction-to-artificial-neural-networks.html>

Artificial Neural Network:



[https://github.com/amueller/introduction\\_to\\_ml\\_with\\_python/blob/master/02-supervised-learning.ipynb](https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb)

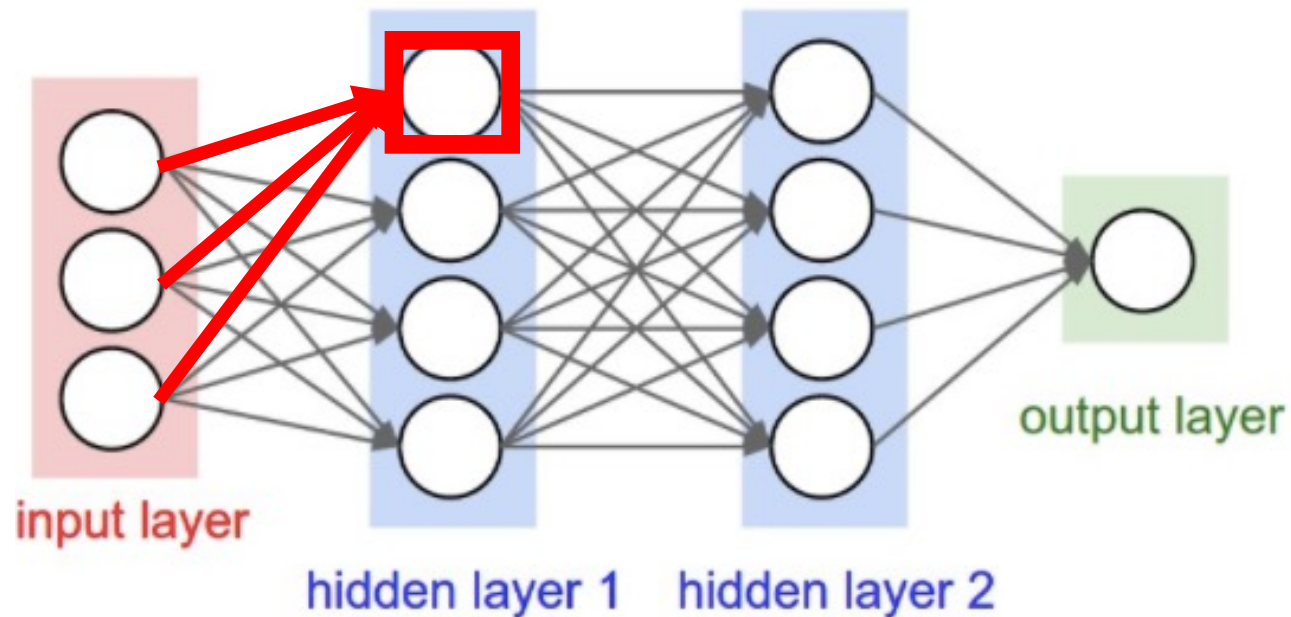
# Neural Network



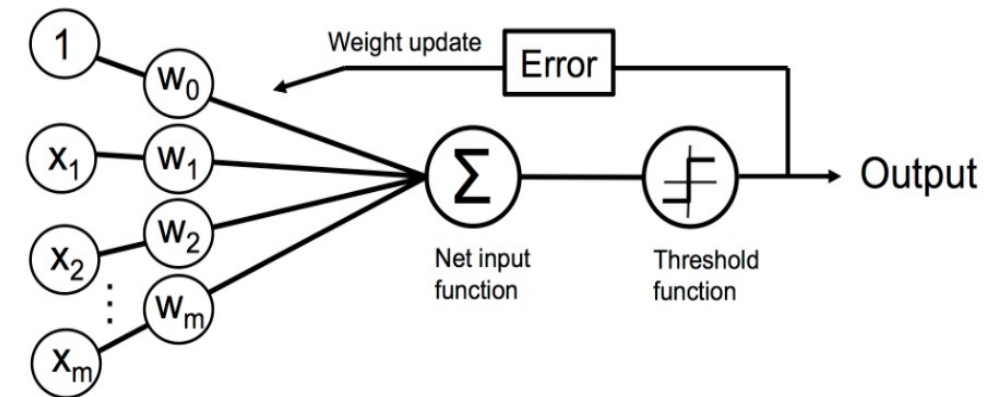
each “hidden layer” uses outputs of units (i.e., neurons) and provides them as inputs to other units (i.e., neurons)

This is a 3-layer neural network (i.e., count number of hidden layers plus output layer)

# Neural Network

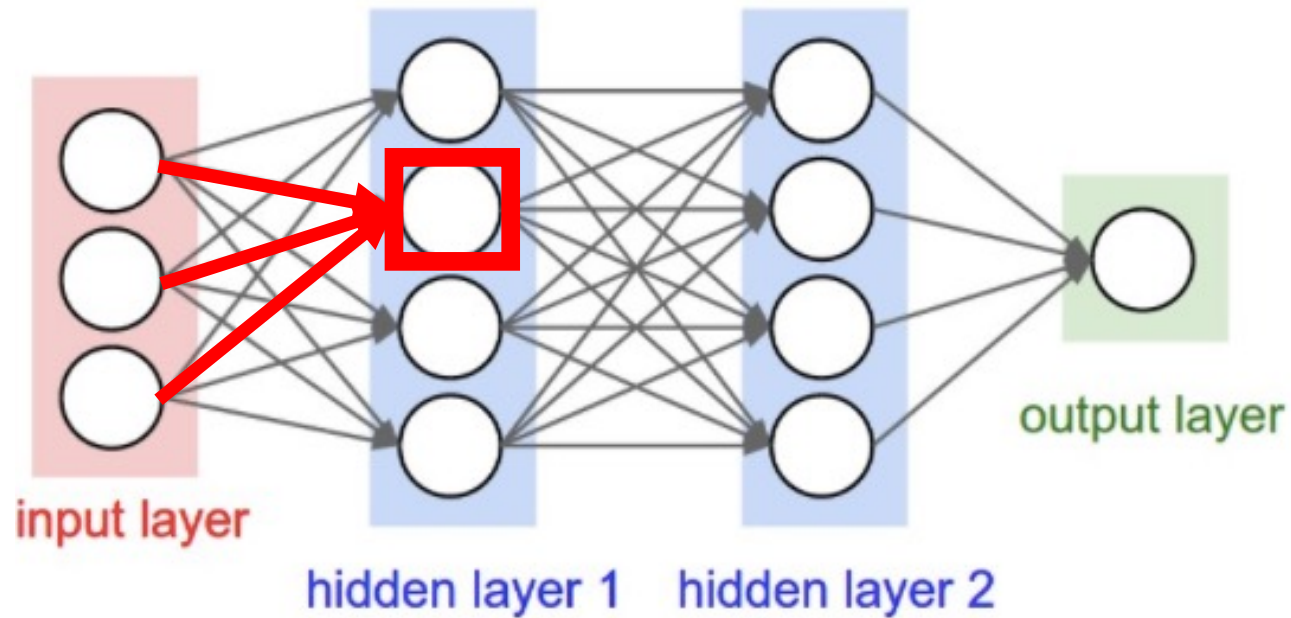


- How does this relate to a perceptron?

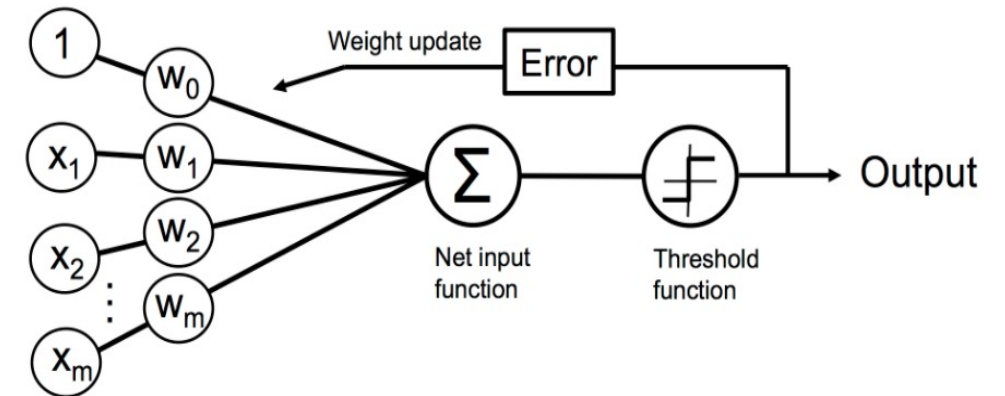


- Unit: takes as input a weighted sum and applies an activation function

# Neural Network

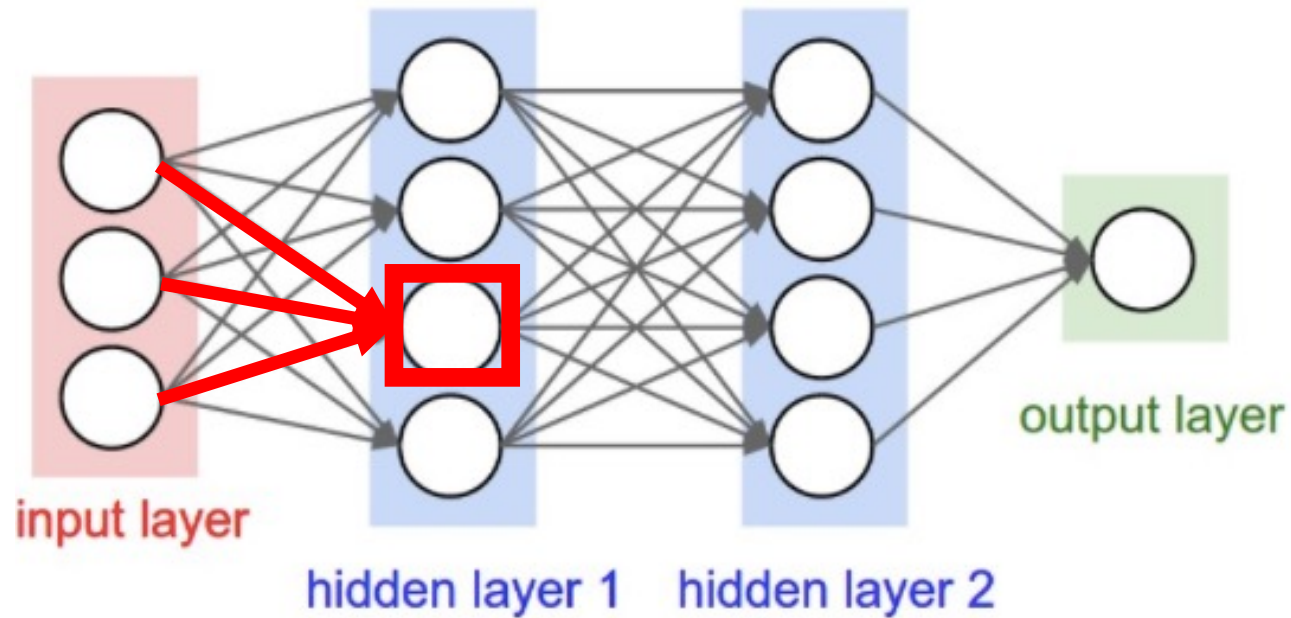


- How does this relate to a perceptron?

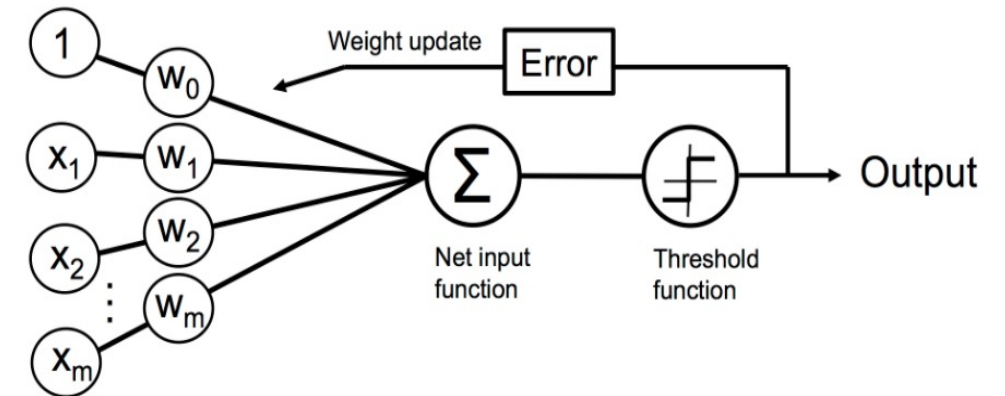


- Unit: takes as input a weighted sum and applies an activation function

# Neural Network



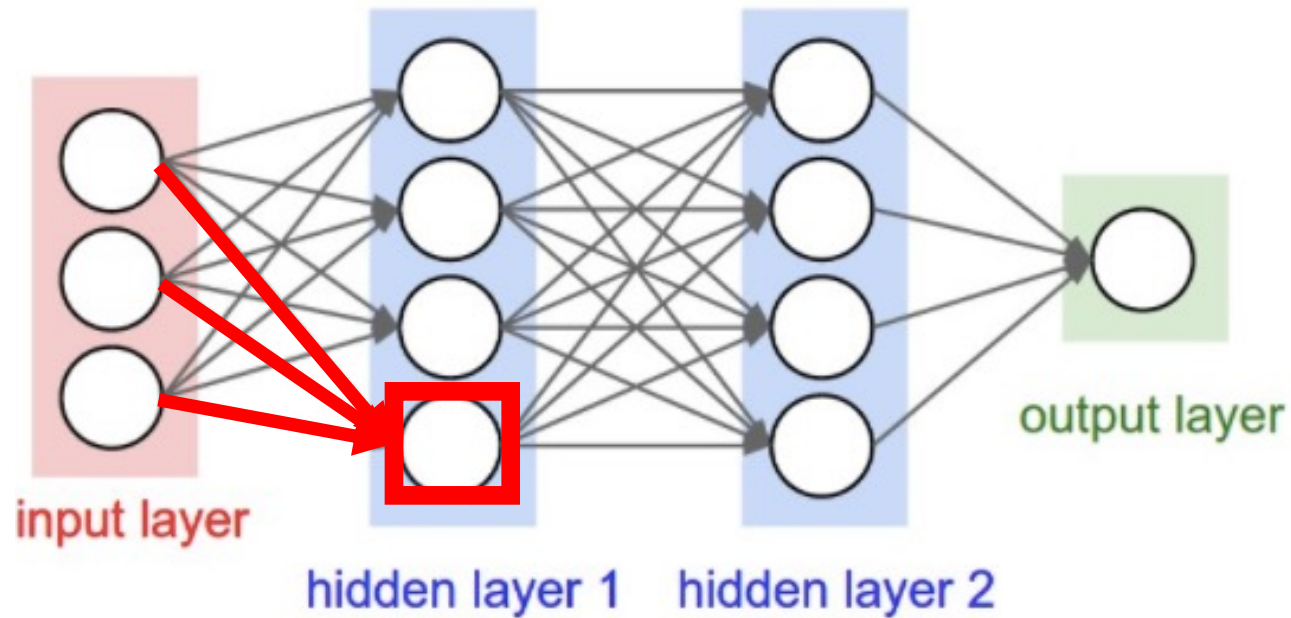
- How does this relate to a perceptron?



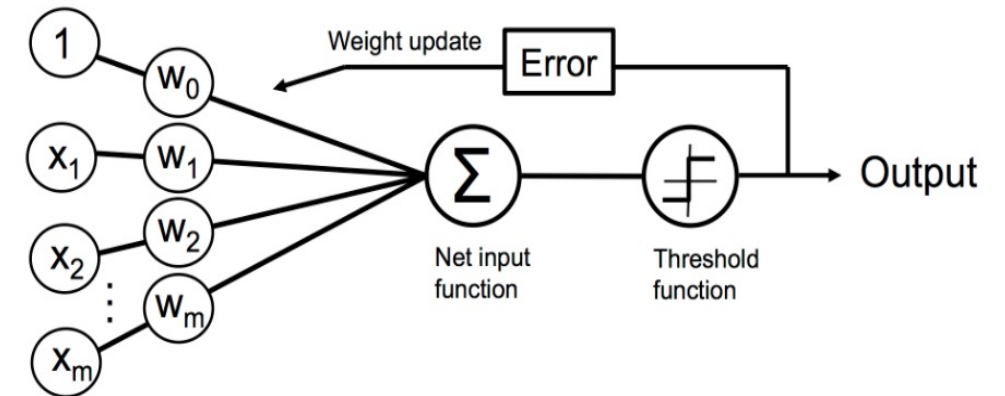
- Unit: takes as input a weighted sum and applies an activation function



# Neural Network

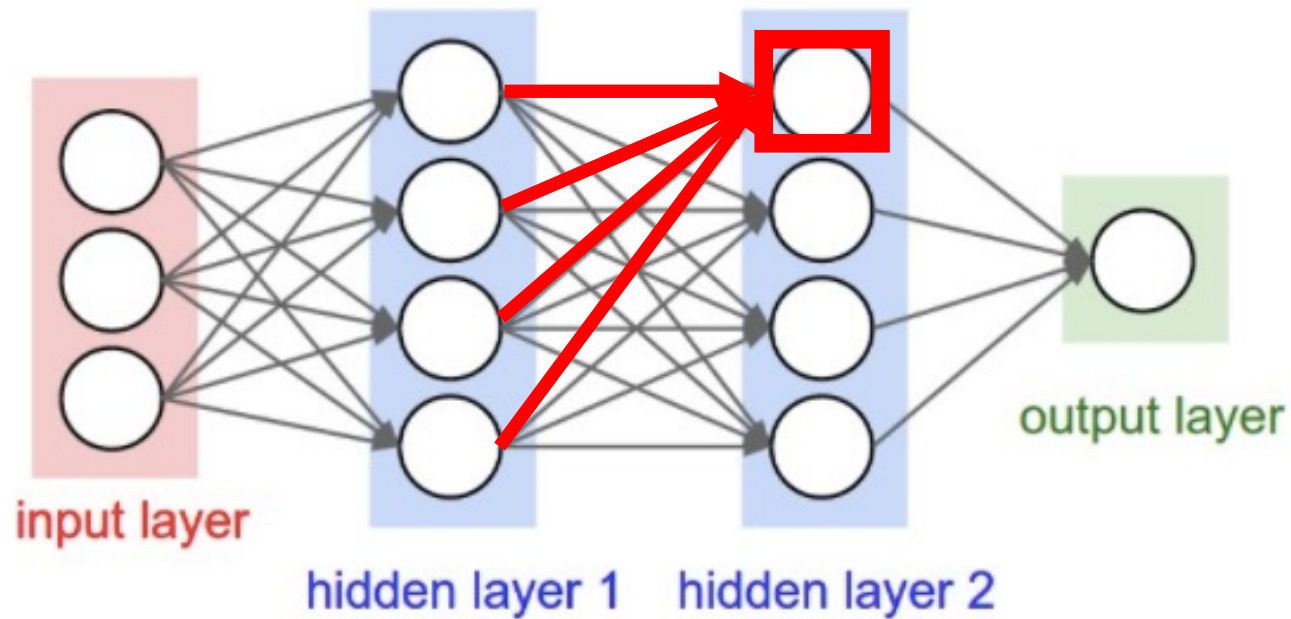


- How does this relate to a perceptron?

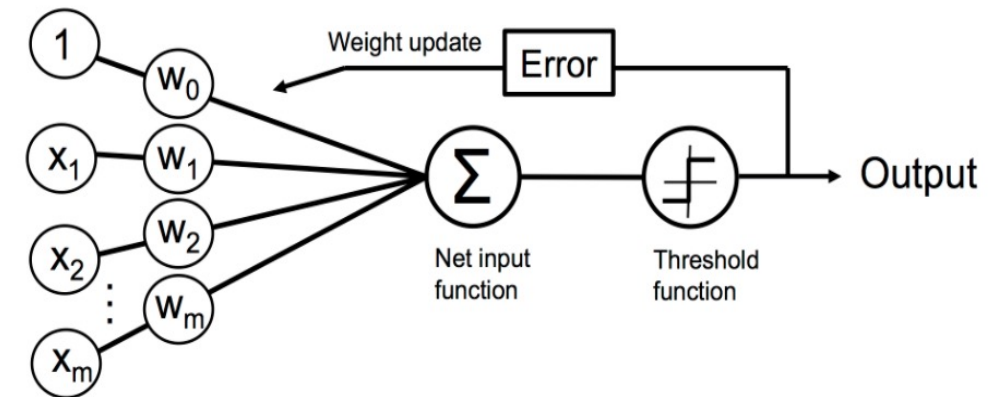


- Unit: takes as input a weighted sum and applies an activation function

# Neural Network

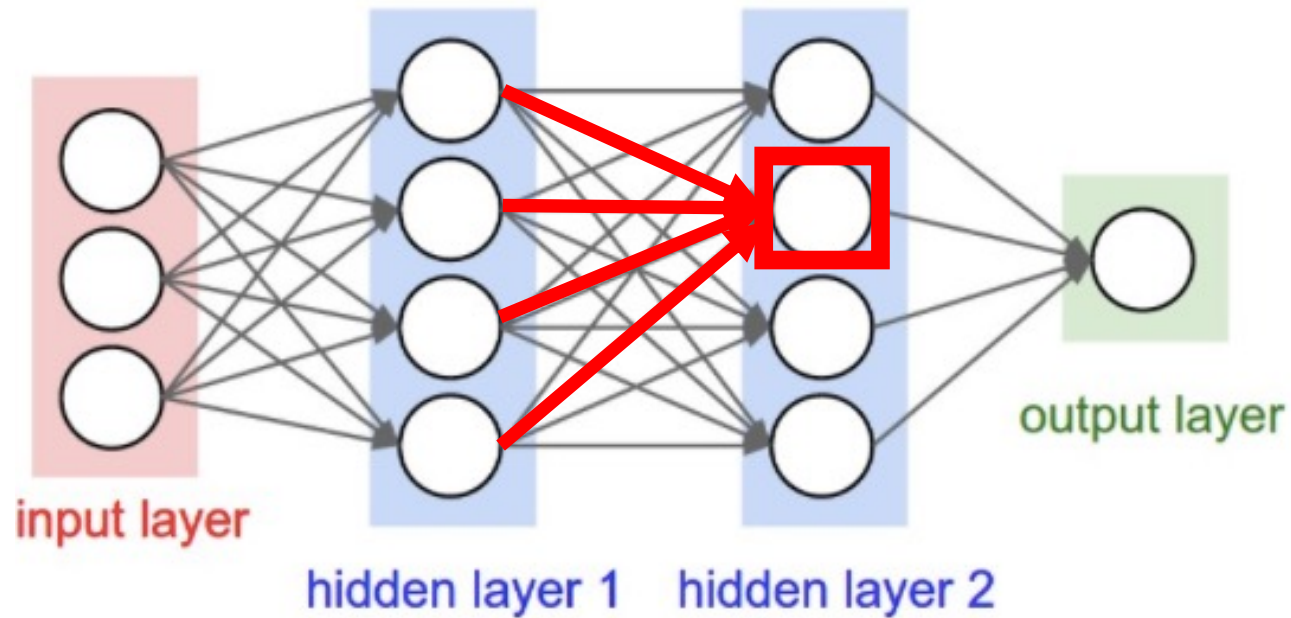


- How does this relate to a perceptron?

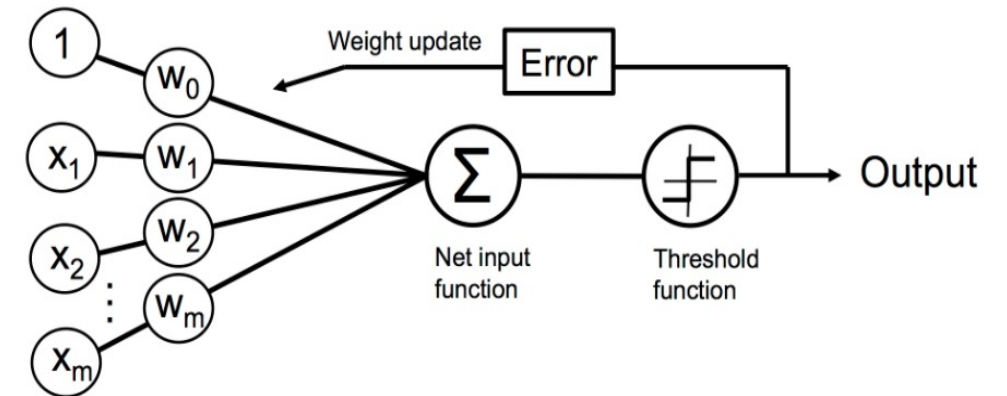


- Unit: takes as input a weighted sum and applies an activation function

# Neural Network



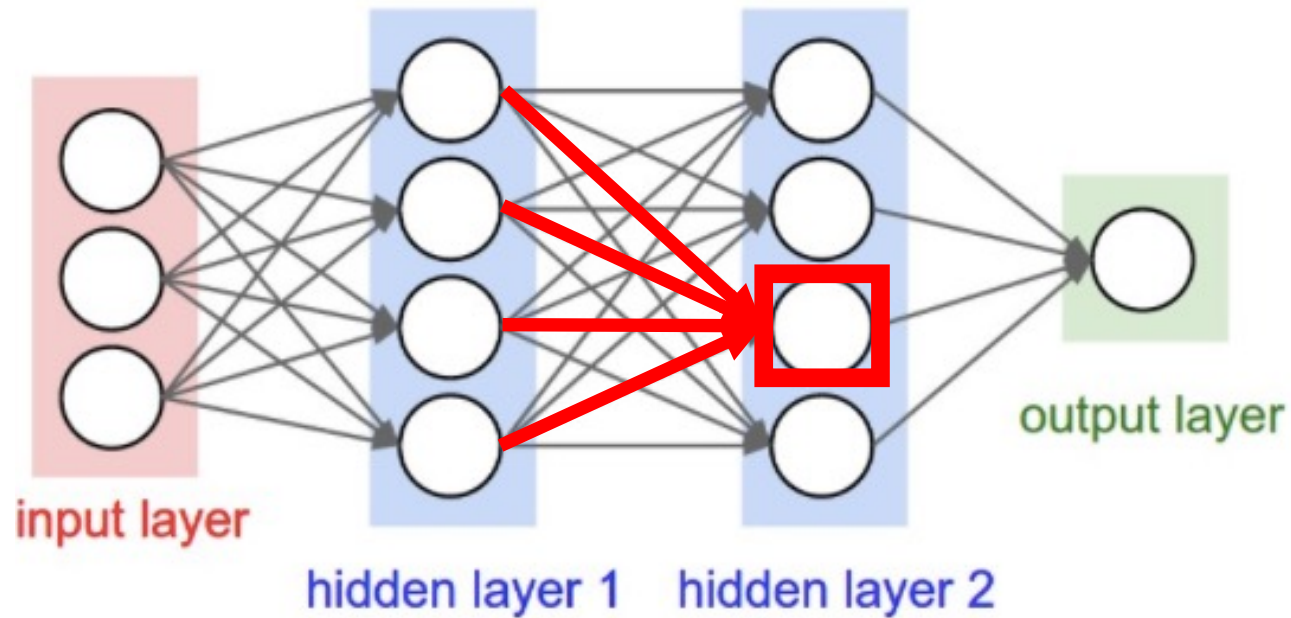
- How does this relate to a perceptron?



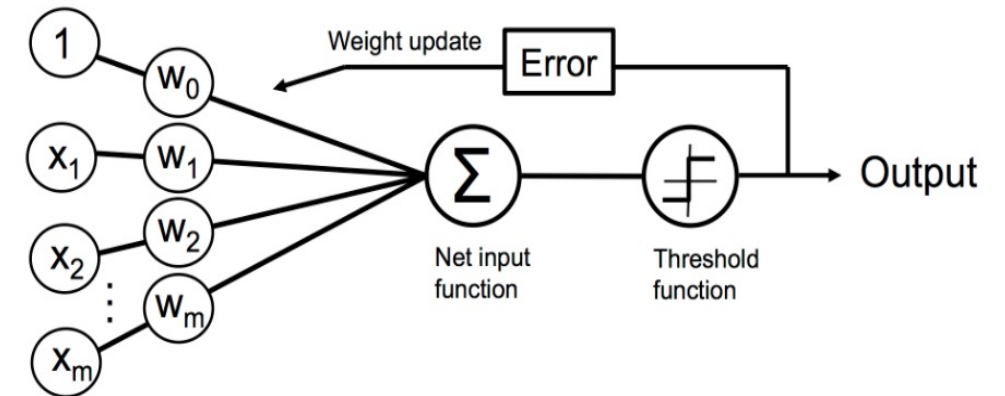
- Unit: takes as input a weighted sum and applies an activation function



# Neural Network

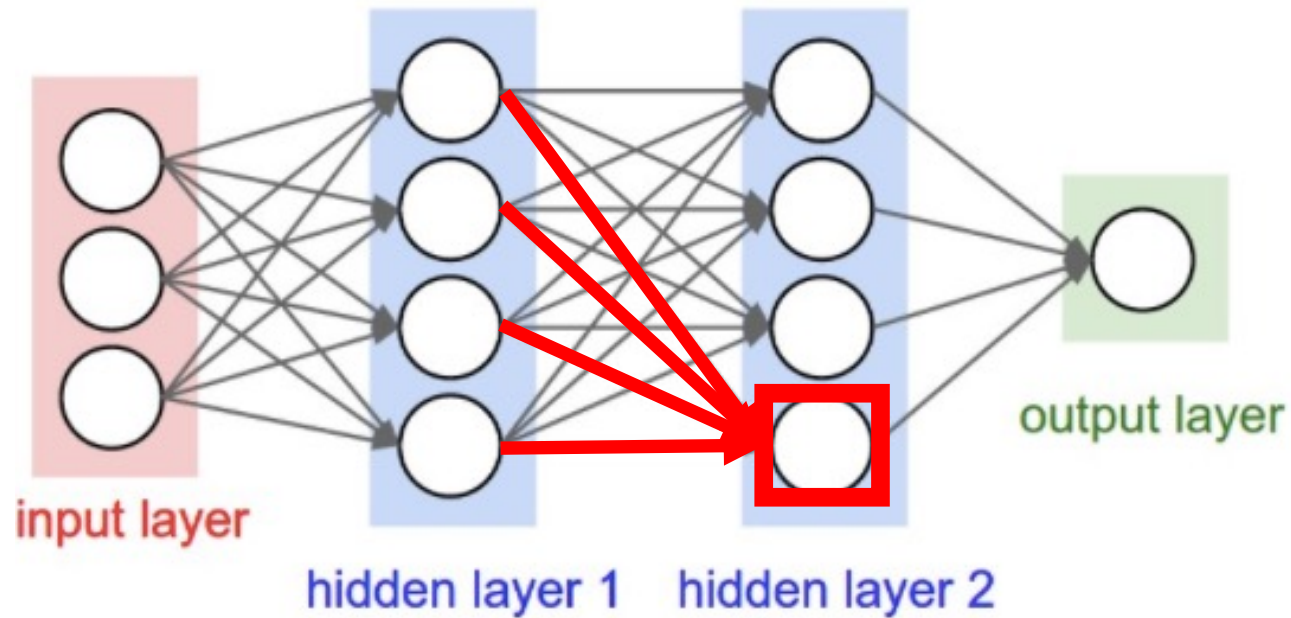


- How does this relate to a perceptron?

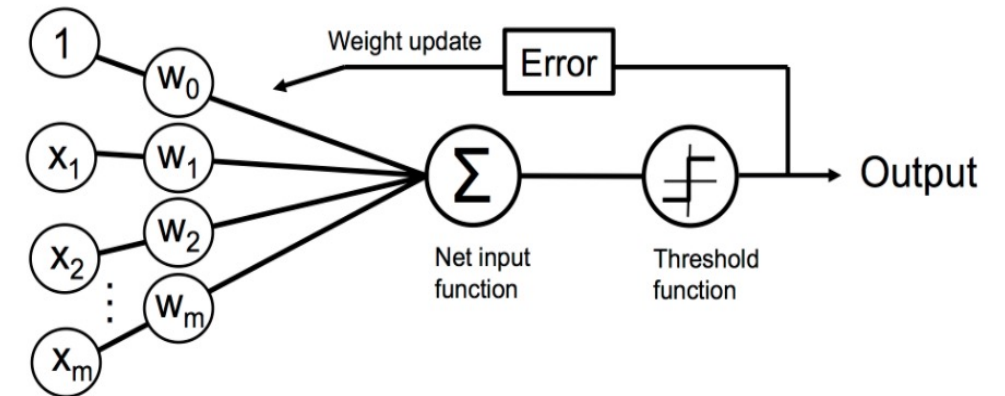


- Unit: takes as input a weighted sum and applies an activation function

# Neural Network

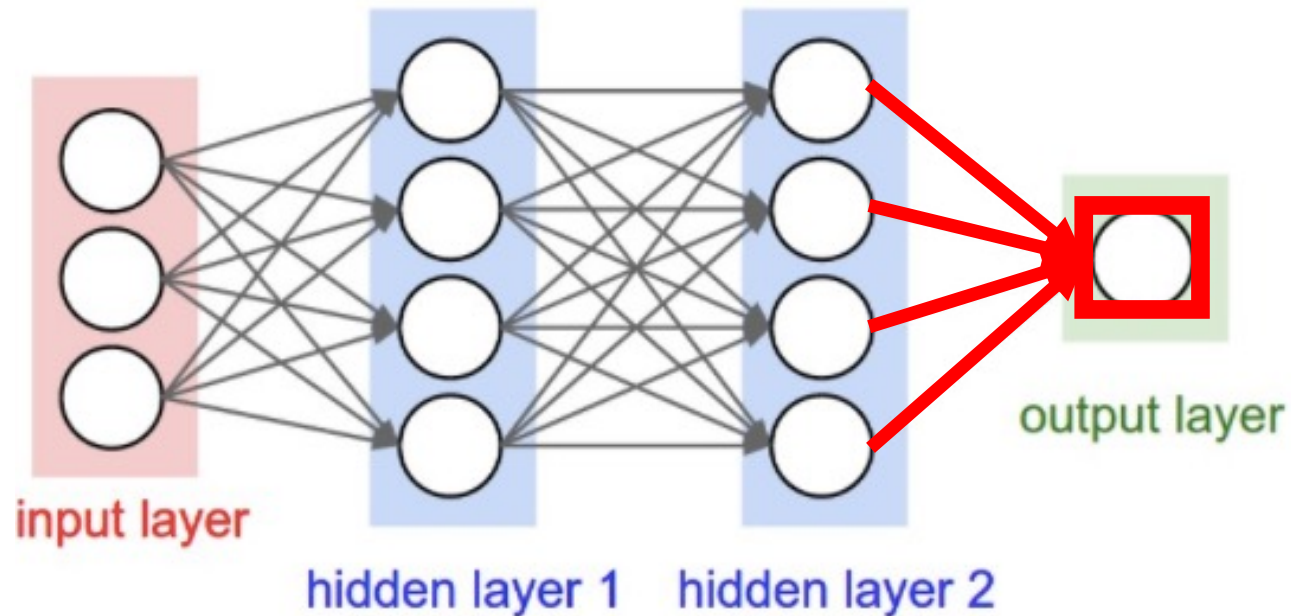


- How does this relate to a perceptron?

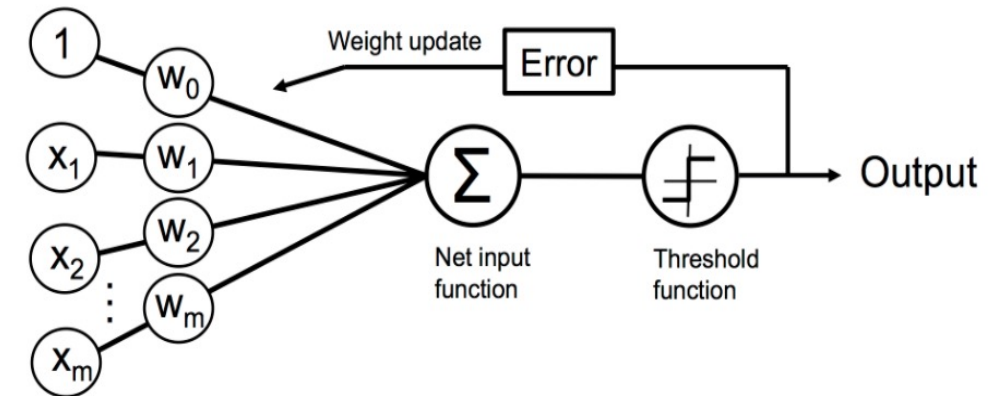


- Unit: takes as input a weighted sum and applies an activation function

# Neural Network

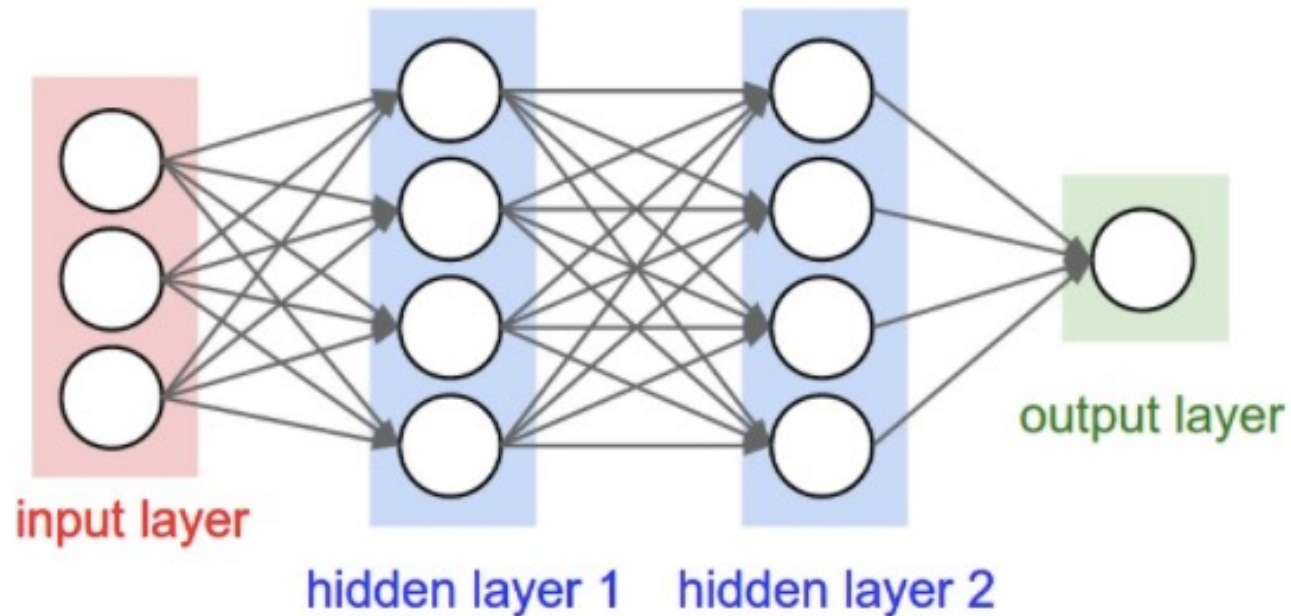


- How does this relate to a perceptron?



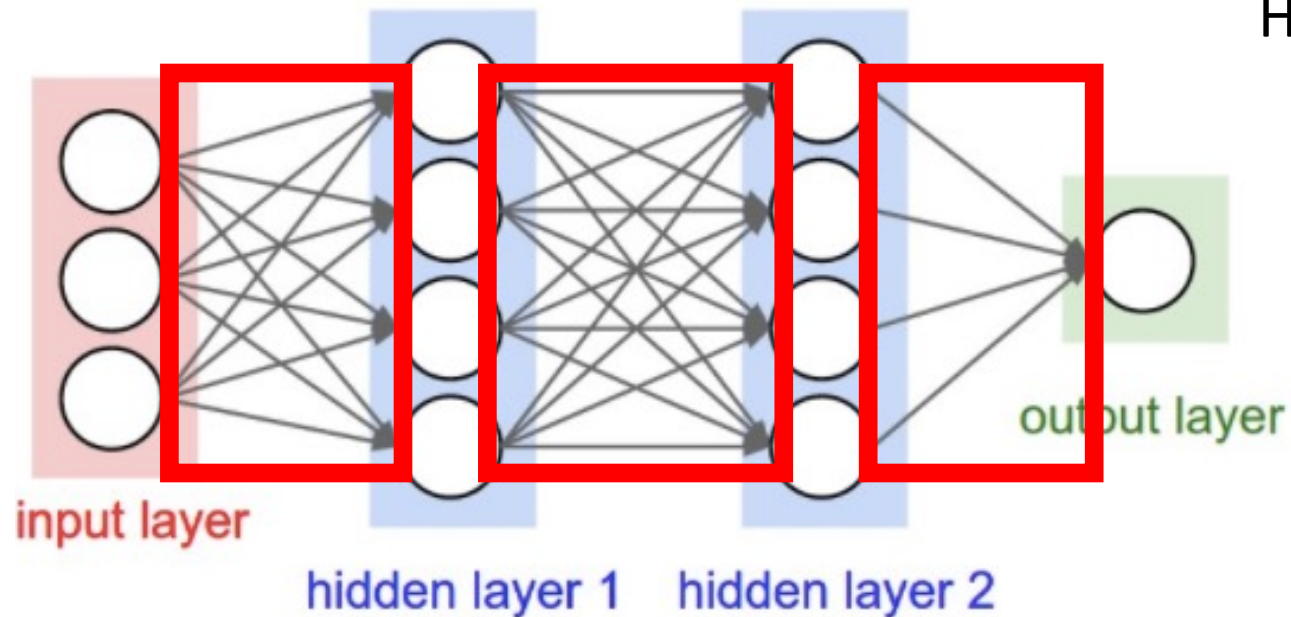
- Unit: takes as input a weighted sum and applies an activation function

# Neural Network



- **Training goal: learn model parameters**
- Layers are called “hidden” because algorithm decides how to use each layer to produce its output

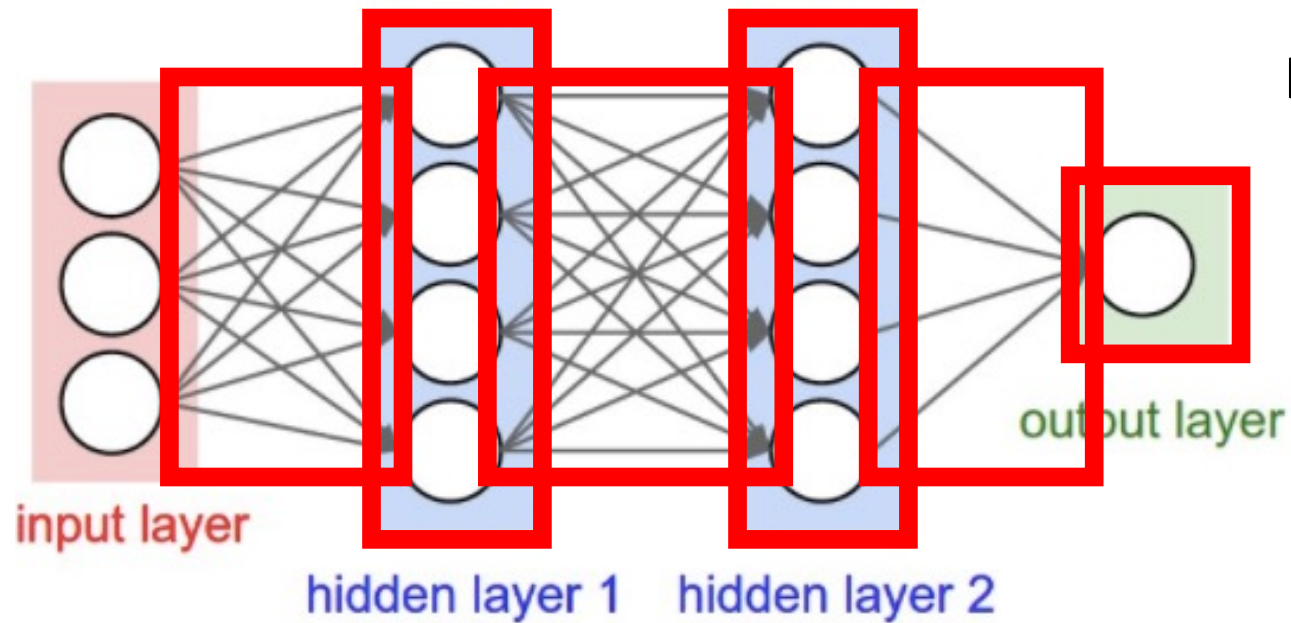
# Neural Network



How many weights are in this model?

- Input to Hidden Layer 1:
  - $3 \times 4 = 12$
- Hidden Layer 1 to Hidden Layer 2:
  - $4 \times 4 = 16$
- Hidden Layer 2 to Output Layer
  - $4 \times 1 = 4$
- Total:
  - $12 + 16 + 4 = 32$

# Neural Network

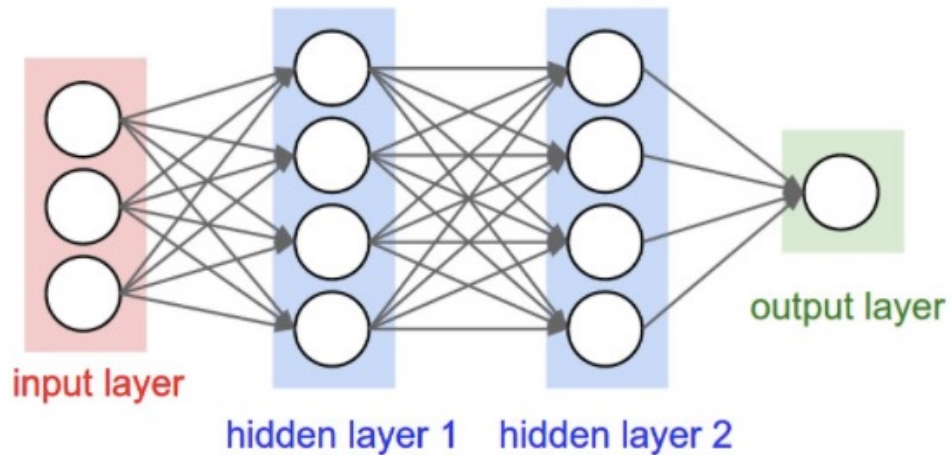


How many parameters are there to learn?

- Number of weights:
  - 32
- Number of biases:
  - $4 + 4 + 1 = 9$
- Total
  - 41



# Fully Connected, Feedforward Neural Networks



- What does it mean for a model to be fully connected?
  - Each unit provides input to each unit in the next layer
- What does it mean for a model to be feedforward?
  - Each layer serves as input to the next layer with no loops

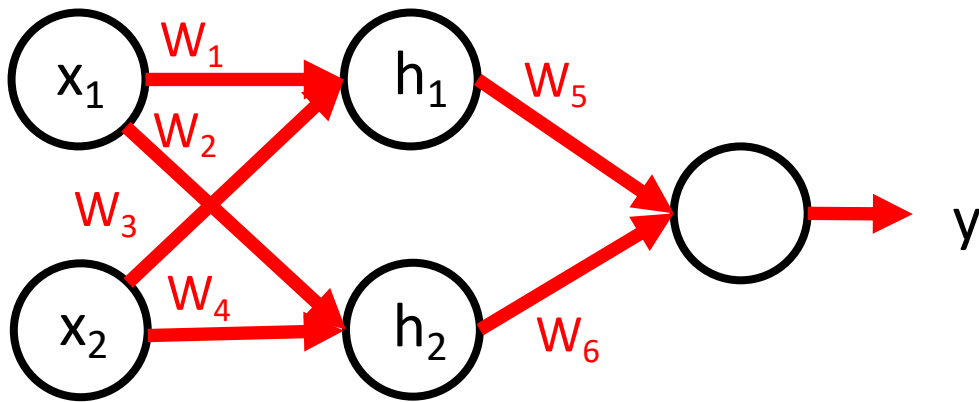
# How Many Layers and Units Should be Used?

To be explored more in lab  
assignment set 1 and this course



# Hidden Layers Alone Are NOT Enough to Model Non-Linear Functions

Key Observation: feedforward networks are just functions chained together  
e.g.,

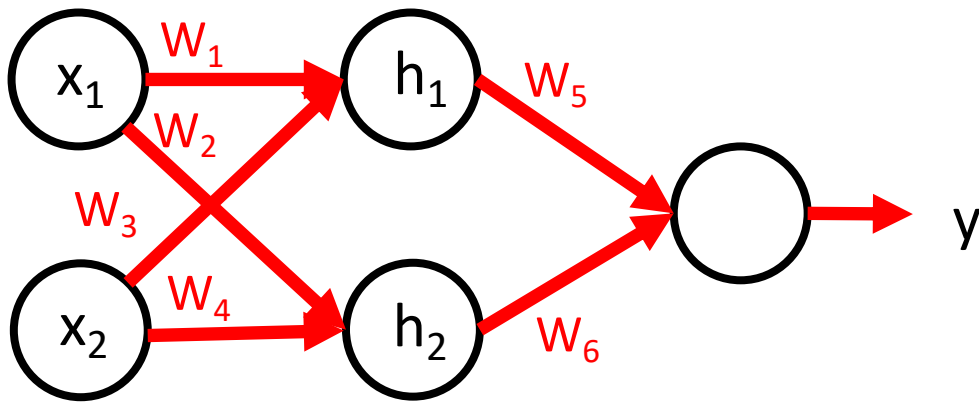


- What is function for  $h_1$ ?
  - $h_1 = w_1x_1 + w_3x_2 + b_1$
- What is function for  $h_2$ ?
  - $h_2 = w_2x_1 + w_4x_2 + b_2$
- What is function for  $y$ ?
  - $y = h_1w_5 + h_2w_6 + b_3$
  - $y = (w_1x_1 + w_3x_2 + b_1)w_5 + (w_2x_1 + w_4x_2 + b_2)w_6 + b_3$
  - $y = w_1w_5x_1 + w_3w_5x_2 + w_5b_1 + w_2w_6x_1 + w_4w_6x_2 + w_6b_2 + b_3$

A chain of LINEAR functions at any depth is still a LINEAR function!

# Hidden Layers Alone Are NOT Enough to Model Non-Linear Functions

Key Observation: feedforward networks are just functions chained together  
e.g.,



- What is function for  $h_1$ ?
  - $h_1 = w_1x_1 + w_3x_2 + b_1$
- What is function for  $h_2$ ?
  - $h_2 = w_2x_1 + w_4x_2 + b_2$
- What is function for  $y$ ?
  - $y = \underbrace{h_1 w_5} + \underbrace{h_2 w_6} + b_3$

Constant x linear function = linear function

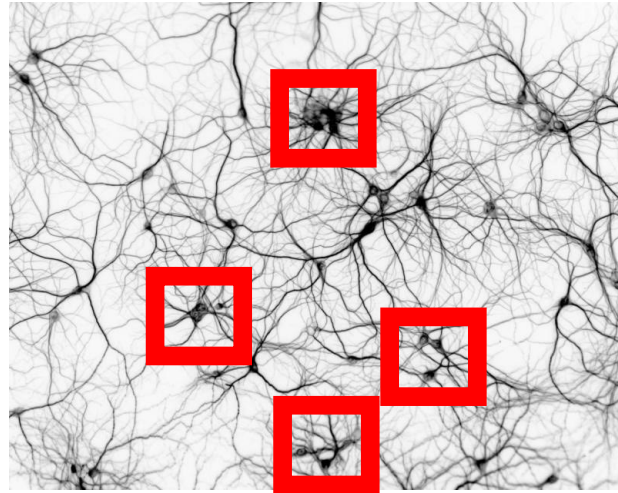
A chain of LINEAR functions at any depth is still a LINEAR function!

# Today's Topics

- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- **Neural network architecture: activation functions**
- Neural network architecture: output units
- Programming tutorial

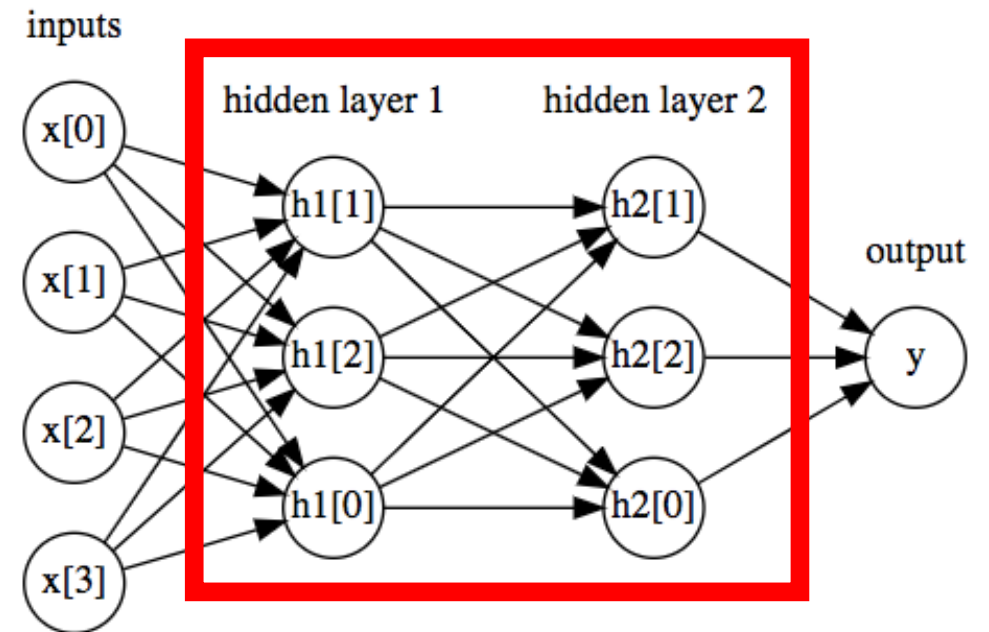
# Key Idea: Use Connected Neurons to Non-linearly Transform Input into Useful Features for Predictions

Biological Neural Network:



<http://www.rzagabe.com/2014/11/03/an-introduction-to-artificial-neural-networks.html>

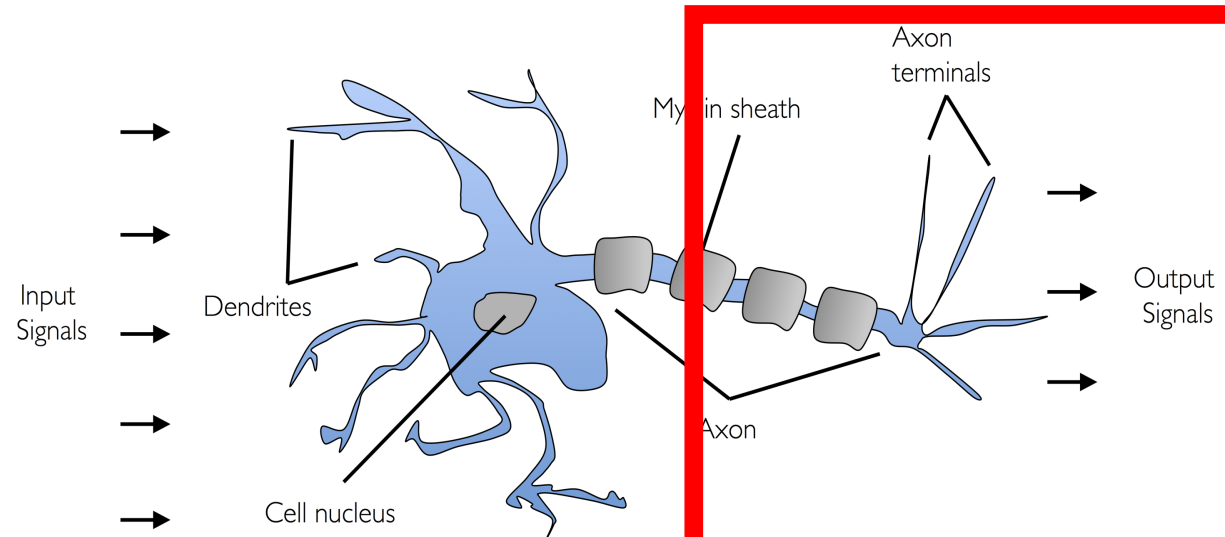
Artificial Neural Network:



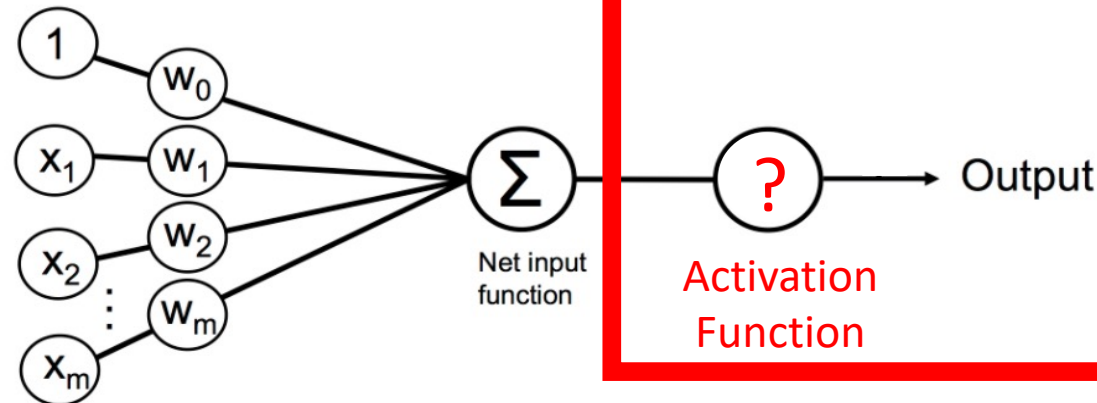
[https://github.com/amueller/introduction\\_to\\_ml\\_with\\_python/blob/master/02-supervised-learning.ipynb](https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb)

# Key Idea: Use Connected Neurons to Non-linearly Transform Input into Useful Features for Predictions

Biological Neuron:



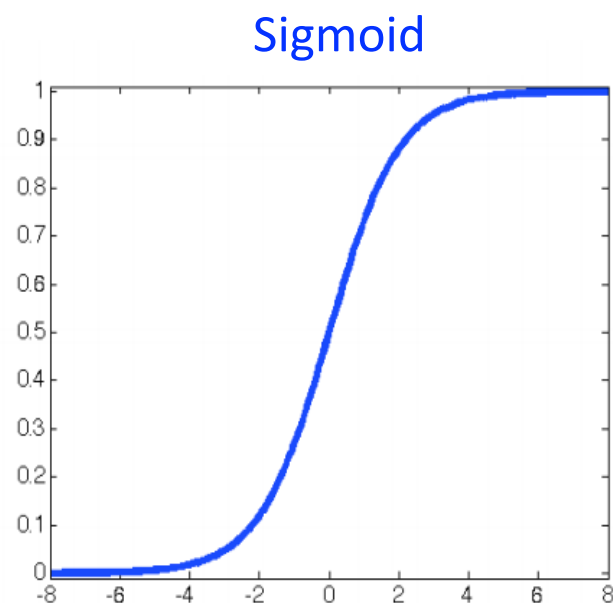
Artificial Neurons  
(e.g., Perceptron):



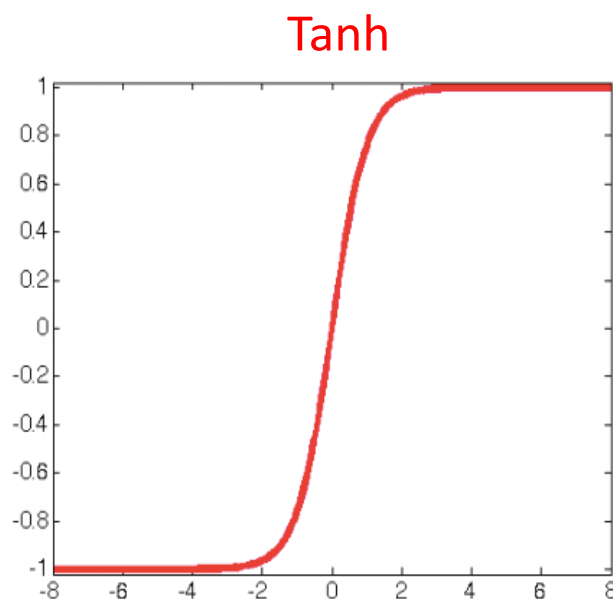
Mimic a neuron firing,  
by having each unit  
apply a non-linear  
“activation” function  
to the weighted input

# Non-Linear Activation Functions

- Each unit applies a non-linear “activation” function to the weighted input to mimic a neuron firing

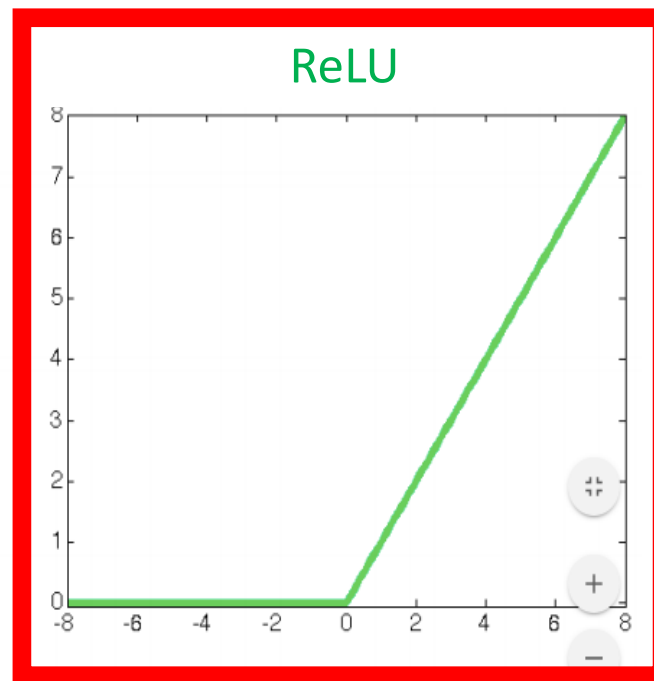


$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



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

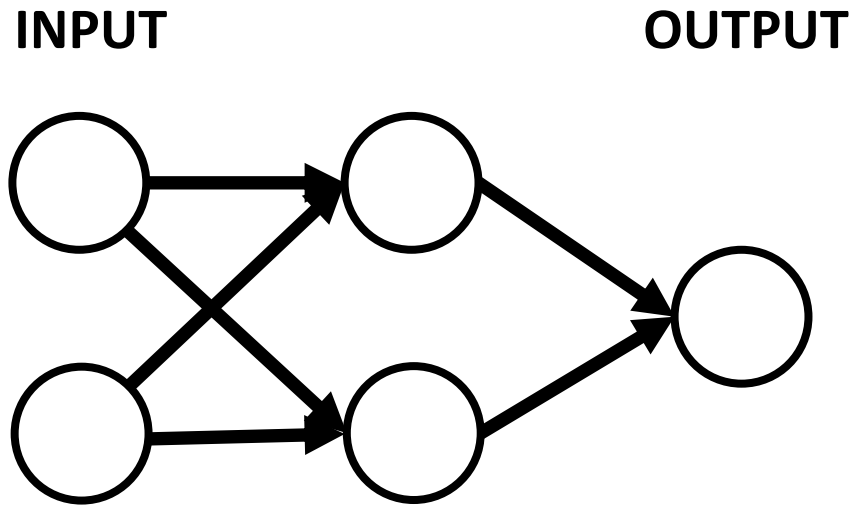
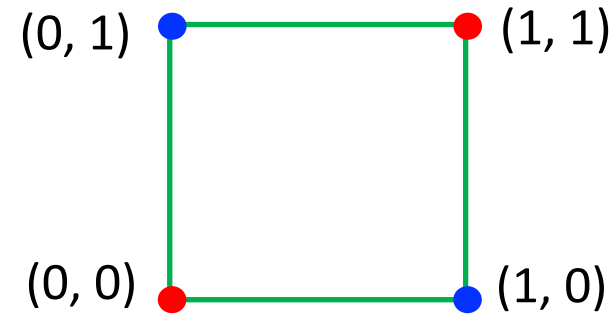
Computationally faster



$$\text{ReLU}(z) = \max(0, z)$$

# Non-Linear Example: Revisiting XOR problem

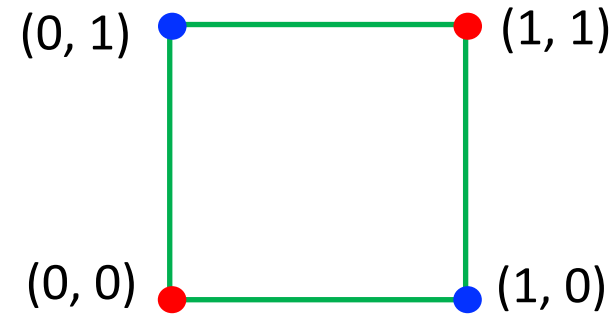
- Non-linear function: separate 1s from 0s:



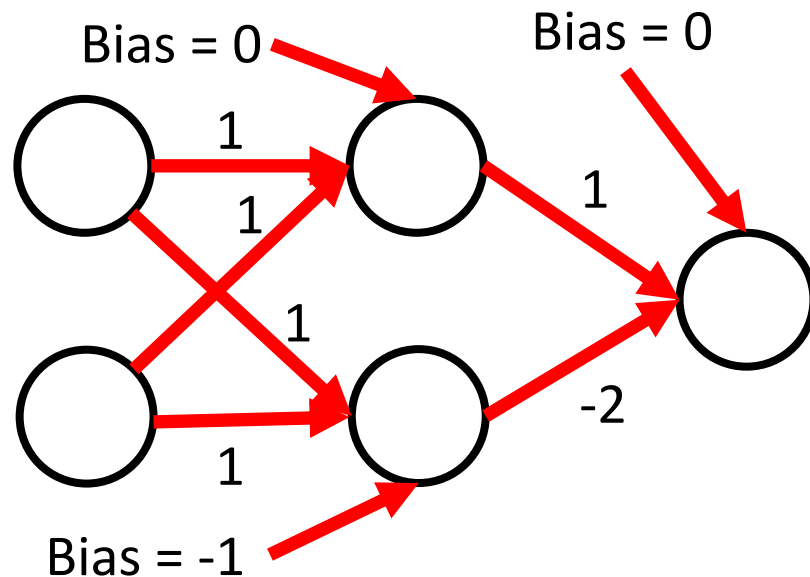
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:

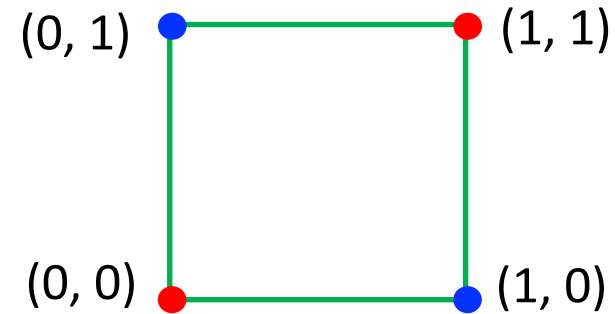


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

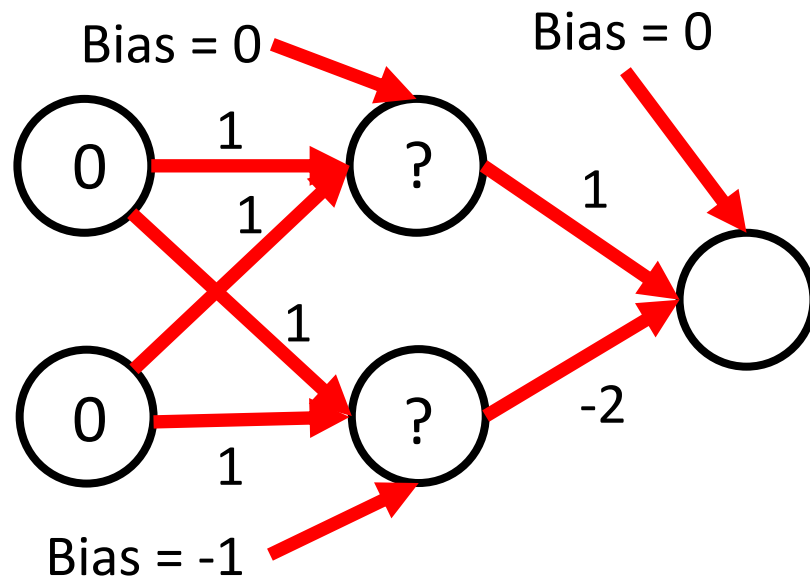


# Non-Linear Example: Revisiting XOR problem

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



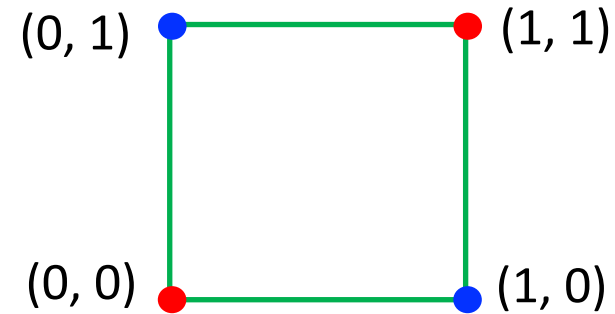
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



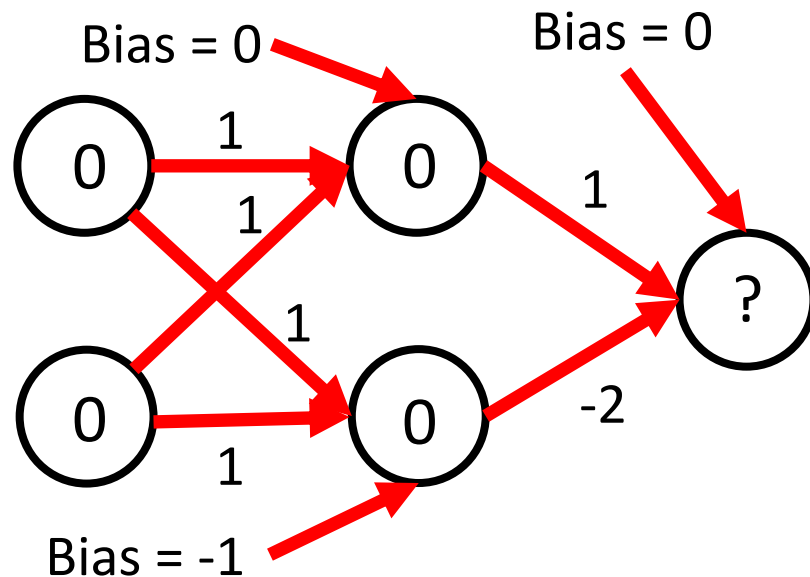
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



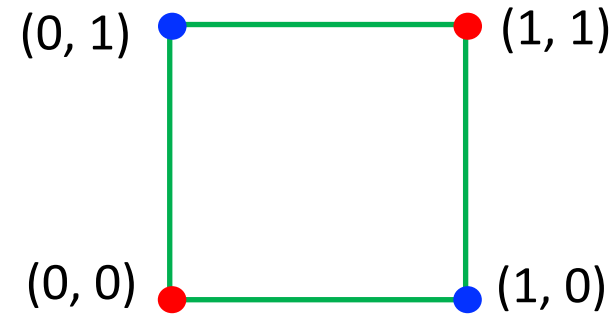
- Approach: ReLU activation function (  $\text{ReLU}(z) = \max(0, z)$  ) with these parameters:



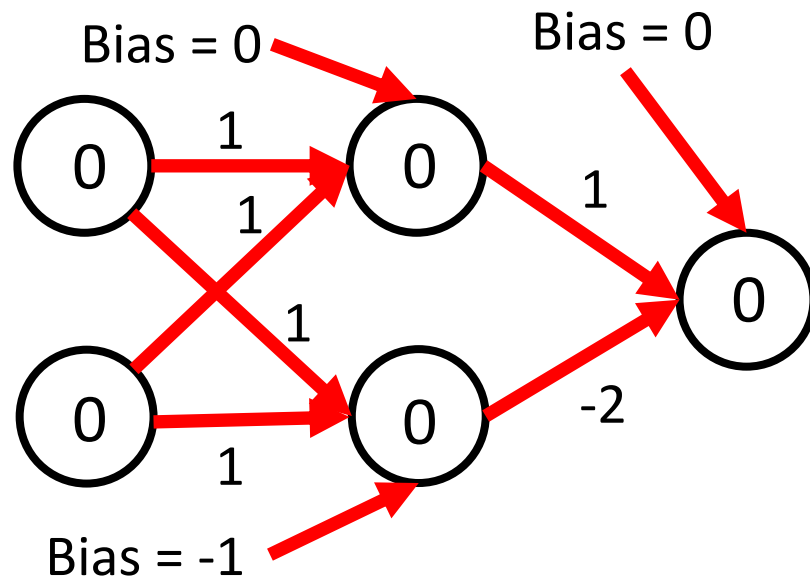
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



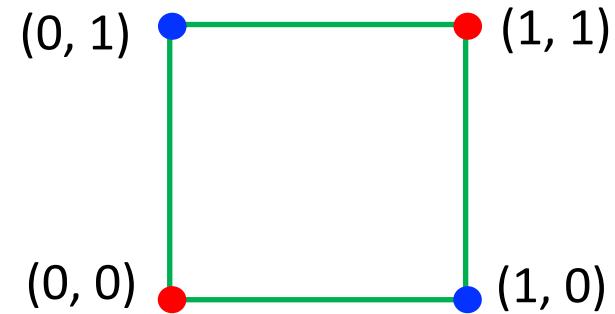
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



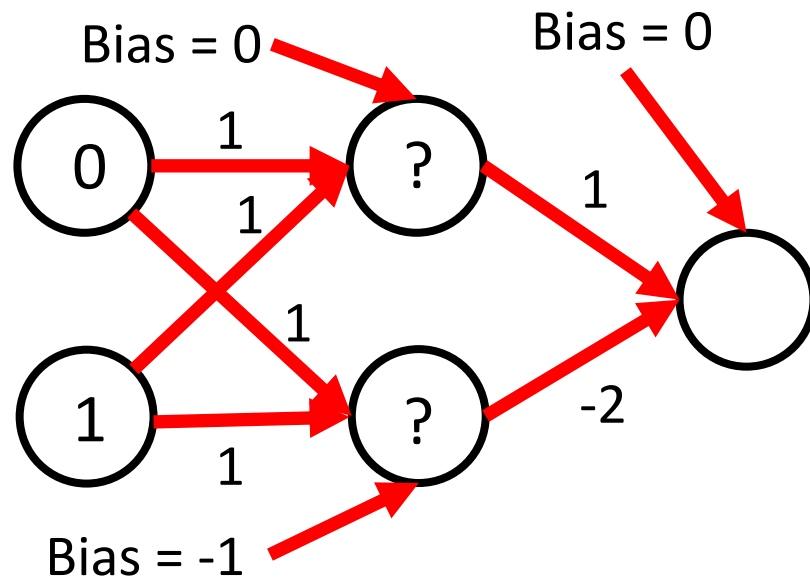
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



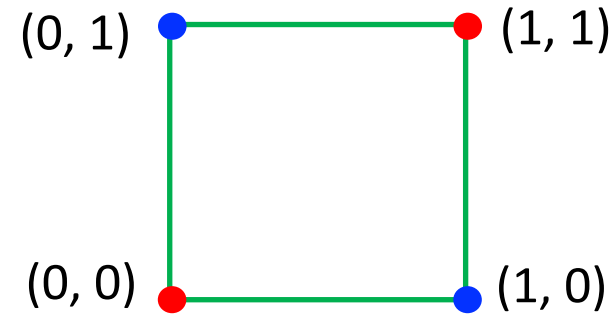
- Approach: ReLU activation function (  $\text{ReLU}(z) = \max(0, z)$  ) with these parameters:



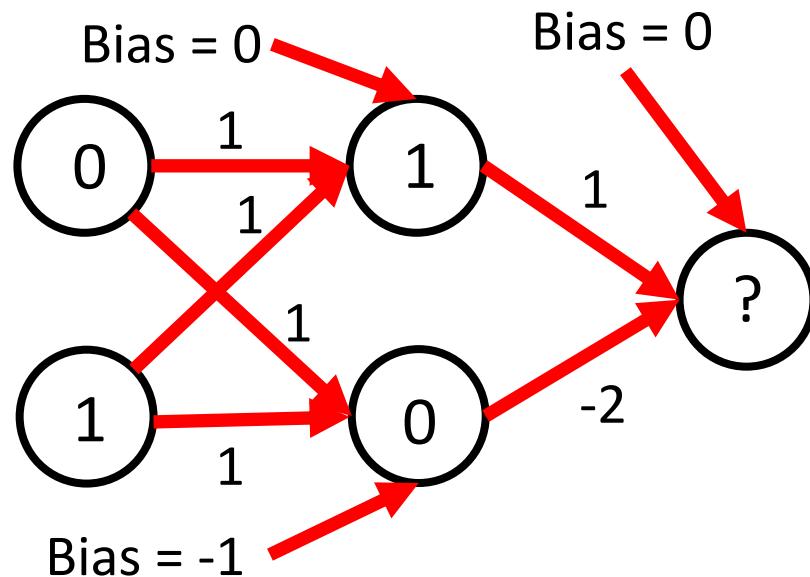
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



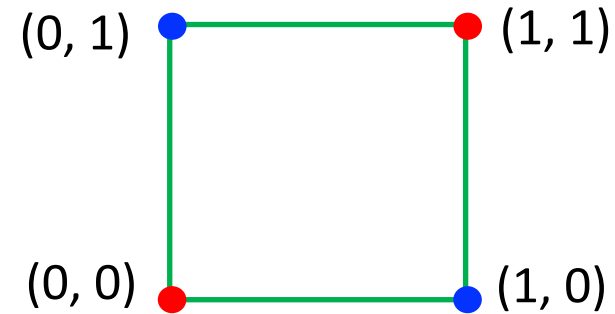
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



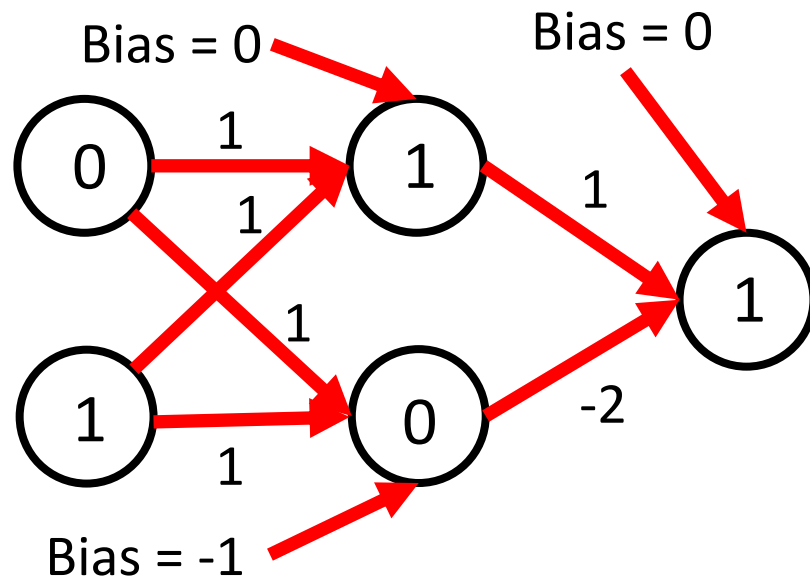
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



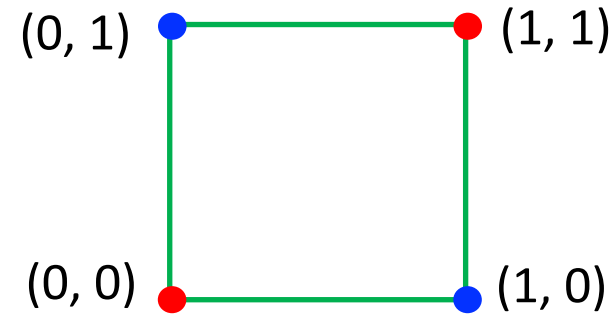
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



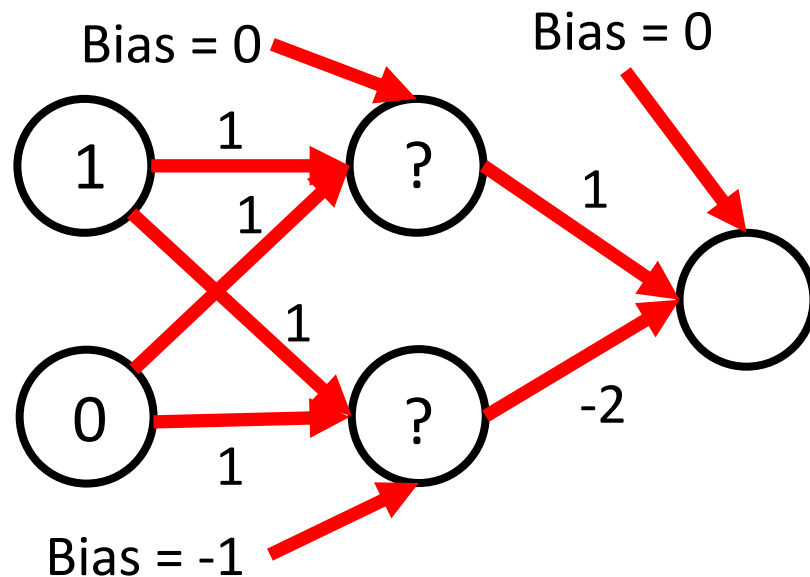
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



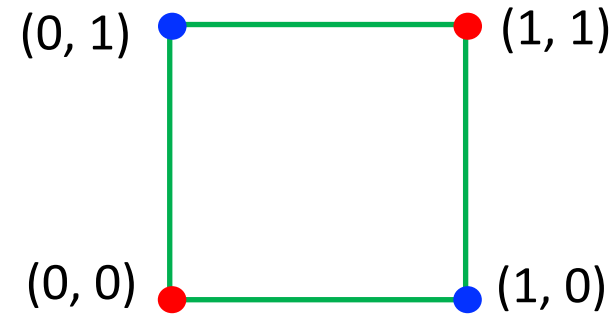
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



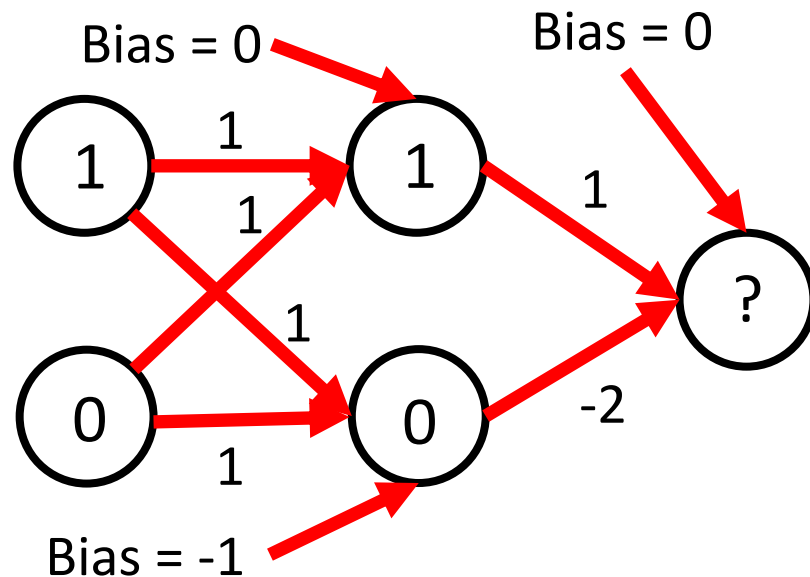
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:

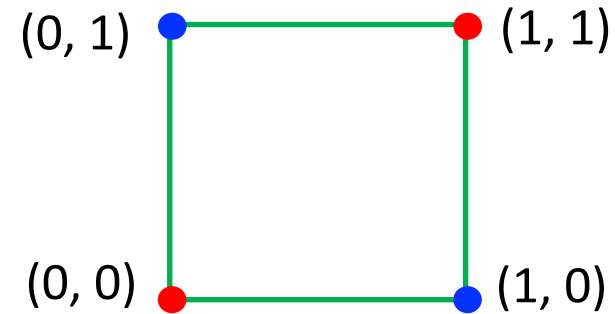


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

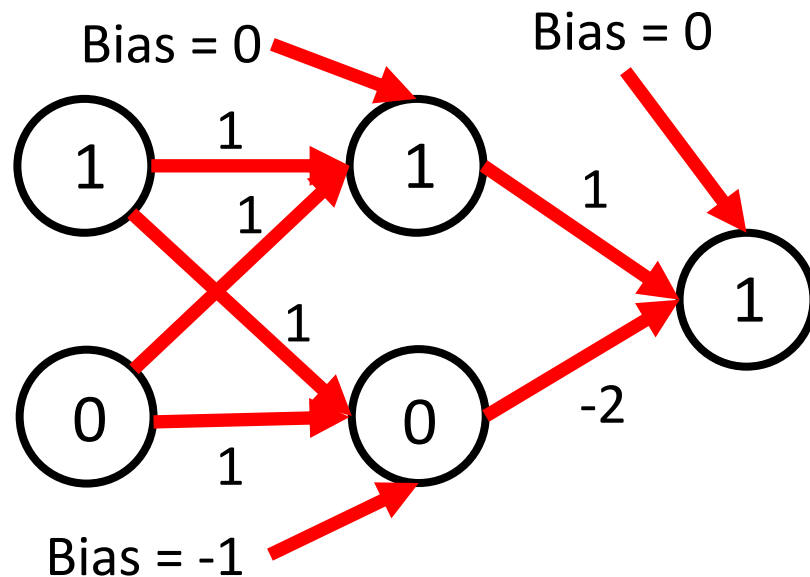


# Non-Linear Example: Revisiting XOR problem

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



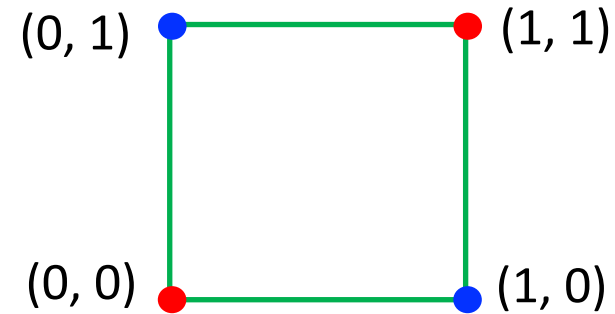
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



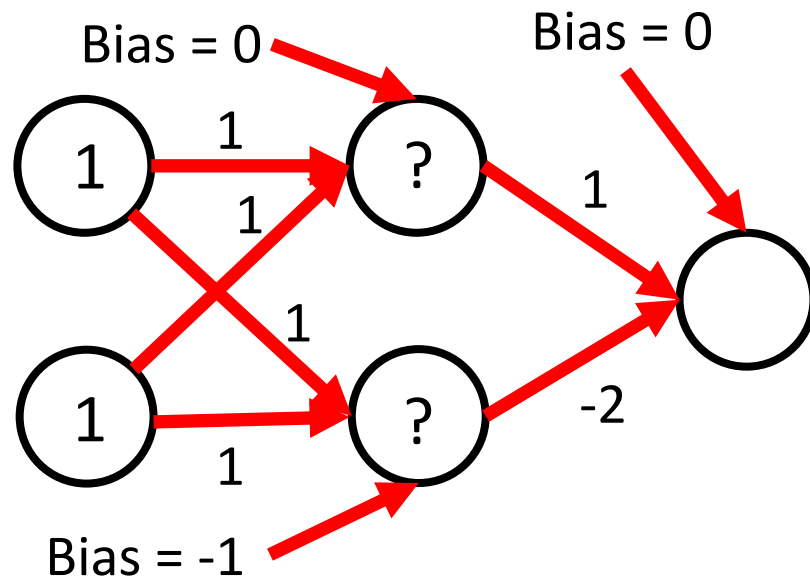
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



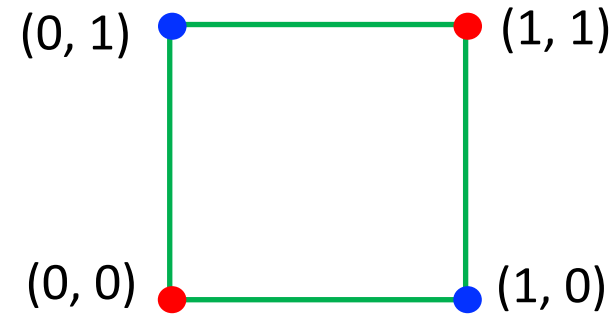
- Approach: ReLU activation function ( $\text{ReLU}(z) = \max(0, z)$ ) with these parameters:



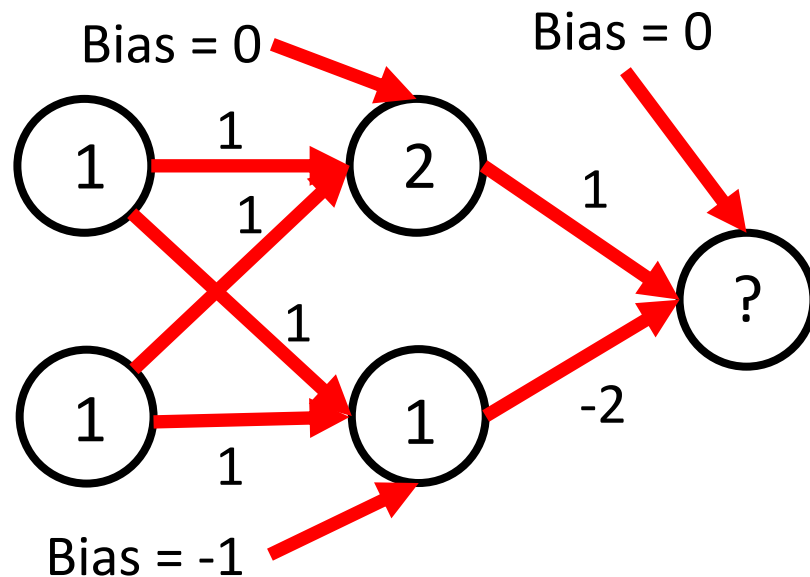
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



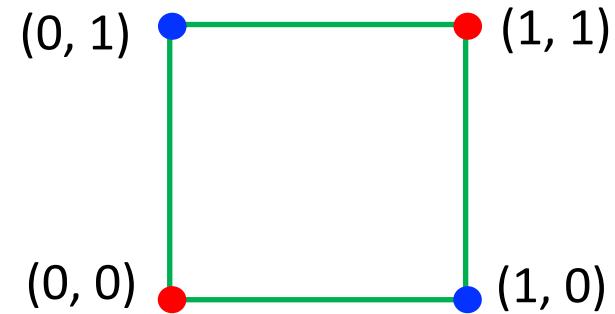
- Approach: ReLU activation function (  $\text{ReLU}(z) = \max(0, z)$  ) with these parameters:



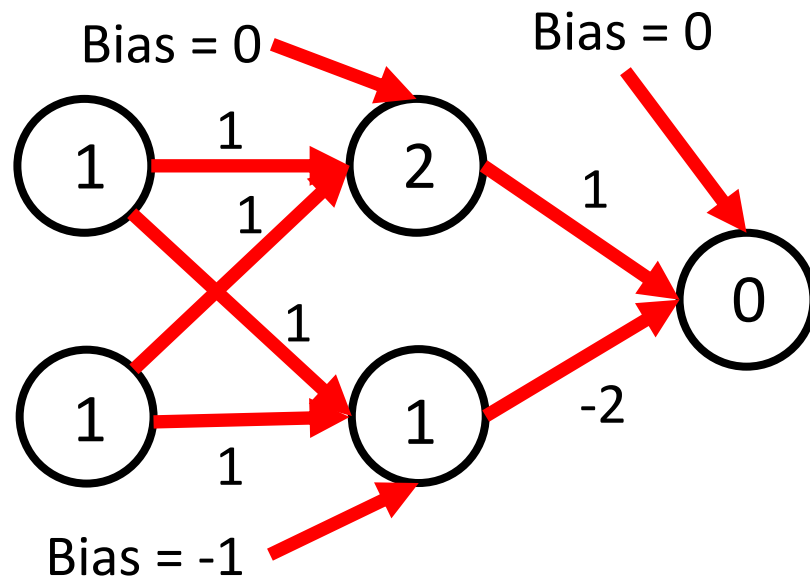
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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



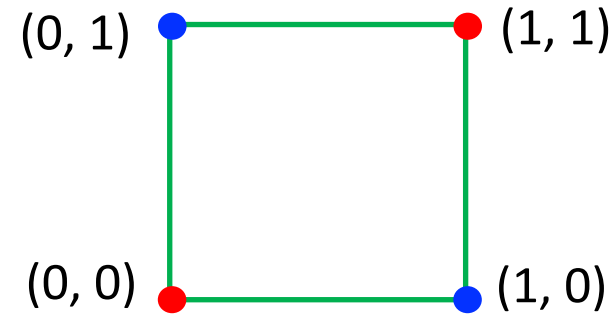
- Approach: ReLU activation function (  $\text{ReLU}(z) = \max(0, z)$  ) with these parameters:



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

# Non-Linear Example: Revisiting XOR problem

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

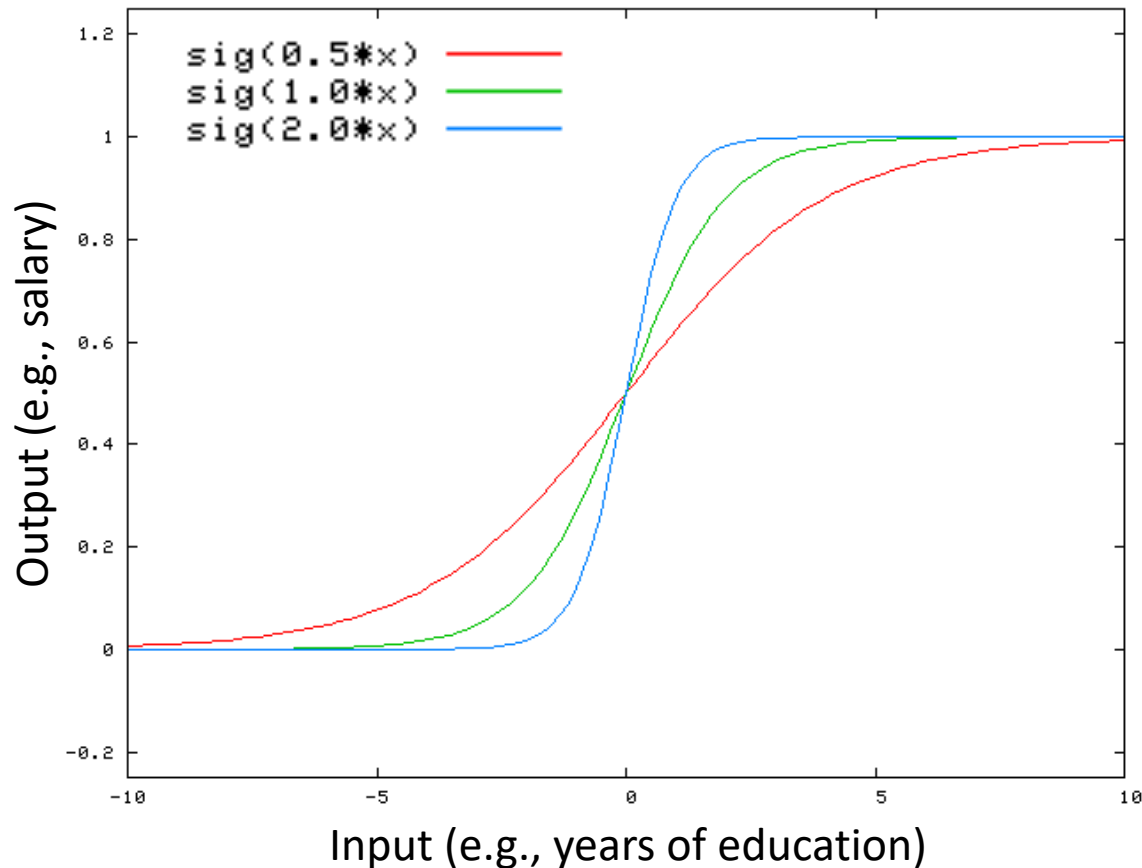


- Approach: ReLU activation function (  $\text{ReLU}(z) = \max(0, z)$  ) with these parameters:

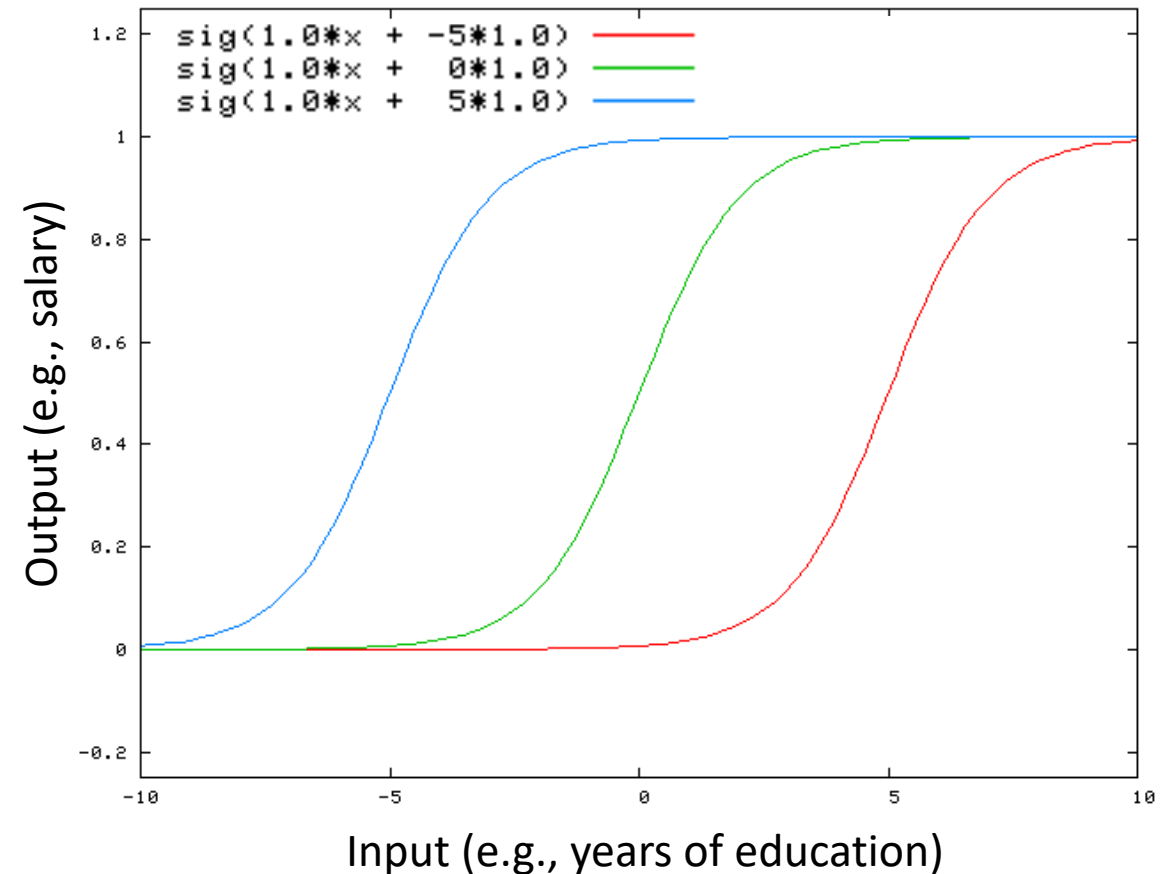
Neural networks can solve XOR problem...  
and so model non-linear functions!

# Activation Functions and Model Parameters (e.g., Sigmoid)

**Weights** determine curvature:



**Biases** determine shifted position:



# Which Activation Functions Should be Used?

To be explored more in lab  
assignment set 1 and this course

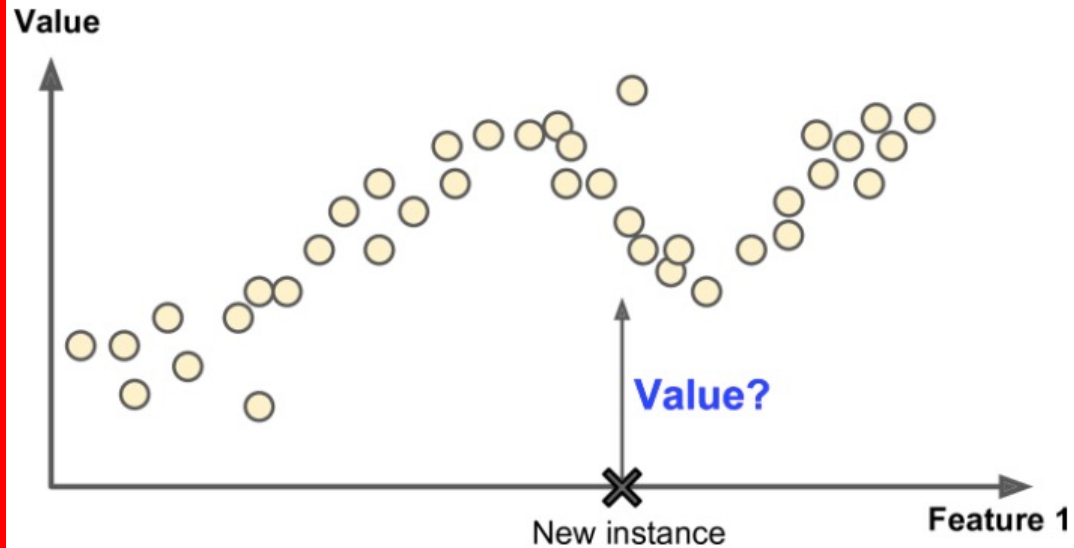
# Today's Topics

- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- Neural network architecture: activation functions
- **Neural network architecture: output units**
- Programming tutorial

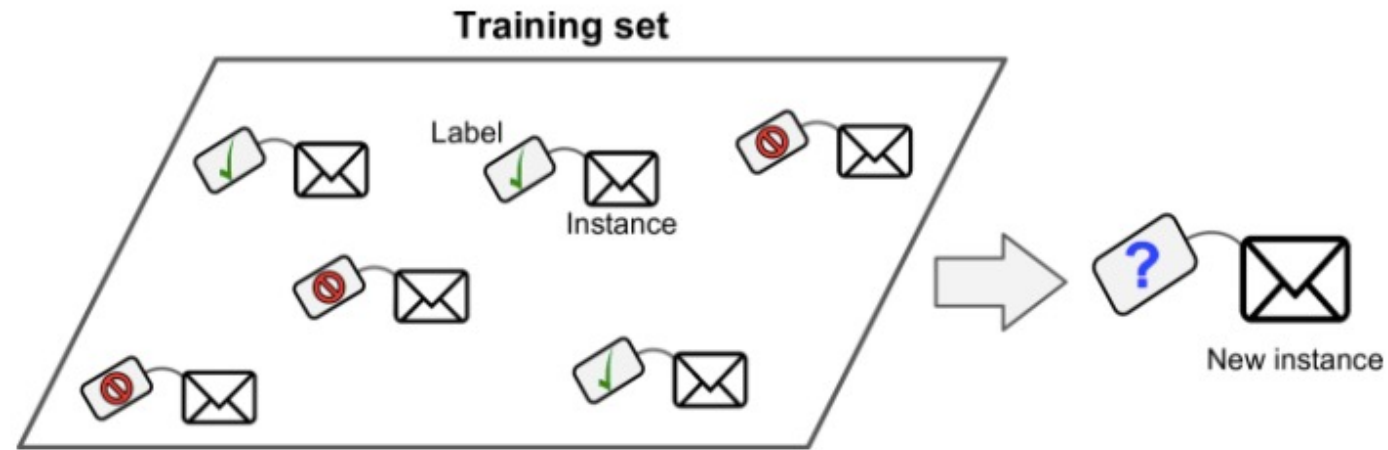


# Desired Output Driven by Task

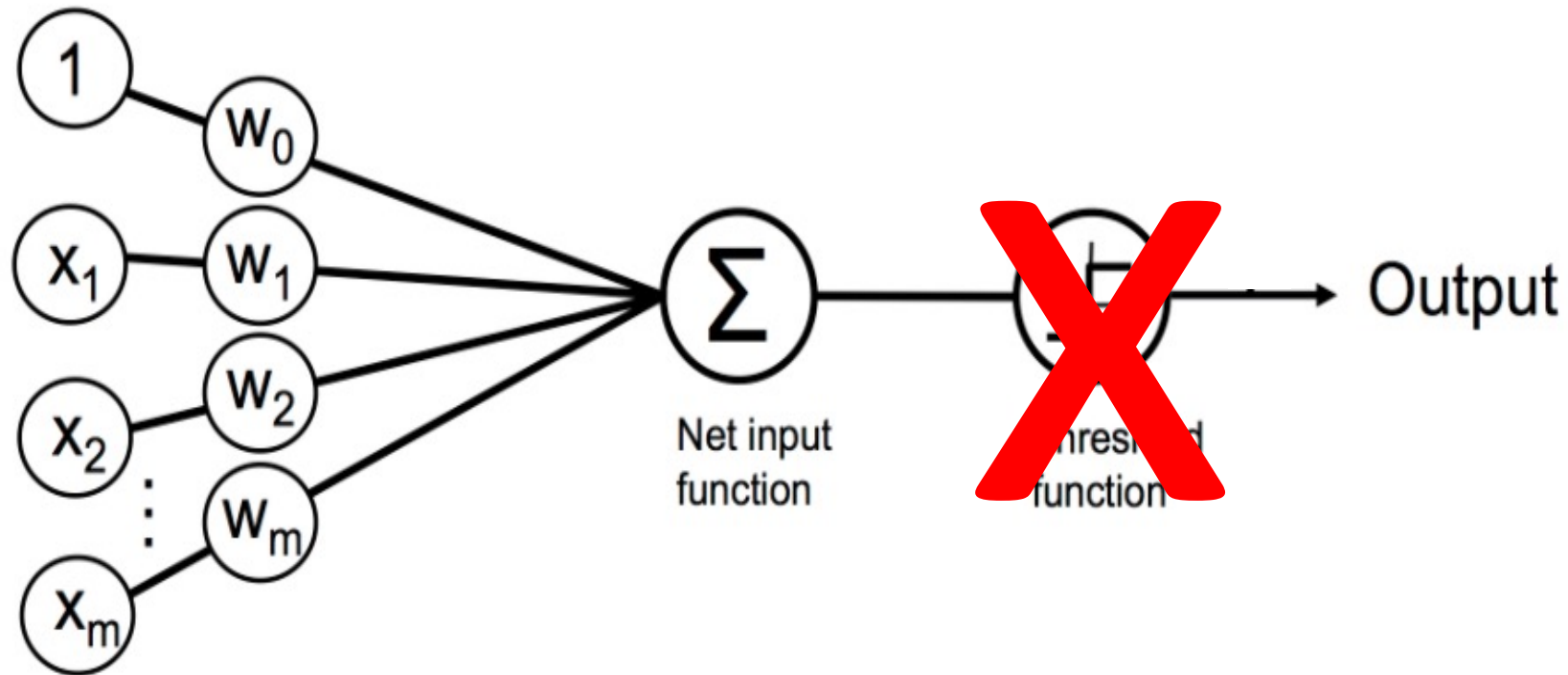
Regression  
(predict **continuous** value)



Classification  
(predict **discrete** value)

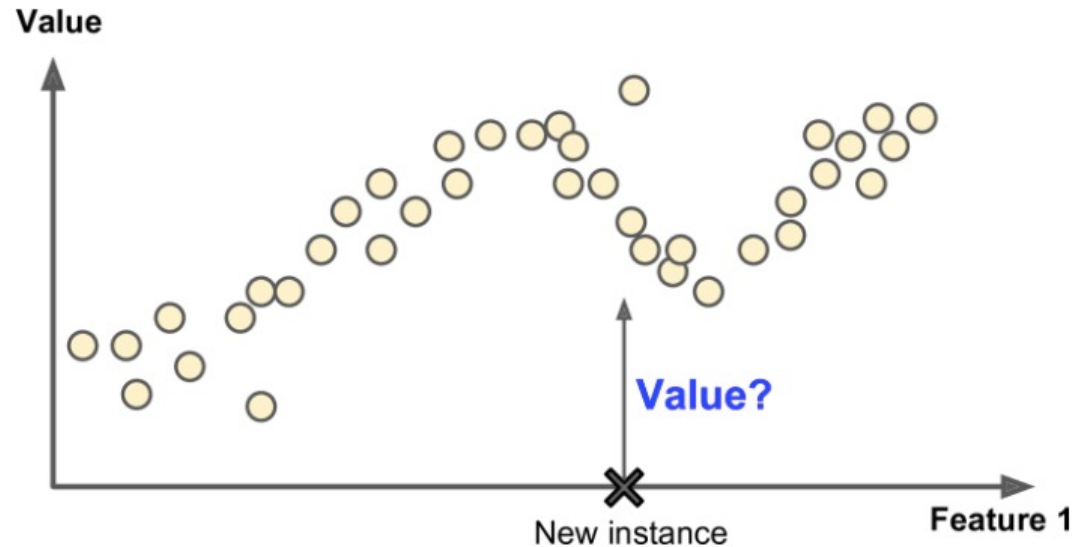


# Linear (No Activation Function)

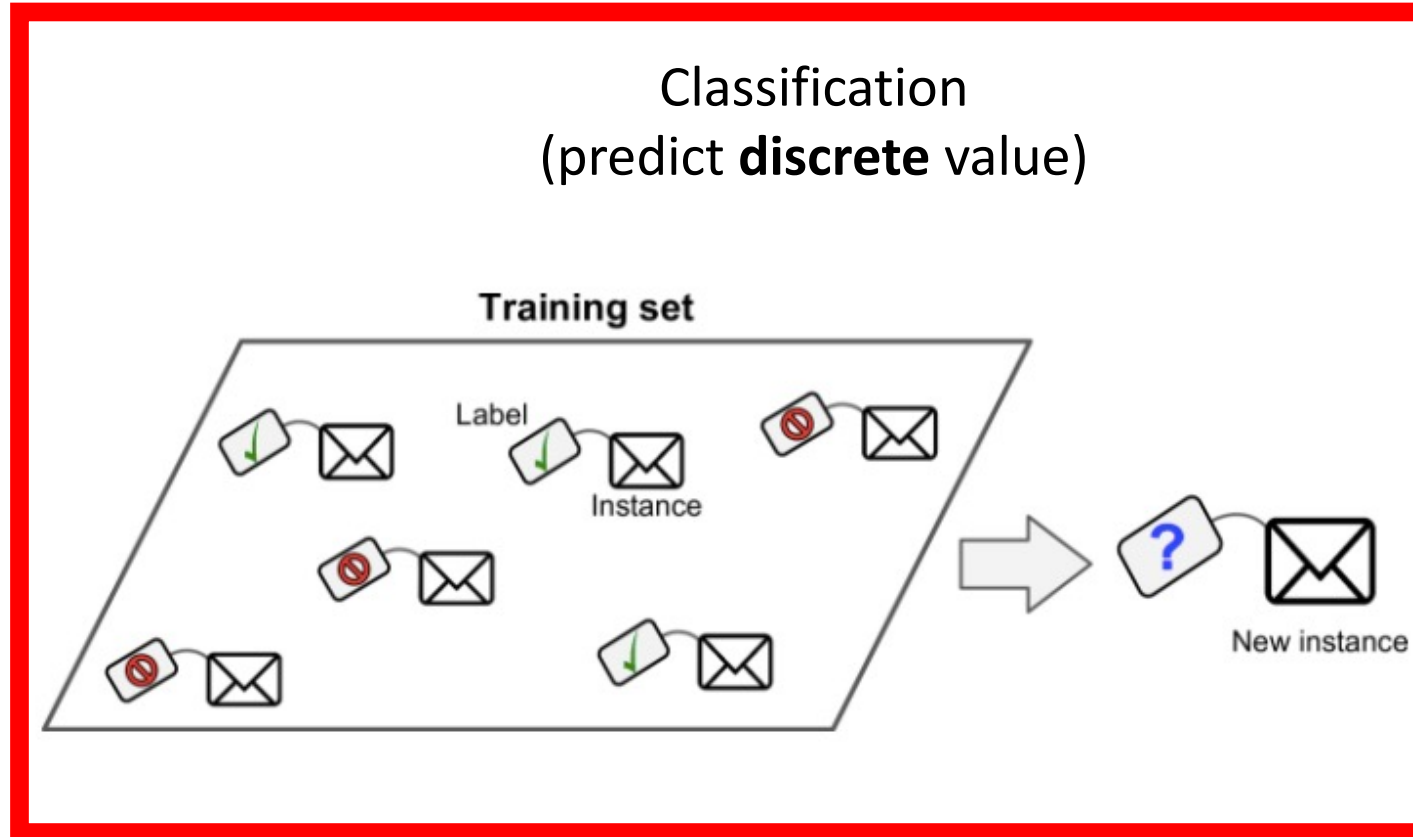


# Desired Output Driven by Task

Regression  
(predict **continuous** value)

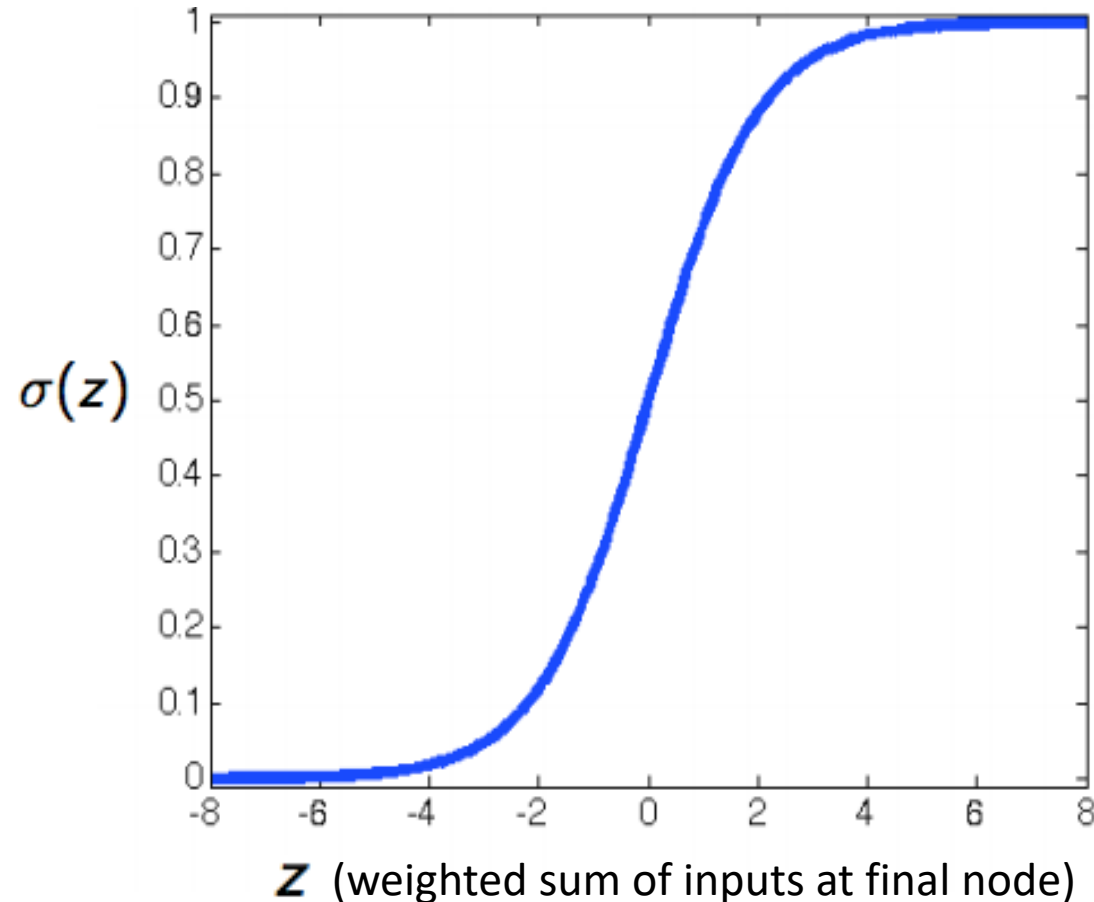


Classification  
(predict **discrete** value)



# Sigmoid (for Binary Classification); aka – Logistic Regression

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



If  $z \geq 0.5$ , output 1; Else, output 0

Why not use  $z$  instead of  $\sigma(z)$ ?

- We want a probability in  $[0, 1]$

What happens to the output as  $z$  becomes more positive?

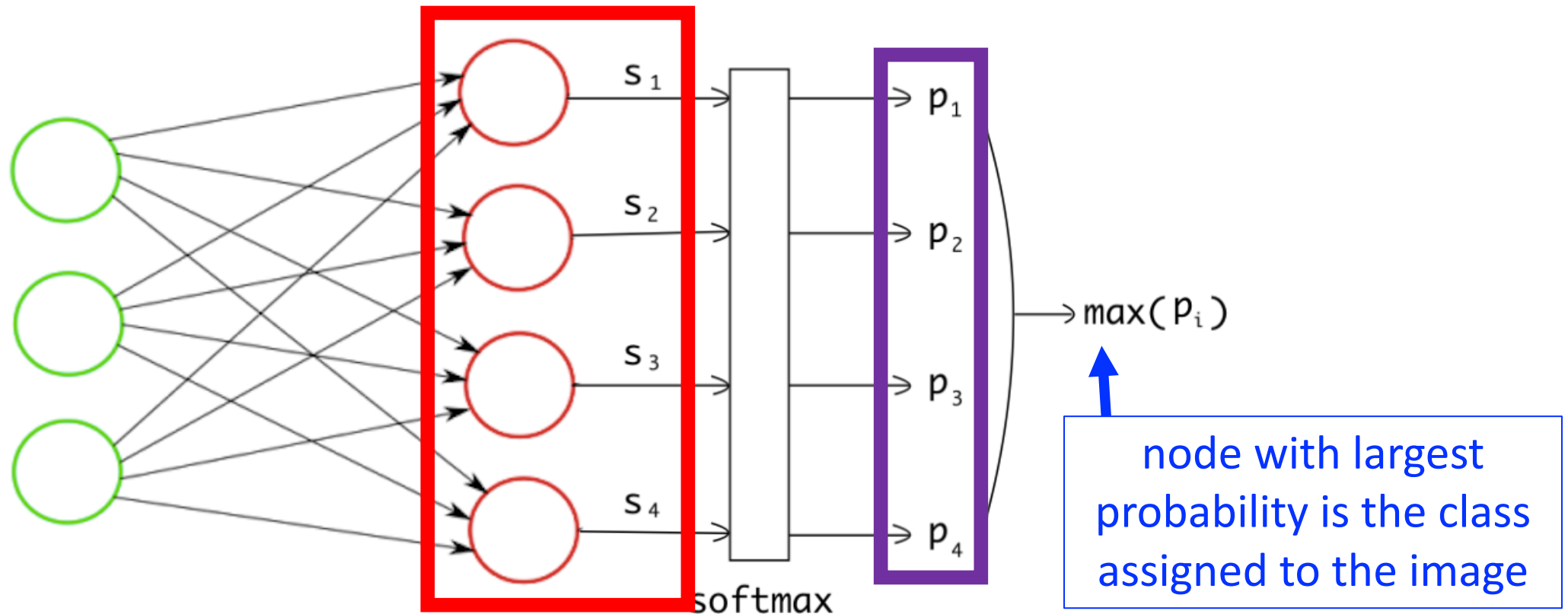
- $e^{-z}$  approaches 0 so value approaches 1

What happens to the output as  $z$  becomes more negative?

- $e^{-z}$  becomes larger so value approaches 0

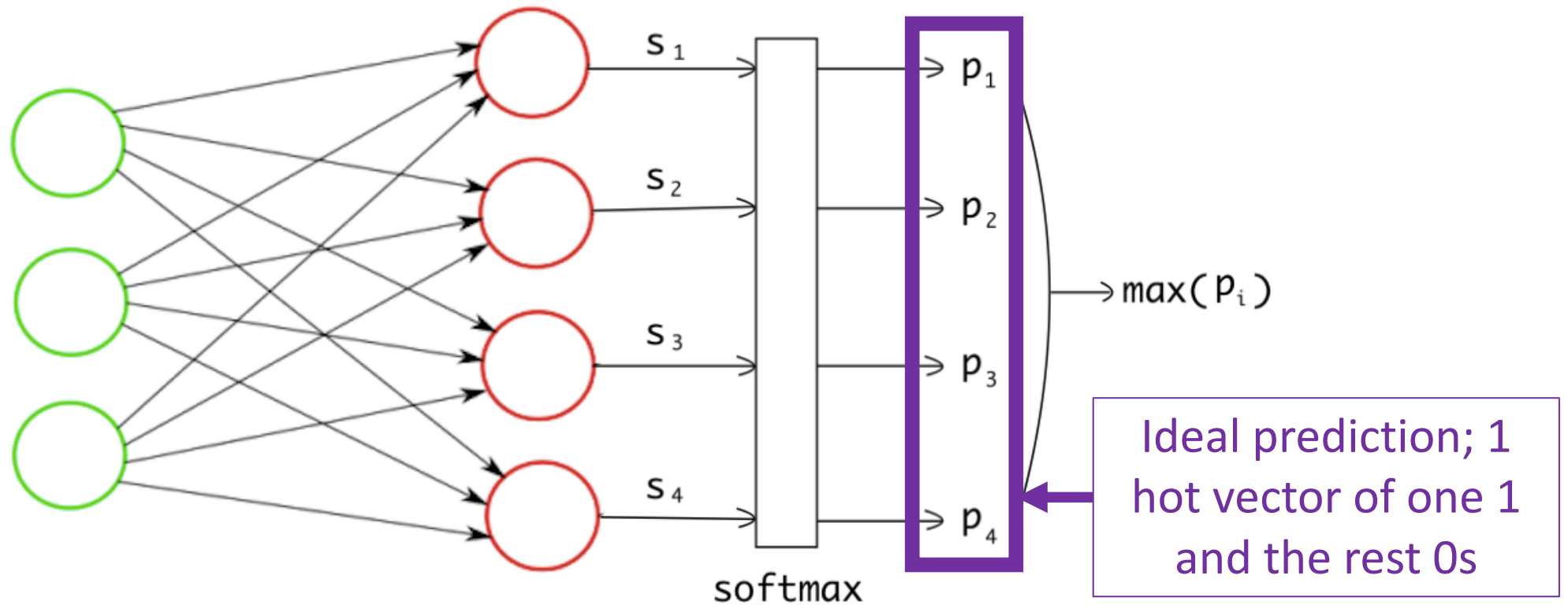
# Softmax (for Multiclass Classification)

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



# Softmax (for Multiclass Classification)

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



# Softmax (for Multiclass Classification)

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

Removes 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

$$e^{z_i}$$

$$\sigma(\mathbf{z})_i =$$

$i = 1, \dots, K$

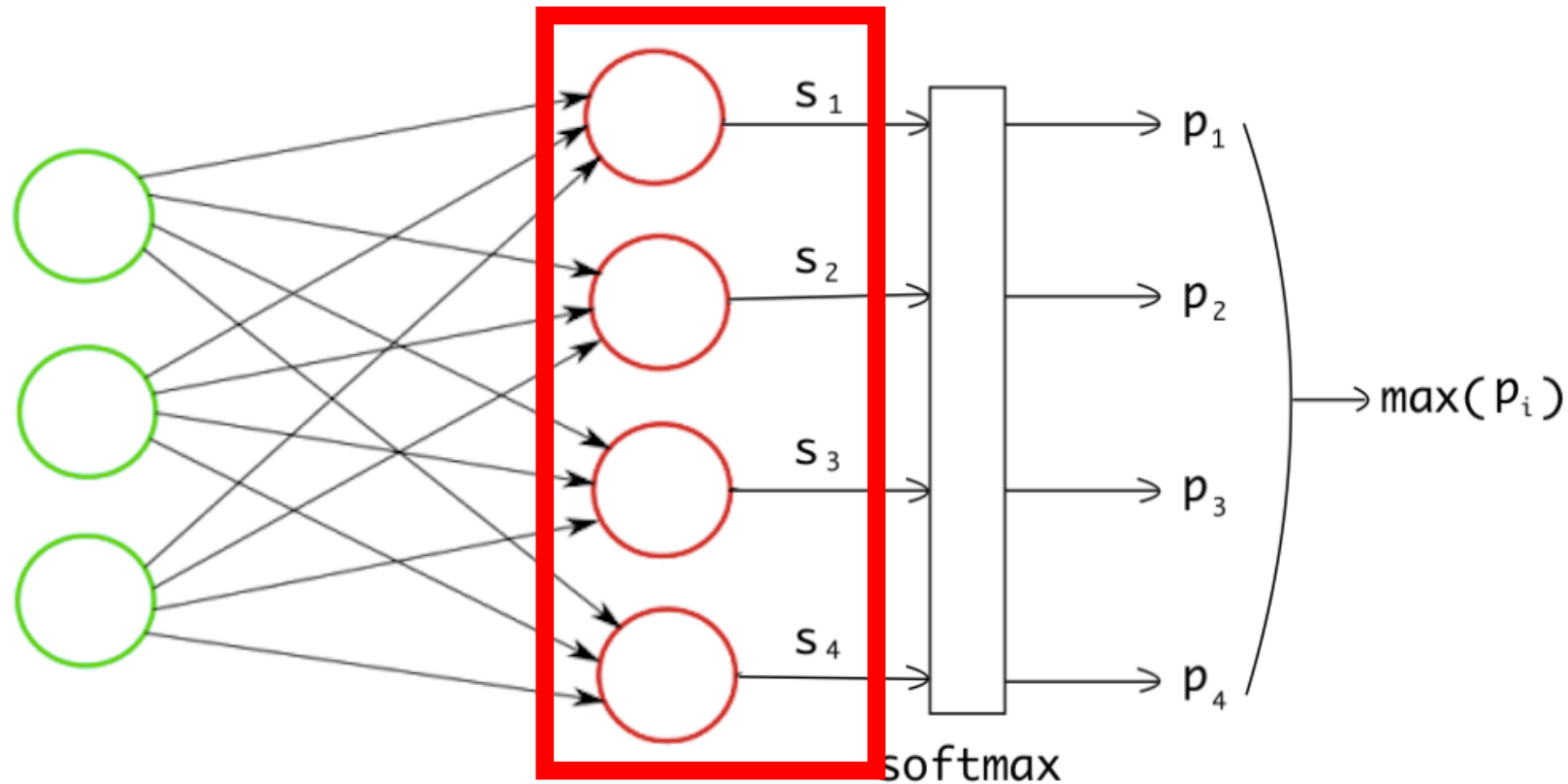
$$\sum_{j=1}^K e^{z_j}$$

Number of classes

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

# Softmax (for Multiclass Classification)

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





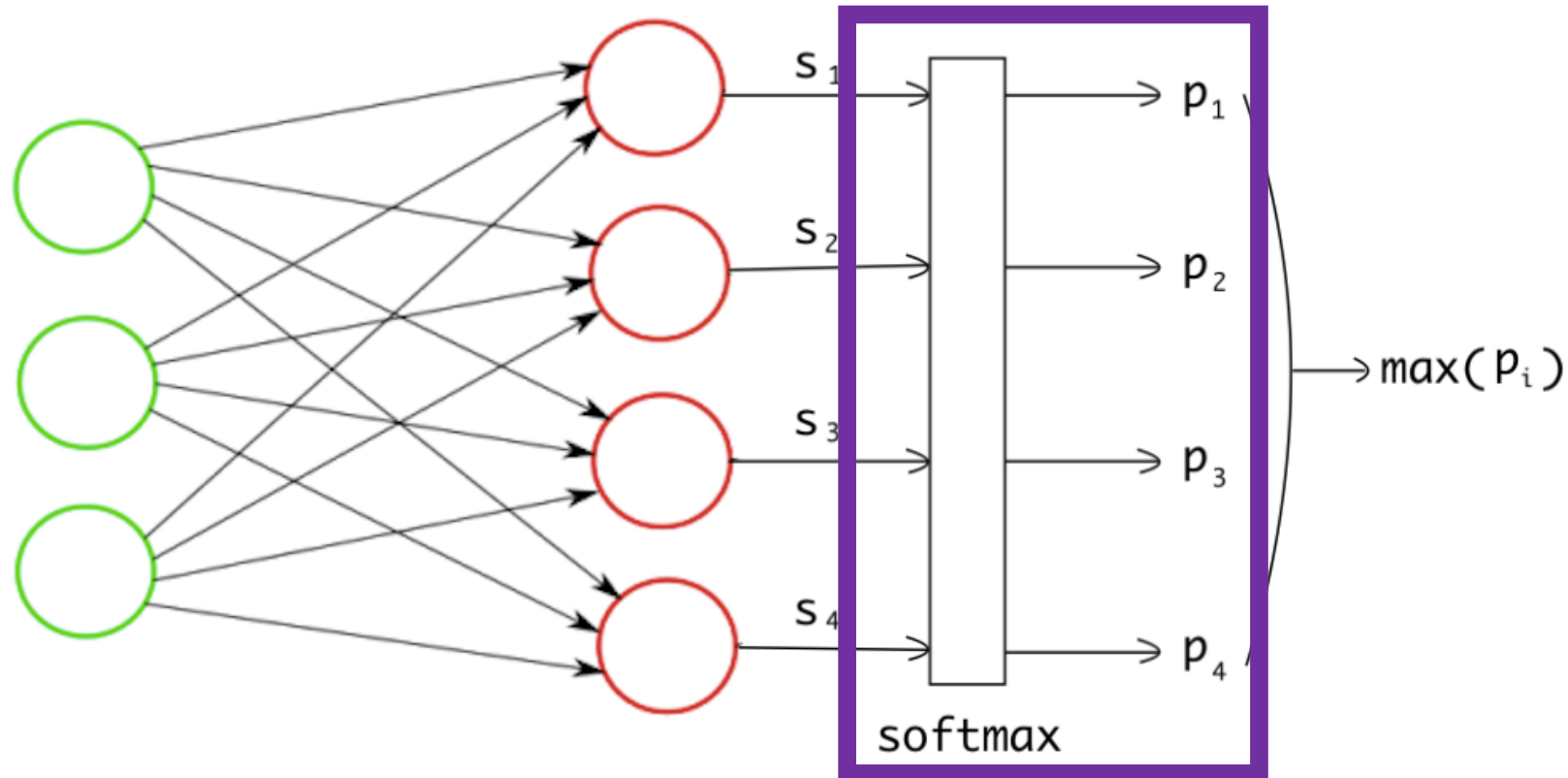
# Softmax (for Multiclass Classification)

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

# Softmax (for Multiclass Classification)

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



# Softmax (for Multiclass Classification)

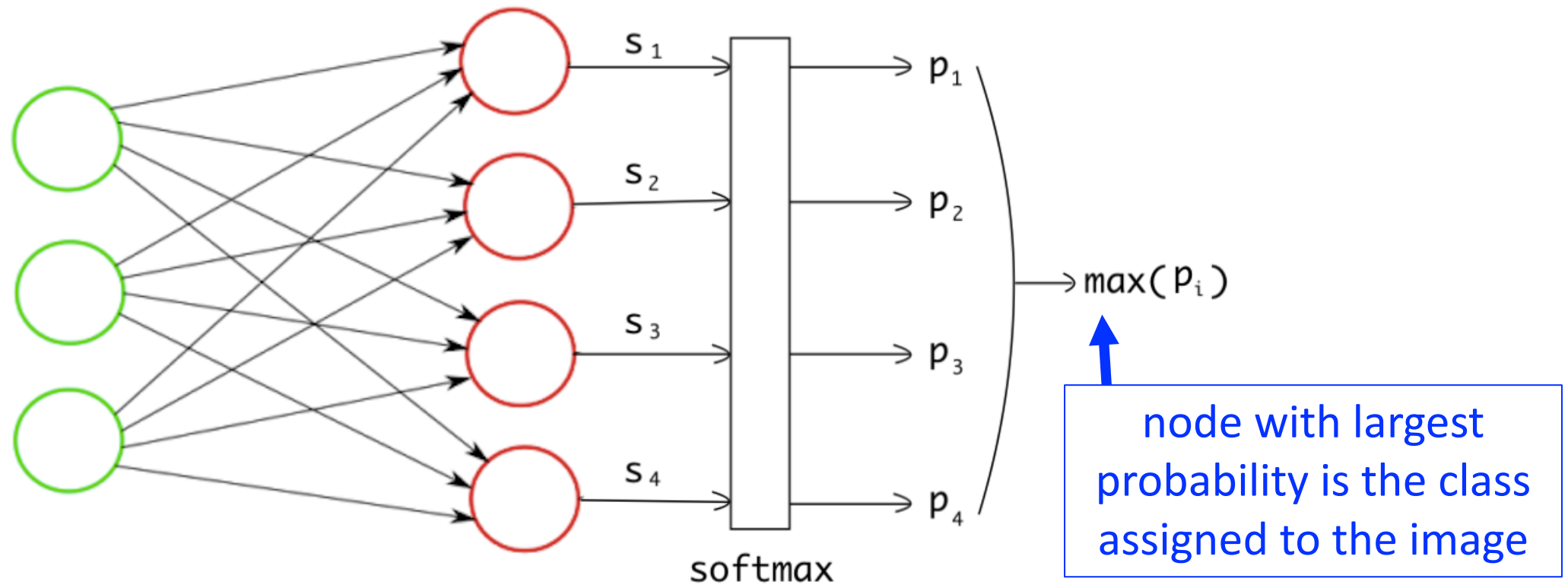
Converts vector of scores into a **probability distribution** that sums to 1; e.g.,

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

	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

# Softmax (for Multiclass Classification)

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



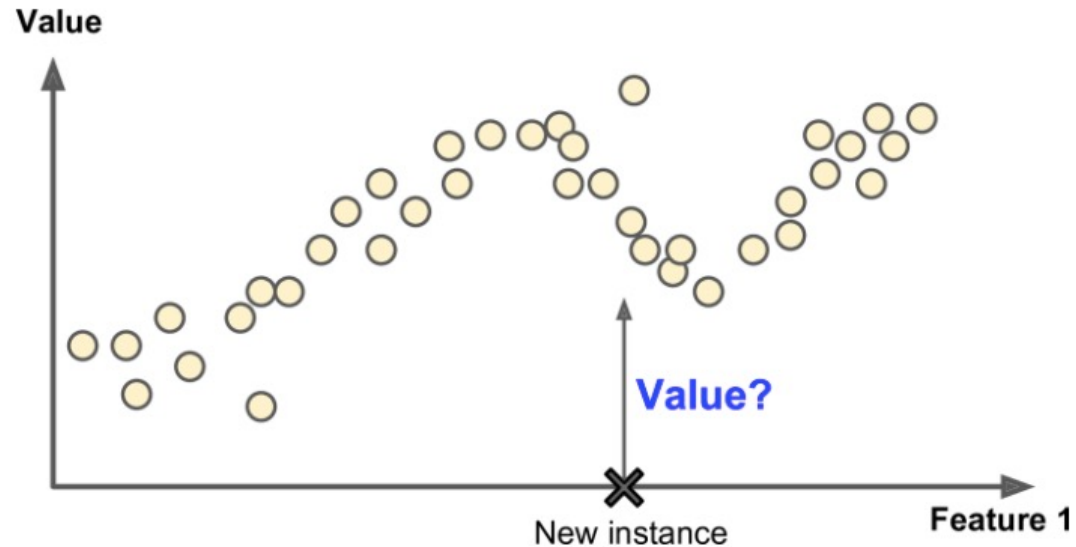
# Softmax (for Multiclass Classification)

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

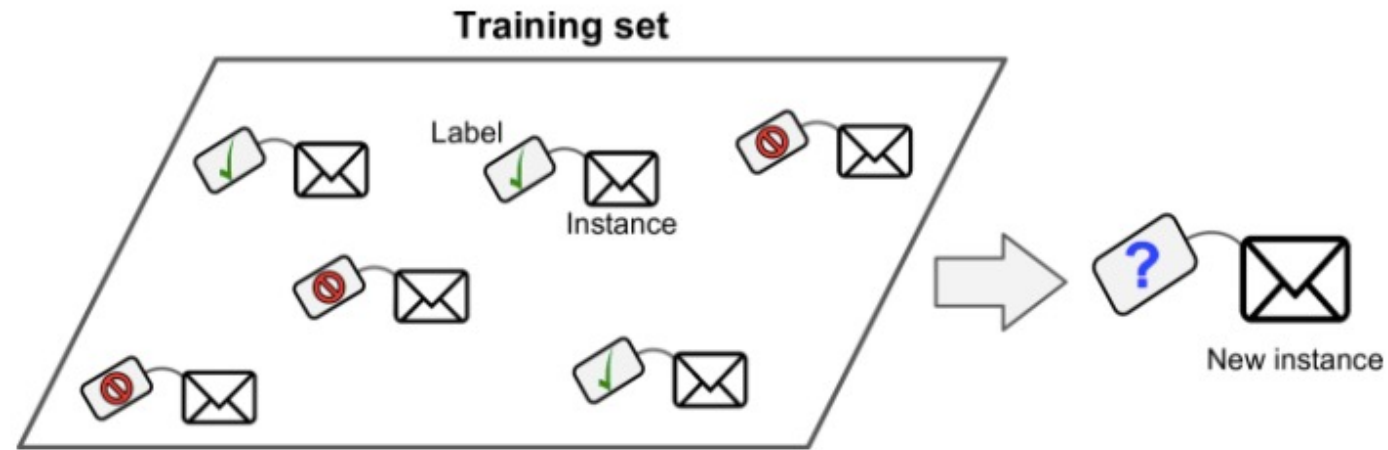
	Scoring Function	Unnormalized Probabilities	Normalized Probabilities
Dog	-3.44	0.0321	0.0006
Cat	1.16	3.1899	0.0596
Boat	-0.81	0.4449	0.0083
Airplane	3.91	49.8990	0.9315

# Desired Output Driven by Task

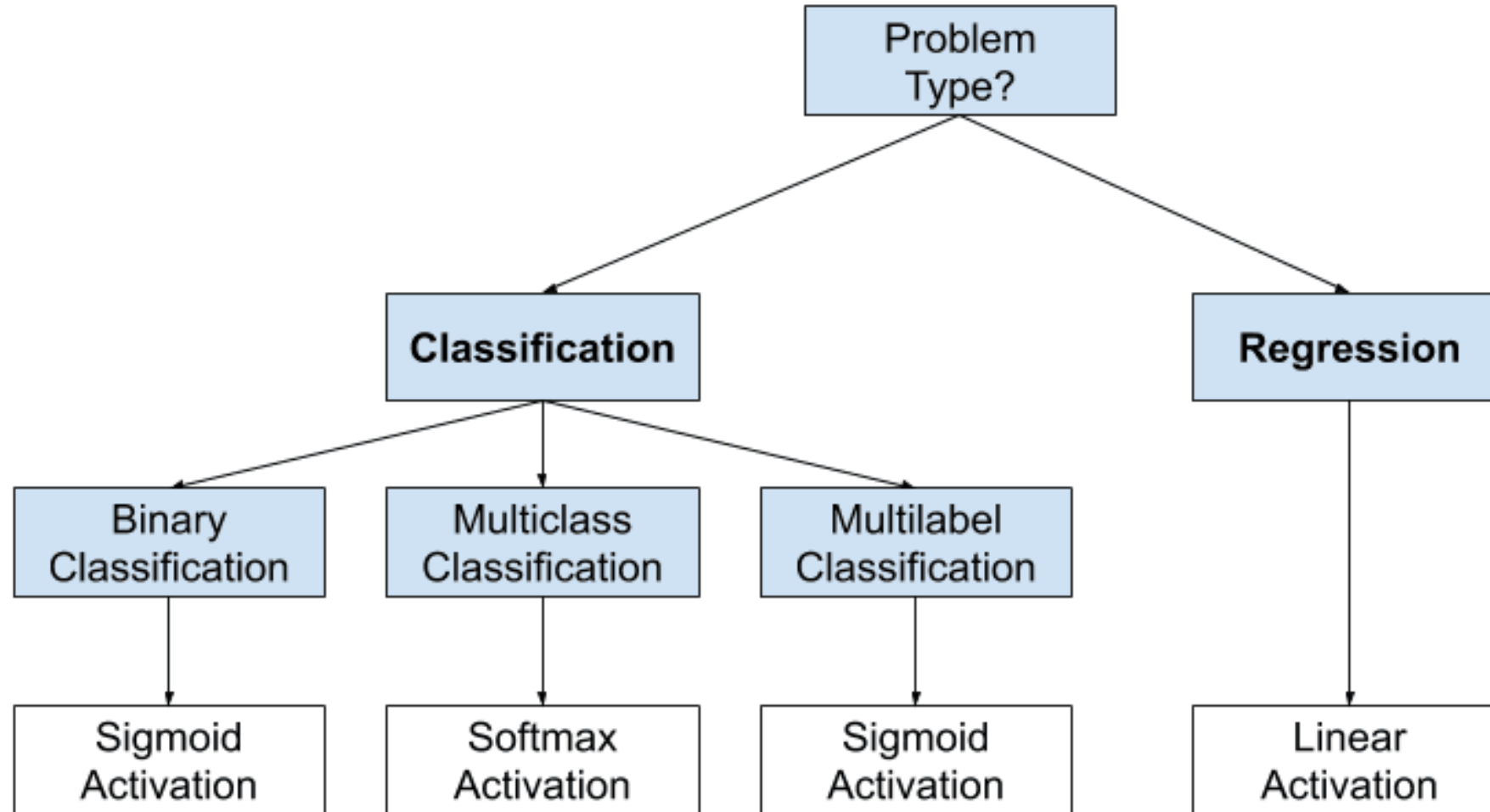
Regression  
(predict **continuous** value)



Classification  
(predict **discrete** value)



# Desired Output Driven by Task



# Today's Topics

- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- Neural network architecture: activation functions
- Neural network architecture: output units
- Programming tutorial: link to code and video in Canvas



# Today's Topics

- Motivation for neural networks: need non-linear models
- Neural network architecture: hidden layers
- Neural network architecture: activation functions
- Neural network architecture: output units
- Programming tutorial



*The End*