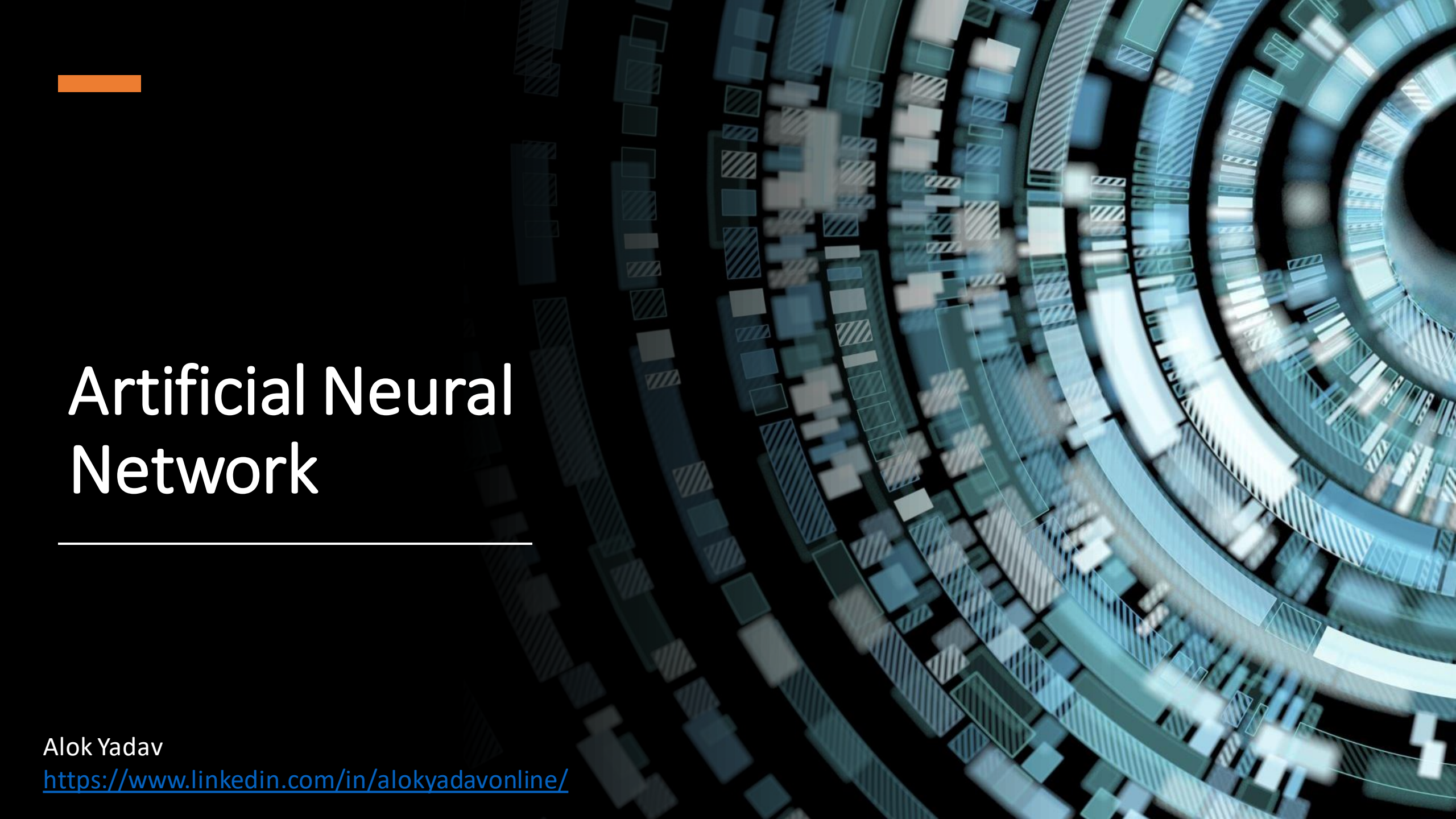
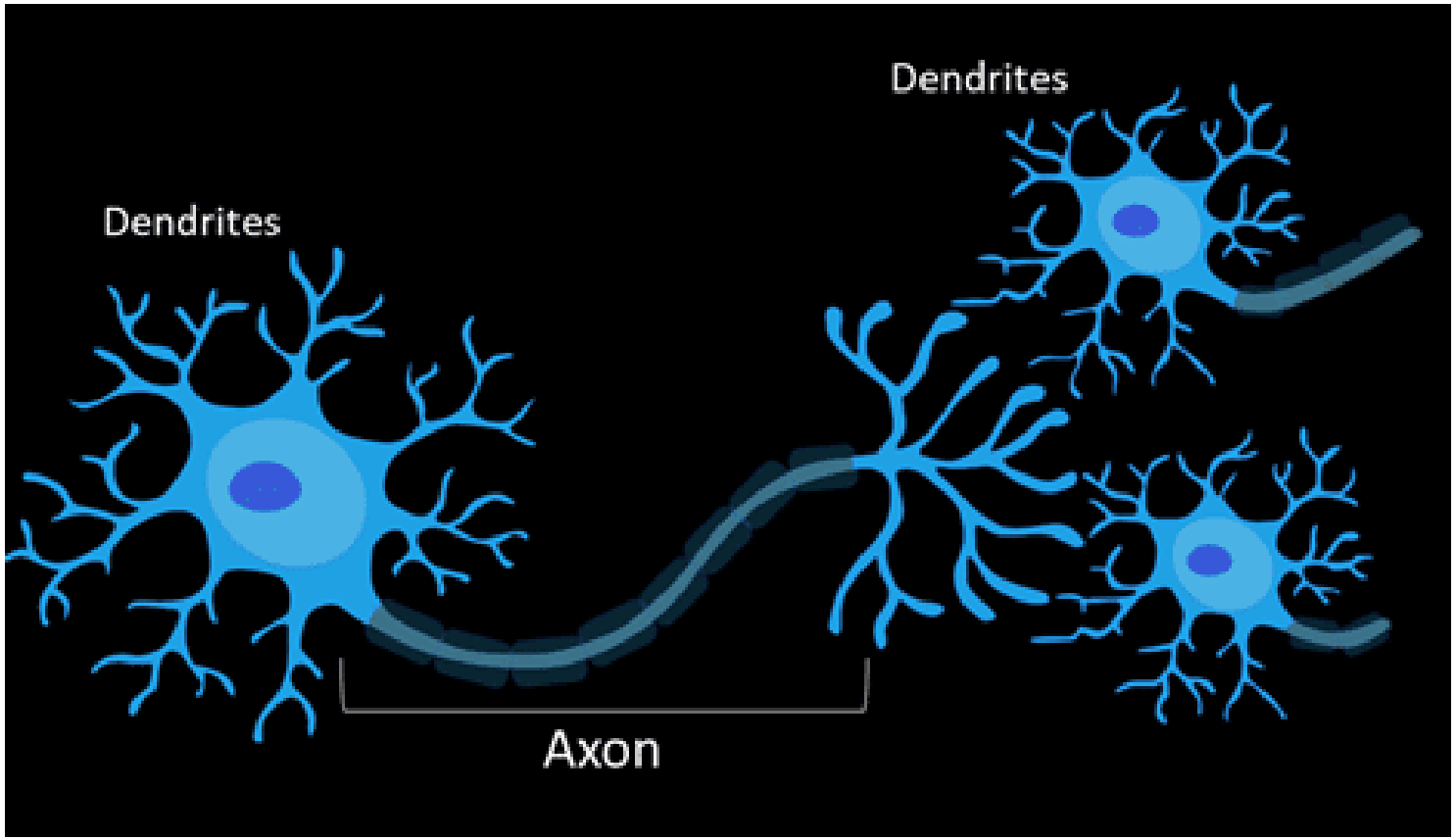


# Artificial Neural Network

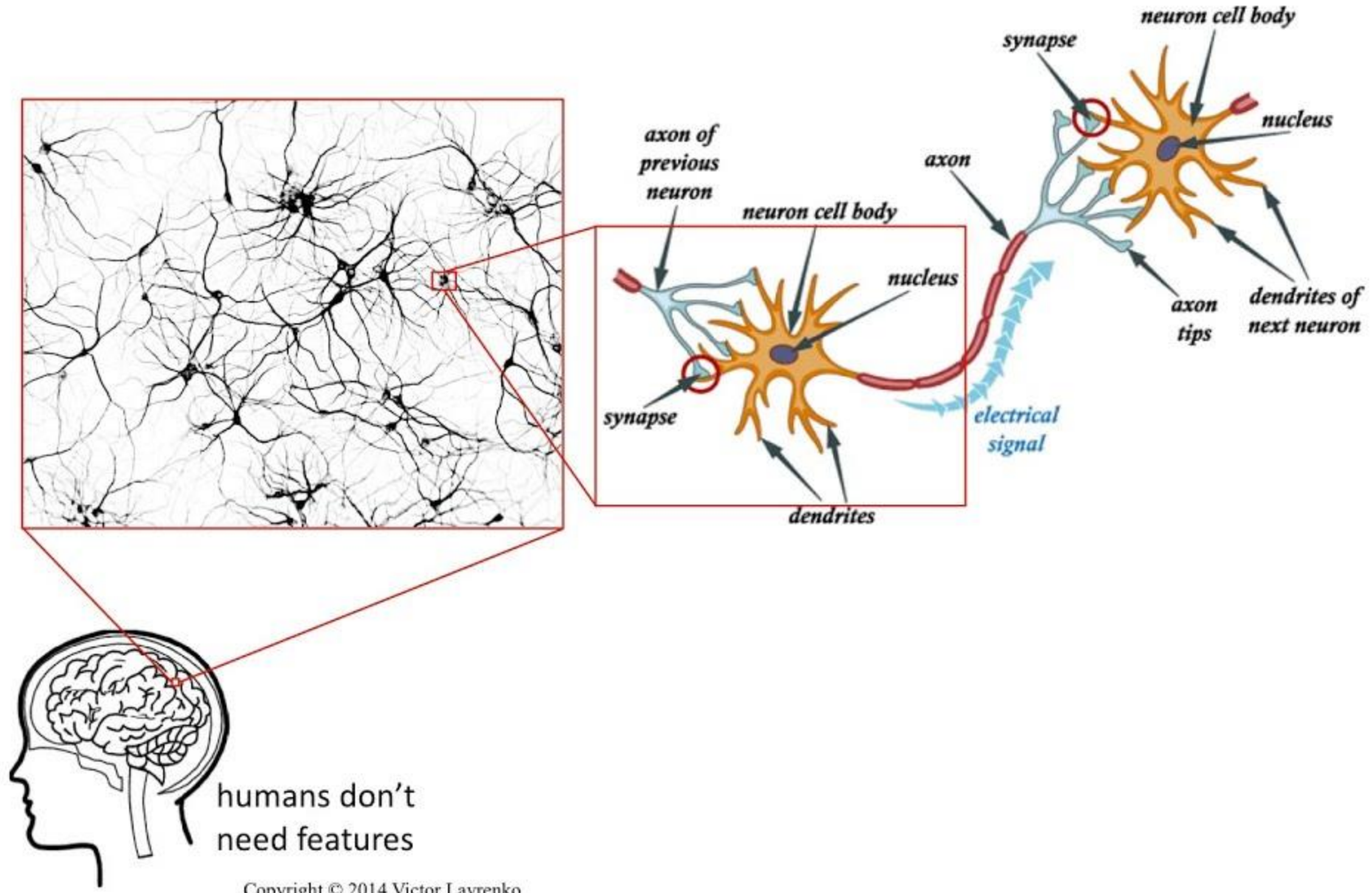
Alok Yadav
https://www.linkedin.com/in/alokyadavonline/




# Biological Motivation

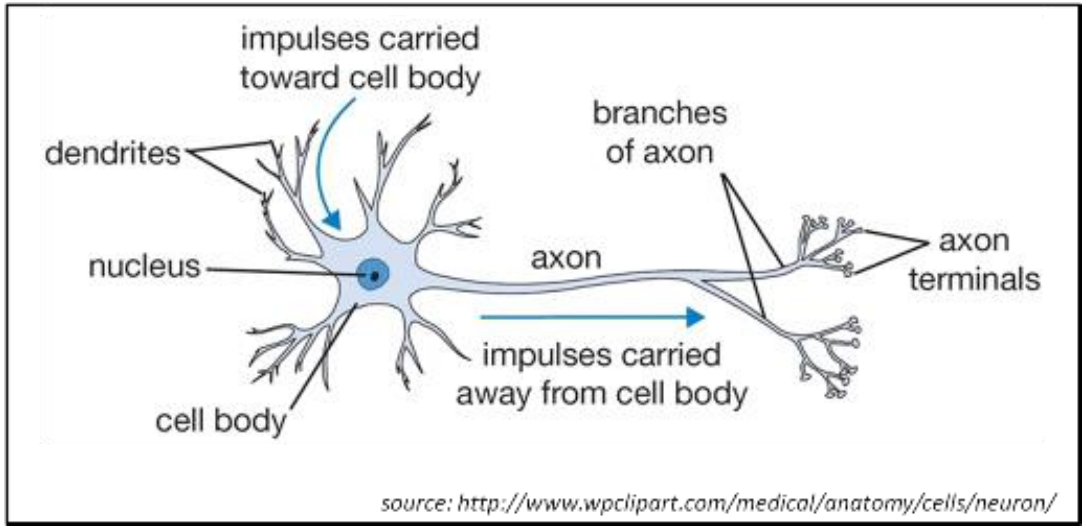
- Motivation behind neural network is human brain. Human brain is called as the best processor even though it works slower than other computers.
- Many researchers thought to make a machine that would work in the prospective of the human brain.
- Human brain contains billion of neurons which are connected to many other neurons to form a network so that if it sees any image, it recognizes the image and processes the output.

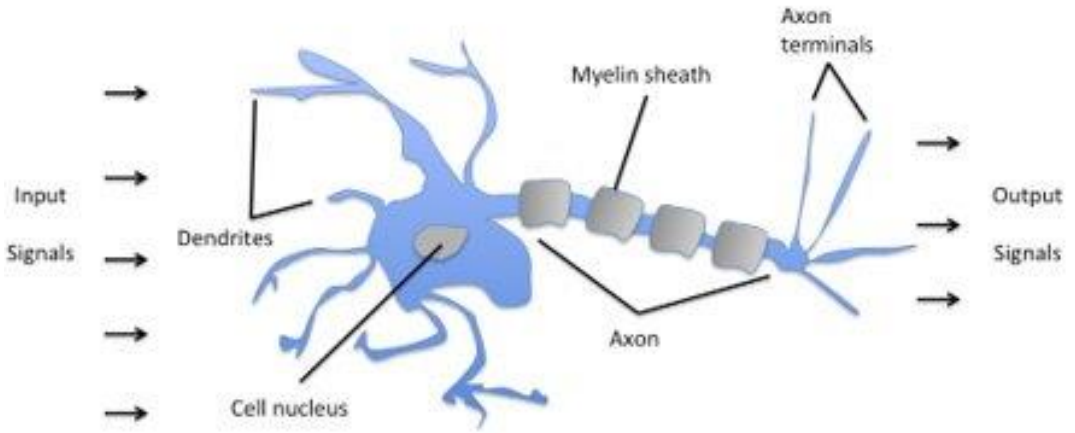
# Neurons and the brain



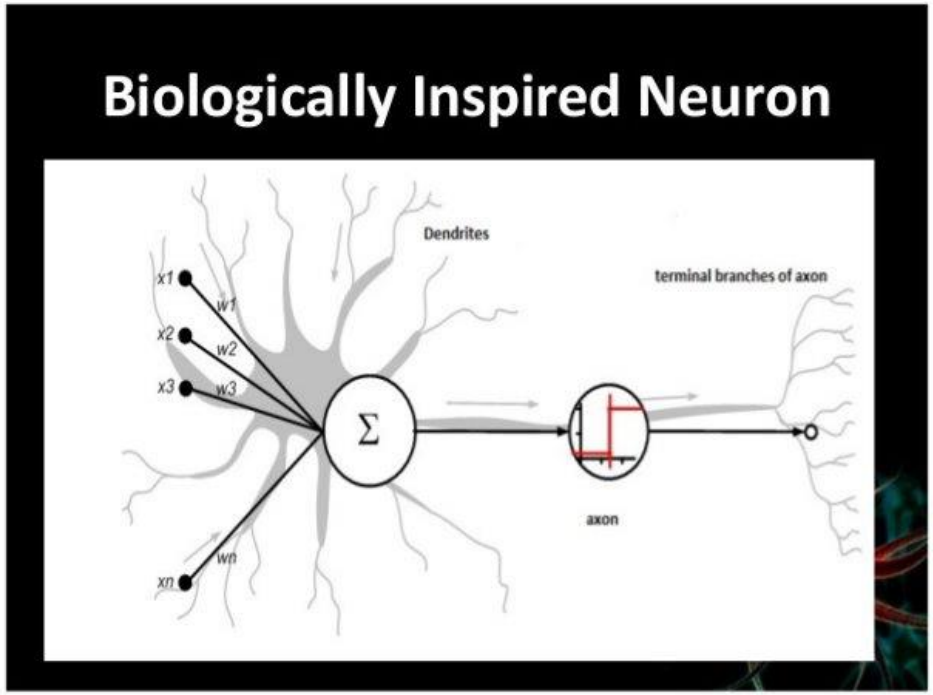
# Basic Concepts of Neural Networks

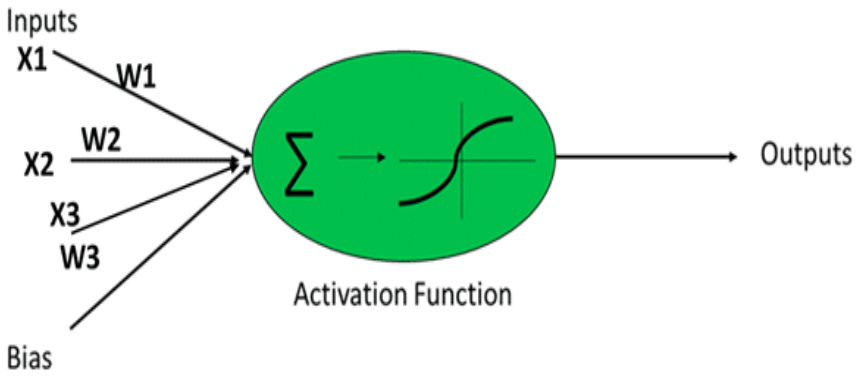
- Biological and artificial neural networks
  - Perceptron （感知器）
    Early neural network structure that uses no hidden layer
  - **Neurons (神经元）**
    Cells (processing elements) of a biological or artificial neural network
  - **Nucleus （神经核）**
    The central processing portion of a neuron
  - **Dendrite （树突）**
    The part of a biological neuron that provides inputs to the cell
  - **Axon （轴突）**
    An outgoing connection (i.e., terminal) from a biological neuron
  - **Synapse （突触）**
    The connection (where the weights are) between processing elements in a neural network




**Schematic of a biological neuron.**




- Dendrite receives signals from other neurons.
- Cell body sums the incoming signals to generate input.
- When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons.
- The amount of signal transmitted depend upon the strength of the connections.
- Connections can be inhibitory, i.e. decreasing strength or excitatory, i.e. increasing strength in nature.

**Perceptron is a single layer neural network**

The Neural Networks work the same way as the perceptron.

So, if you want to know how neural network works, learn how perceptron works.

# The perceptron consists of 4 parts

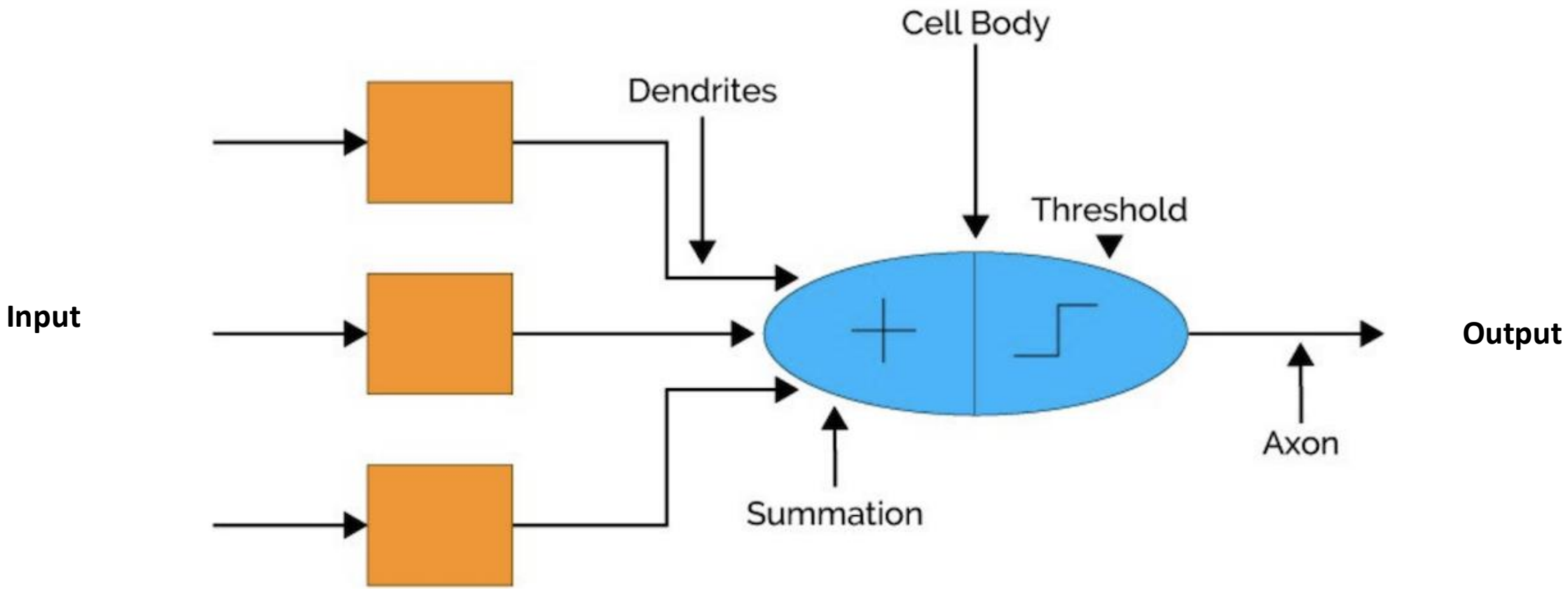
Input values or One input layer

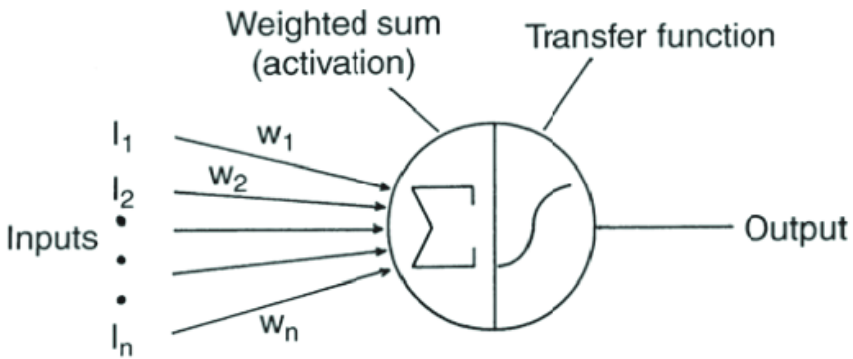
Weights and Bias

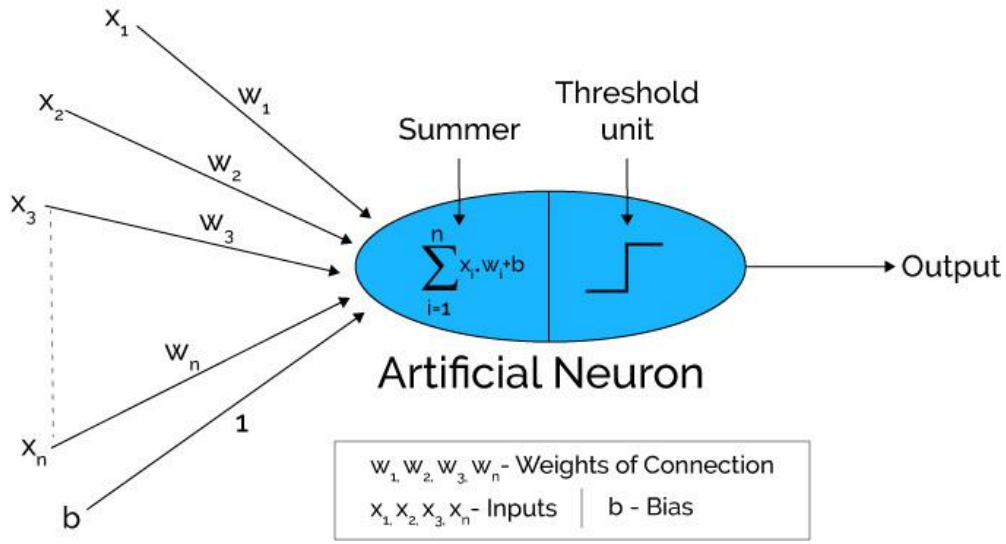
Net sum

Activation Function

# Perceptron

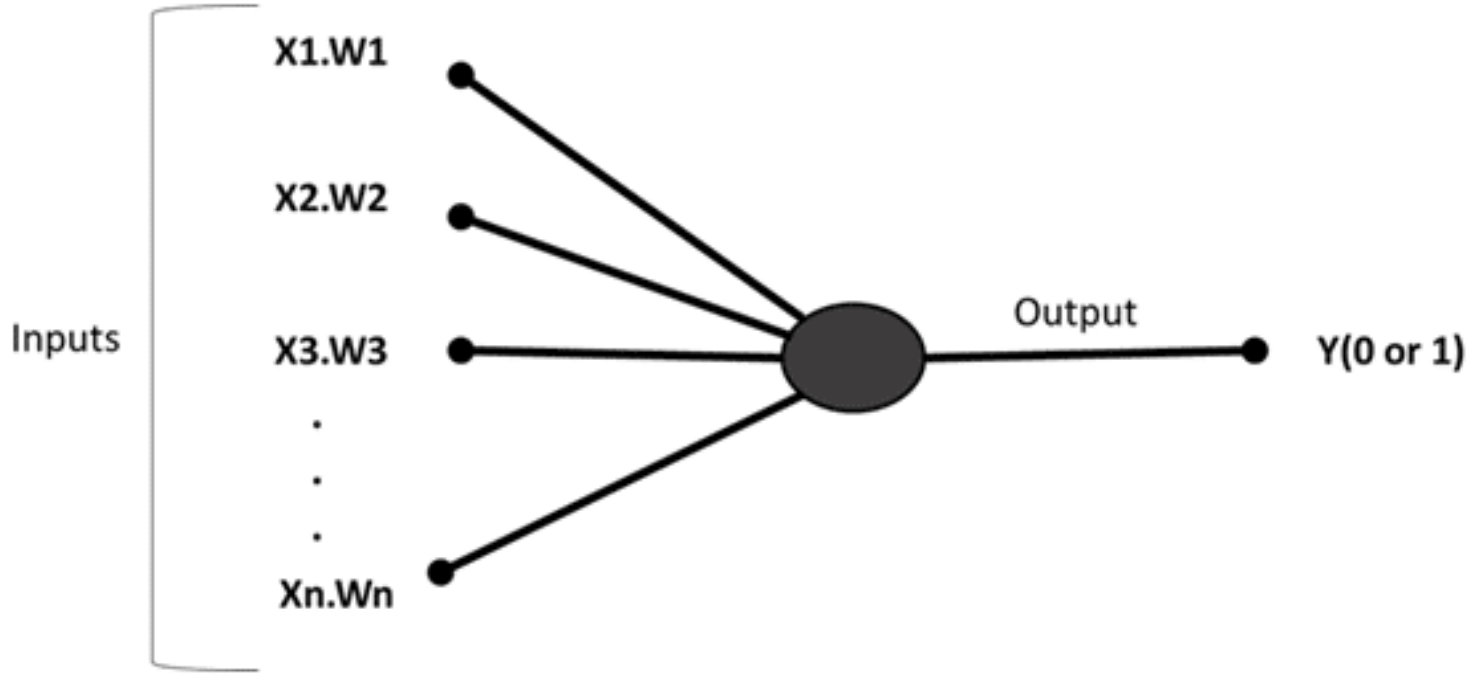

Cell Body
Dendrites
Threshold
Input
Output
Axon
Summation


Weighted sum
(activation)
Transfer function
Inputs
$I_1$
$I_2$
$I_n$
$w_1$
$w_2$
$w_n$
Output

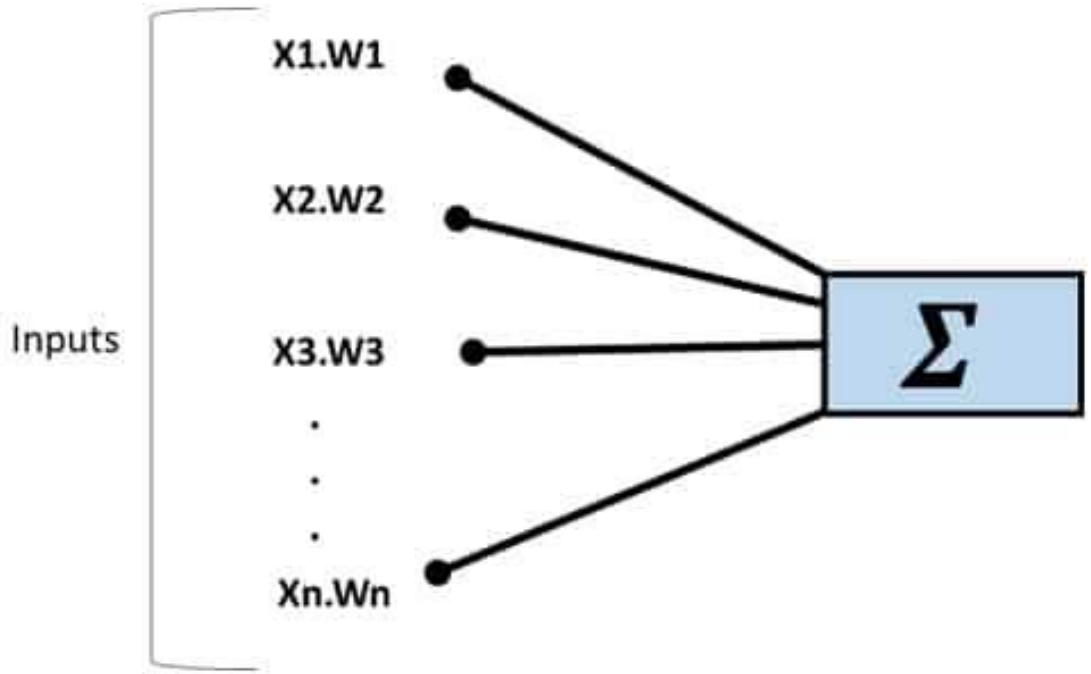

$x_1$
$x_2$
$x_3$
$x_n$
b
$w_1$
$w_2$
$w_3$
$w_n$
1
Summer
Threshold
unit
$\sum_{i=1}^{n} x_i.w_i+b$
Output
Artificial Neuron
$w_1, w_2, w_3, w_n$ - Weights of Connection
$x_1, x_2, x_3, x_n$ - Inputs
b - Bias

# How perceptron works?

**A.** All the inputs X1, X2, X3,…., Xn multiplies with their respective weights.



**B.** All the multiplied values are added.



**C.** Sum of the values are applied to the activation function.
For Example : Unit Step Activation Function.

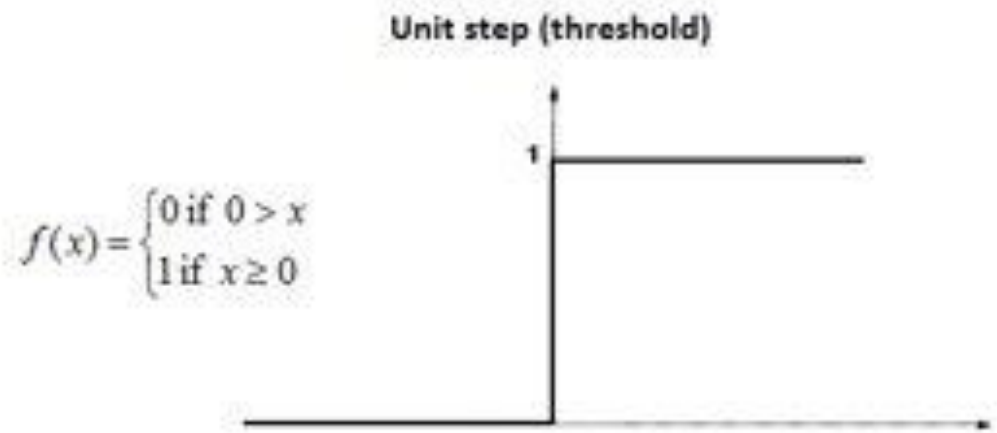
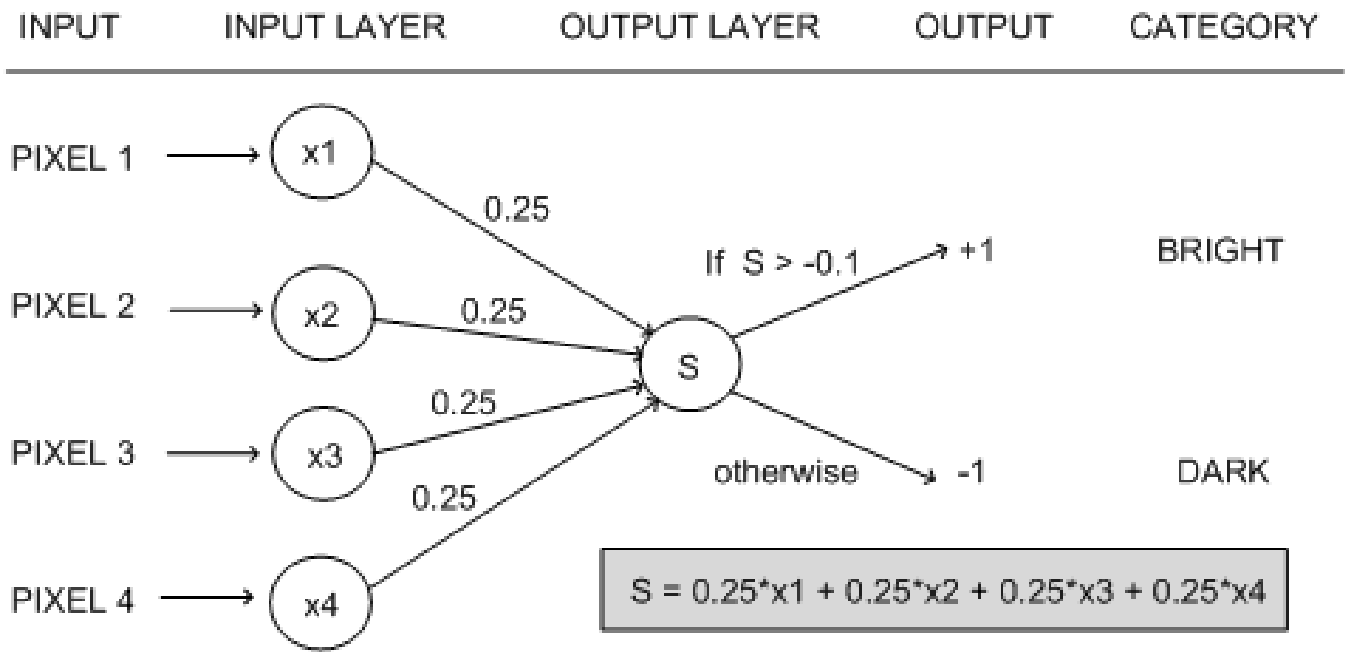



**Fig: Unit Step Activation Function**

# Why do we need Weights and Bias?

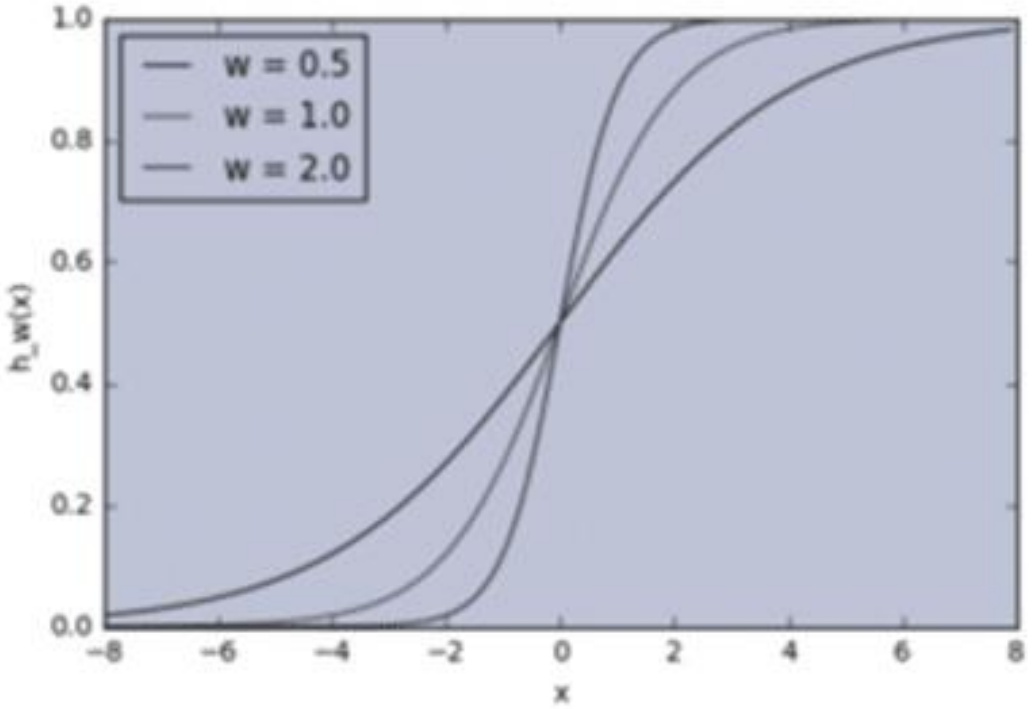
- **Weights** shows the strength of the particular node.
- ***A bias*** value allows you to shift the activation function curve up or down.



# Weights and Bias

- Weights W1, W2, W3,…., Wn shows the strength of a neuron.
- Bias allows you to change/vary the curve of the activation curve.

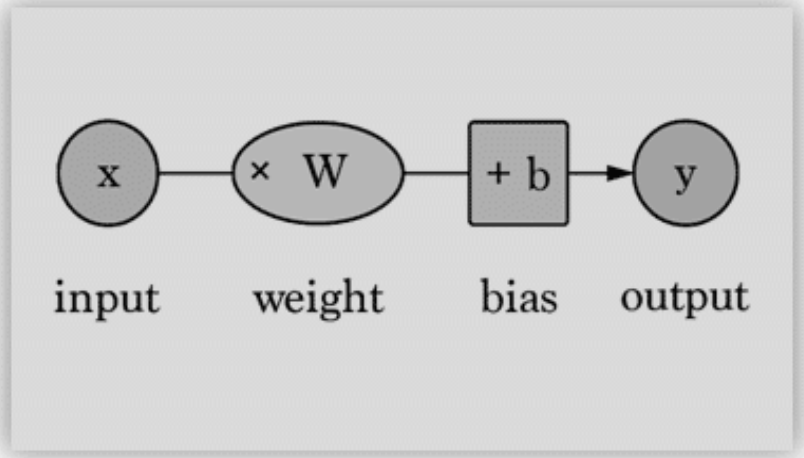
Weight without bias curve graph:

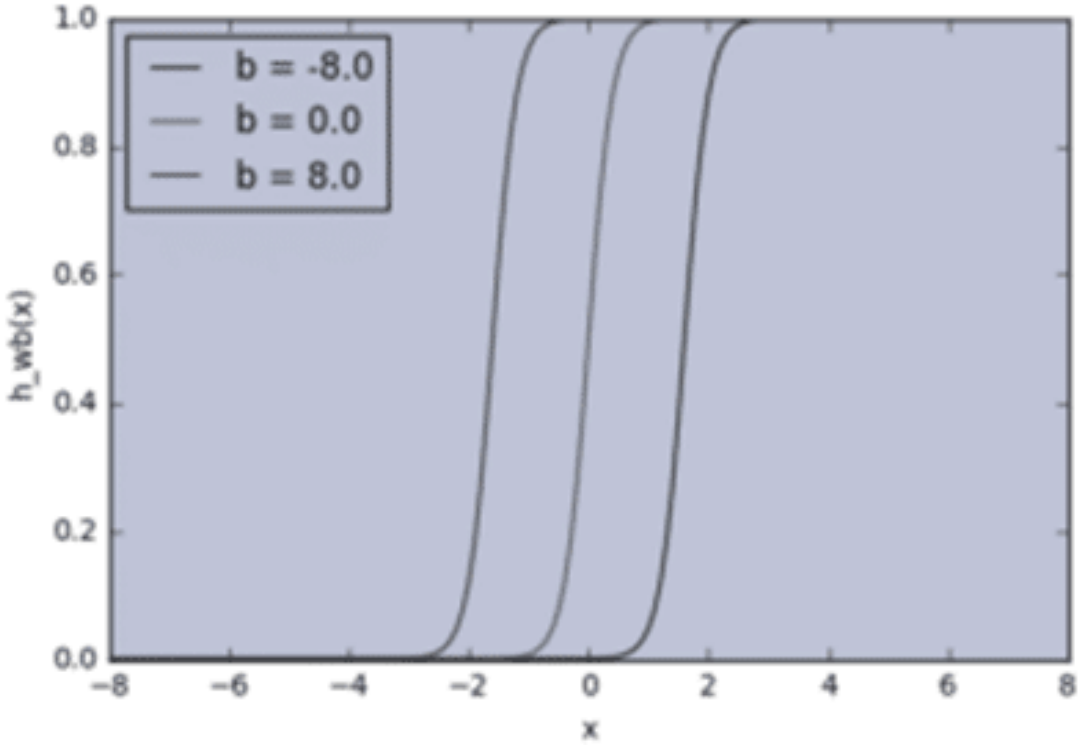



- Here, by changing weights, you can very input and outputs.
- Different weights changes the output slope of the activation function.
- This can be useful to model Input-Output relationships.

- What if you only want output to be changed when X>1? Here, the role of bias starts.

As you can see, by varying the bias b, you can change when the node activates. Without a bias, you cannot vary the output.



# Bias

- To introduce bias in the model, we add an input node ‘1’ and try to find the weight of the particular node.
- For example, if we want to predict a model to find out how much interest rates should be imposed on person A’s credit card whose salary is x, expenditure is y, we make a model for each salary and find the interest rate.
- If the person B does not have any salary, then the interest rate will not be zero, it will a default interest rate is applied, it is called bias
- It is generally introduced in the model to find out one more input node ‘1’ with weight w0.

# Let’s understand with an example.

- Suppose you want to go to a food shop. Based on the three factors you will decide whether to go out or not, i.e.
- Weather is good or not, i.e. X1. Say X1=1 for good weather and X1=0 for bad weather.
- You have vehicle available or not, i.e. X2. Say X2=1 for vehicle available and X2=0 for not having vehicle.
- You have money or not, i.e. X3. Say X3=1 for having money and X3=0 for not having money.
- Based on the conditions, you choose weight on each condition like W1=6 for money as money is the first important thing you must have, W2=2 for vehicle and W3=2 for weather and say you have set threshold to 5.
- In this way, perceptron makes decision making model by calculating X1W1, X2W2, and X3W3 and comparing these values to the desired output.

# How are the weights re-calibrated?

- Re-calibration of weights is an easy, but a lengthy process.
- The only nodes where we know the error rate are the output nodes. Re-calibration of weights on the linkage between hidden node and output node is a function of this error rate on output nodes.
- **But, how do we find the error rate at the hidden nodes?**
- It can be statistically proved that:
- **Error @ H1 = W(H1O1)*Error@O1 + W(H1O2)*Error@O2**
- Using these errors we can re-calibrate the weights of linkage between hidden nodes and the input nodes in a similar fashion. Imagine, that this calculation is done multiple times for each of the observation in the training set.

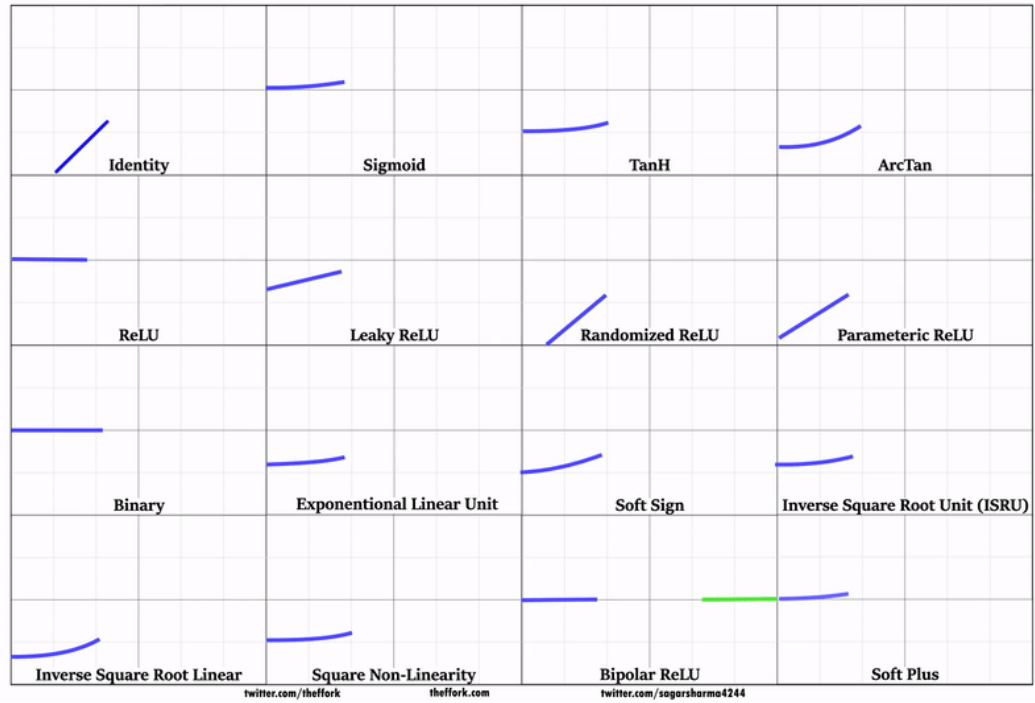
What is Activation Function?

- It is a curve (sigmoid, tanH, ReLU) which is used to map the values of the network between bounded values. This is done for every node in the network. For example, sigmoid can map any range of values between 0 and 1.
- The value of the activation function is then assigned to the node.

Why do we need Activation Function?

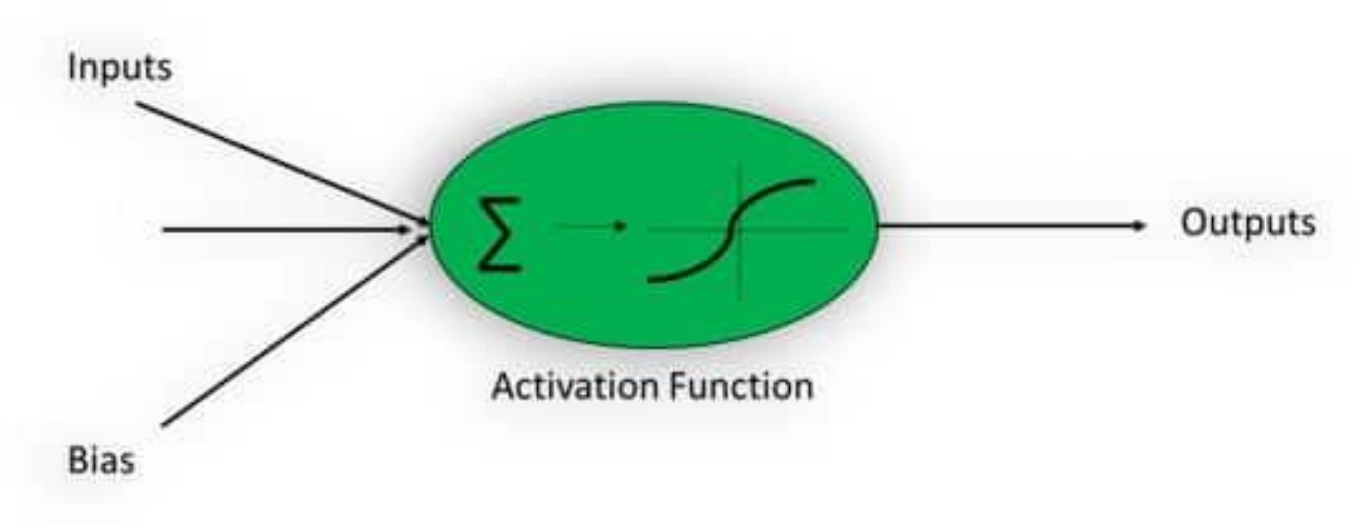
•In short, the activation functions are used to map the input between the required values like (0, 1) or (-1, 1).

- https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
- https://theffork.com/activation-functions-in-neural-networks/
- https://github.com/Sagarsharma4244/Activaiton-Funciton-form-Scratch
- https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975



# Activation Function

- Activation functions are used for non-linear complex functional mappings between the inputs and required variable. They introduce non-linear properties to our Network.
- They convert an input of an artificial neuron to output. That output signal now is used as input in the next layer.




Simply, input between the required values like (0, 1) or (-1, 1) are mapped with the activation function.

# Why Activation Function?

- Activation Function helps to solve the complex non-linear model. Without activation function, output signal will just be a linear function and your neural network will not be able to learn complex data such as audio, image, speech, etc.
- Some commonly used activation functions are:
- Sigmoid or Logistic
- Tanh — Hyperbolic tangent
- ReLu -Rectified linear units

# Activation function

The activation function of the node defines the output of the node. There are 4 most popular activation function:

Step function – It restricts the value of output to 0 and 1.

Rectified linear unit – ReLU is like half of step function, it suppresses the negative values. It is the most popular and utilized function.

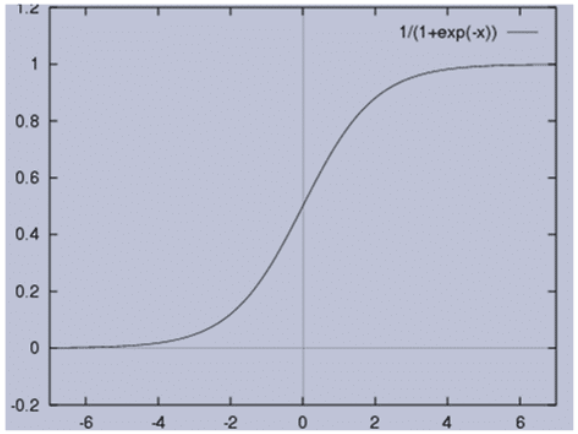
Sigmoid function – Better than step function, it also limits the output from 0 to 1, but it smoothens the value. It is also called probabilities, it is a continuous function. When we have binary problems, we use sigmoid function.

Tanh function – similar to sigmoid, it limits the function from -1 to 1.

## Sigmoid Activation Function:

Sigmoid Activation Function can be represented as:

**f(x) = 1 / 1 + exp(-x)**




- Range of sigmoid function is between 0 and 1.
- It has some disadvantages like slow convergence, vanishing gradient problem or it kill gradient, etc. Output of Sigmoid is not zero centered that makes its gradient to go in different directions.

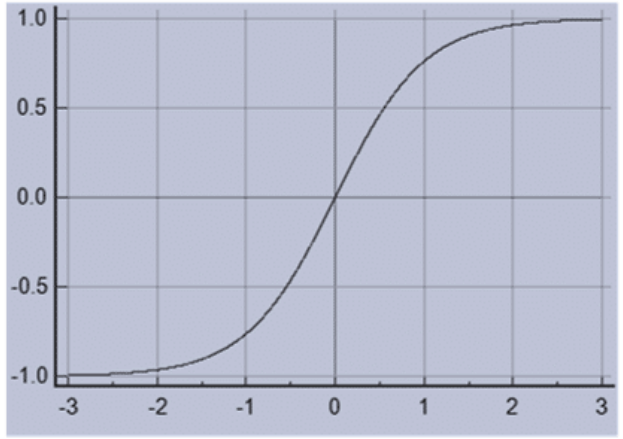
## Tanh- Hyperbolic tangent

Tanh can be represented as:

**f(x) = 1 — exp(-2x) / 1 + exp(-2x)**

It solves the problem occurring with Sigmoid function. Output of Tanh is zero centered because range is between -1 and 1.

Optimization is easy as compared to Sigmoid function.




But still it suffers gradient vanishing problem.

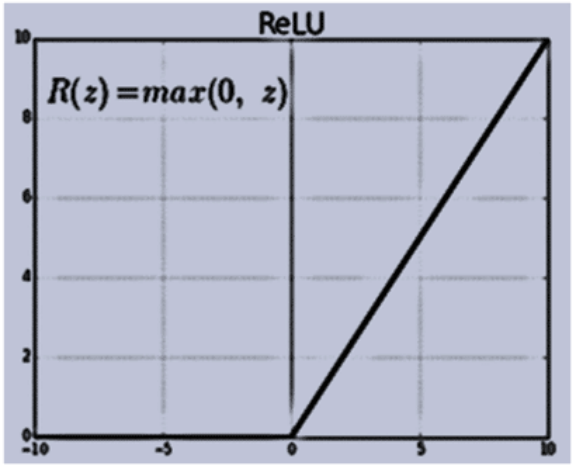
## ReLu- Rectified Linear units

It can be represented as:

R(x) = max(0,x)

if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x

It avoids as well as rectifies vanishing gradient problem. It has six times better convergence as compared to tanh function.




It should be used within hidden layers of the neural network.

# How a neural network knows what biases and weights to use.

- Neural networks often start off with random weights and biases, but then train themselves over and over again till they reach peak performance.
- They do this by calculating the amount of error they currently have. This is called the cost of the neural network.
- This is calculated by finding the difference between the network's prediction, and the desired result and finding the sum of those error values' squares $(\text{target} - \text{output})^2$.
- In the graph, the red line represents the predictions of a simple neural network. The blue dots are the correct predictions.
- The neural network calculates how far the predictions (red line) are from the actual values (blue dots) and squares it (creates the green squares).
- It then adds up those squared values to give you the cost of the neural network.

# Loss function

- The loss function is a measure of the model's performance.
- After you have defined the hidden layers and the activation function, you need to specify the loss function and the optimizer.
- For binary classification, it is common practice to use a binary cross entropy loss function. In the linear regression, you use the mean square error.
- The loss function is an important metric to estimate the performance of the optimizer. During the training, this metric will be minimized. You need to select this quantity carefully depending on the type of problem you are dealing with.

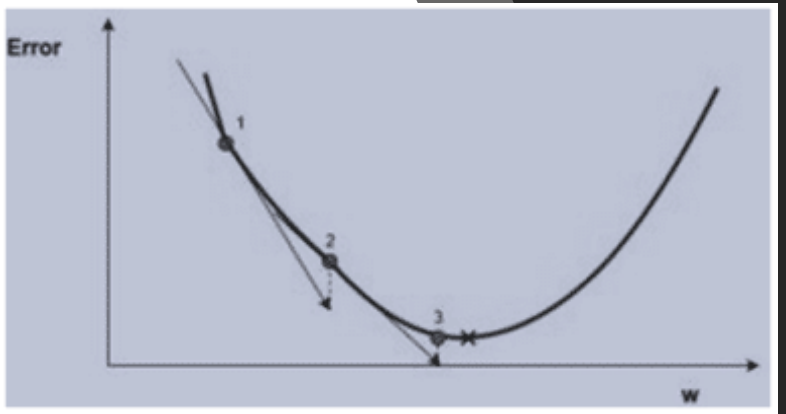
# Optimizer

- The loss function is a measure of the model's performance. The optimizer will help improve the weights of the network in order to decrease the loss. There are different optimizers available, but the most common one is the Stochastic Gradient Descent.

The conventional optimizers are:

- Momentum optimization,
- Nesterov Accelerated Gradient,
- AdaGrad,
- Adam optimization

# Gradient Descent




- Gradient is the slope of the error curve.
- The idea of introducing gradient to reduce the or minimize the error between the desired output and the input. To predict the output based on the every input, weight must be varied to minimize the error.
- Problem is how to vary weight seeing the output error. This can be solved by gradient descent.
- In the above graph, blue plot shows the error, red dot shows the ‘w’ value to minimize the error and the black cross or line show the gradient.
- At point 1, random ‘w’ value is selected with respect to error and gradient is checked.
- If gradient is positive with respect to the increase in w, then step towards will increase the error and if it is negative with respect to increase in ‘w’, then step towards will decrease the error. In this way, gradient shows the direction of the error curve.
- The process of minimizing error will continue till the output value reaches close to the desired output. This is a type of **Backpropagation**.

# Feed-Forward Neural Network

Feed-forward network means data flows in only one direction, i.e. from input to output.

In gradient topic, you have studied about minimizing the error. The main agenda is also to minimize the error and for that there are various methods.

In feed-forward neural network, when the input is given to the network before going to the next process, it guesses the output by judging the input value. After guess, it checks the guessing value to the desired output value. The difference between the guessing value and the desired output is error.

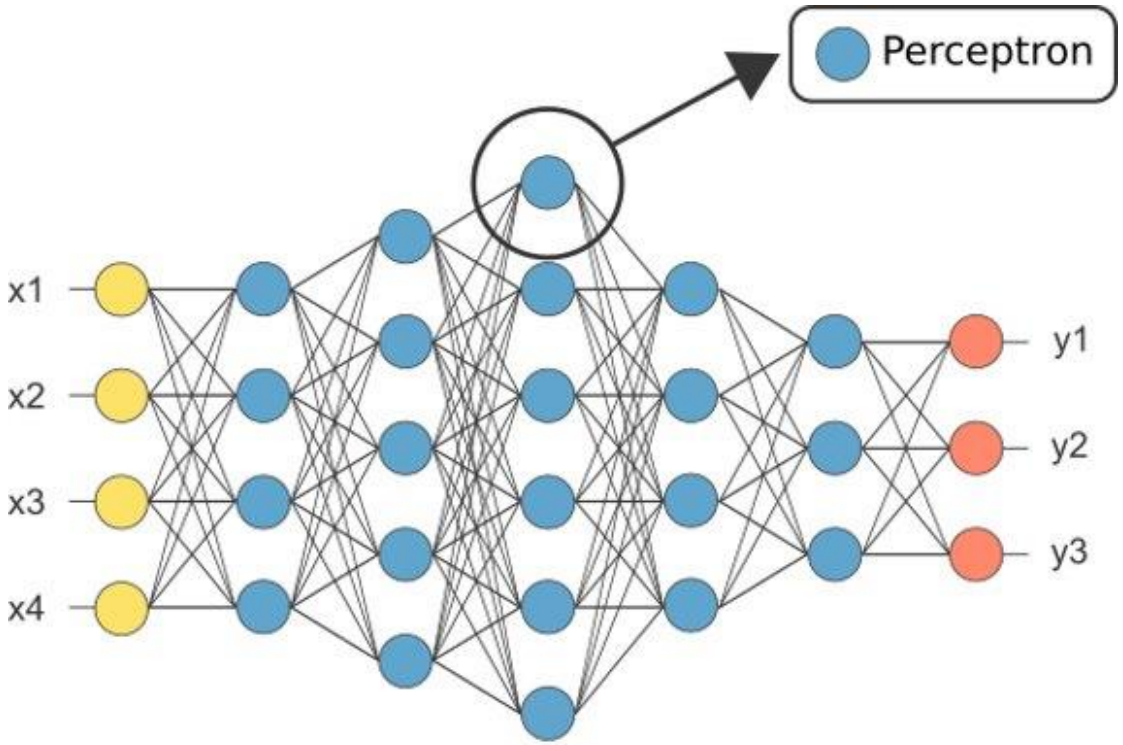
**Guess= input * weight**

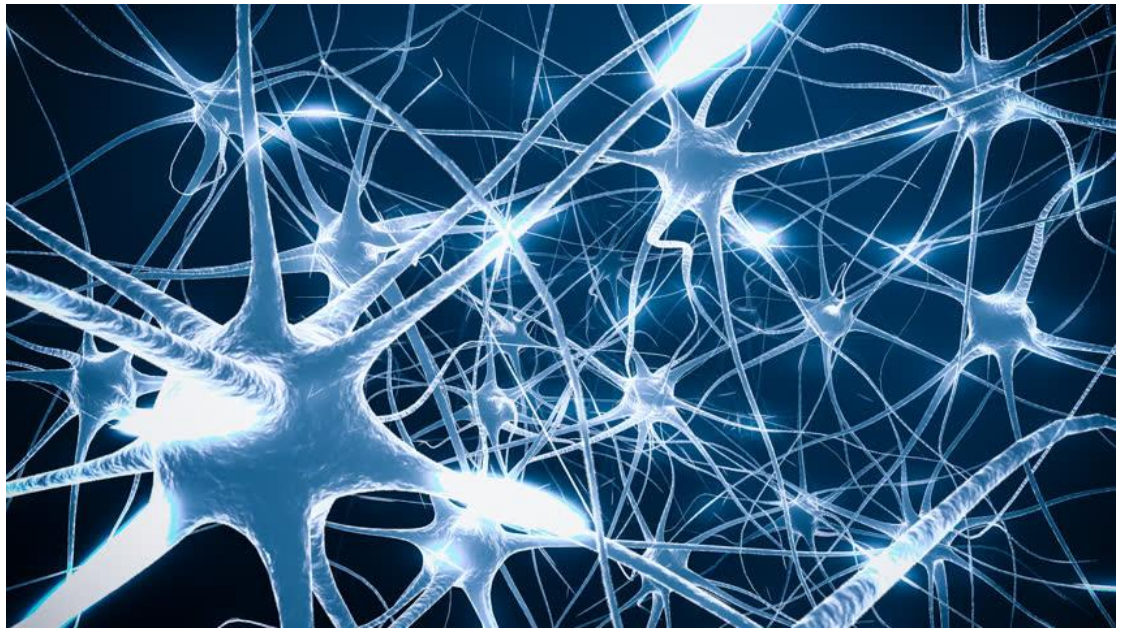
**Error= Desired Output – Guess**

You already know how to minimize the error.

- This is how backpropagation works. It uses gradient descent algorithm.

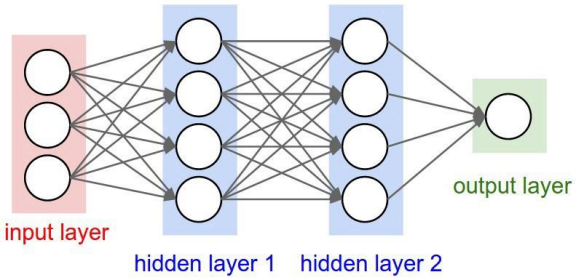
- This is how backpropagation works. It uses gradient descent algorithm.


Perceptron
x1
x2
x3
x4
y1
y2
y3



A neural network is made up of 3 main parts:

Input layer
This is literally the layer that inputs information for the neural network to process. Each circle represents 1 feature (a piece of information). This can be anything. It could be the square footage of your house for a house price prediction program, or the value of a pixel on a screen for a computer vision program




Hidden layers
These layers do all the processing for neural networks. You can have as many of these as you want. Generally speaking, the more hidden layers you have, the more accurate the neural network will be. Each layer consists of nodes that mimic our brains' neurons. These nodes receive information from the previous layer's nodes, multiply it by **weight** and then add a **bias** to it. Each line in the diagram represents a weight.

Output Layer
This layer simply brings together the information from the last hidden layer of the network to output all the information you need from the program.

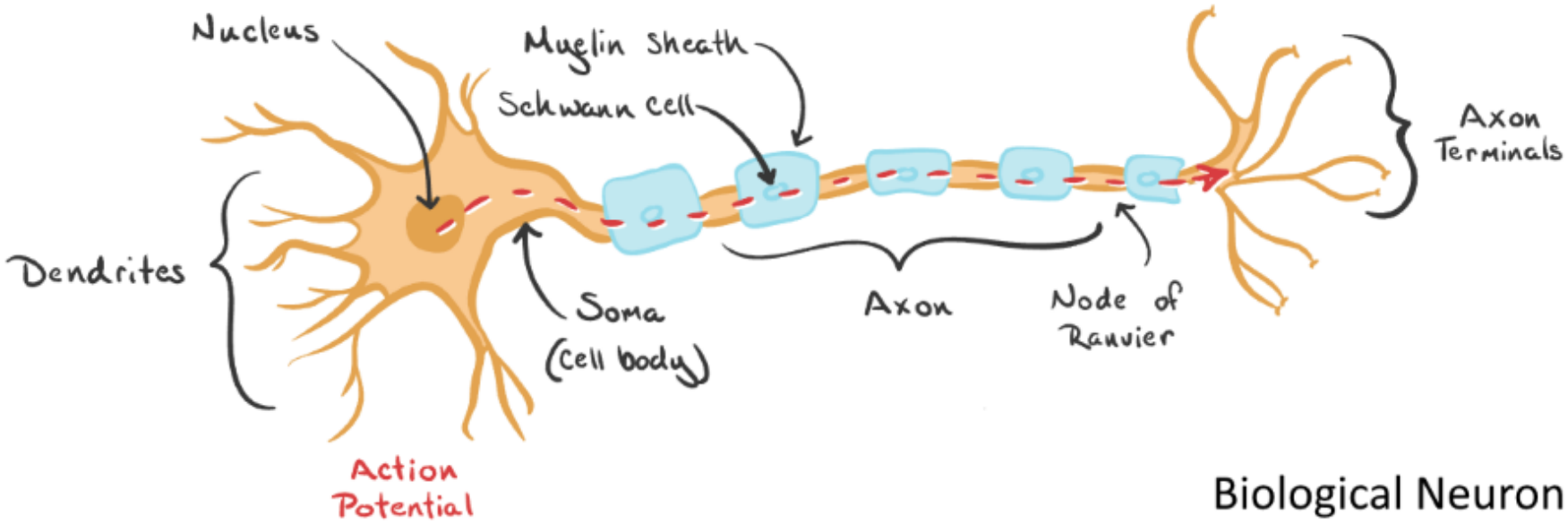
- To sum up, neural networks take information from the input layer, process it in the hidden layers, and output the desired information in the output layer. This whole process of running a neural network is called **forward propagation.**

# Optimizing the Weights and Biases

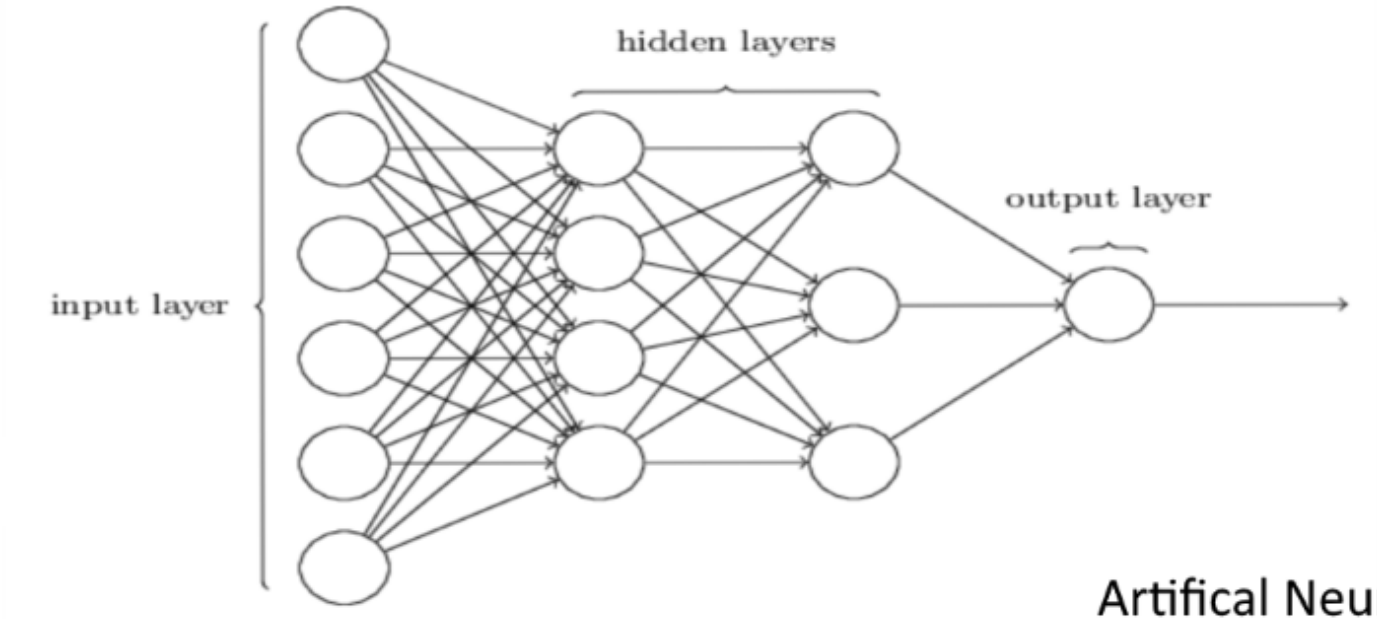
- The entire goal of training the neural network is to minimize the cost. Neural networks do this using a process called **backpropagation**.
- This seems like a complicated word but its quite simple.
- forward propagation is when you run information through a neural network to give you a result.
- Backward propagation is literally the same thing but backward.
- You just start at the output layer and run the neural network in reverse to optimize the weights and biases.
- The math is quite complicated and beyond the scope of this session, but you can always go search the internet if you're interested in learning it!

# Framework of artificial neural networks (ANN)

1. Assign Random weights to all the linkages to start the algorithm
2. Using the inputs and the (Input ->Hidden node) linkages find the activation rate of Hidden Nodes
3. Using the activation rate of Hidden nodes and linkages to Output, find the activation rate of Output Nodes
4. Find the error rate at the output node and recalibrate all the linkages between Hidden Nodes and Output Nodes
5. Using the Weights and error found at Output node, cascade down the error to Hidden Nodes
6. Recalibrate the weights between hidden node and the input nodes
7. Repeat the process till the convergence criterion is met
8. Using the final linkage weights score the activation rate of the output nodes




Biological Neuron




Artifical Neural Network

# Training Perceptrons

- The most common deep learning algorithm for supervised training of the multilayer perceptrons is known as **backpropagation.**

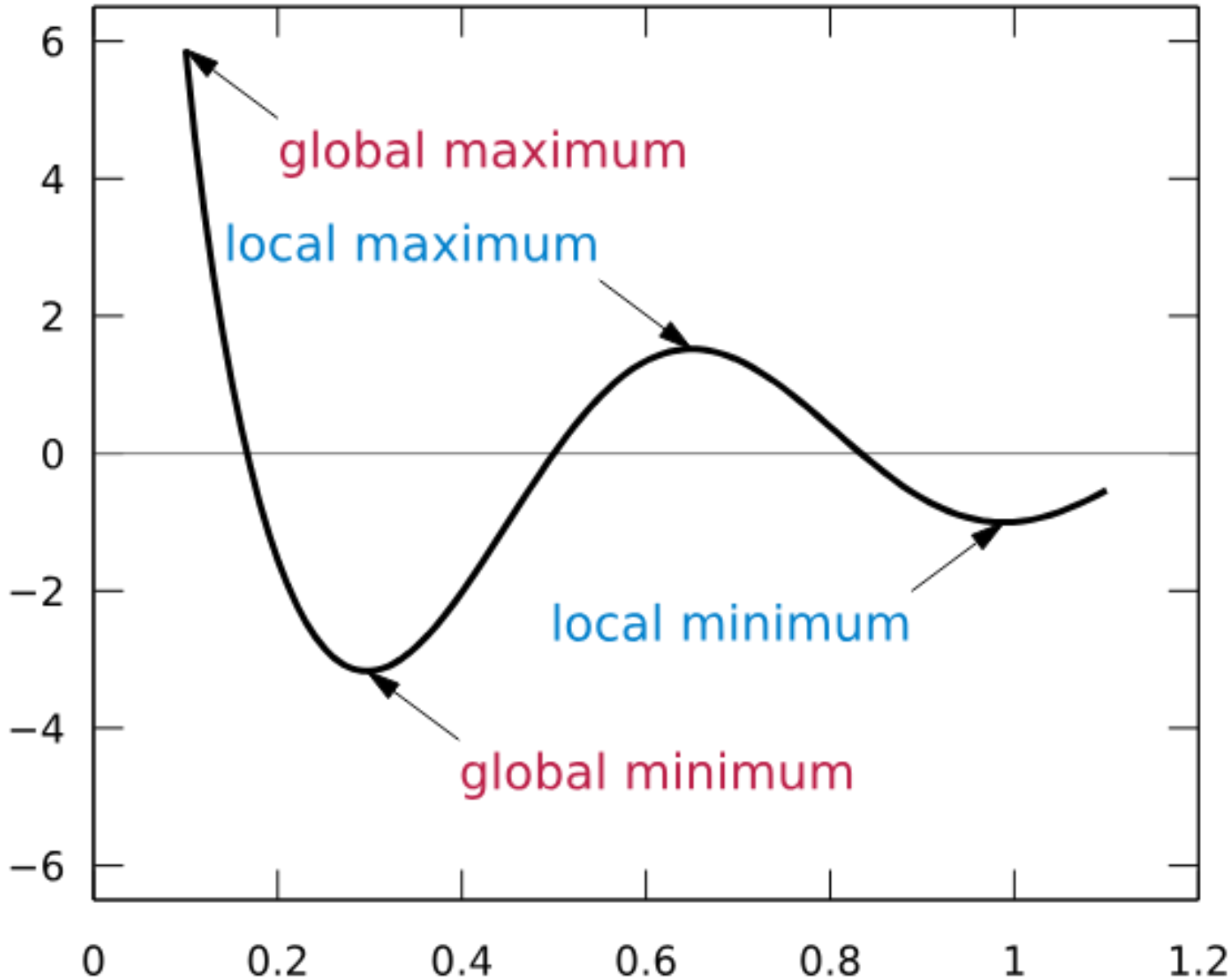
The basic procedure:

- A training sample is presented and propagated forward through the network.
- The output error is calculated, typically the mean squared error:

$$E = \frac{1}{2}(t - y)^2$$

Where *t* is the target value and *y* is the actual network output. Other error calculations are also acceptable, but the MSE is a good choice.

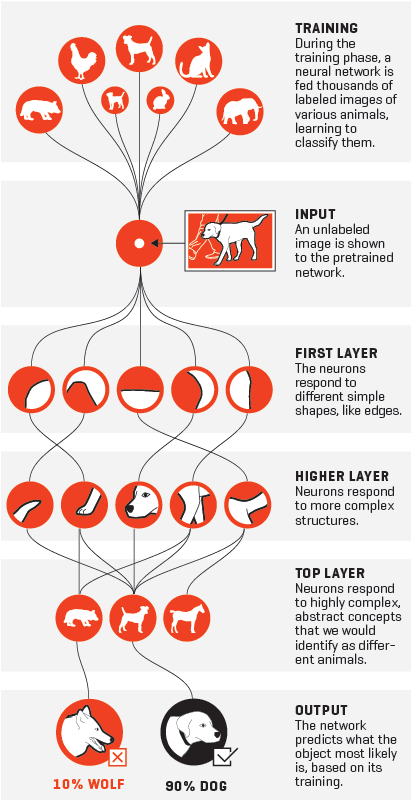
Network error is minimized using a method called stochastic gradient descent.


6
4
2
0
−2
−4
−6
0
0.2
0.4
0.6
0.8
1
1.2
global maximum
local maximum
local minimum
global minimum

# Types of Neural Network

- Convolutional Neural Network(CNN)
- Recursive Neural Network(RNN)
- Recurrent neural network (RNN)
- Long short-term memory (LSTM)

## HOW NEURAL NETWORKS RECOGNIZE A DOG IN A PHOTO




**TRAINING**
During the training phase, a neural network is fed thousands of labeled images of various animals, learning to classify them.

**INPUT**
An unlabeled image is shown to the pretrained network.

**FIRST LAYER**
The neurons respond to different simple shapes, like edges.

**HIGHER LAYER**
Neurons respond to more complex structures.

**TOP LAYER**
Neurons respond to highly complex, abstract concepts that we would identify as different animals.

**OUTPUT**
The network predicts what the object most likely is, based on its training.