



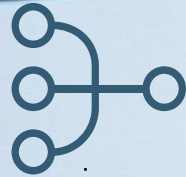
# INTRODUCTION TO NEURAL NETWORKS

A BRANCH OF DEEP LEARNING

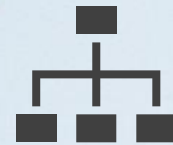
# AGENDA



WHAT IS  
NEURAL  
NETWORK



COMPONENTS  
OF A NEURAL  
NETWORK



TRAINING A  
NEURAL  
NETWORK



Q&A

# WHAT IS NEURAL NETWORK

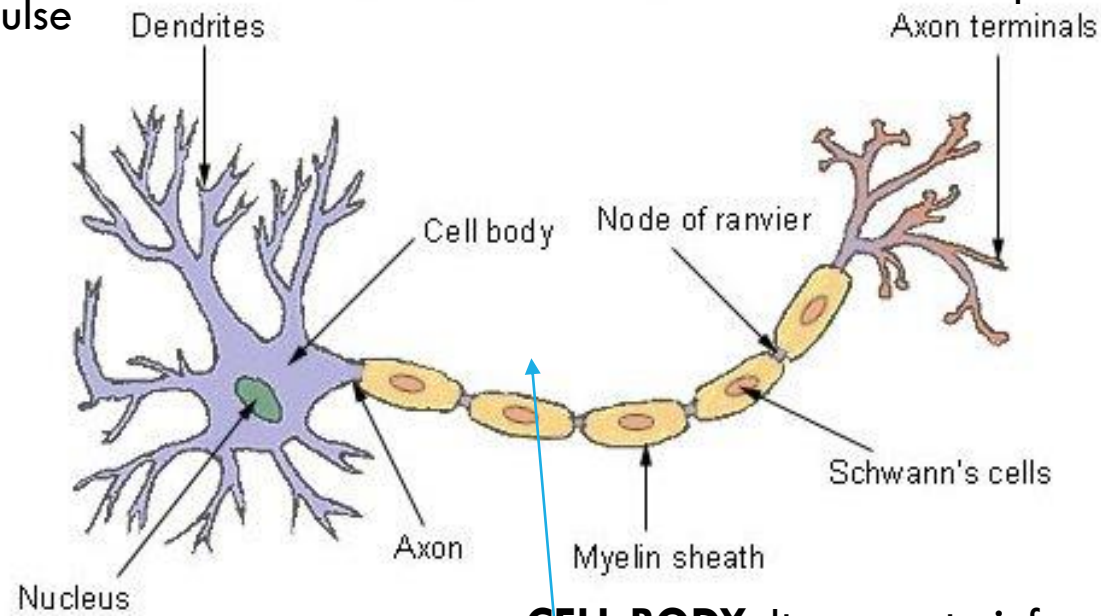


# INTUITION BEHIND NEURAL NETWORK

Neural Networks (NN), also called as **Artificial Neural Network** is named after its artificial representation of working of a human being's *nervous system*.

**DENDRITES:** It takes the inputs from other neurons in form of electrical impulse

**AXON TERMINALS:** It transmits output in form of electrical impulse



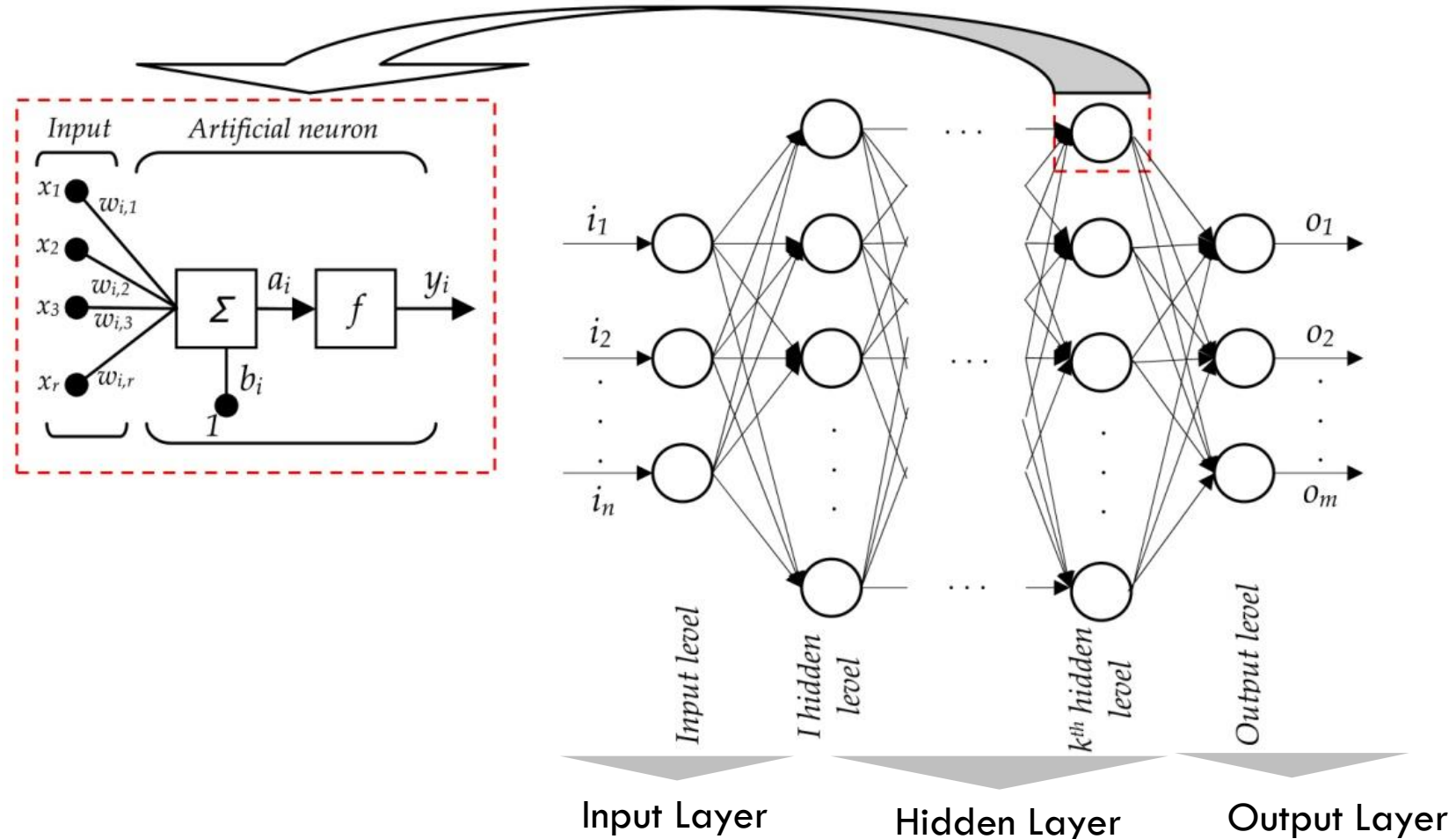
**CELL BODY:** It generate inferences from those inputs and decide what action to take

# NEURAL NETWORK STRUCTURE

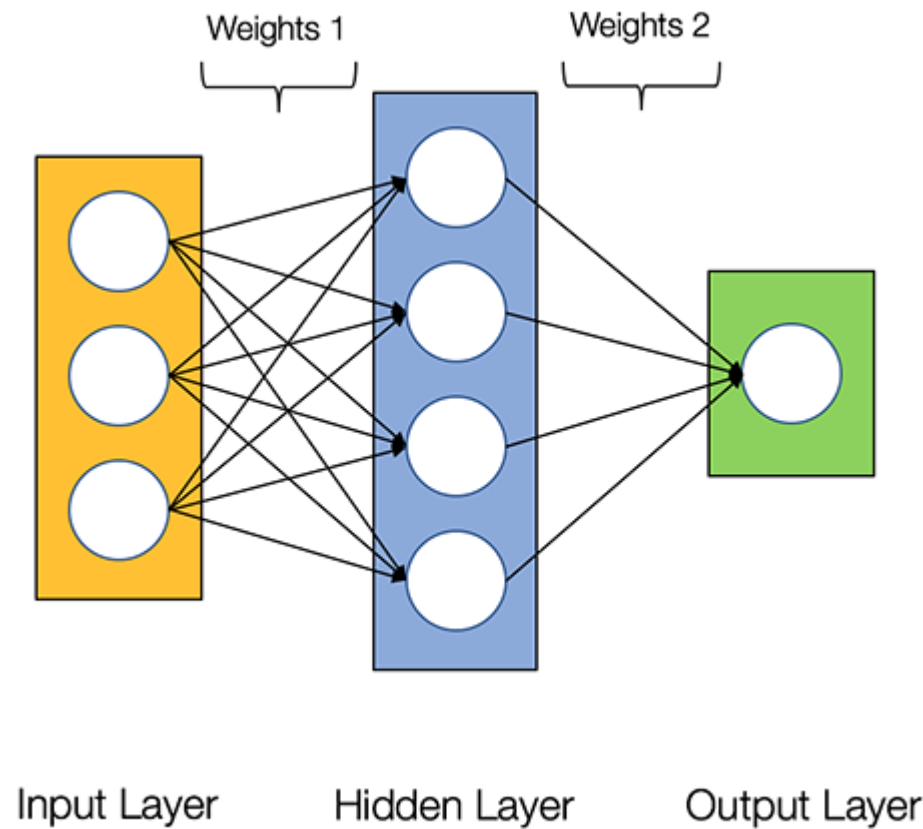
**Input Layer:** The training observations are fed through these neurons

**Hidden Layers:** These are the intermediate layers between input and output which help the Neural Network learn the complicated relationships involved in data.

**Output Layer:** The final output is extracted from previous two layers



# AN EXAMPLE OF 2 LAYER ANN





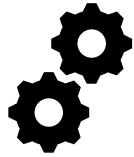
# COMPONENTS OF A NEURAL NETWORK



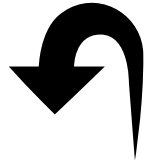
# COMPONENTS OF A NEURAL NETWORK



**NEURON CONNECTIONS**



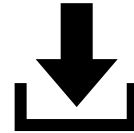
**CONNECTIONS**



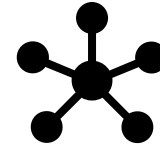
**BIAS**



**ACTIVATION  
FUNCTION**



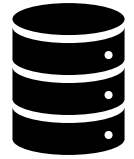
**INPUT  
LAYER**



**HIDDEN  
LAYERS**



**OUTPUT  
LAYER**



**WEIGHT**



# COMPONENTS OF A NEURAL NETWORK : NEURON



NEURON

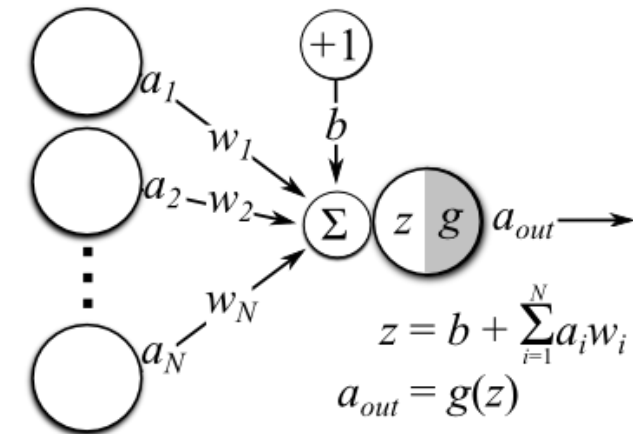
It is the **basic unit of a neural network**. It gets certain number of inputs and a bias value. When a signal(value) arrives, it gets multiplied by a weight value.

$W_i = \text{weights, } b = \text{bias}$

$$z = x_1 * w_1 + x_2 * w_2 + \dots + x_n * w_n + b * 1$$

$$\hat{y} = a_{out} = \text{sigmoid}(z)$$

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



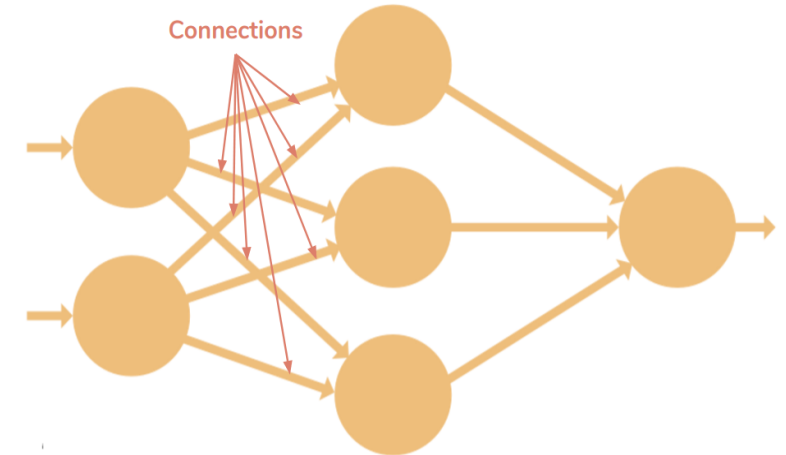
# COMPONENTS OF A NEURAL NETWORK : CONNECTIONS



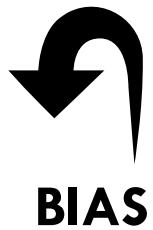
## CONNECTIONS

**It connects one neuron in one layer to another neuron in other layer or the same layer.**

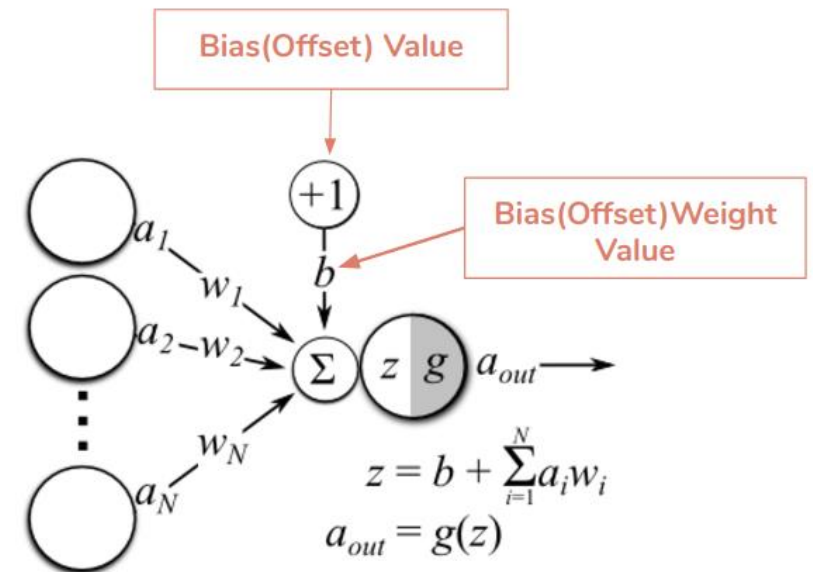
A connection always has a weight value associated with it. Goal of the training is to update this weight value to decrease the loss(error).



# COMPONENTS OF A NEURAL NETWORK : BIAS



**Bias(Offset)**—It is an extra input to neurons and it is always 1, and has it's own connection weight. This makes sure that even when all the inputs are none (all 0's) there will be an activation in the neuron.



# COMPONENTS OF A NEURAL NETWORK : ACTIVATION FUNCTION



## ACTIVATION FUNCTION








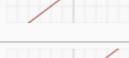

**Activation Function (Transfer Function)**—Activation functions are used to introduce non-linearity to neural networks.

It squashes the values in a smaller range viz. a Sigmoid activation function squashes values between a range 0 to 1.

There are many activation functions used in deep learning industry and **ReLU, SeLU and TanH** are preferred over sigmoid activation function

$$a = f\left(\sum_{i=0}^N w_i x_i\right)$$

## Different kind of Activation Functions

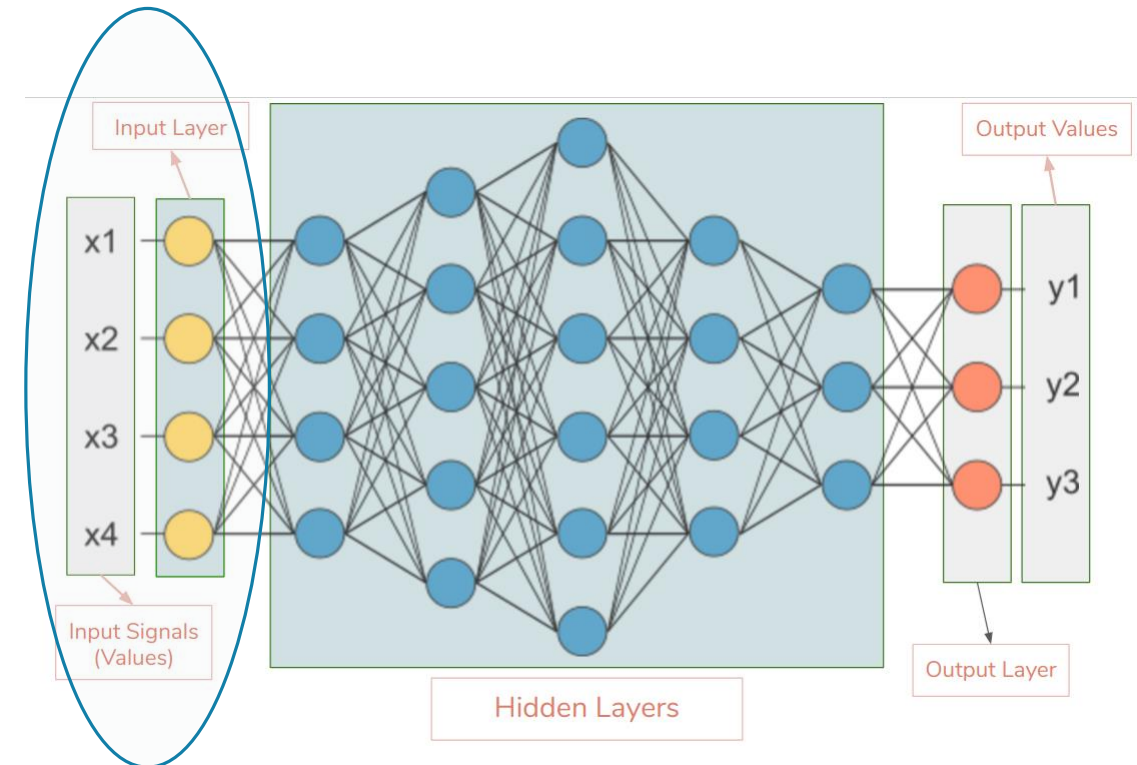
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

# COMPONENTS OF A NEURAL NETWORK : INPUT LAYER

↓  
**INPUT  
LAYER**

This is the first layer in the neural network. It takes **input signals (values)** and passes them on to the next layer.

It doesn't apply any operations on the input signals(values) & has no weights and biases values associated.



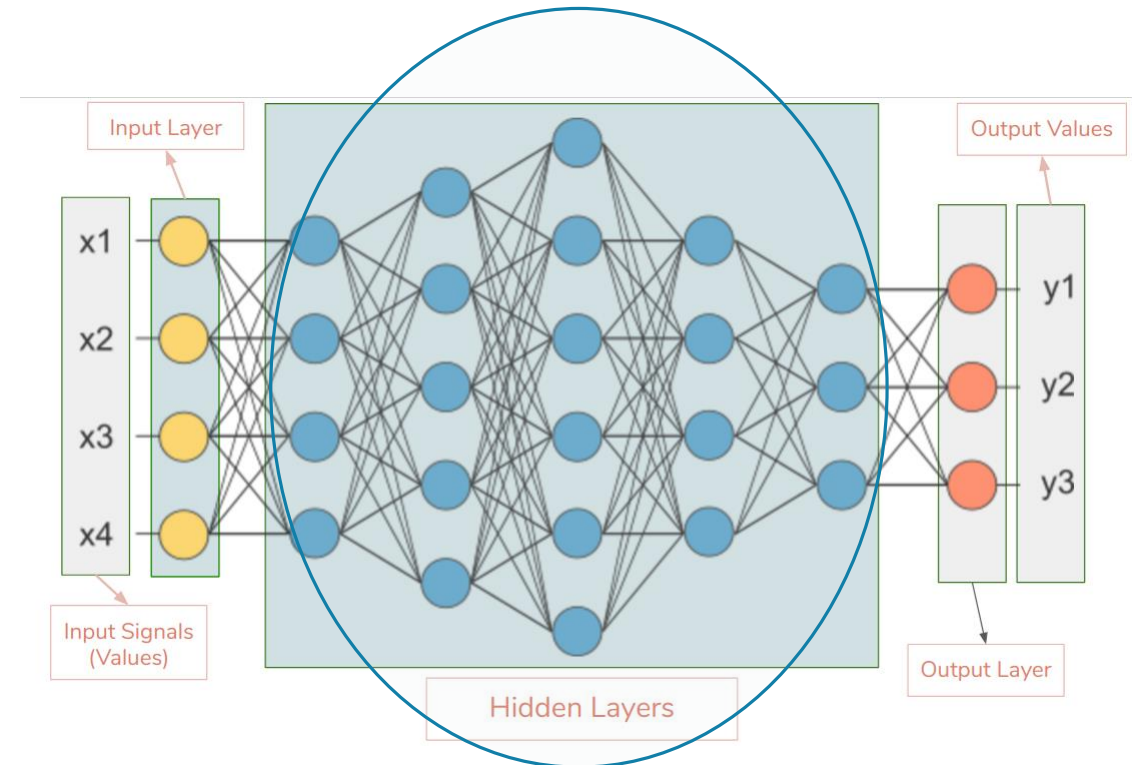
**INPUT  
LAYER**

# COMPONENTS OF A NEURAL NETWORK : HIDDEN LAYER

## ↓ HIDDEN LAYER

**Hidden layers have neurons(nodes)** which apply different transformations to the input data.

One hidden layer is a collection of neurons stacked vertically(Representation).



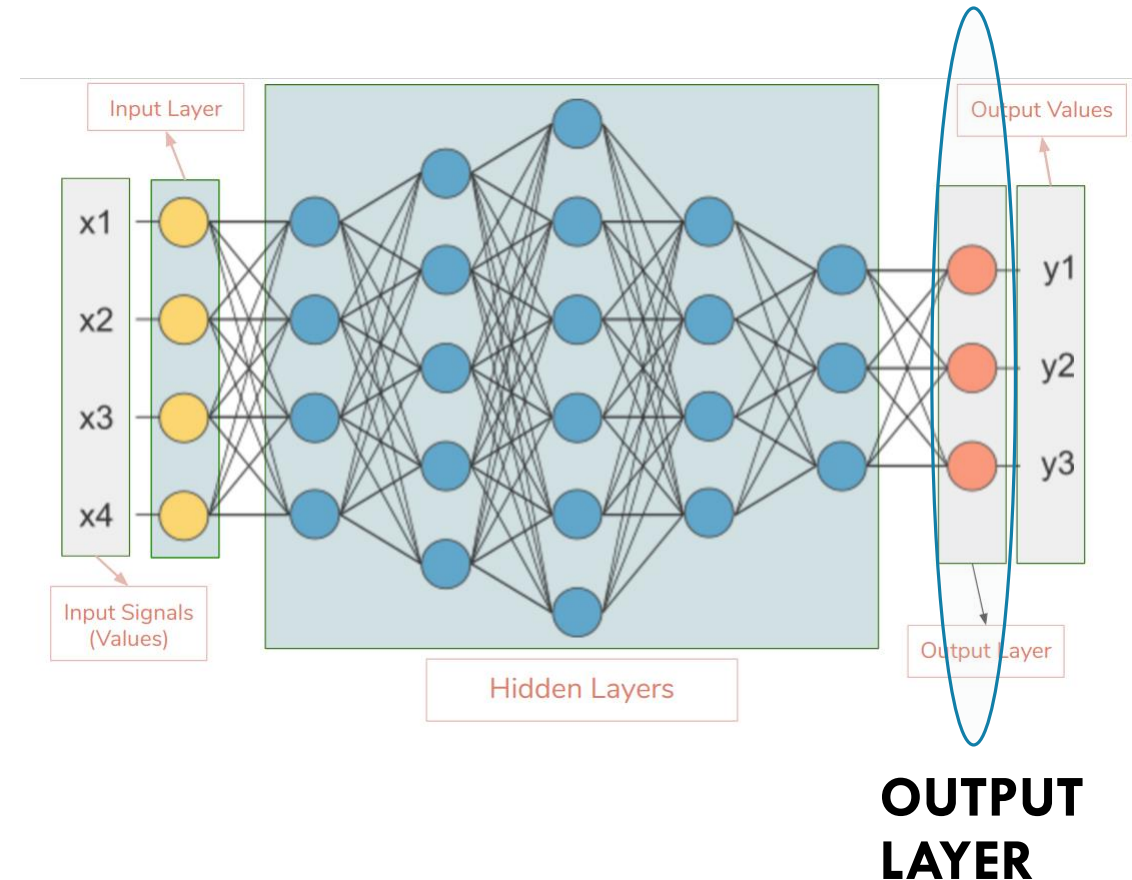
## HIDDEN LAYERS

# COMPONENTS OF A NEURAL NETWORK : OUTPUT LAYER

## ✓ OUTPUT LAYER

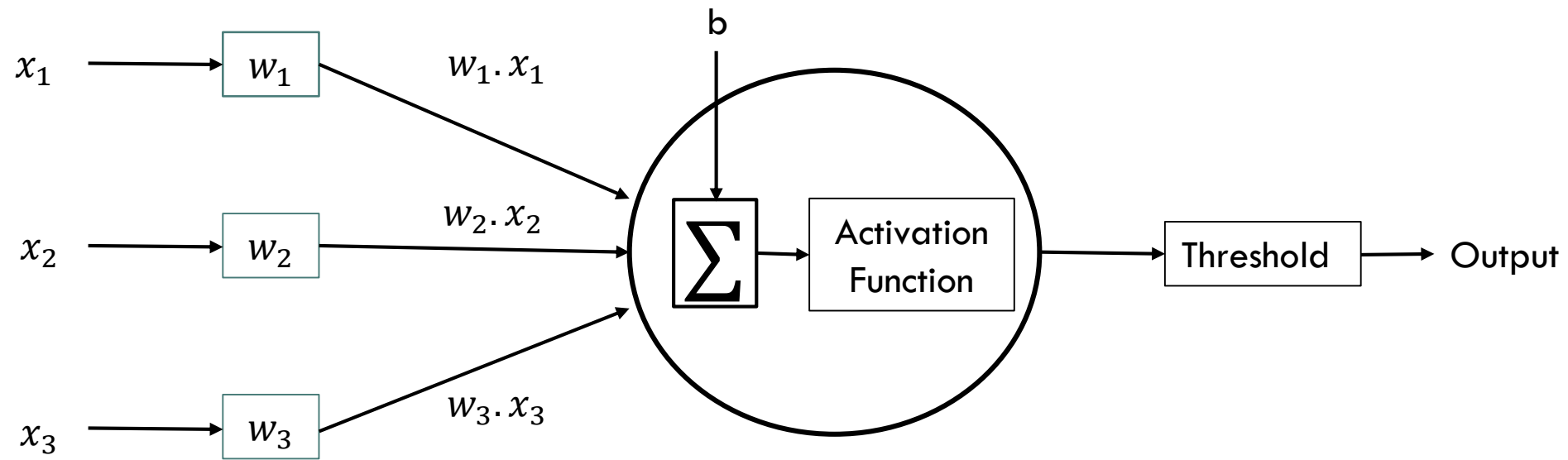
**This layer is the last layer in the network & receives input from the last hidden layer.**

With this layer we can get desired number of values and in a desired range.





# KNITTING IT ALL TOGETHER



A COMPLETE NEURAL NETWORK

# TRAINING A NEURAL NETWORK



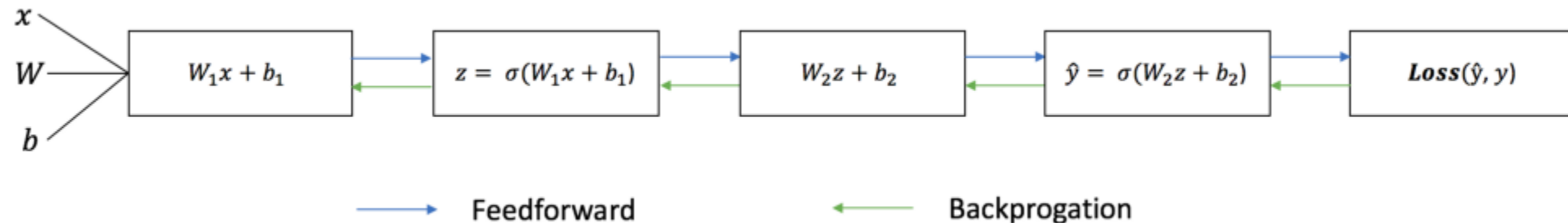
# TRAINING A NEURAL NETWORK

Naturally, the right values for the weights and biases determines the strength of the predictions. The process of fine-tuning the weights and biases from the input data is known as **training the Neural Network**.

Each iteration of the training process consists of the following steps:

Calculating the predicted output  $\hat{y}$ , known as **feedforward**

Updating the weights and biases, known as **backpropagation**



# TRAINING A NEURAL NETWORK : COST FUNCTION

The loss function computes the error for a single training example. **The cost function is the average of the loss functions of the entire training set.**

**Different kind of cost function are:**

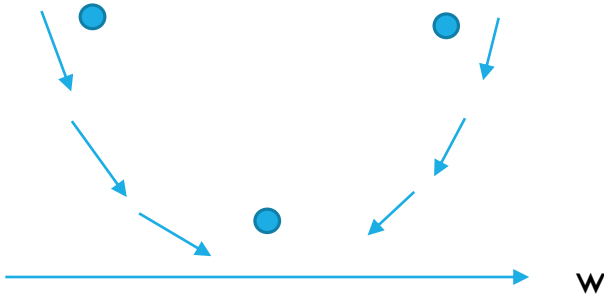
*'mse'*: for mean squared error.

*'binary\_crossentropy'*: for binary logarithmic loss (logloss).

*'categorical\_crossentropy'*: for multi-class logarithmic loss (logloss).

# TRAINING A NEURAL NETWORK : GRADIENT DESCENT

C

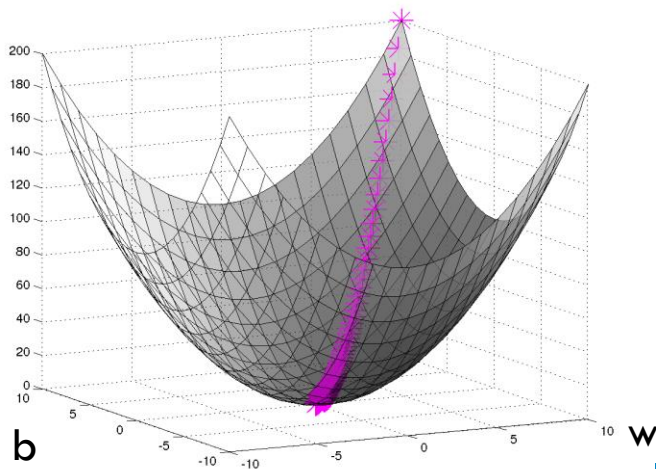


Updating  $w, b$  such that we end up at the minimum point of Cost function. The partial derivatives are nothing but slopes at that point with respect to that parameter.  $\alpha$  here is the **learning rate**.

$$w := w - \alpha \frac{\partial C(w)}{\partial w}$$

$$b := b - \alpha \frac{\partial C(b)}{\partial b}$$

C



When the slope is positive  $w/b$  will decrease and when the slope is negative  $w/b$  will increase which will end up at the parameter stabilizing at the point where  $C$  is minimum.

**Please note here we are reaping the benefits of a carefully chosen loss function here. Had the loss function been non convex we would not have had a global minima**

# PUTTING IT ALL TOGETHER

We can iteratively update  $w$  and  $b$  to minimize the cost function using gradient descent which will result in our neural network updating weights with every iteration. For a large number of iterations if the cost function is sufficiently reduced then the weights will be such that the neural network can accurately predict the labels

$$w := w - \alpha dw$$

$$b := b - \alpha db$$

# Q&A

