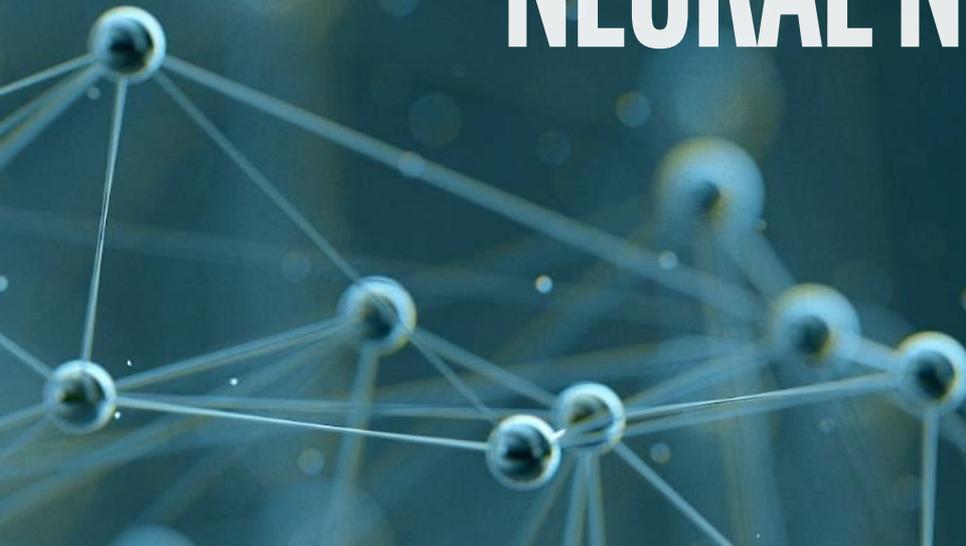


# INTRODUCTION TO NEURAL NETWORKS



# INPUT NORMALIZATION

# NORMALIZATION AND RESCALING OF INPUTS

Ideally, we want all inputs to have roughly the same scale

Helps stabilize learning

How to adjust inputs?

# SHIFTING AND RESCALING

To have inputs range [0, 1]:

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

To have inputs range [-1, 1]:

$$x_i = 2 \left( \frac{x_i - x_{min}}{x_{max} - x_{min}} \right) - 1$$

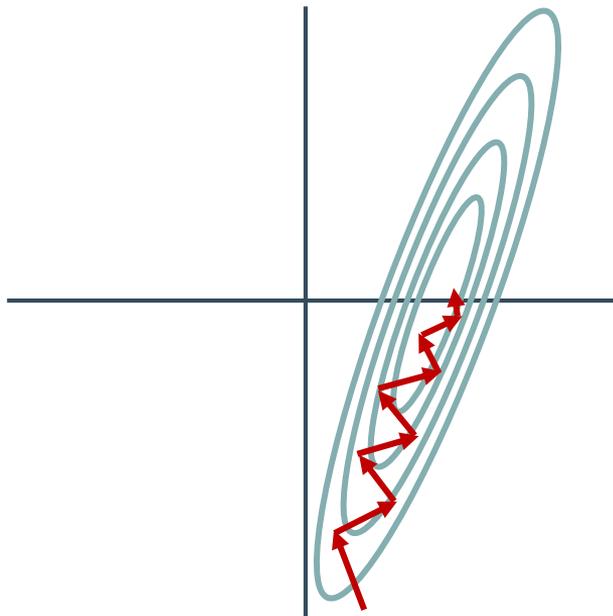
# NORMALIZING

Mean zero, standard deviation one

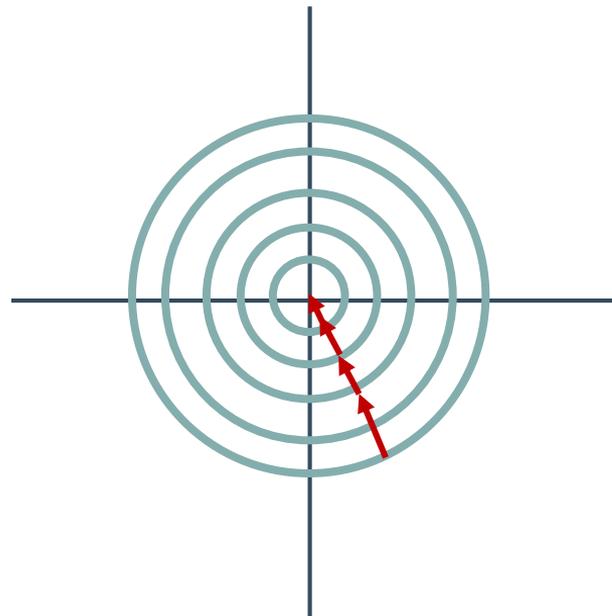
$$x_i = \frac{x_i - \bar{x}}{\sigma}; \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

# OVERHEAD VIEW OF COST CURVE

Un-Normalized



Normalized



# REGULARIZATION

# REGULARIZATION

Goal: prevent overfitting

Want network to generalize beyond training data

How to do this?

One way: stop large weights from dominating

How? Penalize models with large weights

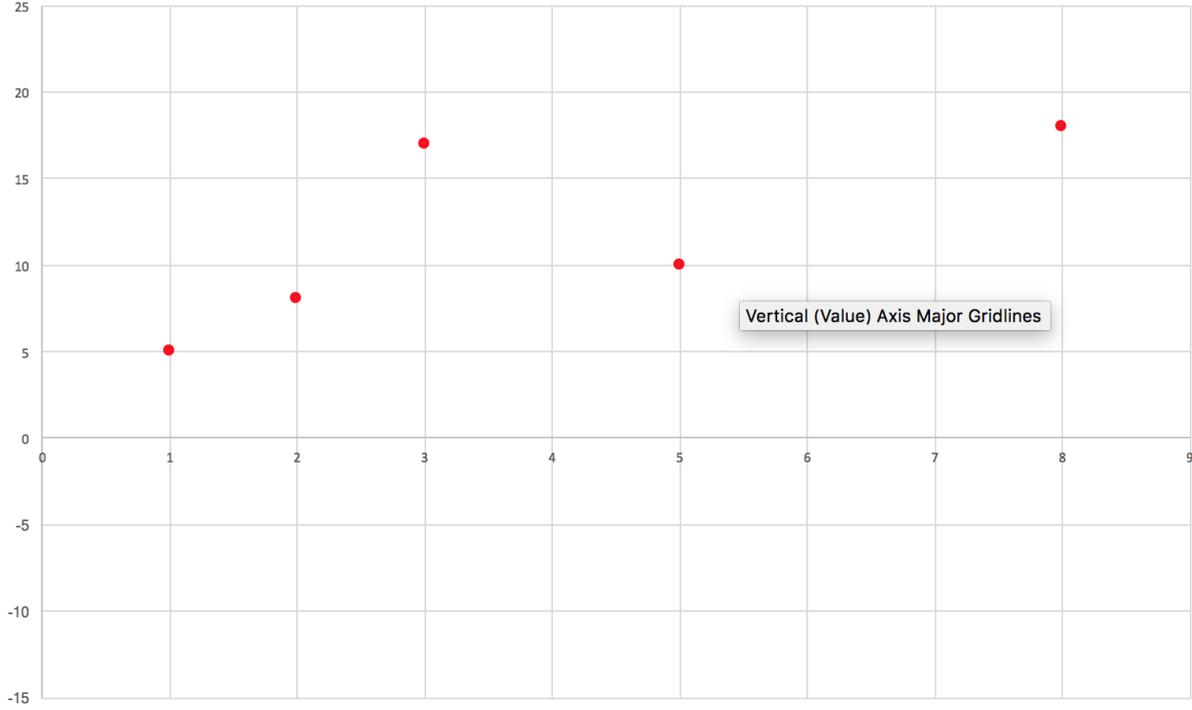
# L2 REGULARIZATION

Adjust loss function to include a second term

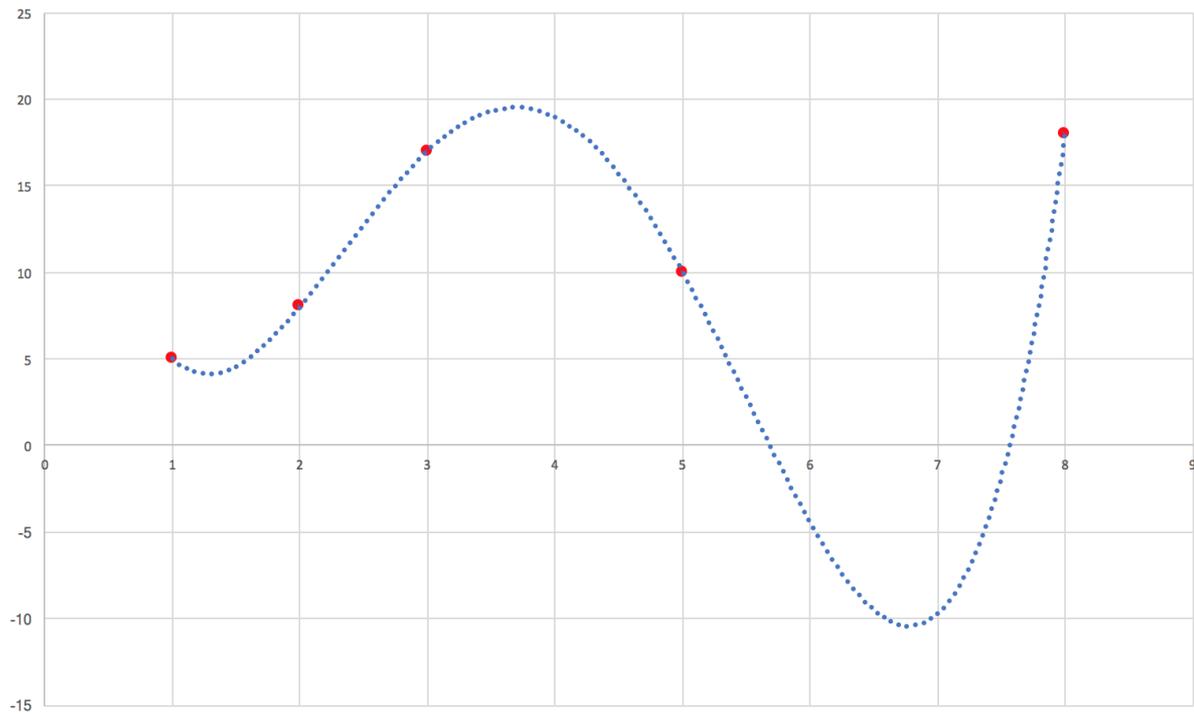
$$J = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^m w_j^2$$

- Bigger sum of weights costs us more
- $\lambda$  is the regularization hyper-parameter
  - Bigger  $\lambda$  means more regularization
  - More penalty for big weights
  - Less attention to raw data fit

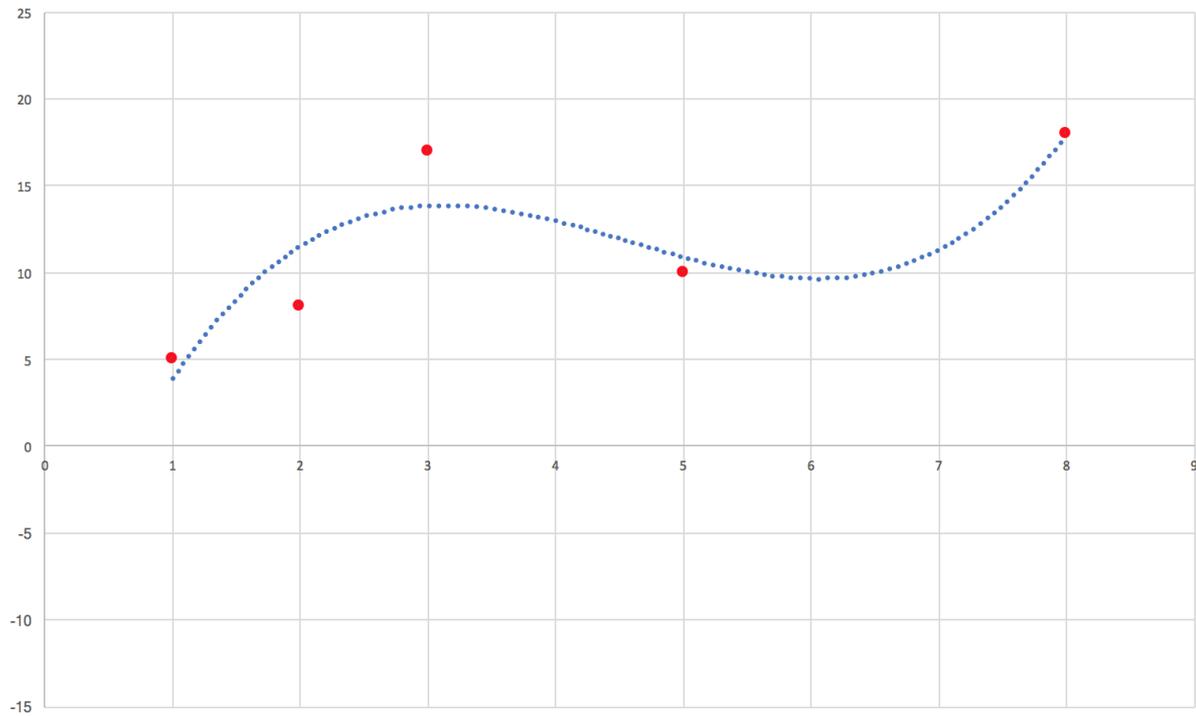
# SCATTER PLOT



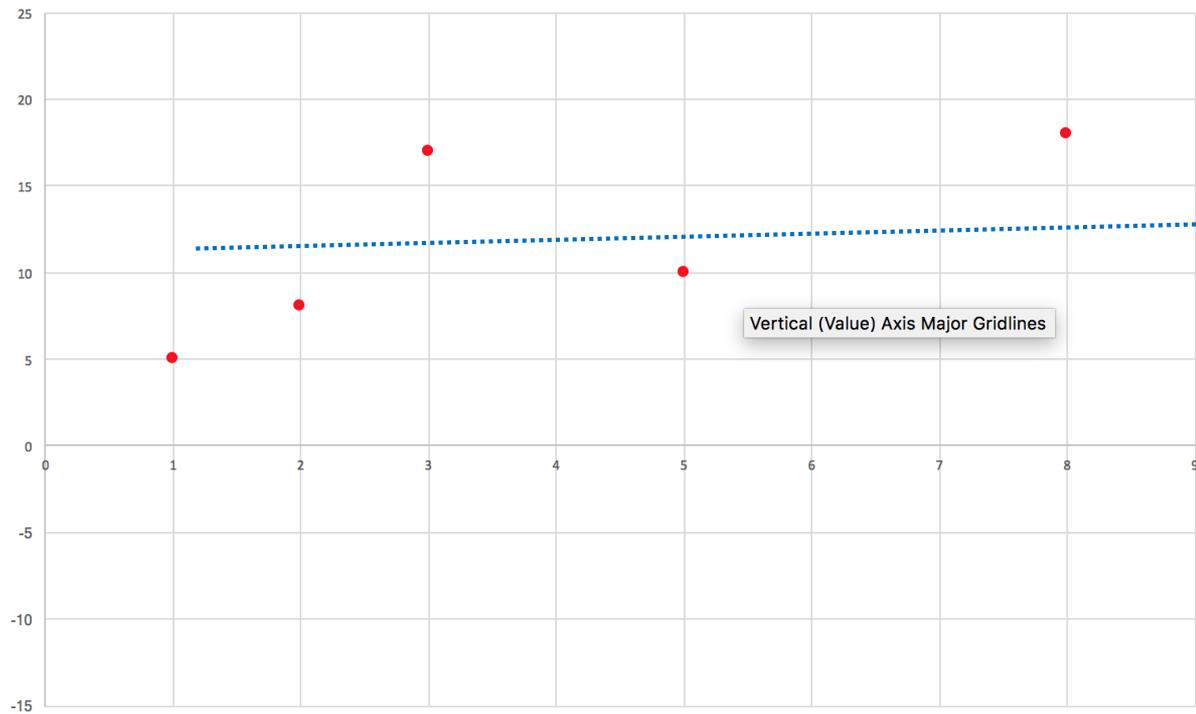
# OVERFITTING



# POSSIBLE MODEL AFTER REGULARIZATION



# TOO MUCH REGULARIZATION!



# DIFFERENT TYPES OF REGULARIZATION

We'll see different techniques that act as a form of regularization

- Not always just a term of the loss function
- Dropout (next class!)

**Key takeaway:**

- regularization = less memorization

# NEURAL NETWORKS

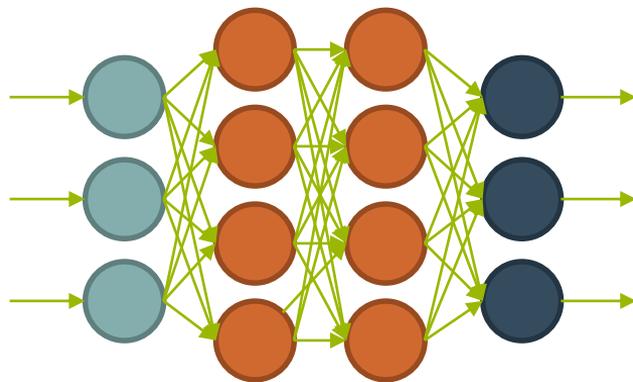
# BASIC IDEA

Use biology as inspiration for math model

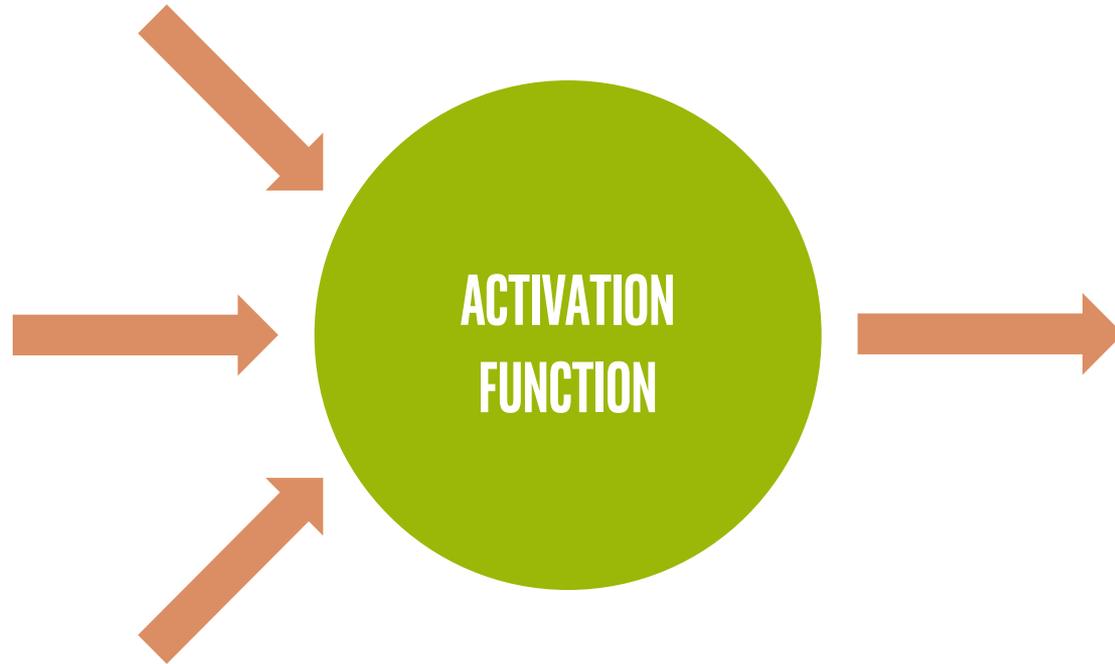
## Neurons:

- Get signals from previous neurons
- Generate signal (or not) according to inputs
- Pass that signal on to future neurons

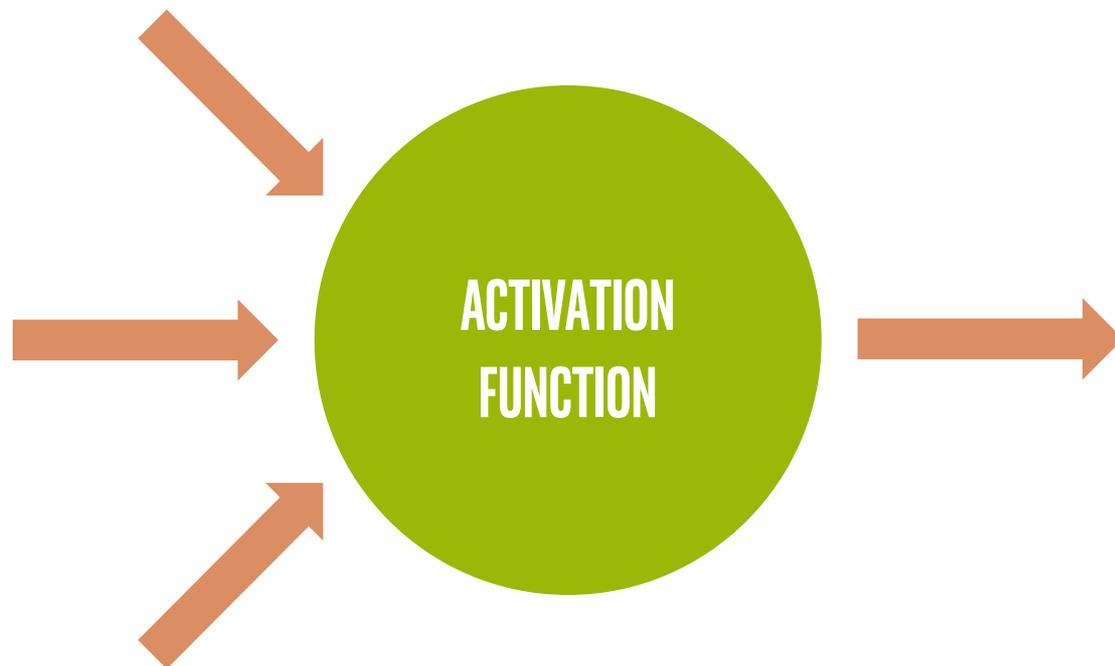
By layering many neurons, can create complex model



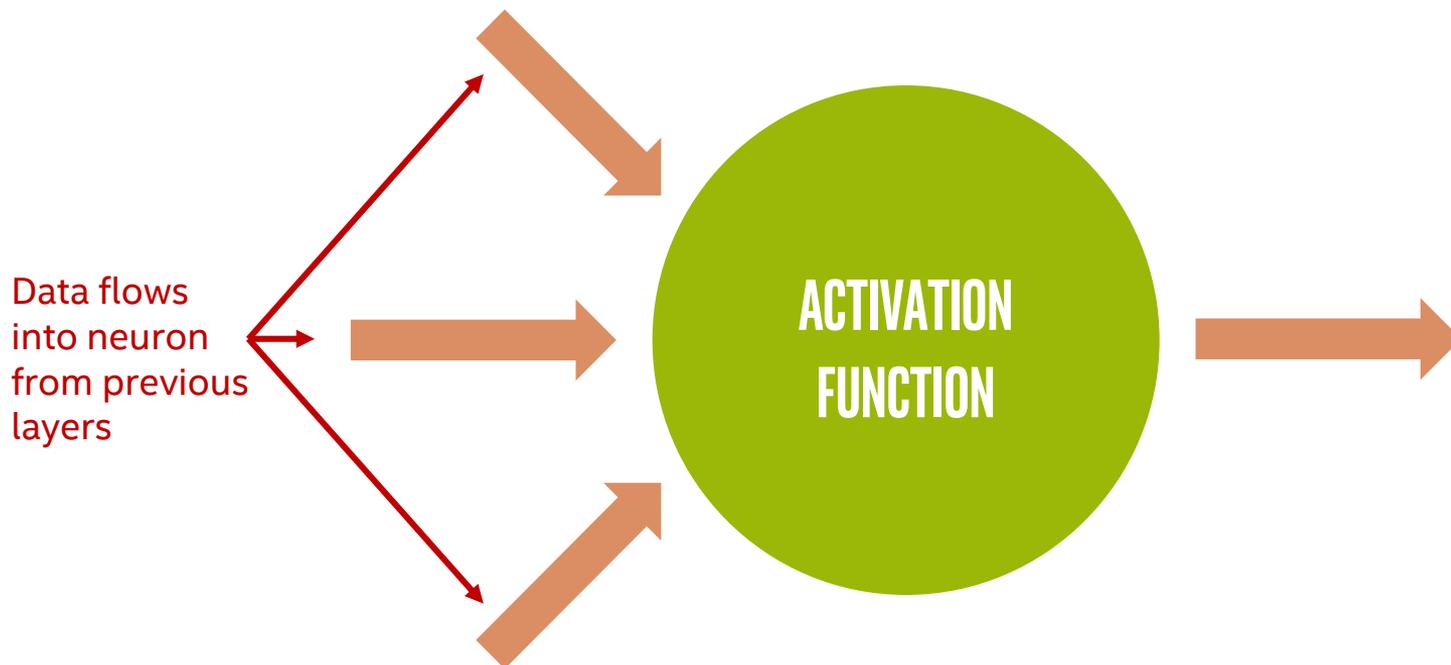
# THE BASIC “NEURON” VISUALIZATION



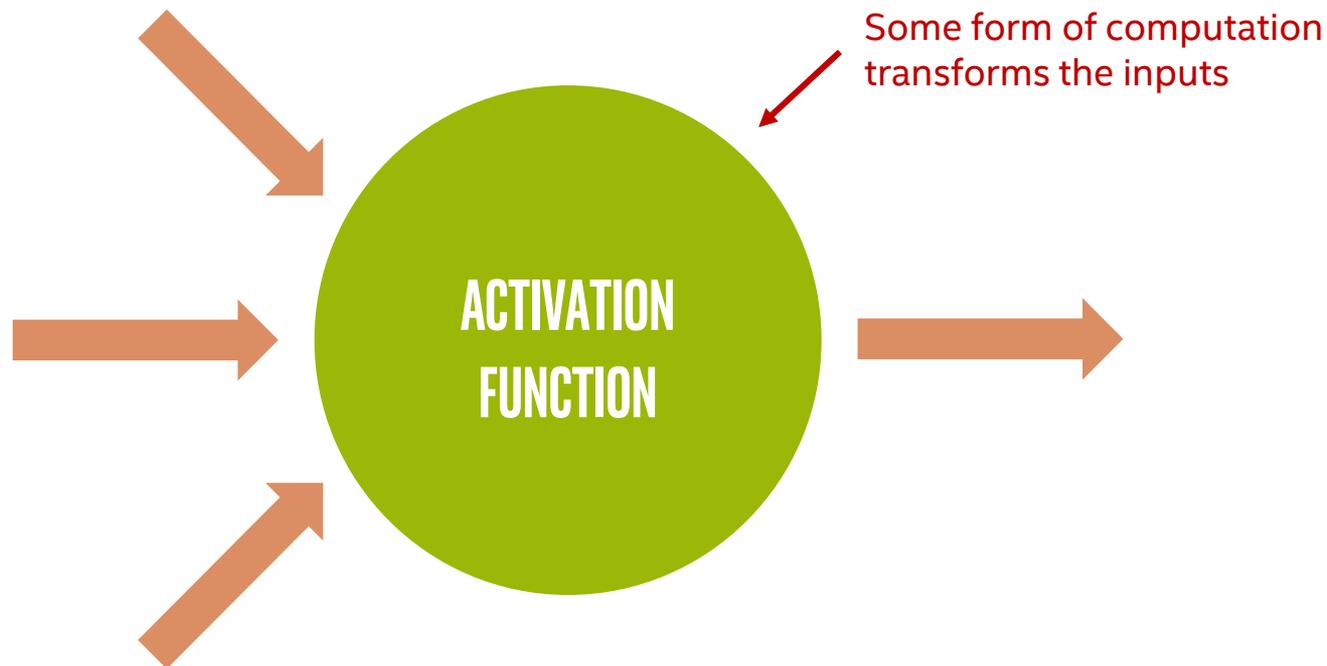
# READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH



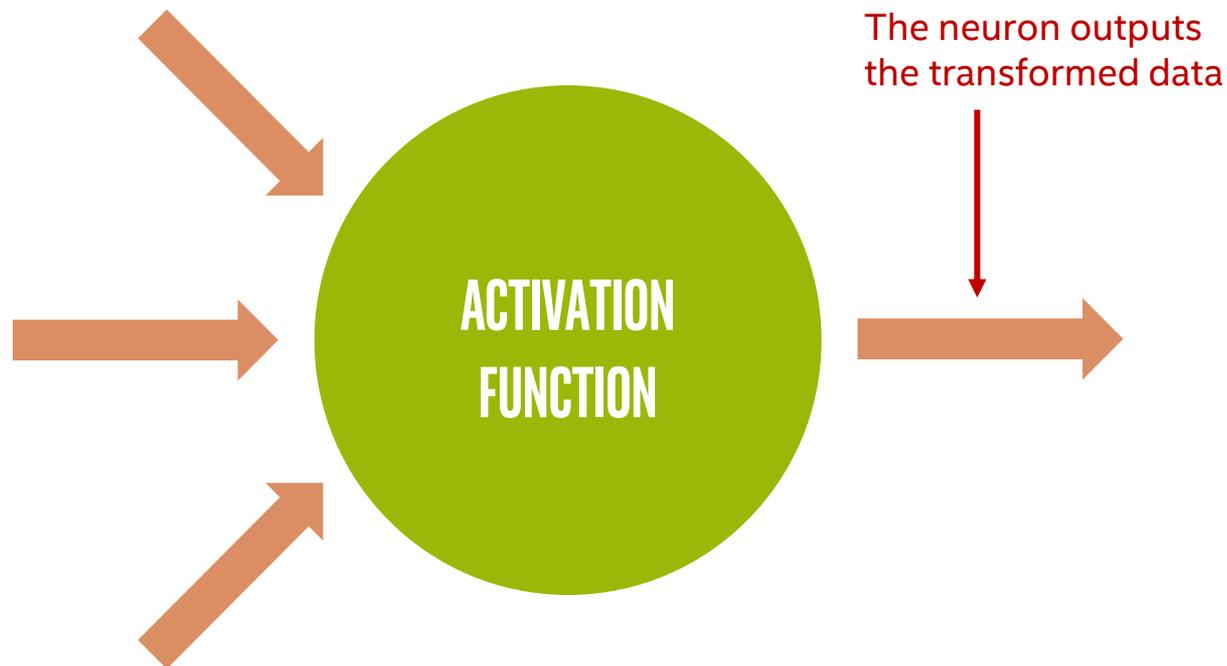
# READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH



# READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH



# READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH



# MATHEMATICAL DESCRIPTION OF A NEURON

$$z = b + \sum_{i=1}^m W_i \cdot x_i$$

$$= W^t x + b$$

$$a = f(z)$$

# MATHEMATICAL DESCRIPTION OF A NEURON

$$\begin{aligned} z &= b + \sum_{i=1}^m W_i \cdot x_i \\ &= W^t x + b \\ a &= f(z) \end{aligned}$$

$z = \text{net input}$

$m \text{ inputs}$

# MATHEMATICAL DESCRIPTION OF A NEURON

$$z = b + \sum_{i=1}^m W_i \cdot x_i$$

Vectorized



$$= W^t x + b$$

$$a = f(z)$$

# MATHEMATICAL DESCRIPTION OF A NEURON

$$z = b + \sum_{i=1}^m W_i \cdot x_i$$

$$= W^t x + b$$

Activation value

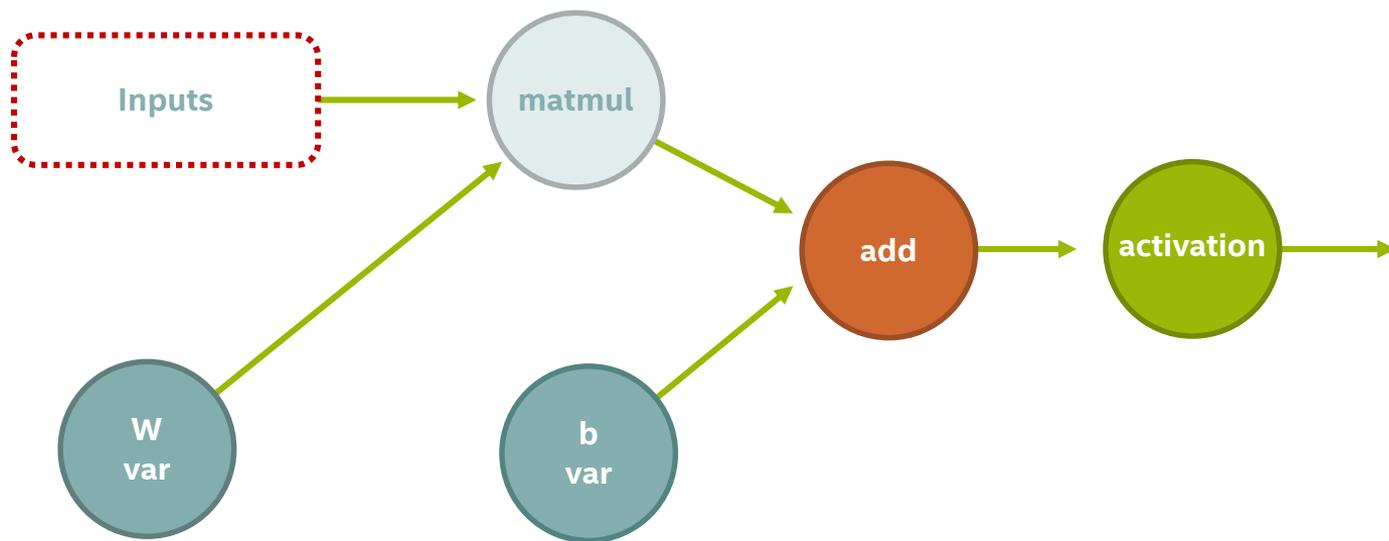


$$a = f(z)$$

Activation function

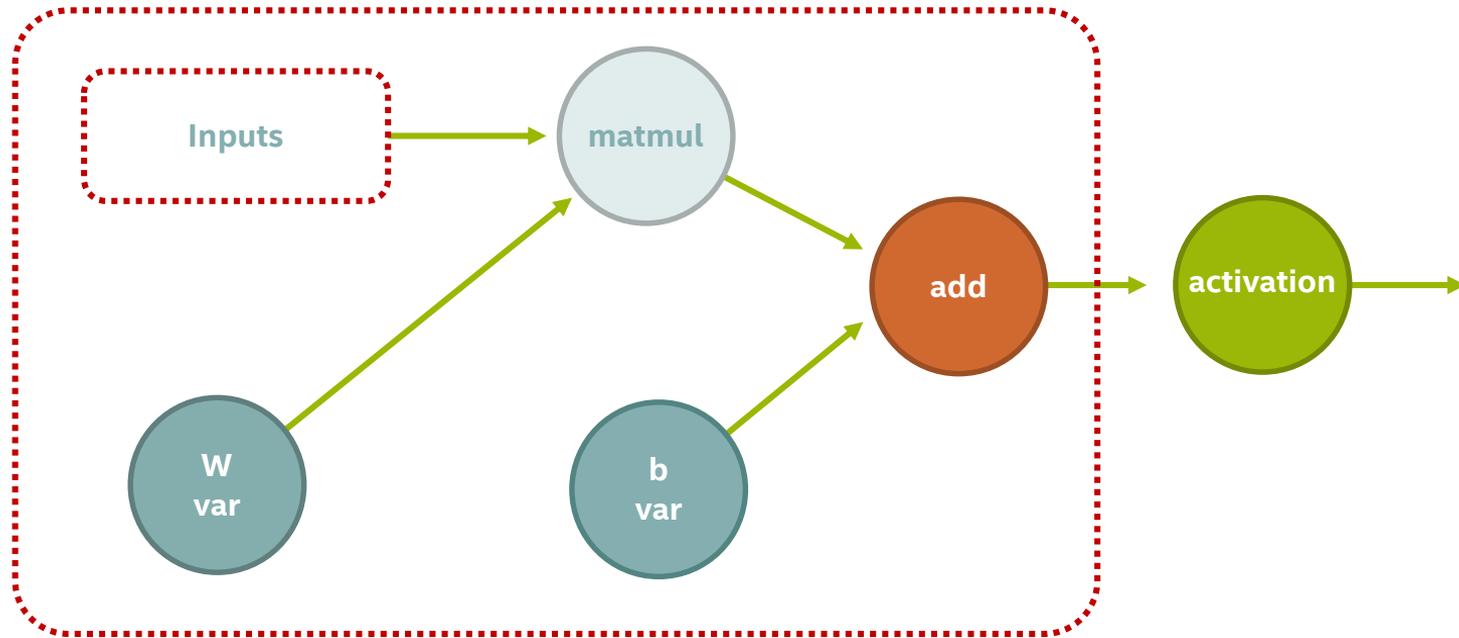


# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)

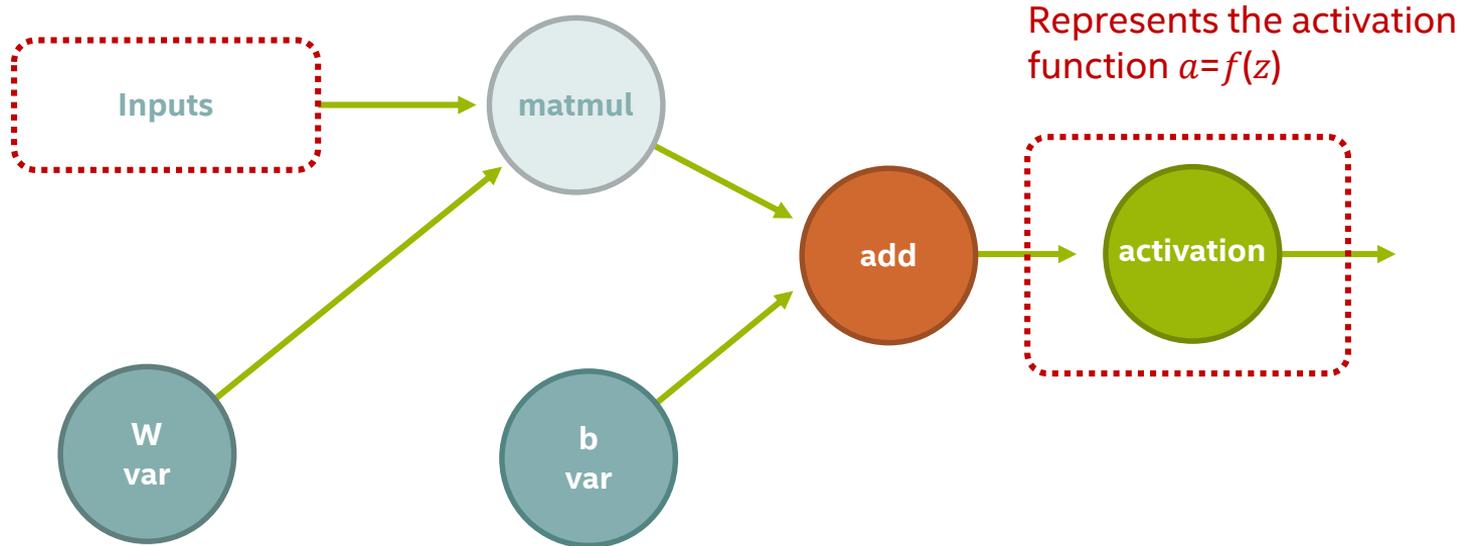


# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)

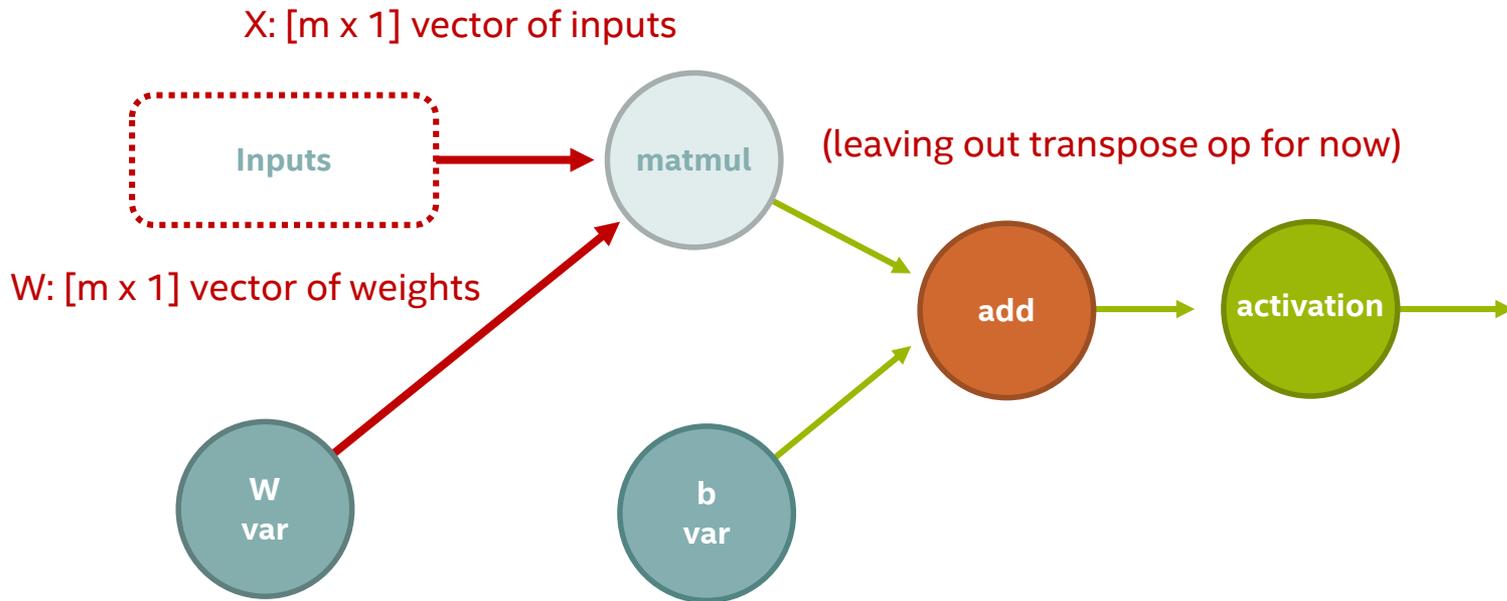
Represents the function  $z = W^tX + b$



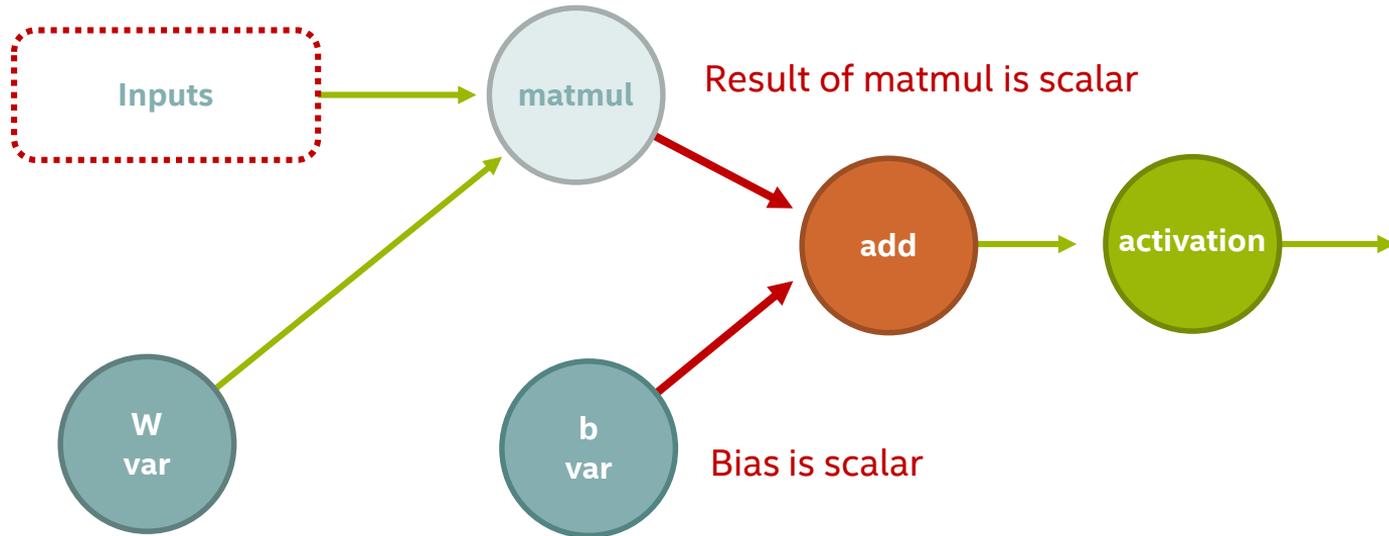
# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)



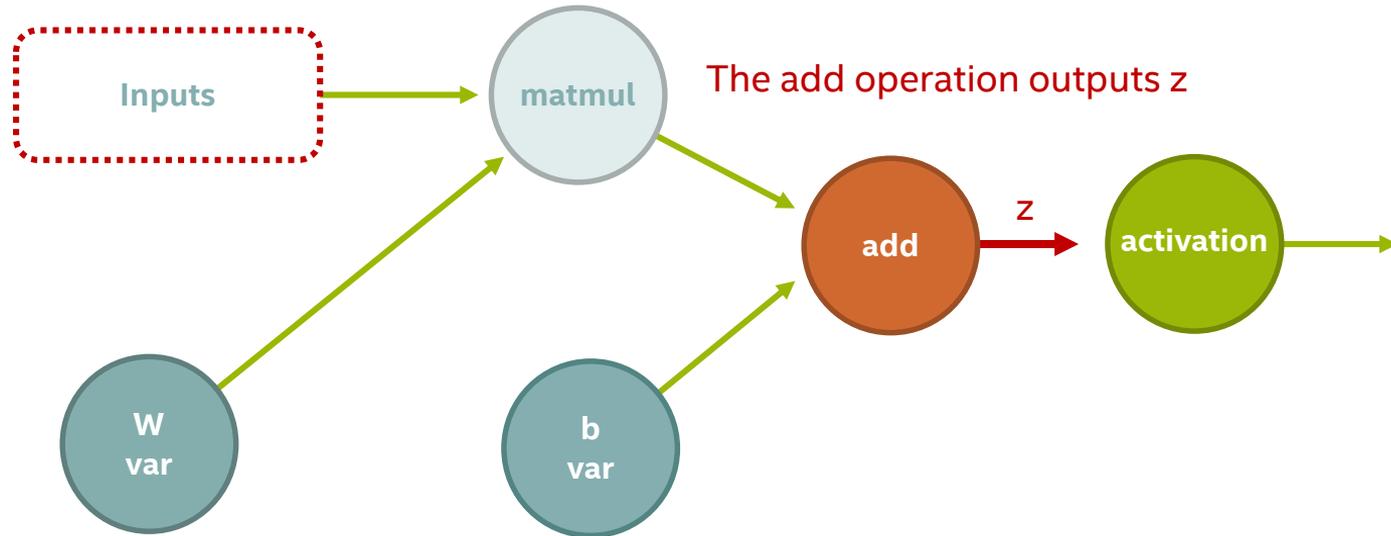
# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)



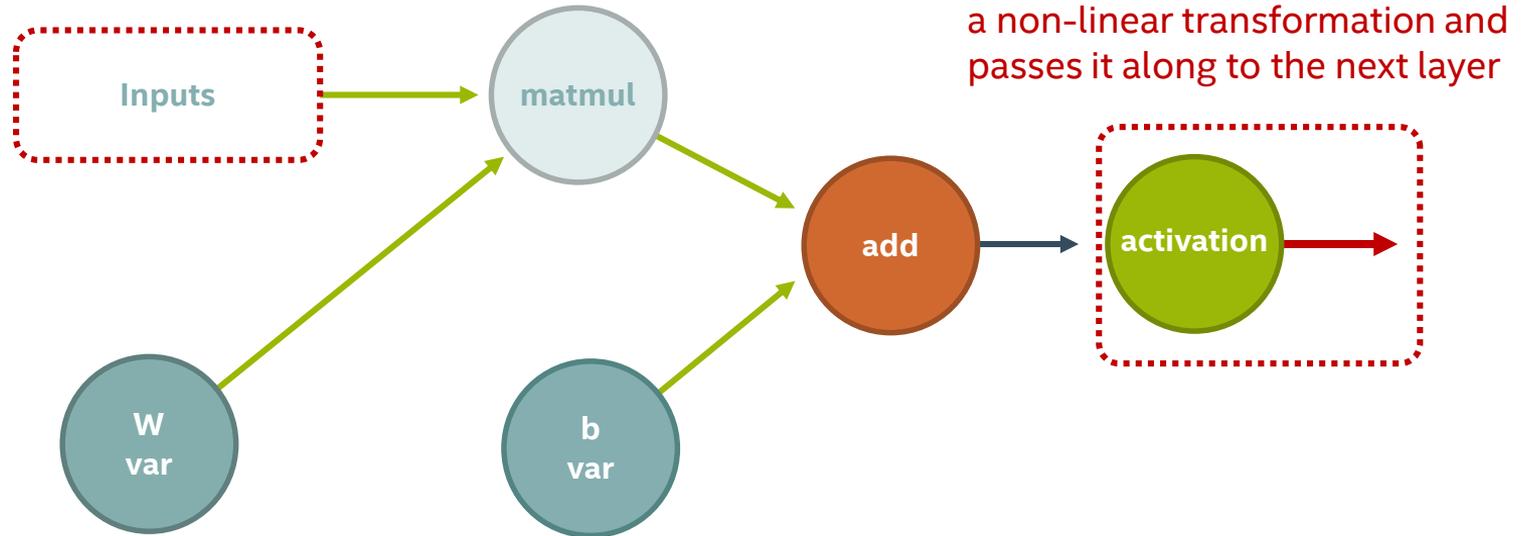
# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)



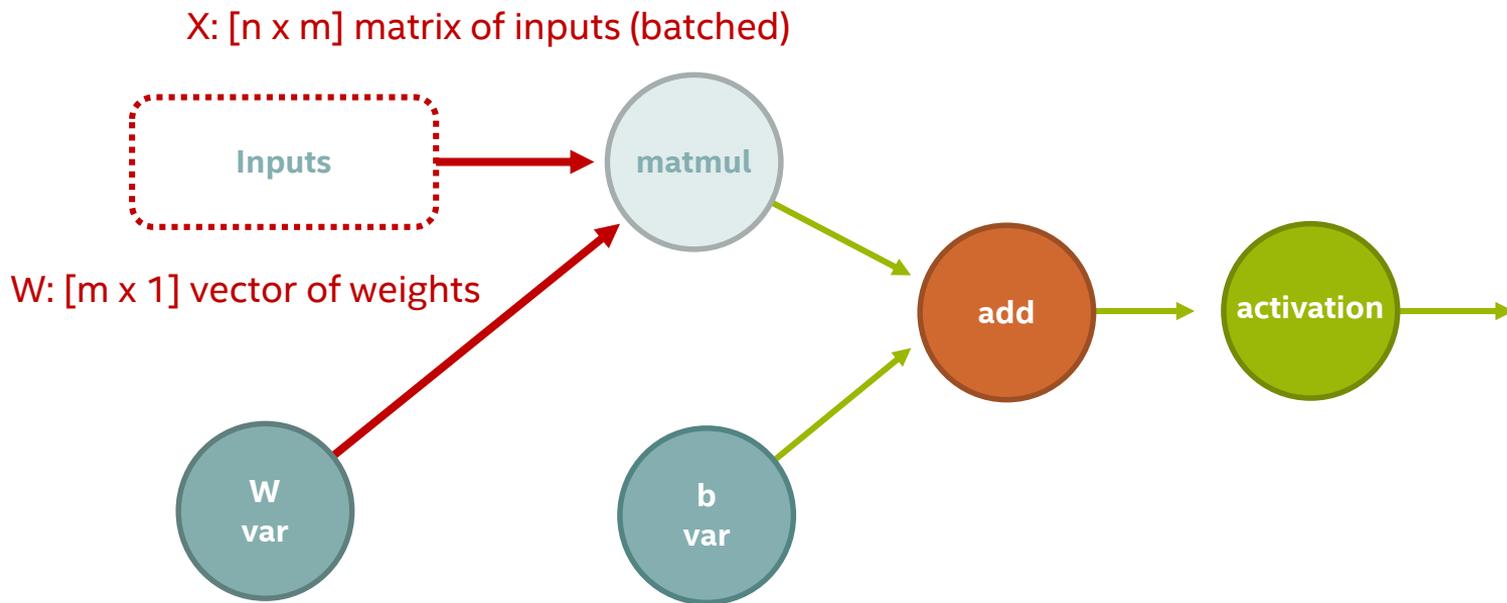
# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)



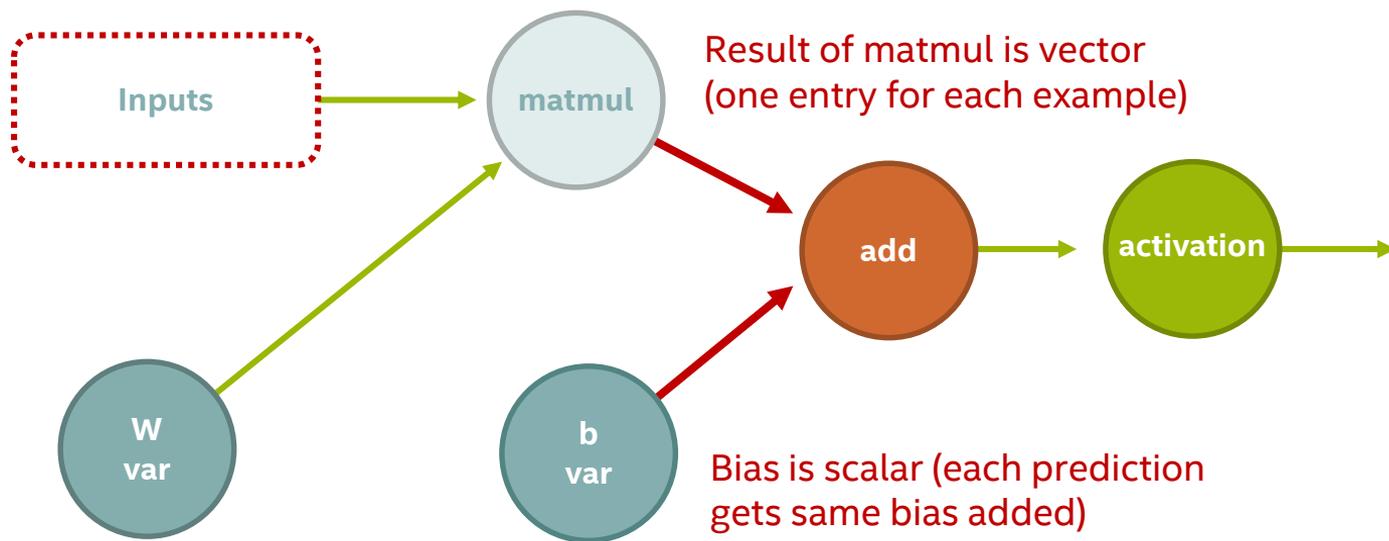
# INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)



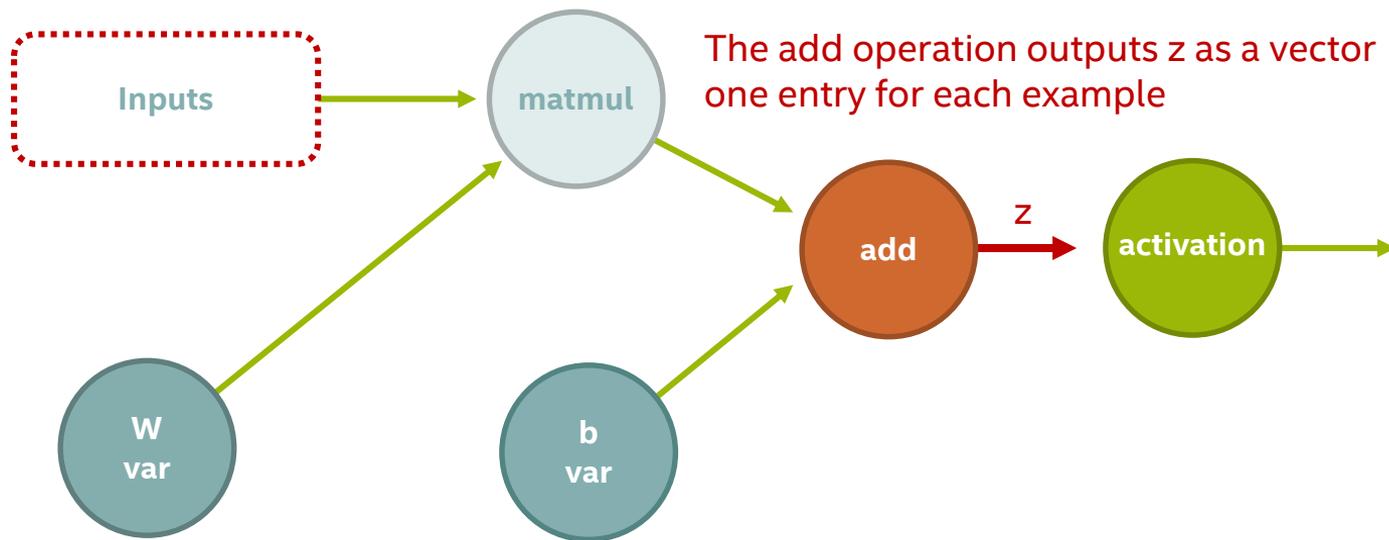
# BATCHED VERSION



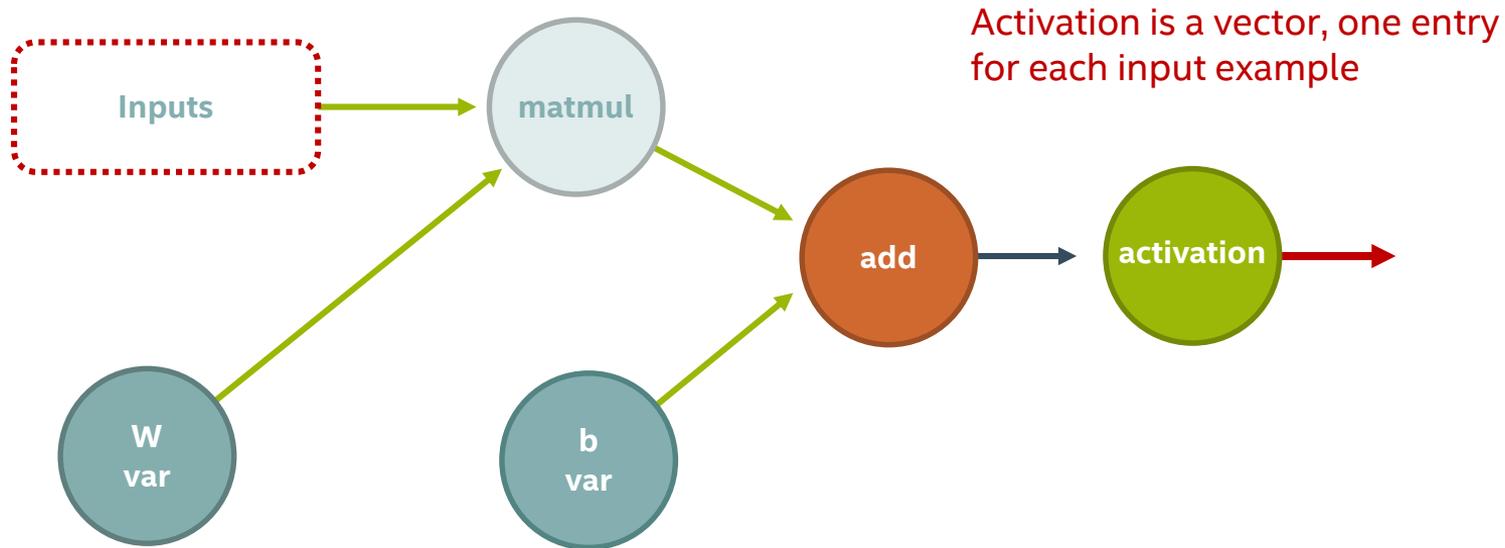
# BATCHED VERSION



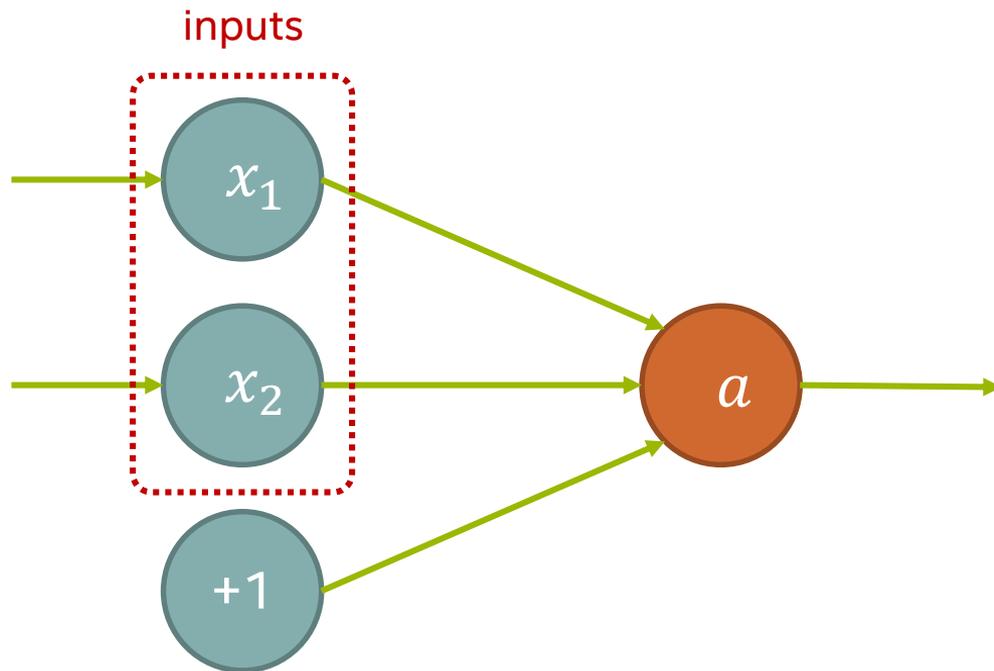
# BATCHED VERSION



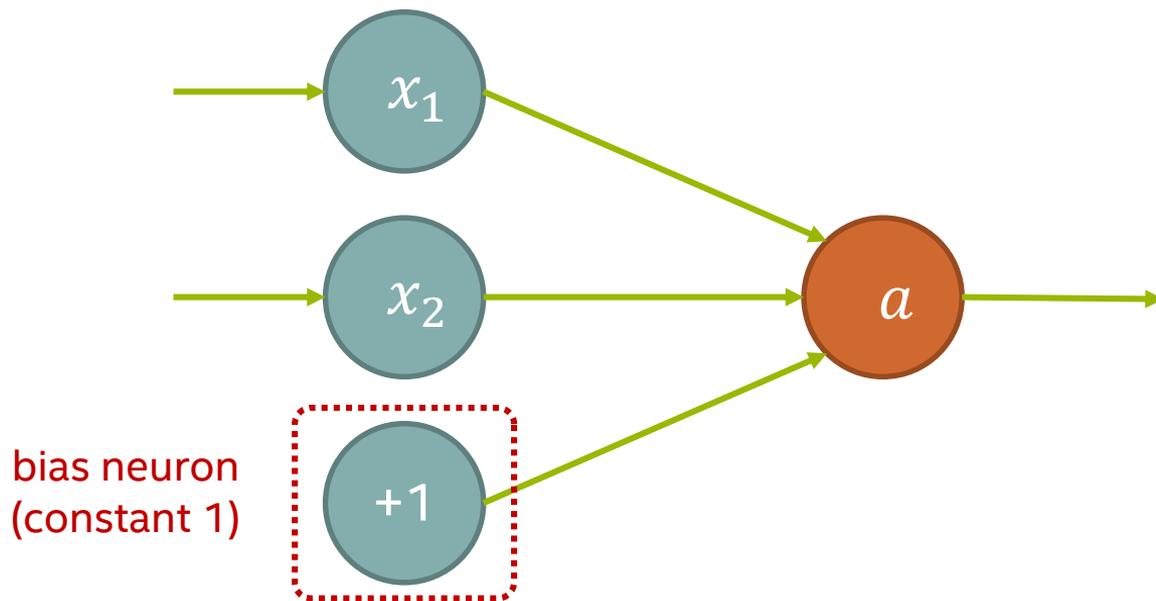
# BATCHED VERSION



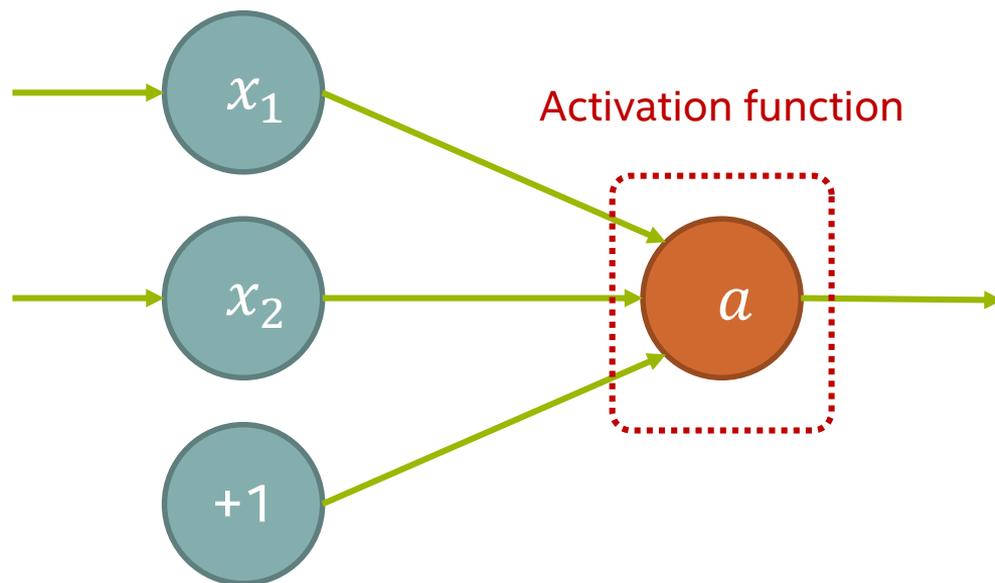
# A CLASSICAL VISUALIZATION OF NEURONS



# A CLASSICAL VISUALIZATION OF NEURONS

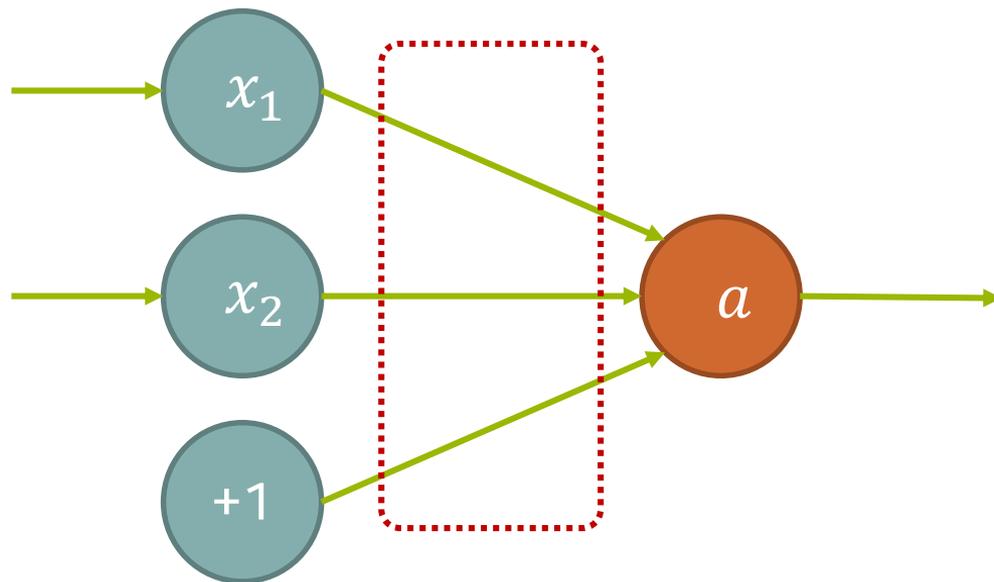


# A CLASSICAL VISUALIZATION OF NEURONS

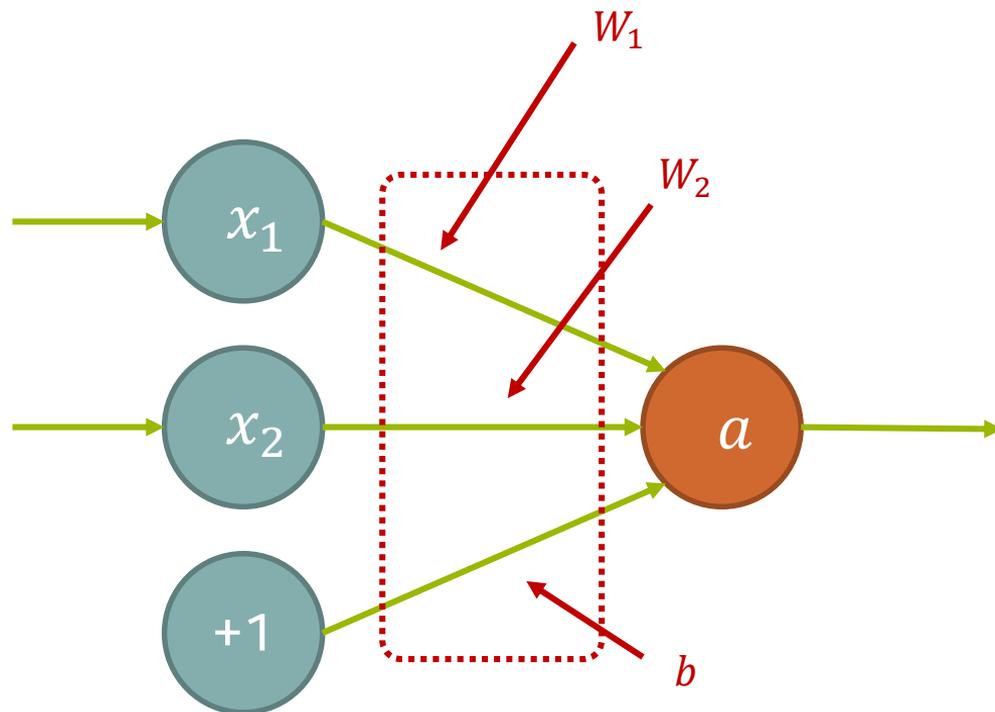


# WHERE ARE THE WEIGHTS?

Weights are shown to be arrows  
in classical visualizations of NNs

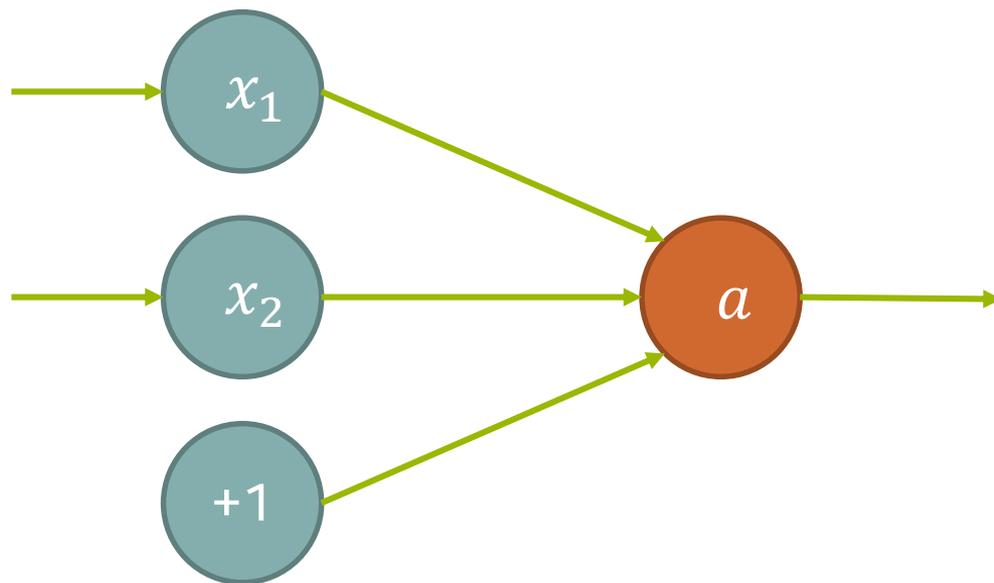


# WHERE ARE THE WEIGHTS?

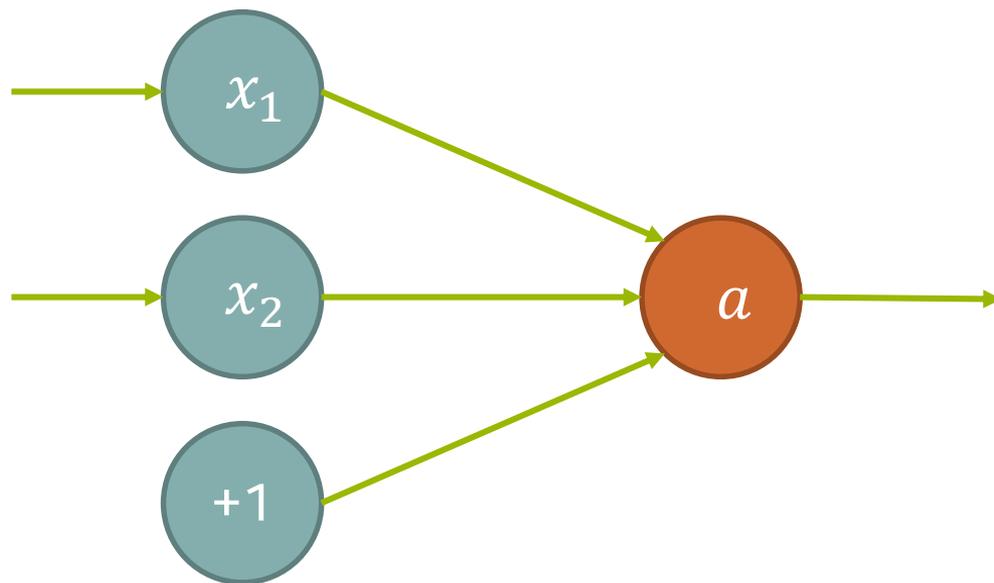


# WHERE IS THE NET VALUE (z)?

Not shown! Usually given  
via formulas in papers



To keep visual noise down, we'll use this notation for now



# OUR FIRST ACTIVATION FUNCTION

# BASIC IDEA

Model inspired by biological neurons

Biological neurons either pass no signal, full signal, or something in between

Want a function that is like this and has an easy derivative.

# SIGMOID (LOGISTIC)

Value at  $z \ll 0$ ?  $\approx 0$

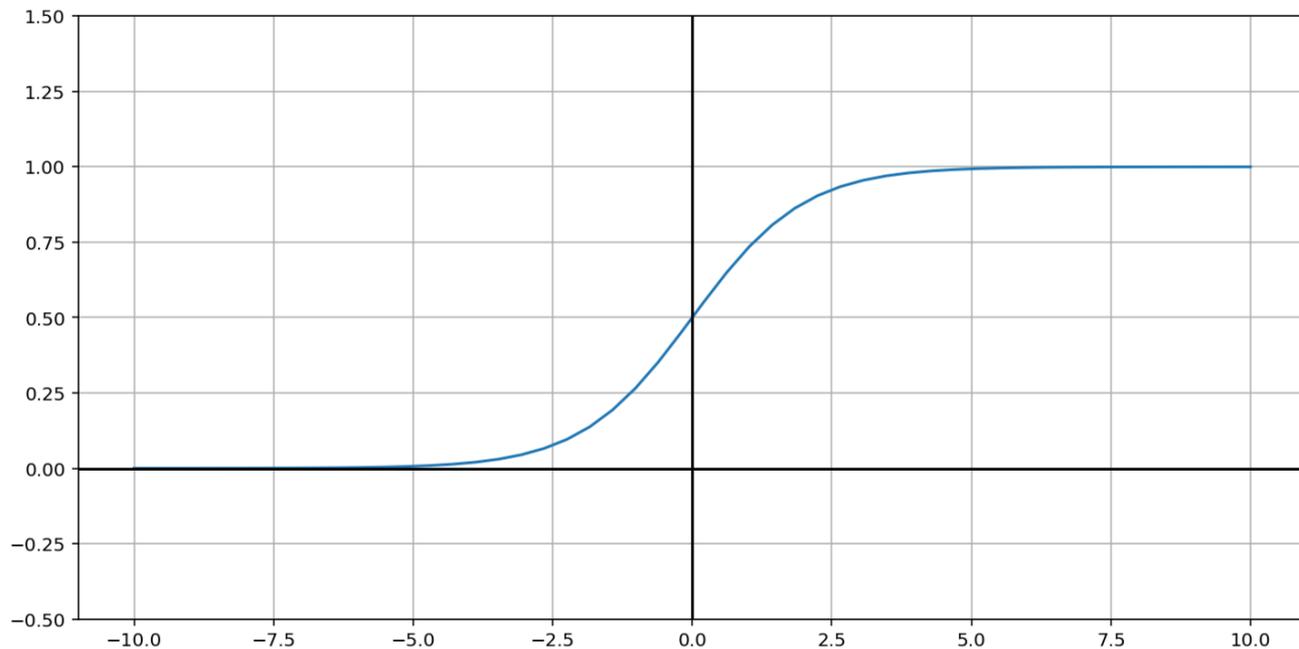
Value at  $z=0$ ?  $= 0.5$

Value at  $z \gg 0$ ?  $\approx 1$

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

# SIGMOID (LOGISTIC)

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



# EASY DERIVATIVE?

Quotient rule

$$\frac{d}{dx} \cdot \frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

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

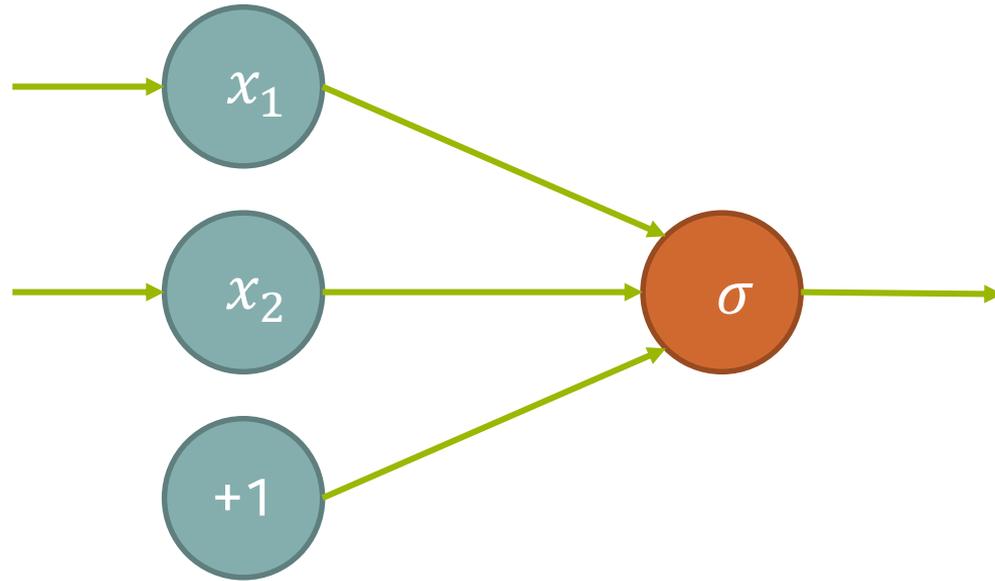
$$\sigma'(z) = \frac{0 - (-e^{-z})}{(1 + e^{-z})^2} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

$$= \frac{1 + e^{-z} - 1}{(1 + e^{-z})^2} = \frac{\cancel{1 + e^{-z}}}{(1 + e^{-z})^{\cancel{2}}} - \frac{1}{(1 + e^{-z})^2}$$

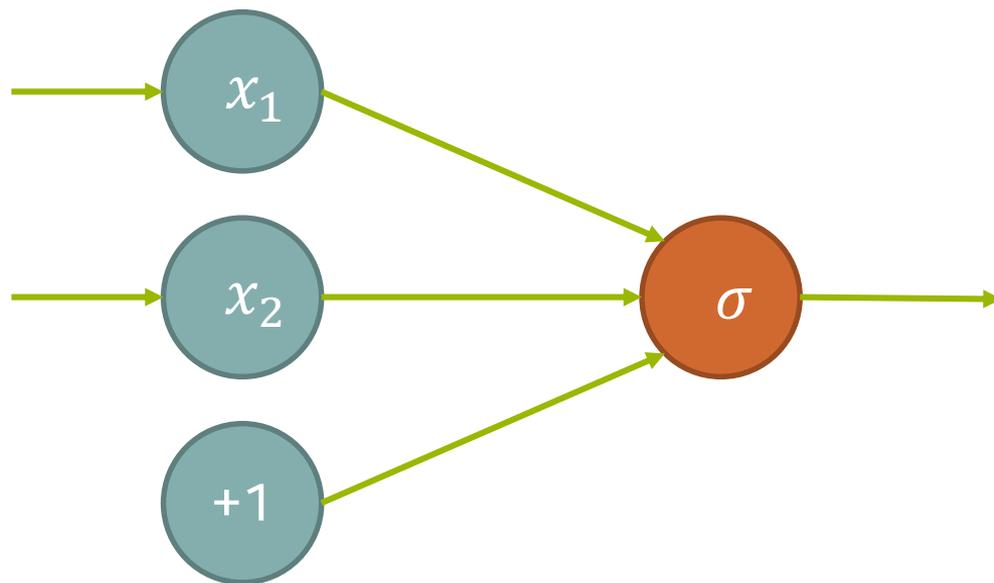
$$= \frac{1}{1 + e^{-z}} - \frac{1}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \left( 1 - \frac{1}{1 + e^{-z}} \right)$$

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

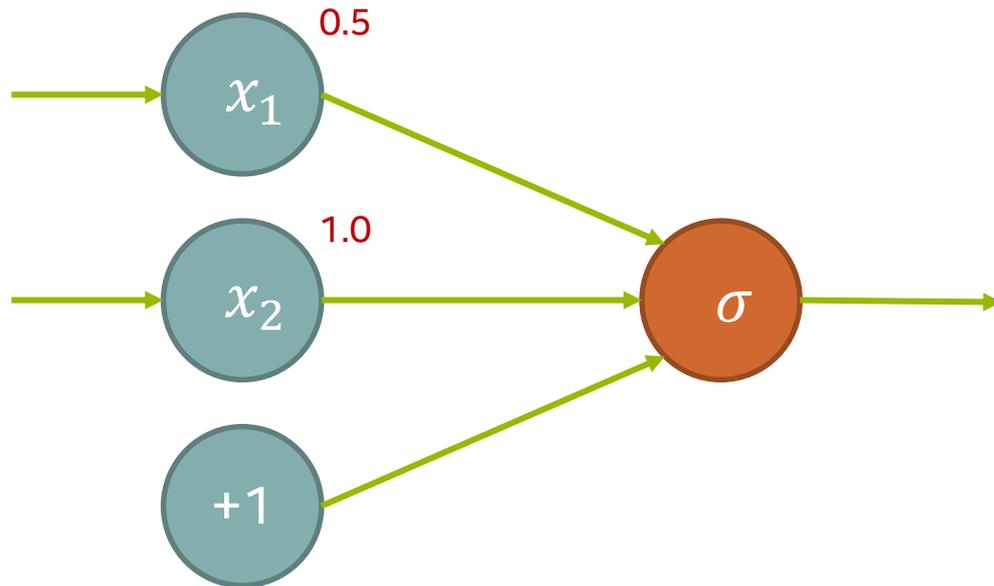
We can plug in the sigmoid as our activation function



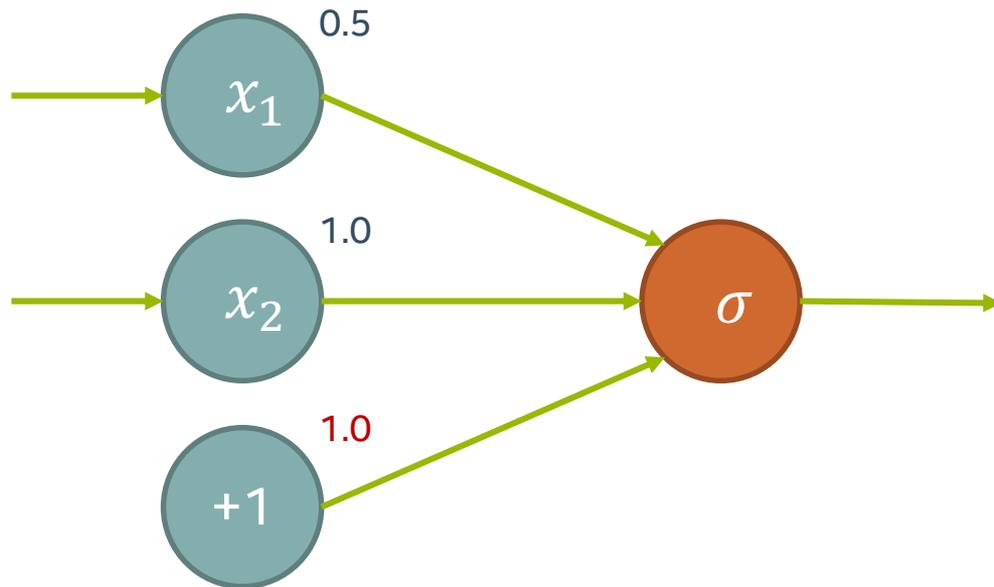
Depending on the inputs and weights, the neuron will output a value between (0, 1)



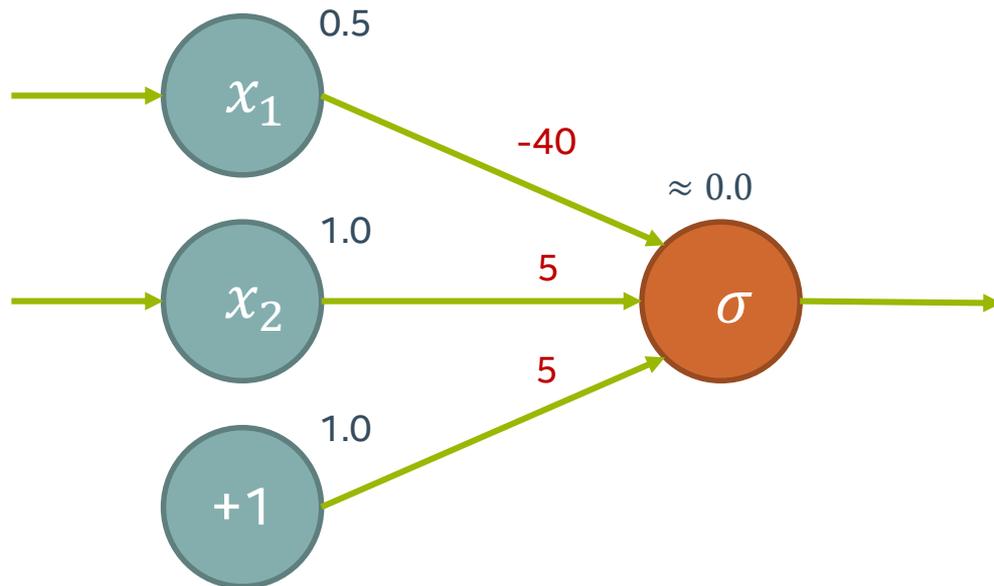
As an example, assume that  $x_1$  outputs 0.5, and  $x_2$  outputs 1.0



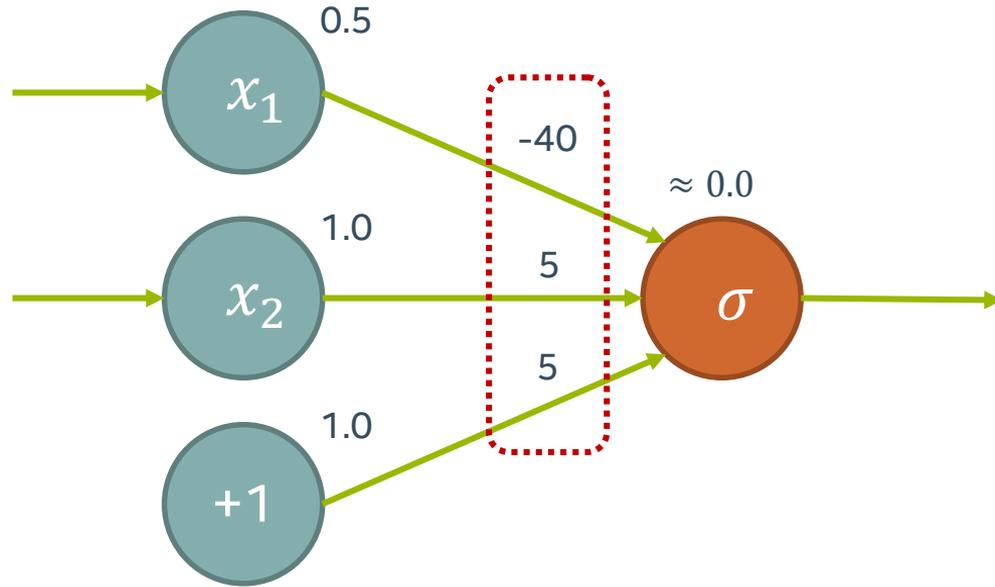
Recall that the bias neuron always outputs 1.0



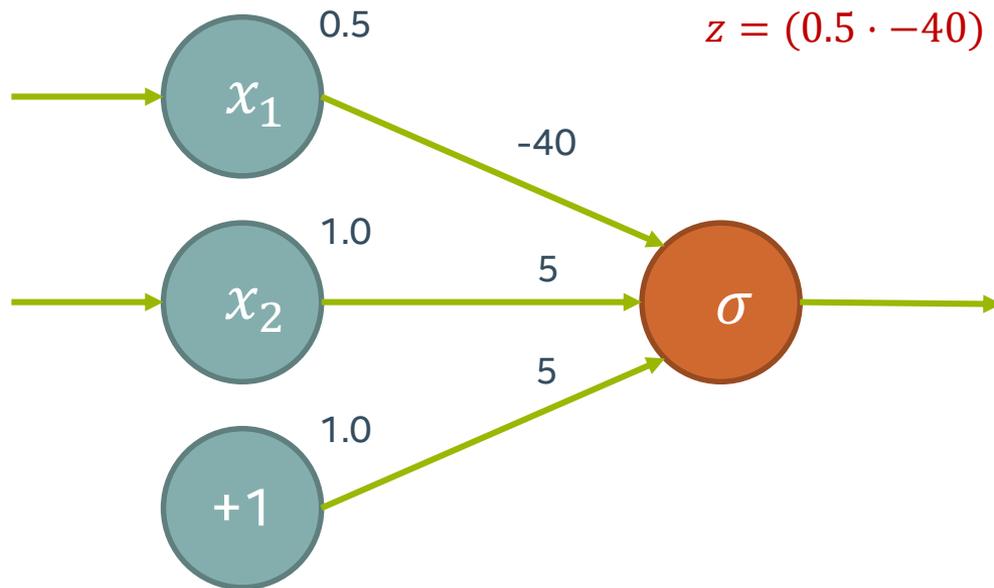
We can now indicate the weights on the connecting edges



When training, we adjust the weights parameters to create outputs that better fit the data

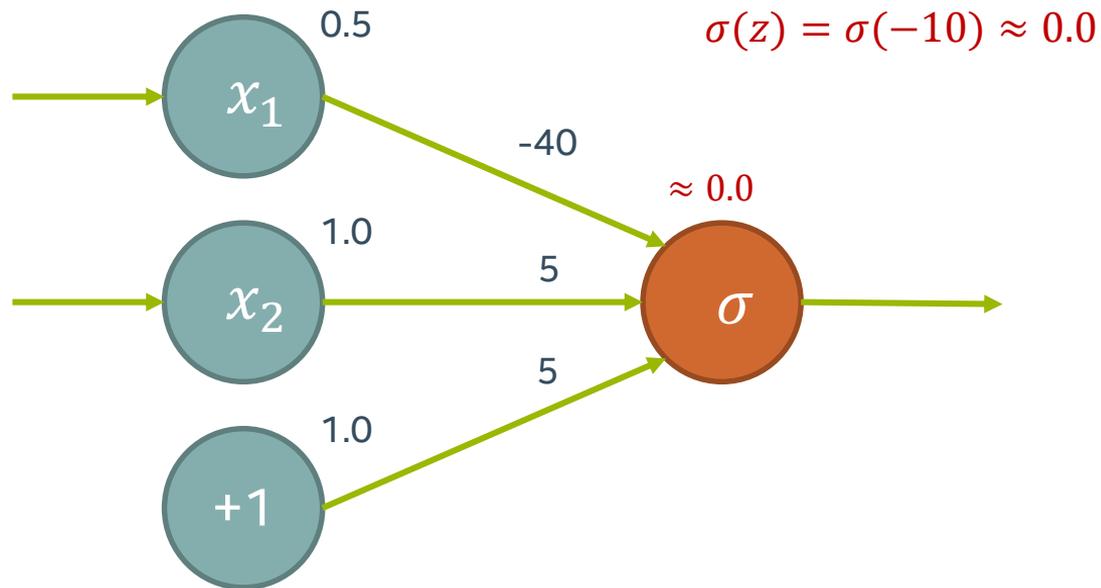


With these values, we end up with net  $z = -10$

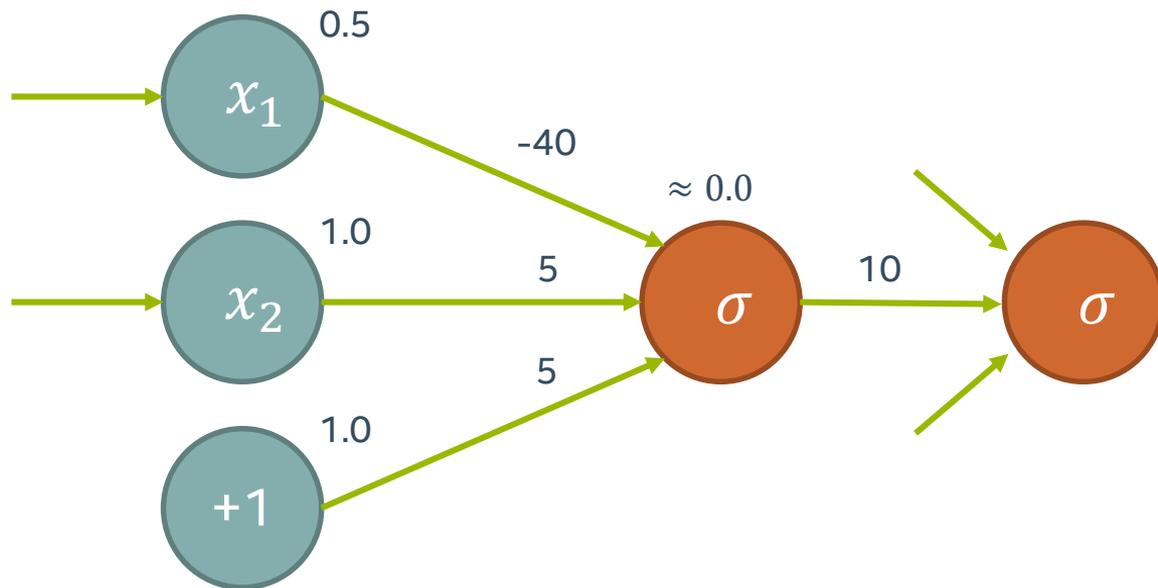


$$z = (0.5 \cdot -40) + (1.0 \cdot 5) + (1.0 \cdot 5)$$

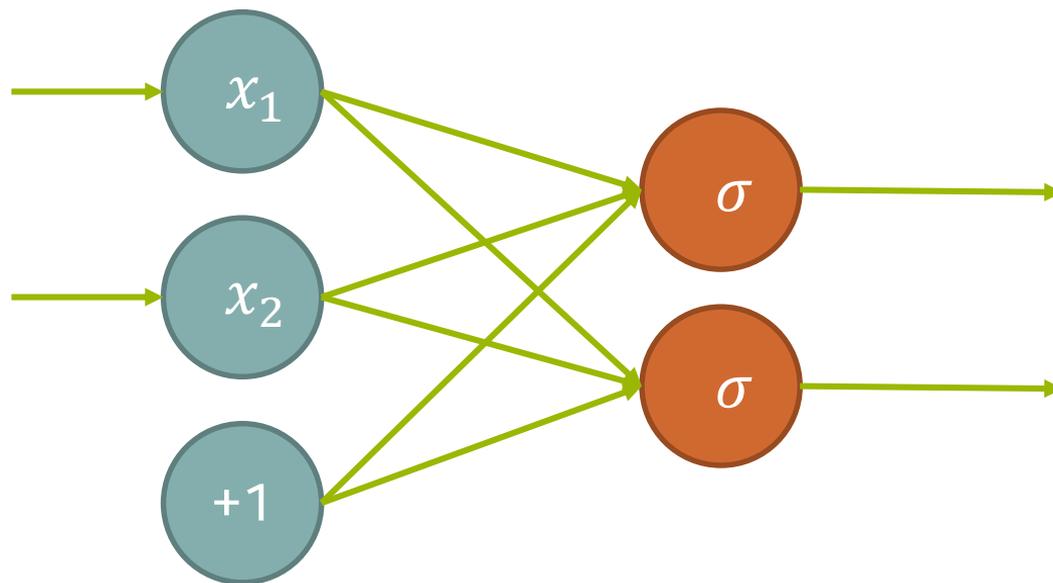
We then plug  $z$  into the sigmoid function to get our output



The output can then be passed onto another neuron, with a weight associated with that connection



Inputs don't need to be limited to passing data into a single neuron. They can pass data to as many as we like.



# LAYERS OF NEURONS

# NEURAL LAYERS

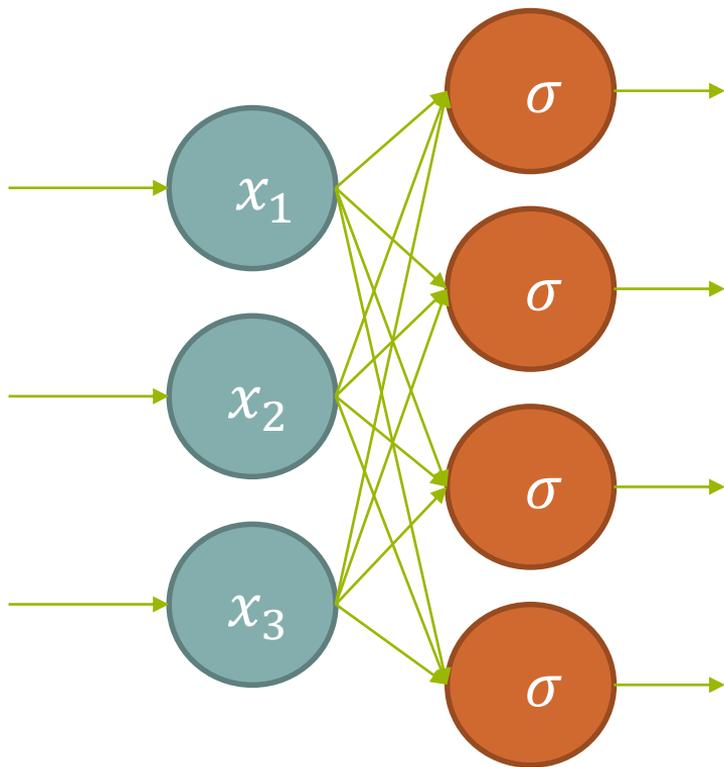
Typically, neurons are grouped into *layers*.

Each neuron in the layer receives input from the same neurons

- **Weights** are different for each neuron

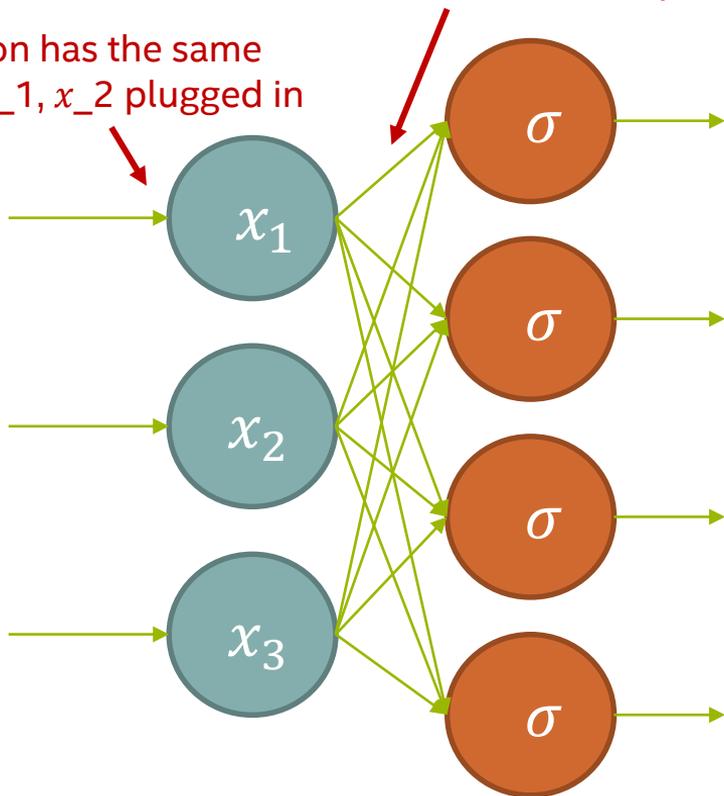
All neurons in this layer output to the same neurons in a subsequent layer

# A SINGLE NEURAL LAYER



# A SINGLE NEURAL LAYER

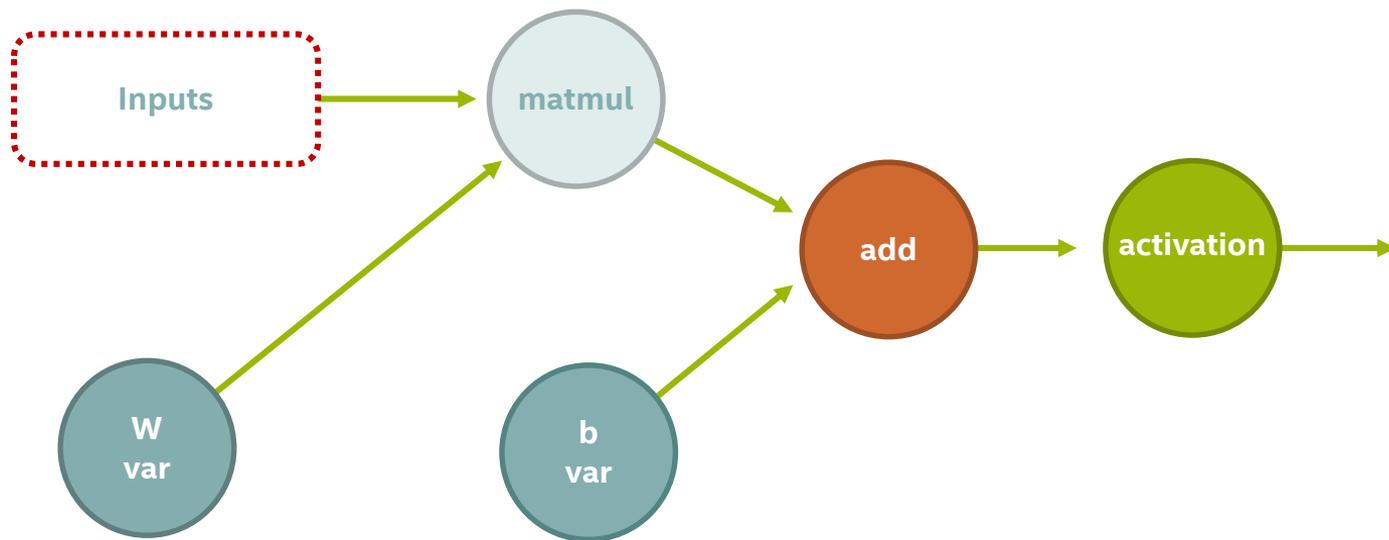
Each neuron has the same value for  $x_1, x_2$  plugged in



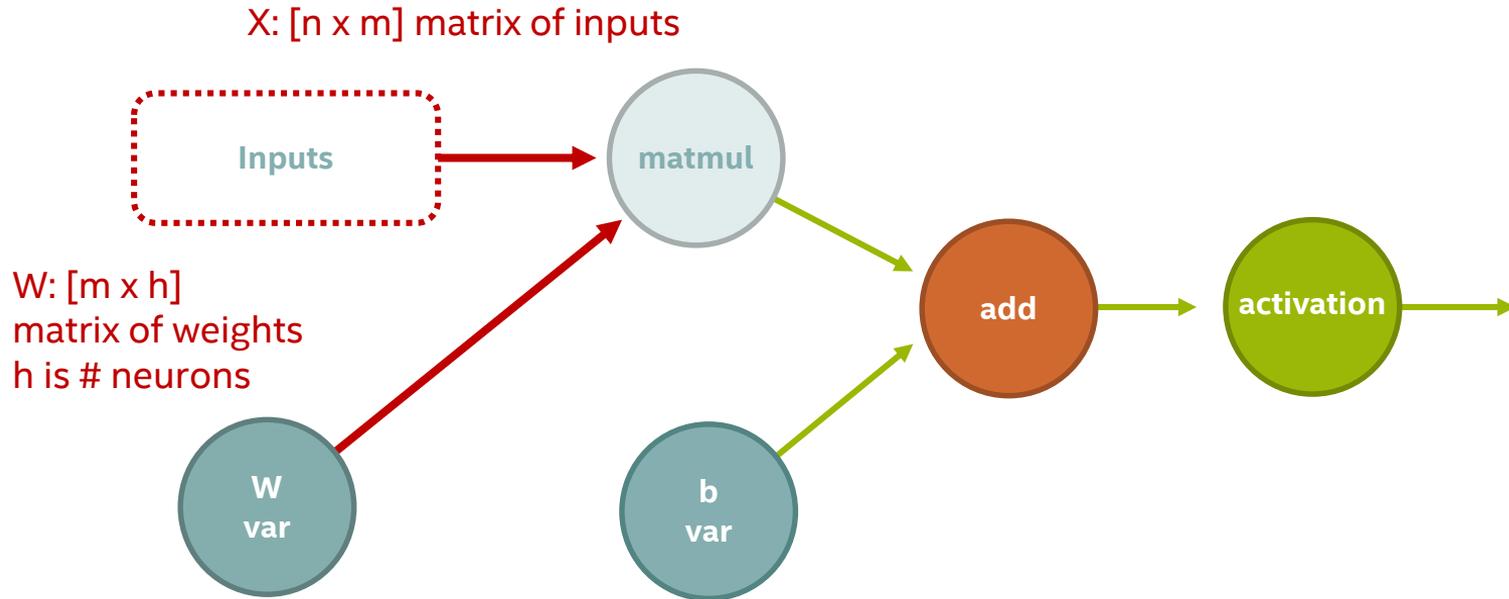
But having different weights means neurons respond to inputs differently

# WHAT DOES A LAYER OF NEURONS LOOK LIKE IN TF?

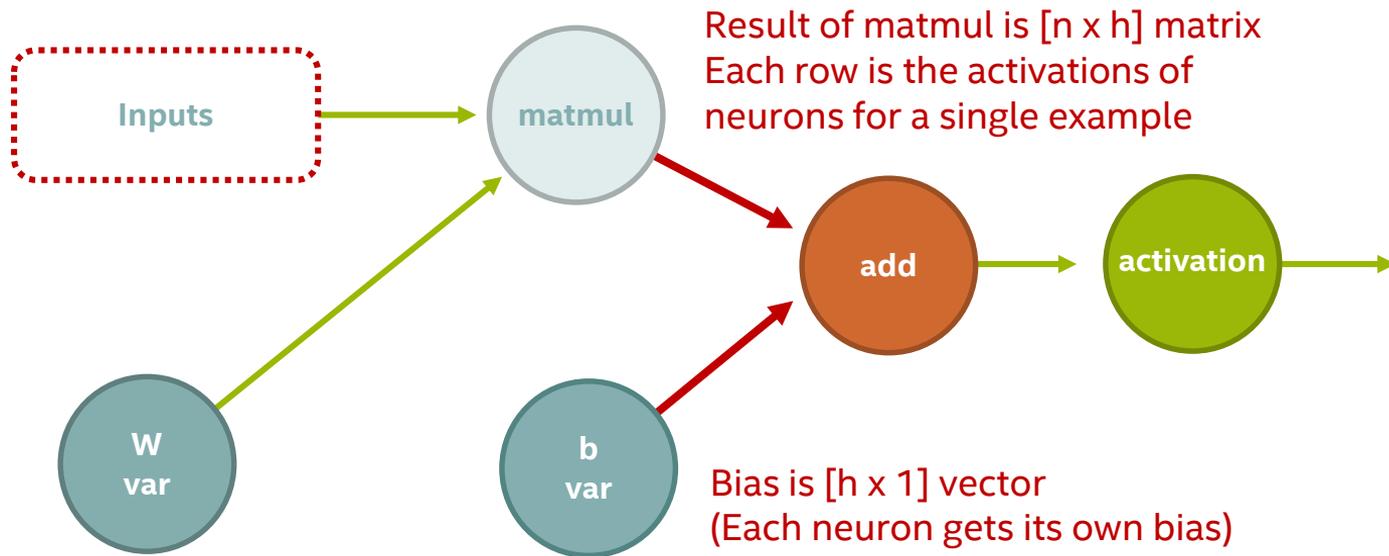
Almost identical to single neuron!



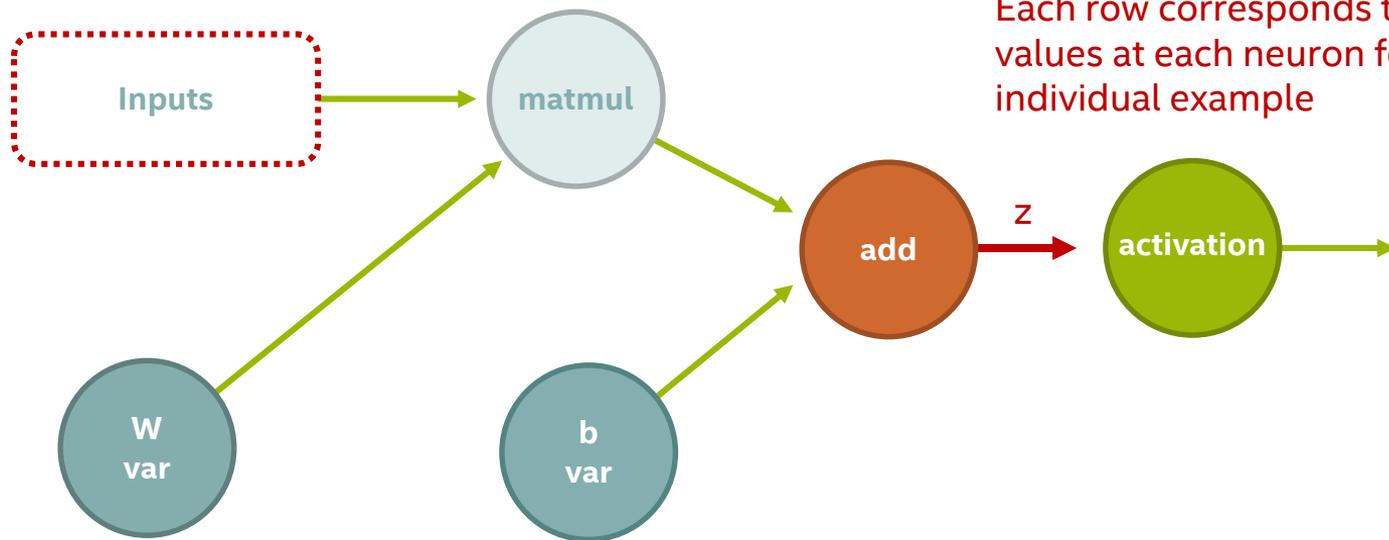
# WEIGHTS: NOW A MATRIX (INSTEAD OF VECTOR)



# BIAS IS NOW A VECTOR (INSTEAD OF A SCALAR)



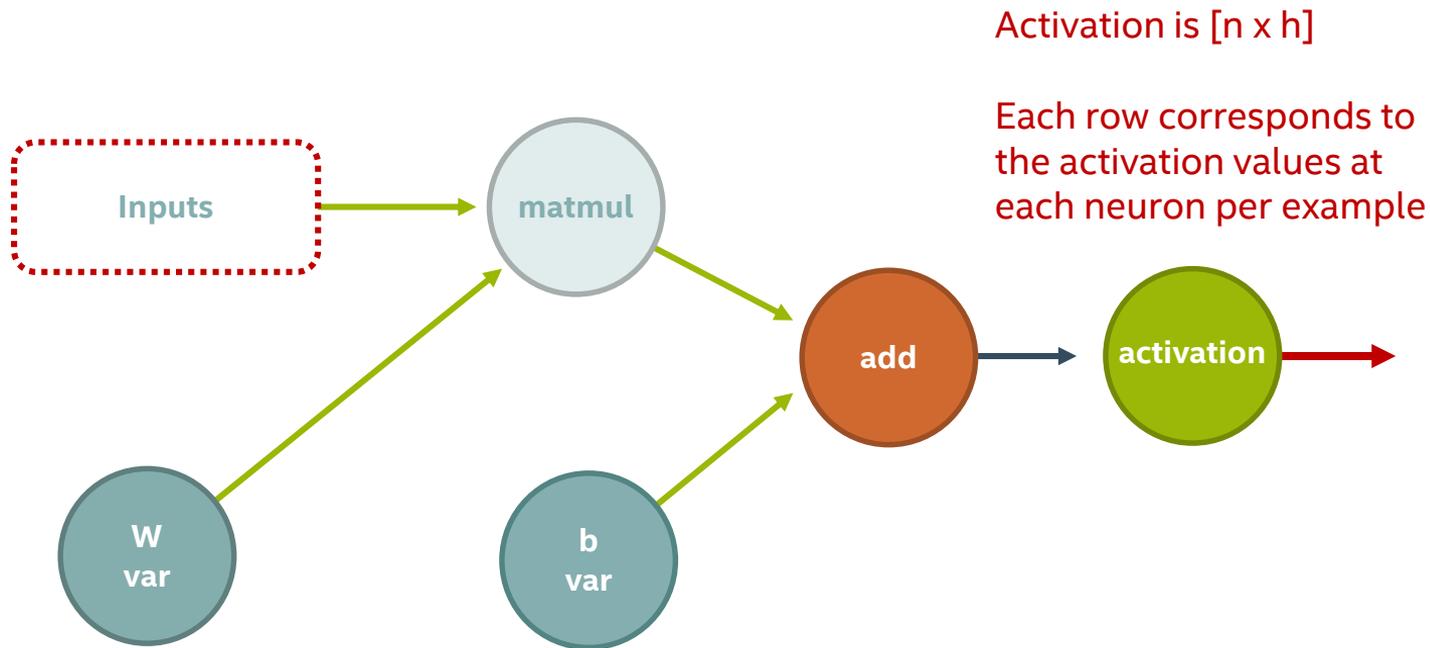
# Z IS A MATRIX



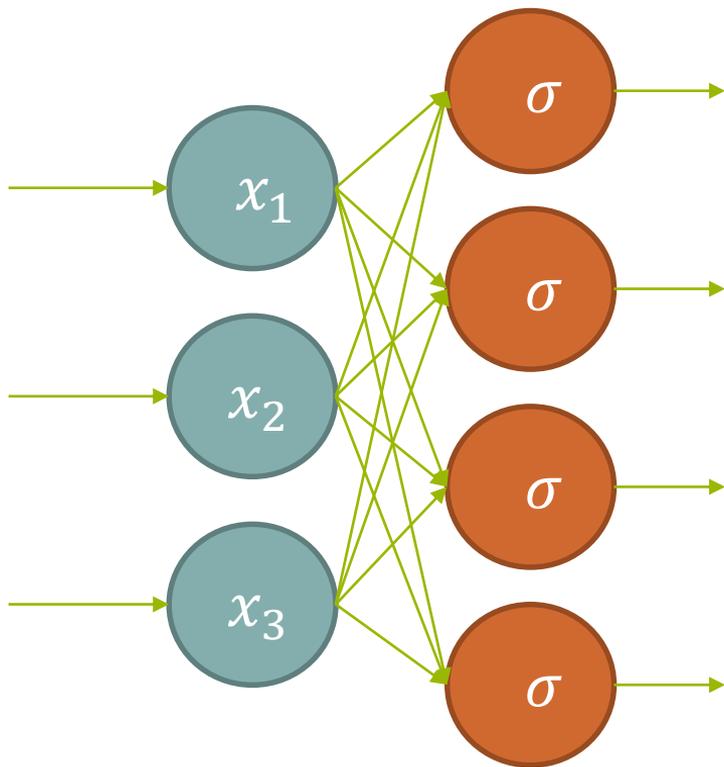
Add outputs an  $[n \times h]$  matrix.

Each row corresponds to the  $z$  values at each neuron for an individual example

# ACTIVATION IS A MATRIX



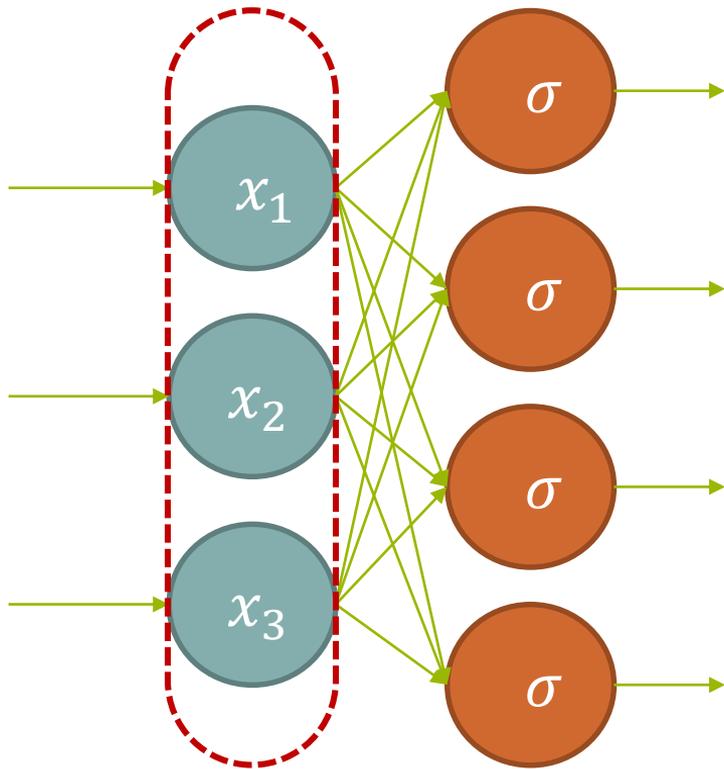
# MATRIX MULTIPLICATION AS LAYER TRANSFORMS



We dictate the size of each layer by defining different sized weights

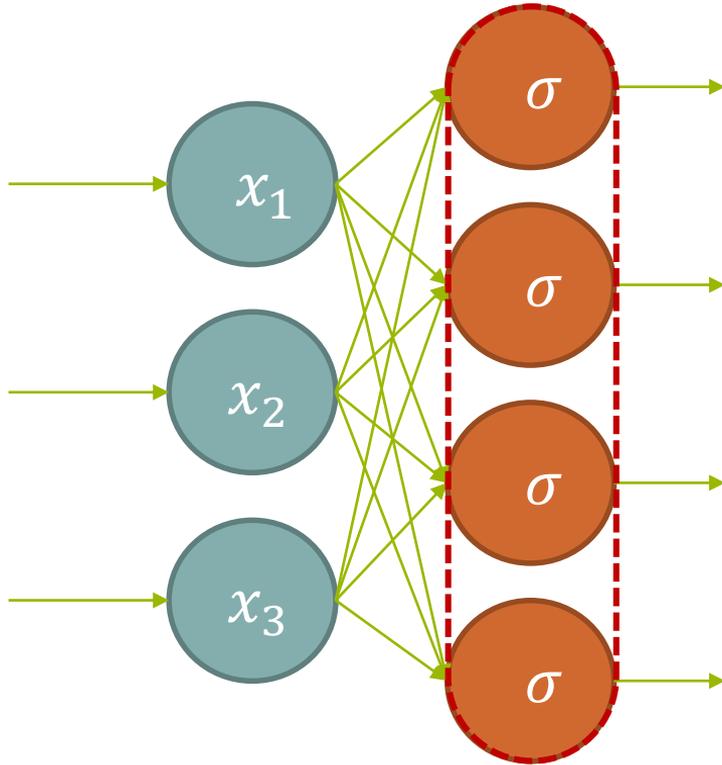
Bias isn't shown here (usually "implied")

*current*<sub>value</sub> =  $X \in \mathbb{R}^{n \times 3}$



X is [n x 3]  
(n data points)

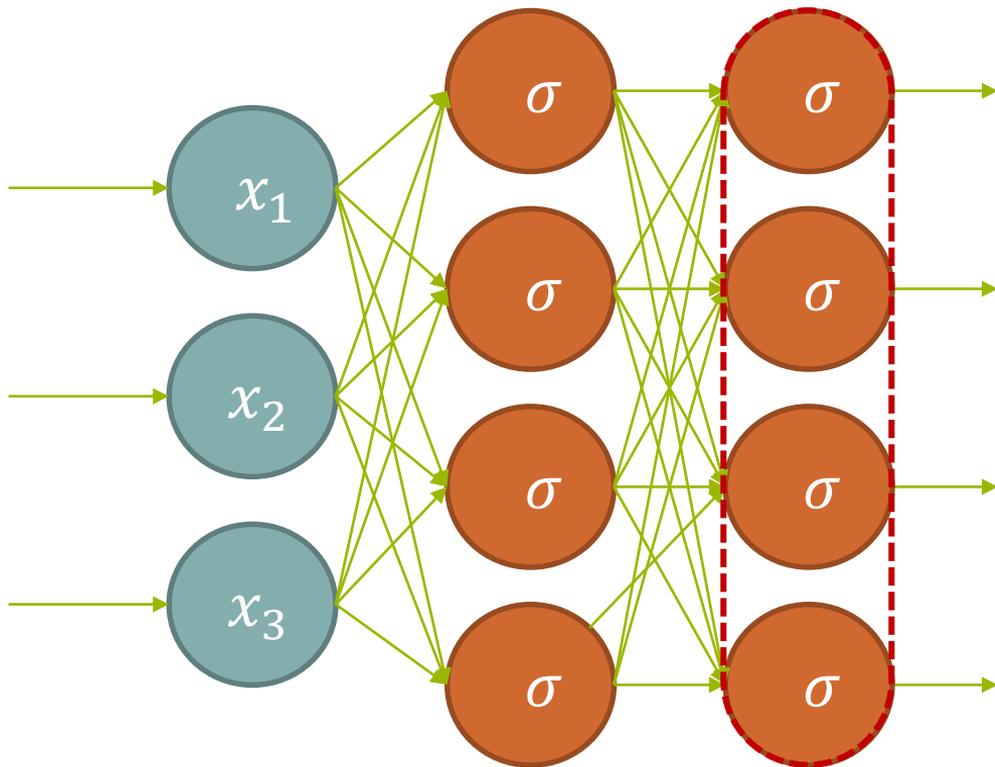
$$\text{current\_value} = XW^{(1)} \in \mathbb{R}^{n \times 4}$$



$X$  is  $[n \times 3]$   
( $n$  data points)

$W^{(1)}$  is  $[3 \times 4]$

$$\text{current\_value} = XW^{(1)}W^{(2)} \in \mathbb{R}^{n \times 4}$$

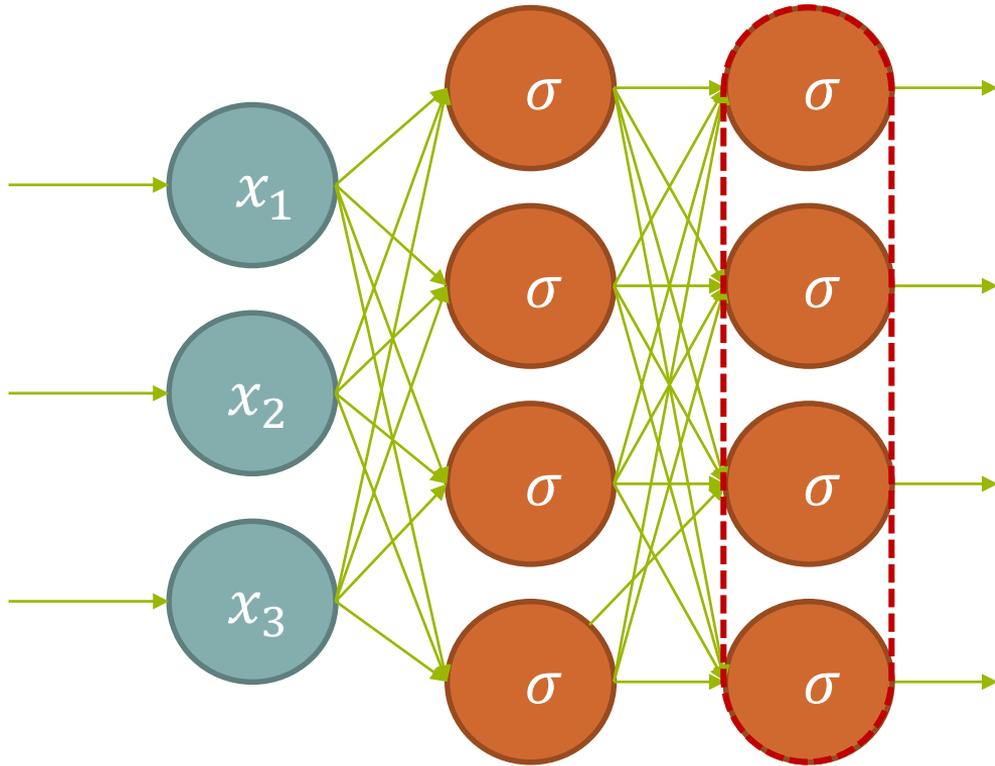


$X$  is  $[n \times 3]$   
( $n$  data points)

$W^{(1)}$  is  $[3 \times 4]$

$W^{(2)}$  is  $[4 \times 4]$

$$\text{current\_value} = XW^{(1)}W^{(2)} \in \mathbb{R}^{n \times 4}$$



$X$  is  $[n \times 3]$   
( $n$  data points)

$W^{(1)}$  is  $[3 \times 4]$

$W^{(2)}$  is  $[4 \times 4]$

Weights dictate  
number of neurons!

## WEIGHT SIZES, IN GENERAL:

$W \Rightarrow [\text{num\_previous\_neurons}, \text{num\_new\_neurons}]$

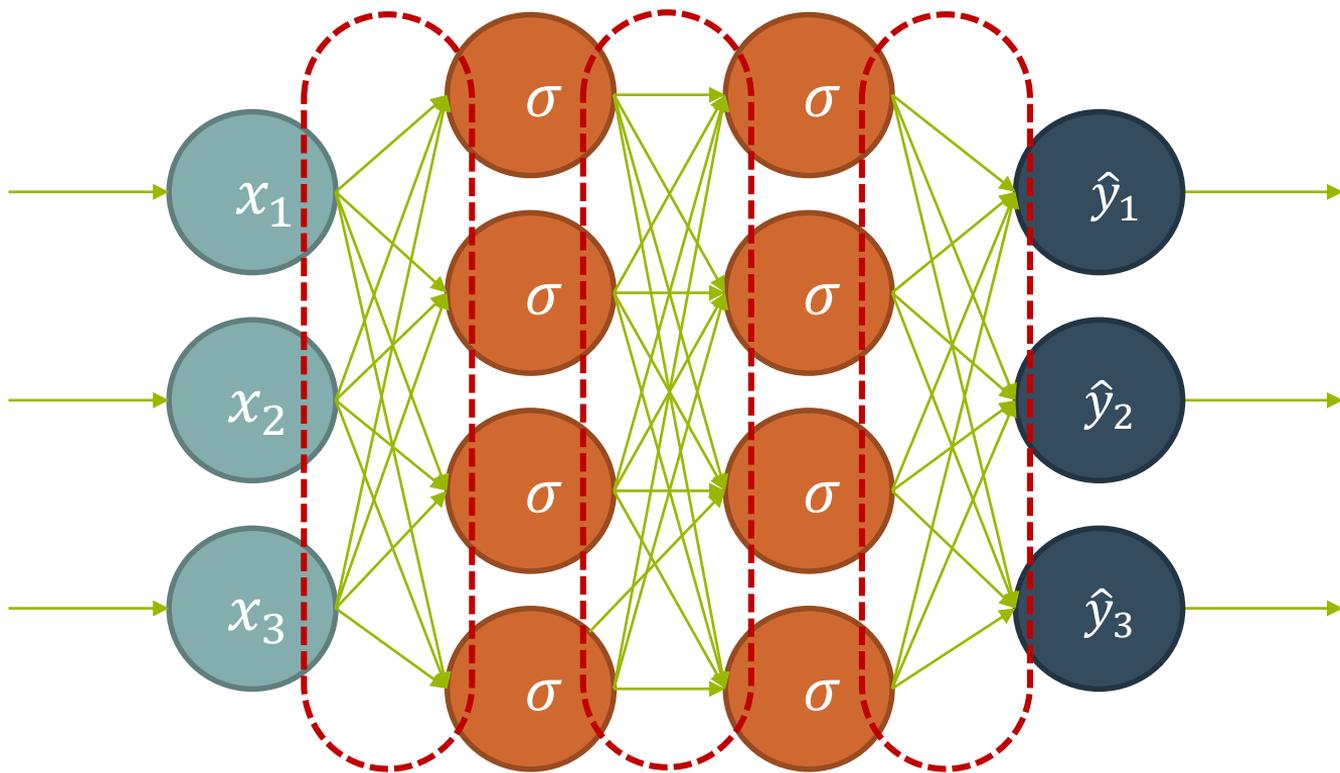
$\Rightarrow [\text{previous\_size}, \text{current\_size}]$

$b \Rightarrow [\text{num\_new\_neurons}]$

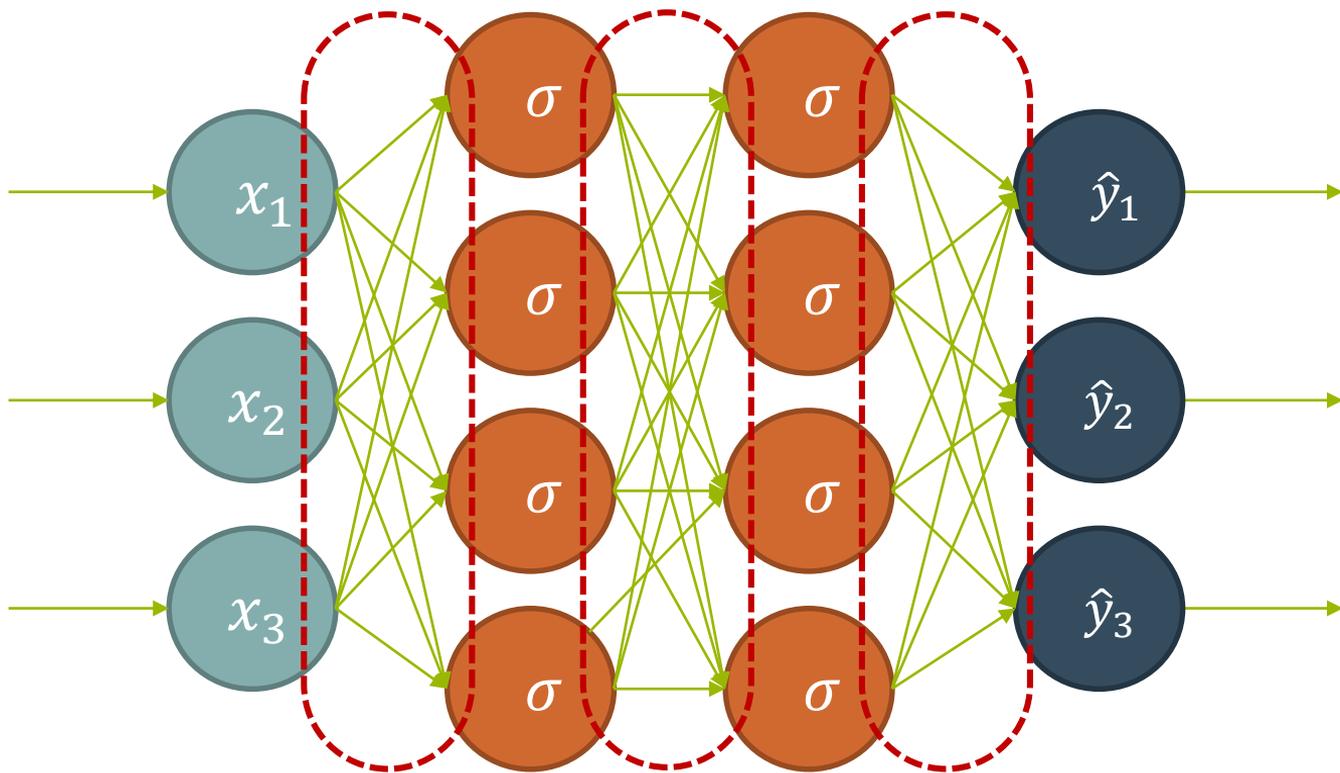
$\Rightarrow [\text{current\_size}]$

# A FEEDFORWARD NETWORK

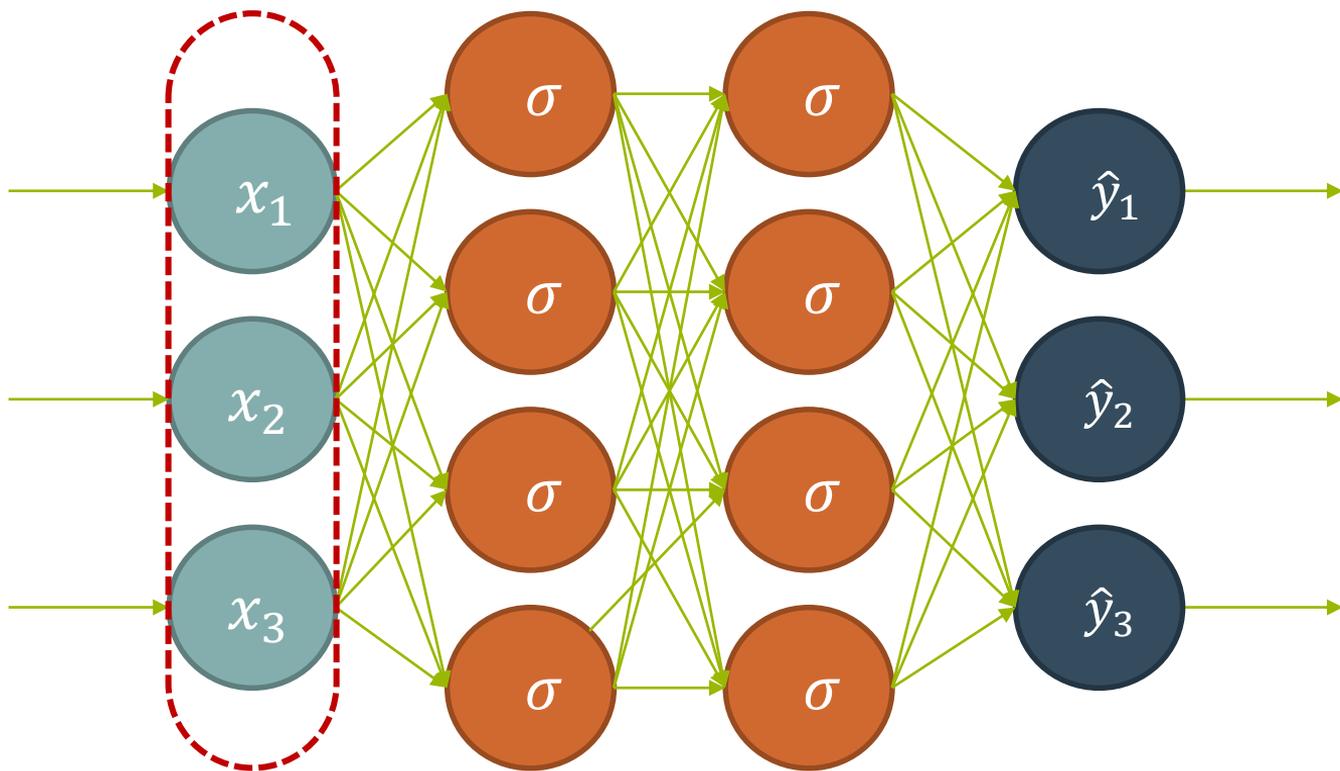
# FEEDFORWARD NEURAL NETWORK



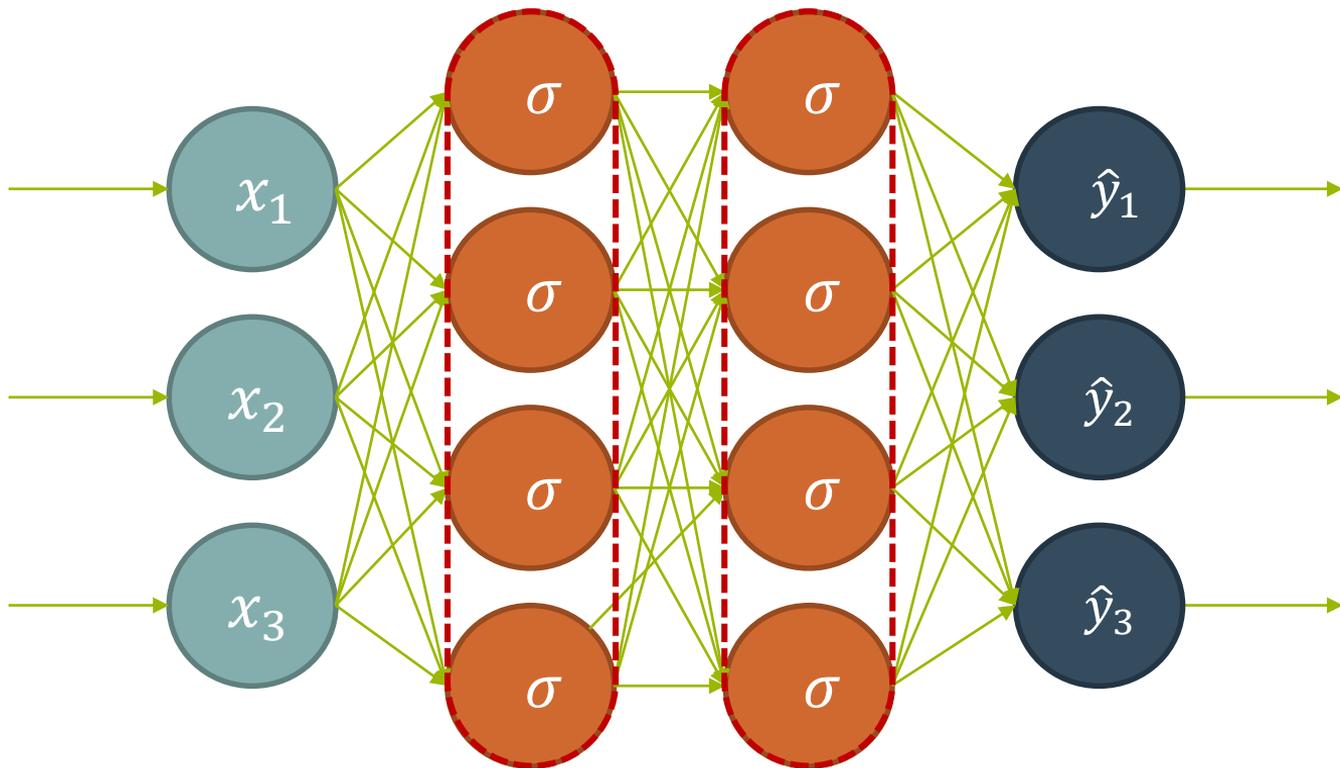
# WEIGHTS



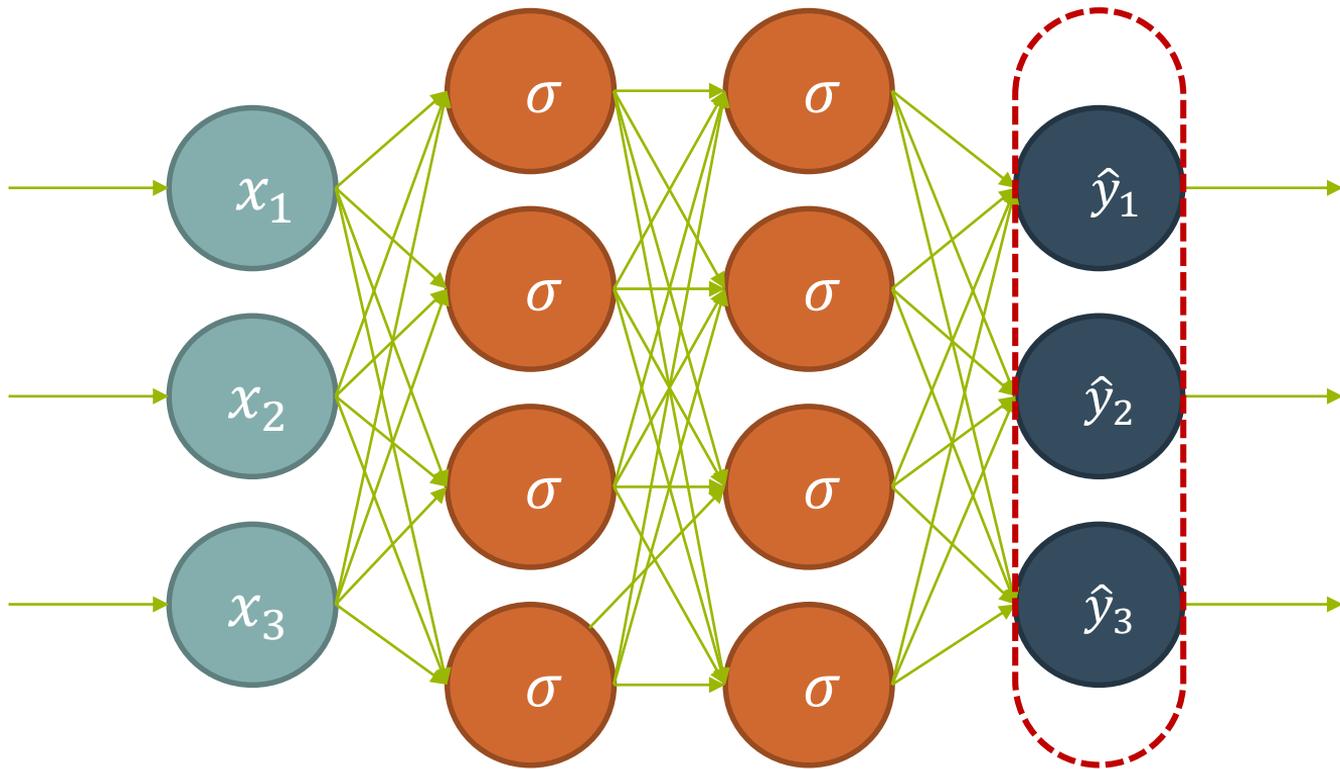
# INPUT LAYER



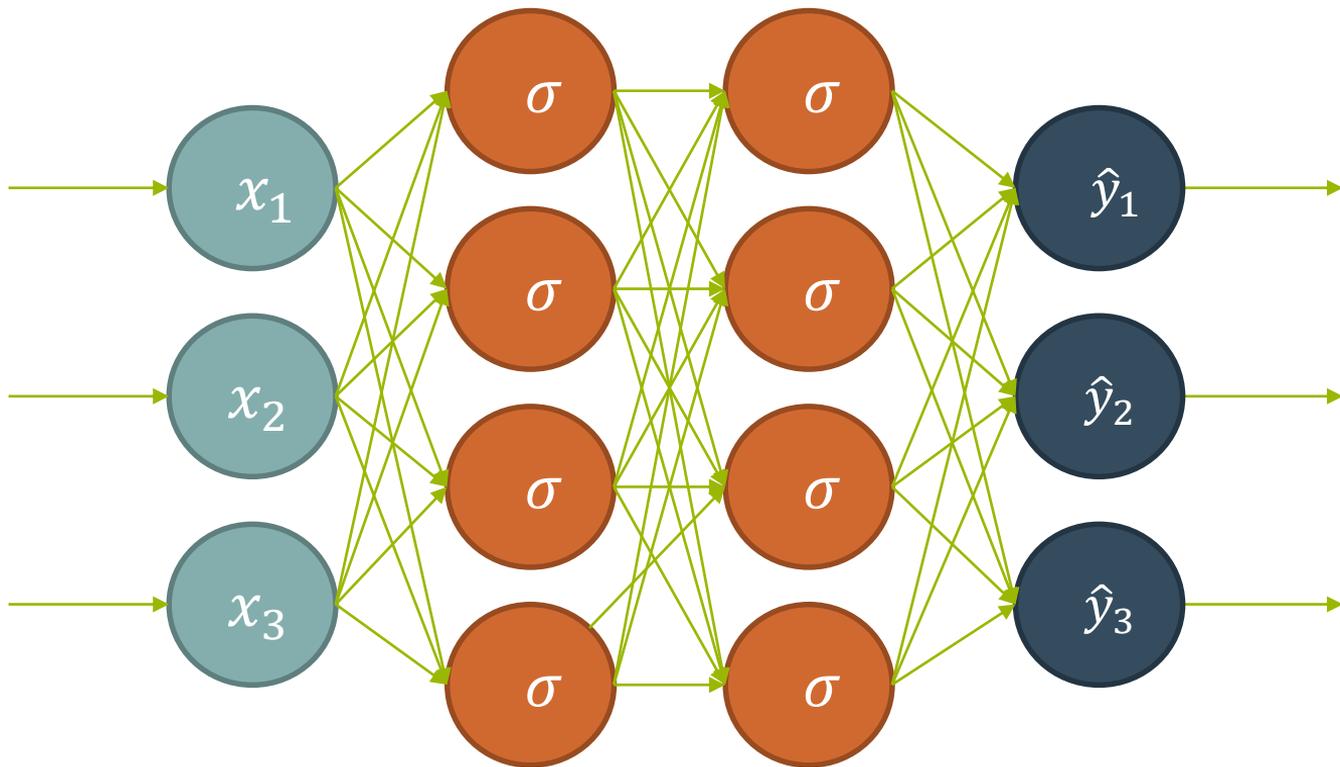
# HIDDEN LAYERS



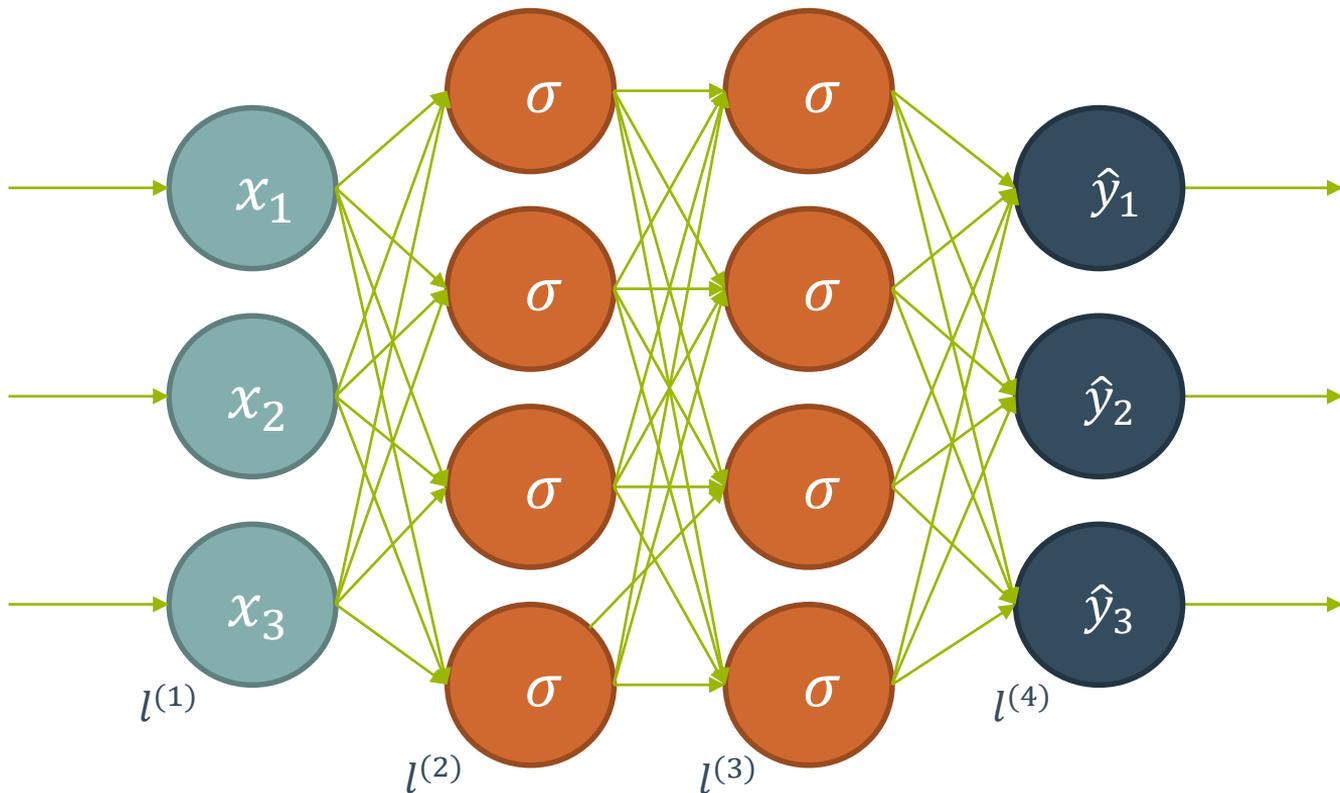
# OUTPUT LAYER



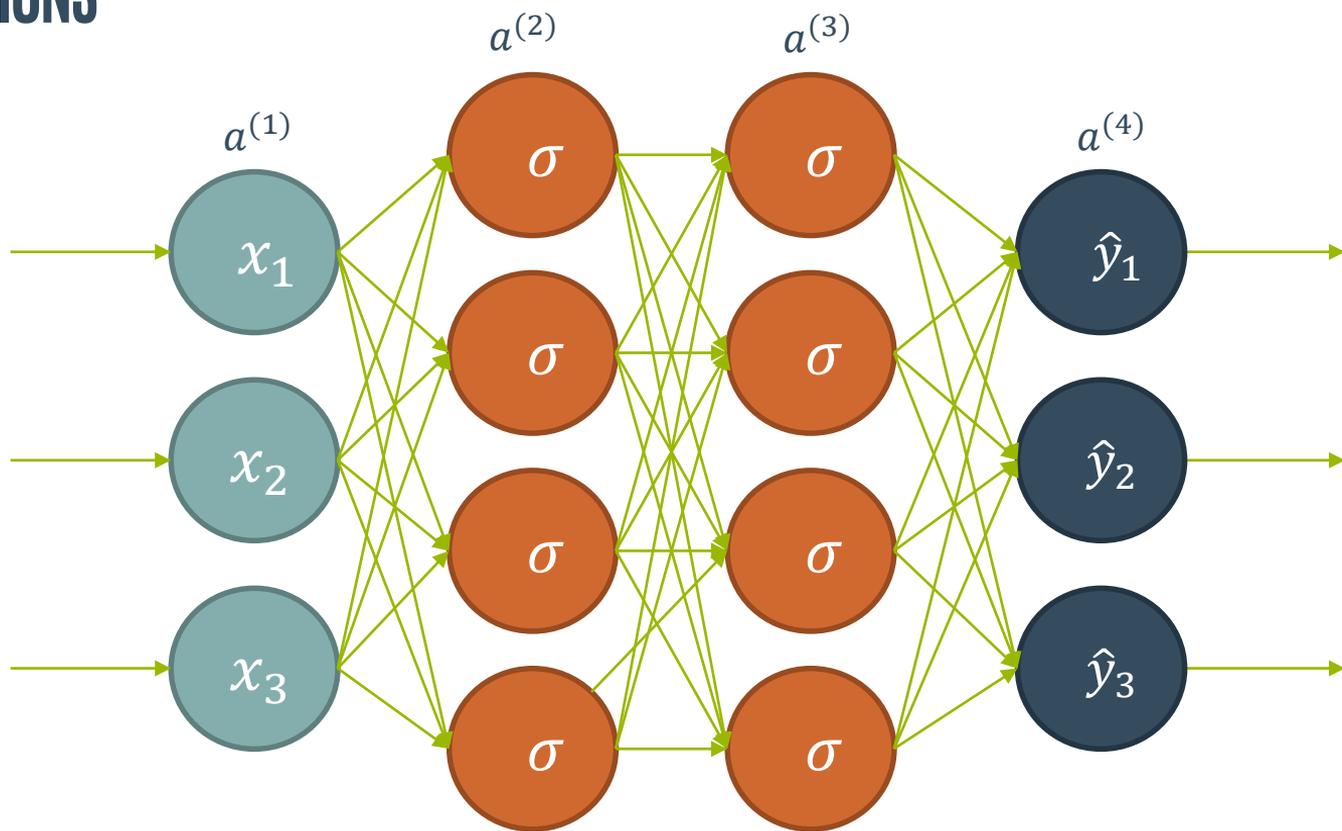
# ANNOTATIONS



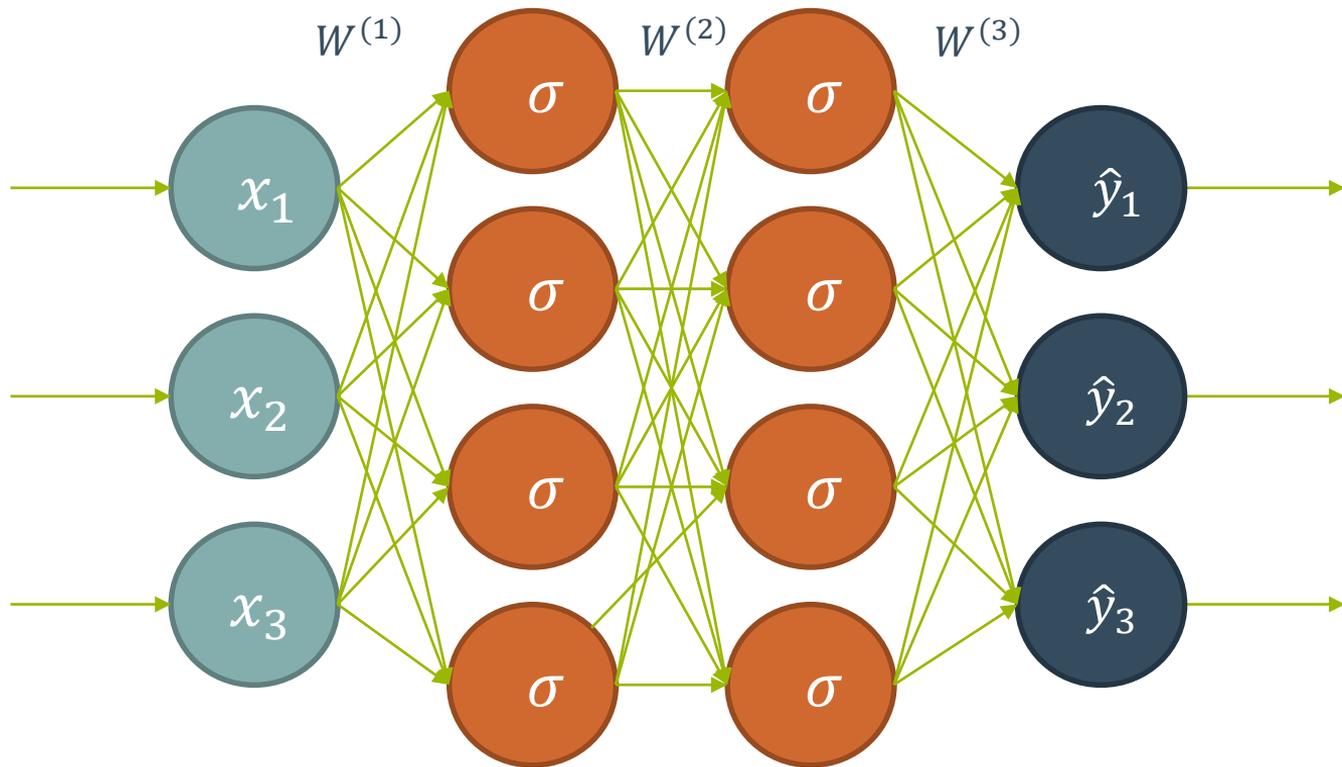
# LAYER NUMBERS



# ACTIVATIONS



# WEIGHTS



# NET INPUTS

