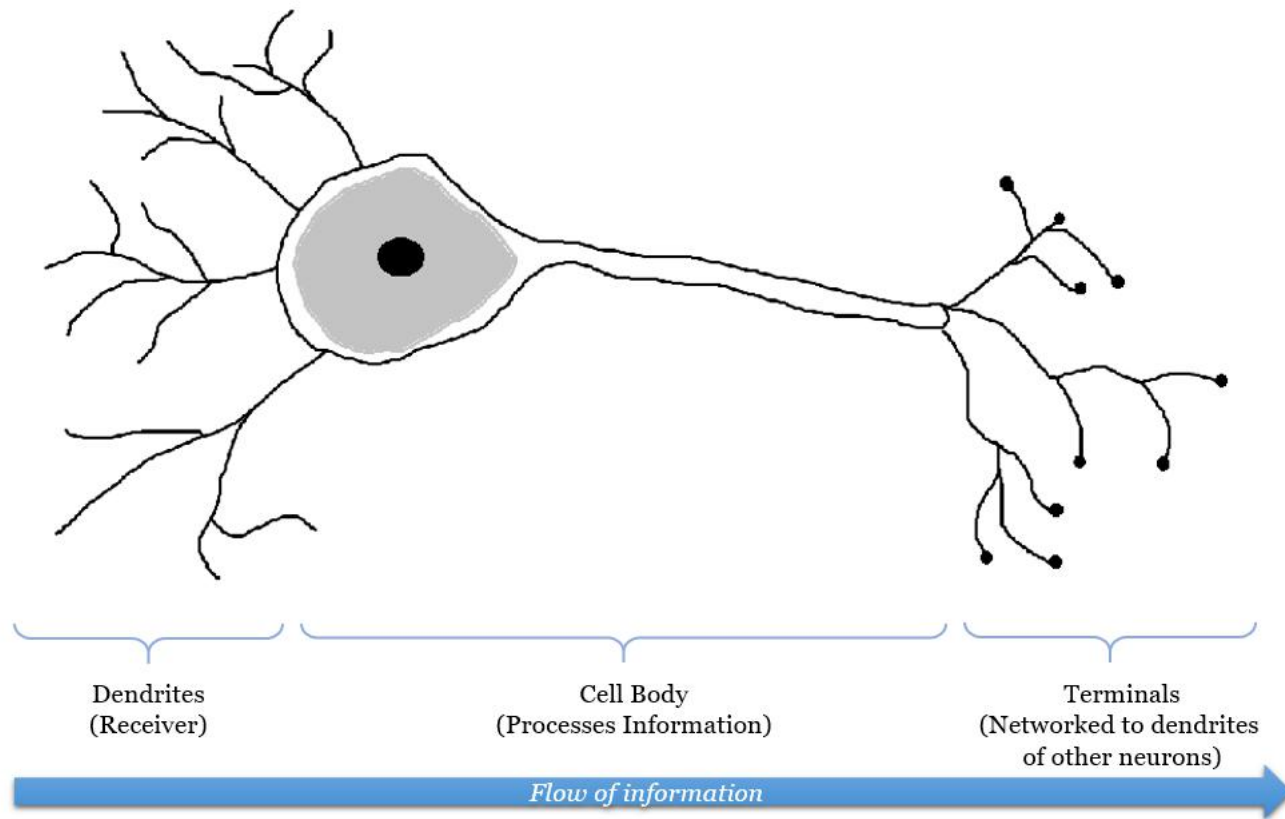


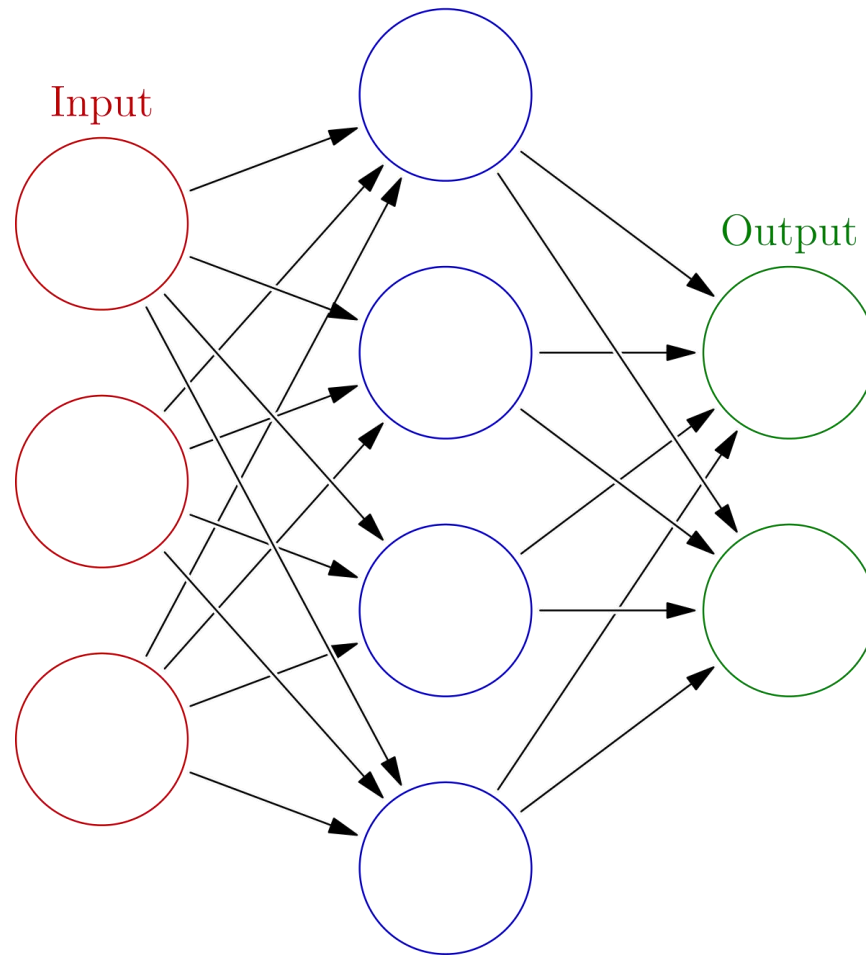
# Neural Network

Krishna Modi, DCS

# Biological Neuron



# Architecture of Neural Network



# Example

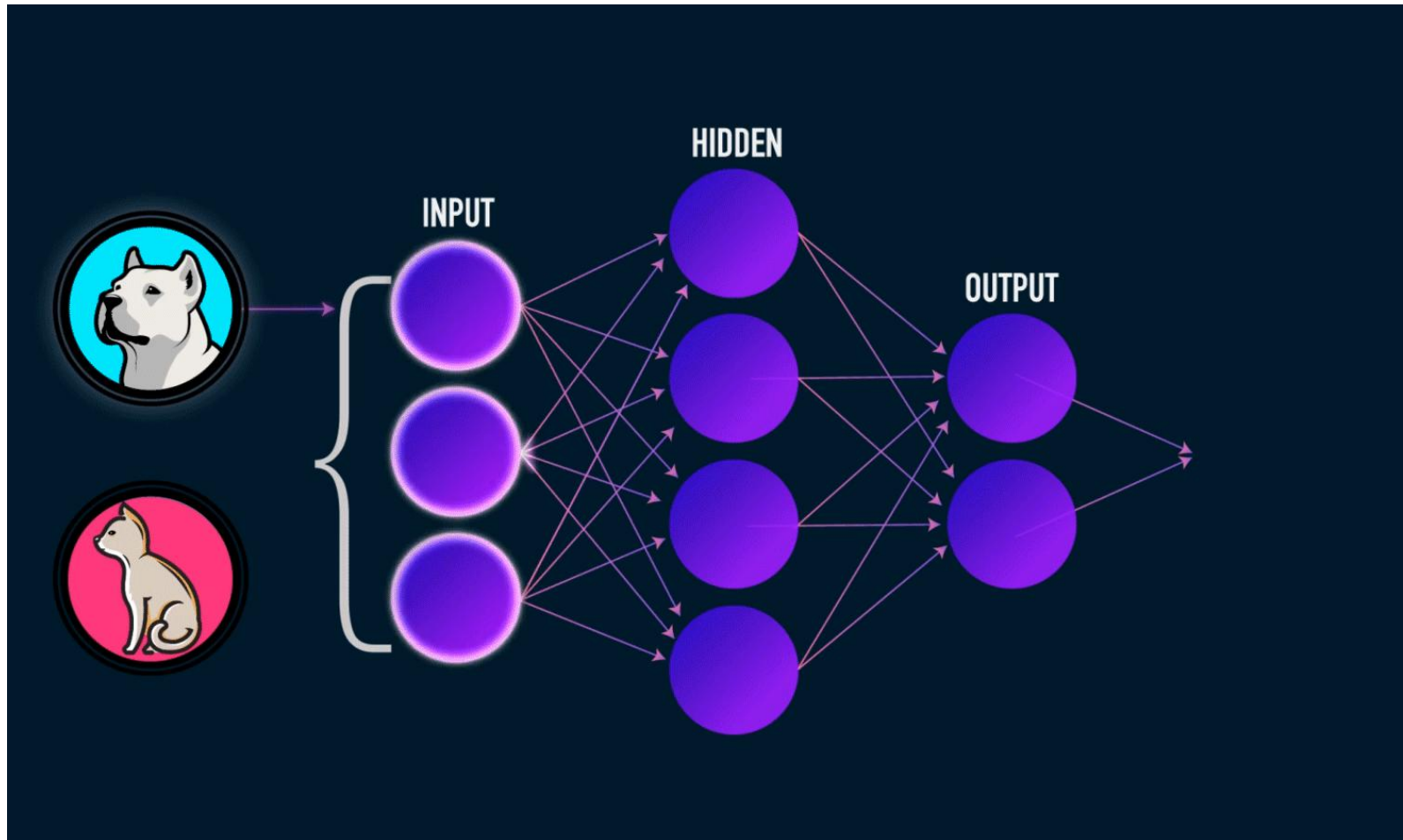
Attribute				Classes
Gender	Car ownership	Travel Cost (\$)/km	Income level	Transportation mode
Male	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Female	1	Cheap	Medium	Train
Female	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Male	0	Standard	Medium	Train
Female	1	Standard	Medium	Train
Female	1	Expensive	High	Car
Male	2	Expensive	Medium	Car
Female	2	Expensive	High	Car

# Example

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

# Elements of Neural Network

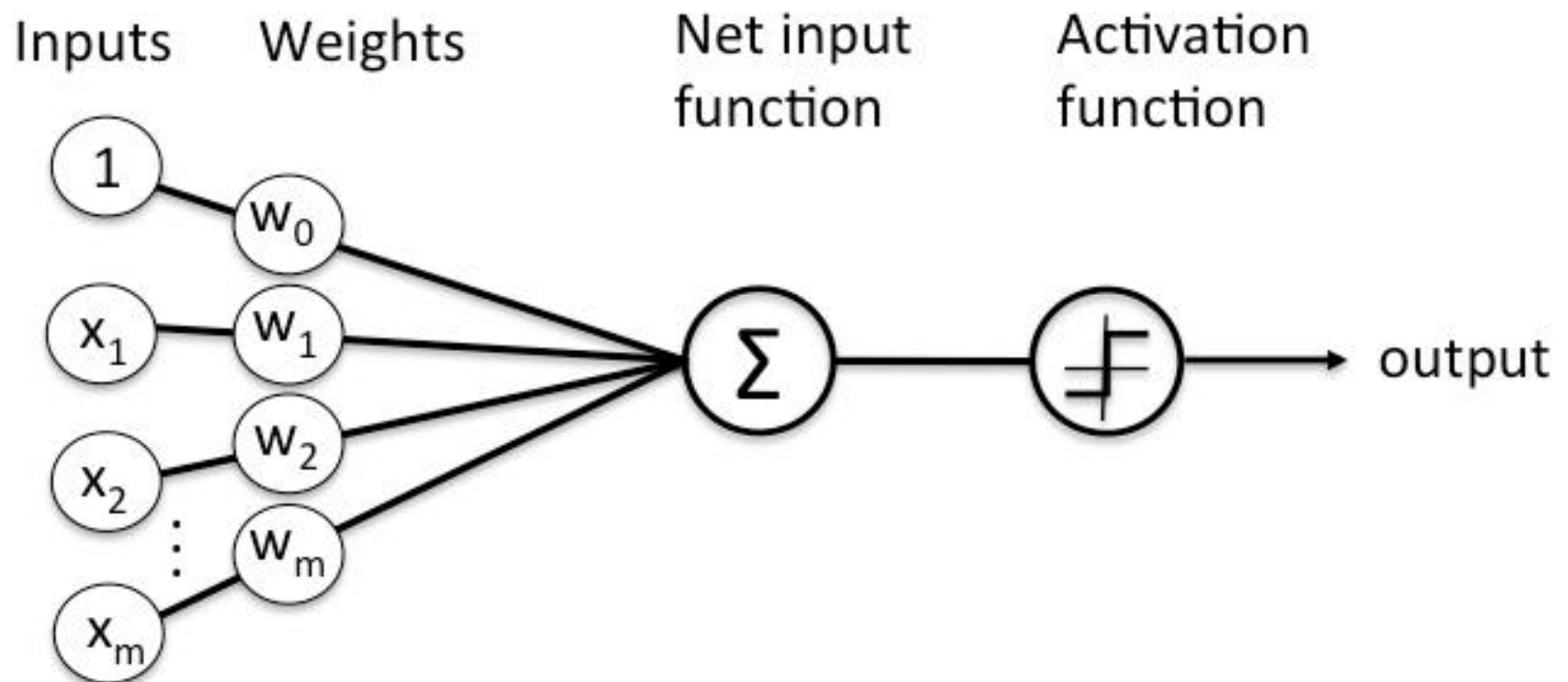
- **Input Layer :-** This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.
- **Hidden Layer :-** Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.
- **Output Layer :-** This layer bring up the information learned by the network to the outer world.



# Working of ANN

- ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value**.
- Each link is associated with **weight**. ANNs are capable of learning, which takes place by altering weight values.

# Artificial Neuron



# Activation Function

- Function used to get output of the neuron.
- It is also called as transfer function.
- It is used to determine output of neural network.

# TYPES OF ACTIVATION FUNCTIONS

Krishna Modi, DCS

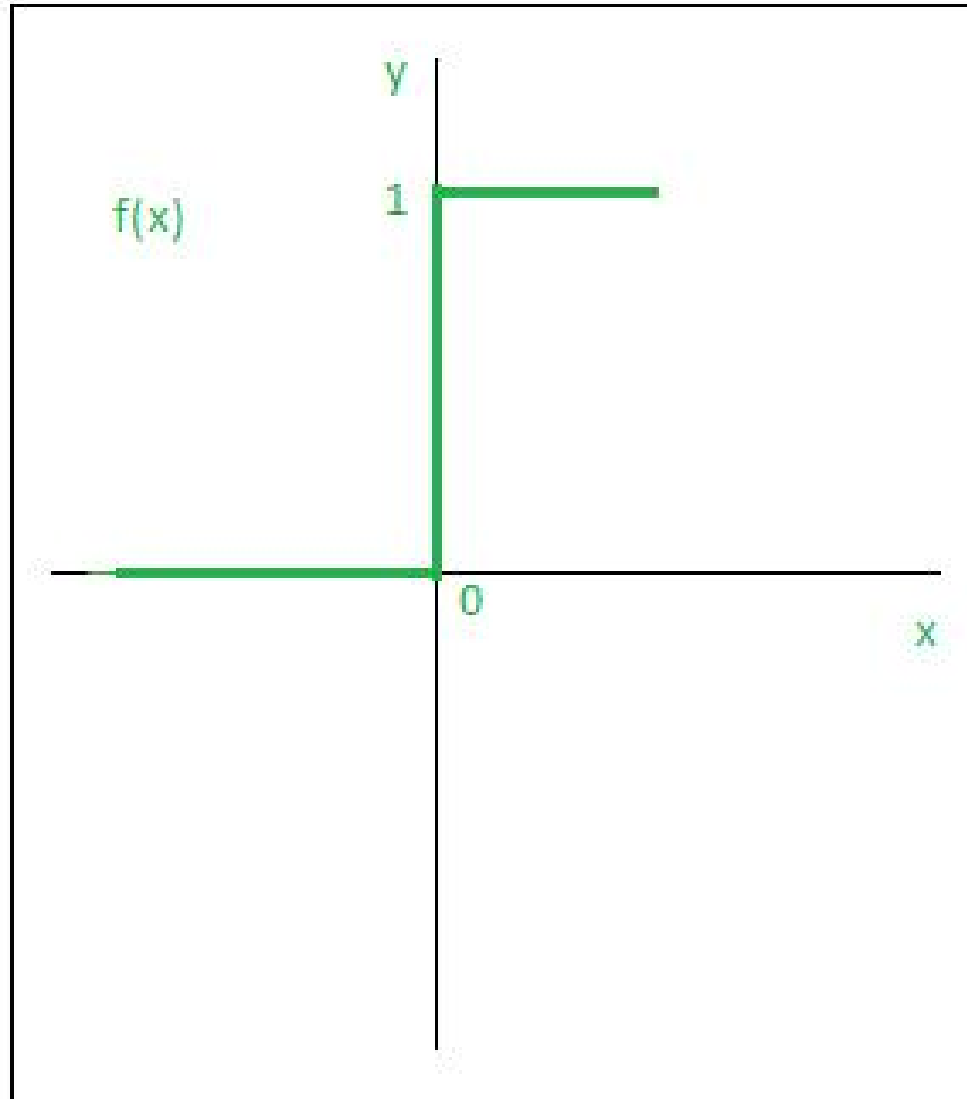
# Step Function

- Step Function is one of the simplest kind of activation functions. In this, we consider a threshold value and if the value of net input say  $y$  is greater than the threshold then the neuron is activated.

Mathematically,

$$f(x)=1, \text{ if } x \geq 0$$

$$f(x)=0, \text{ if } x < 0$$

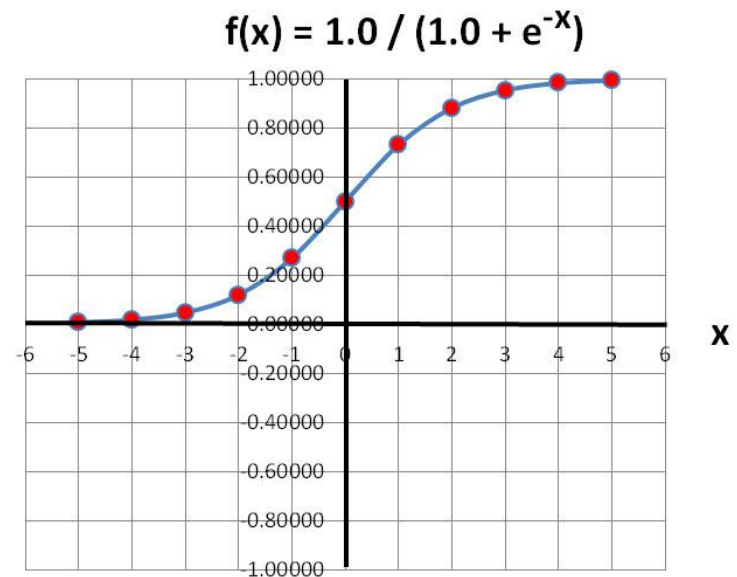


# Sigmoid Function

Widely used  
activation fu

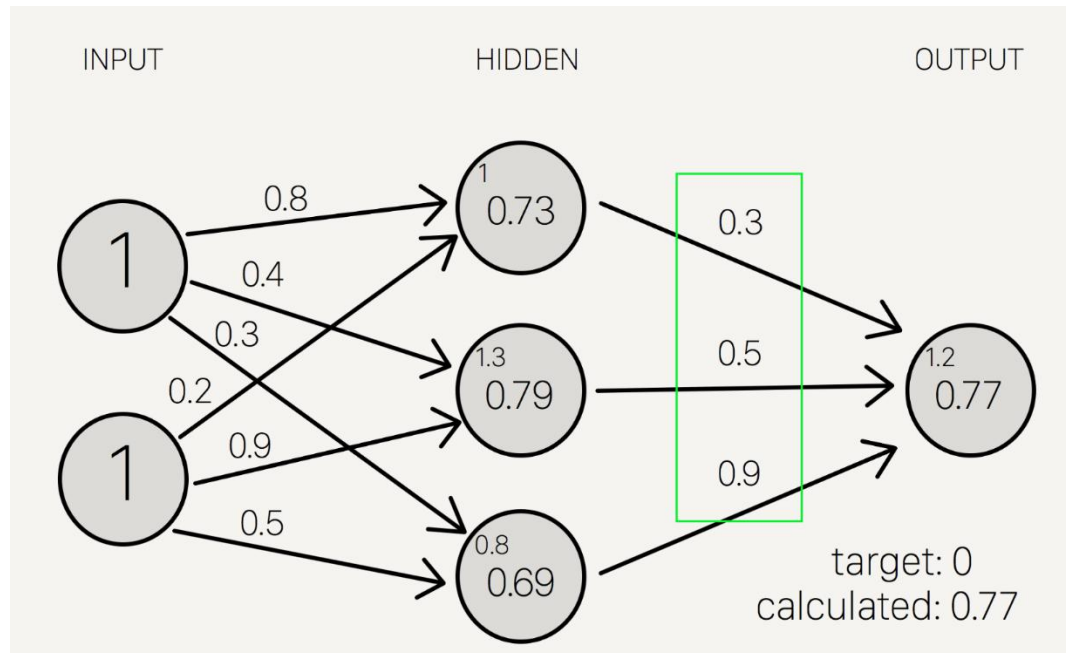
$$S(t) = \frac{1}{1 + e^{-t}}$$

x	f(x)
-5	0.00669
-4	0.01799
-3	0.04743
-2	0.11920
-1	0.26894
0	0.50000
1	0.73106
2	0.88080
3	0.95257
4	0.98201
5	0.99331



The Log-Sigmoid Function

# Example



Example of Neural Network

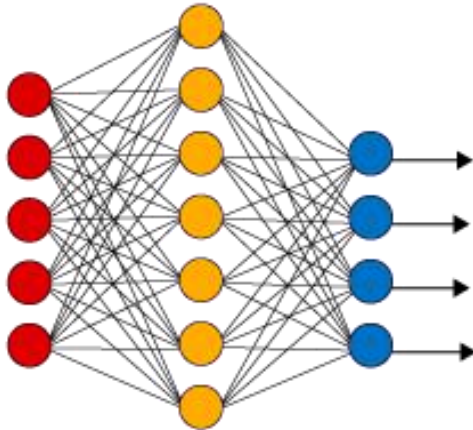
# Applications

- Natural Language Processing
- Facial Recognition
- Lip reading
- Image processing
- Fraud detection
- Weather Forecasting

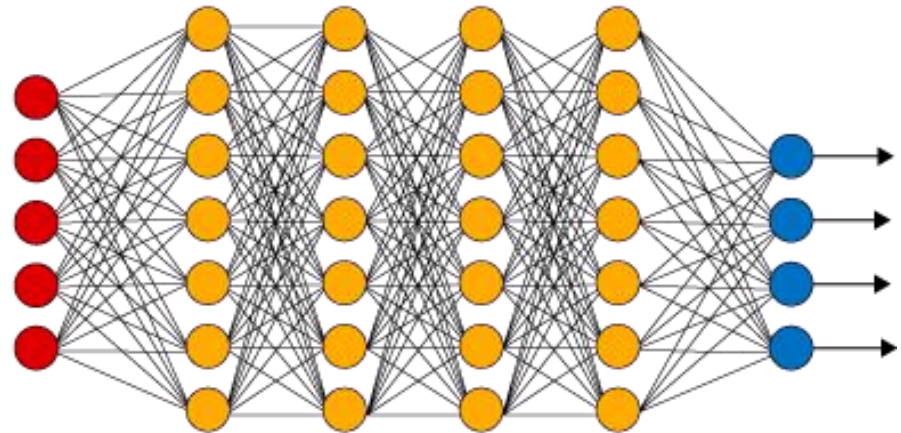


# Deep Learning

Simple Neural Network



Deep Learning Neural Network



● Input Layer

● Hidden Layer

● Output Layer

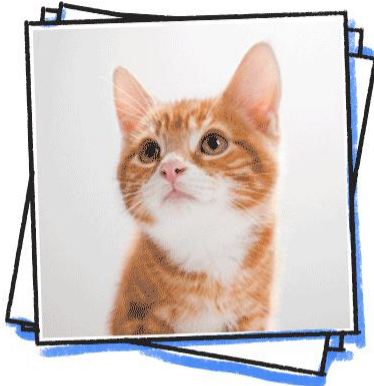
# Videos

- [Video 1](#)
- [Video 2](#)

# Perceptron

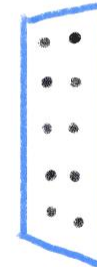
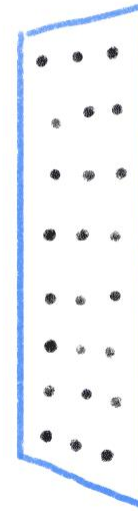
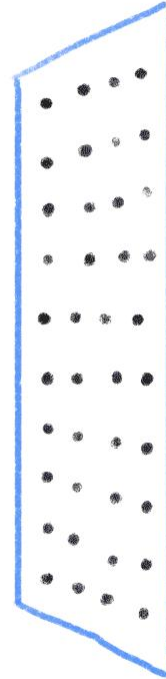
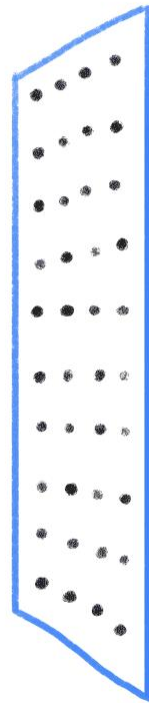
***Perceptron is a single layer neural network and  
a multi-layer perceptron is called Neural Networks.***

CAT



(LBELED)  
PHOTOS

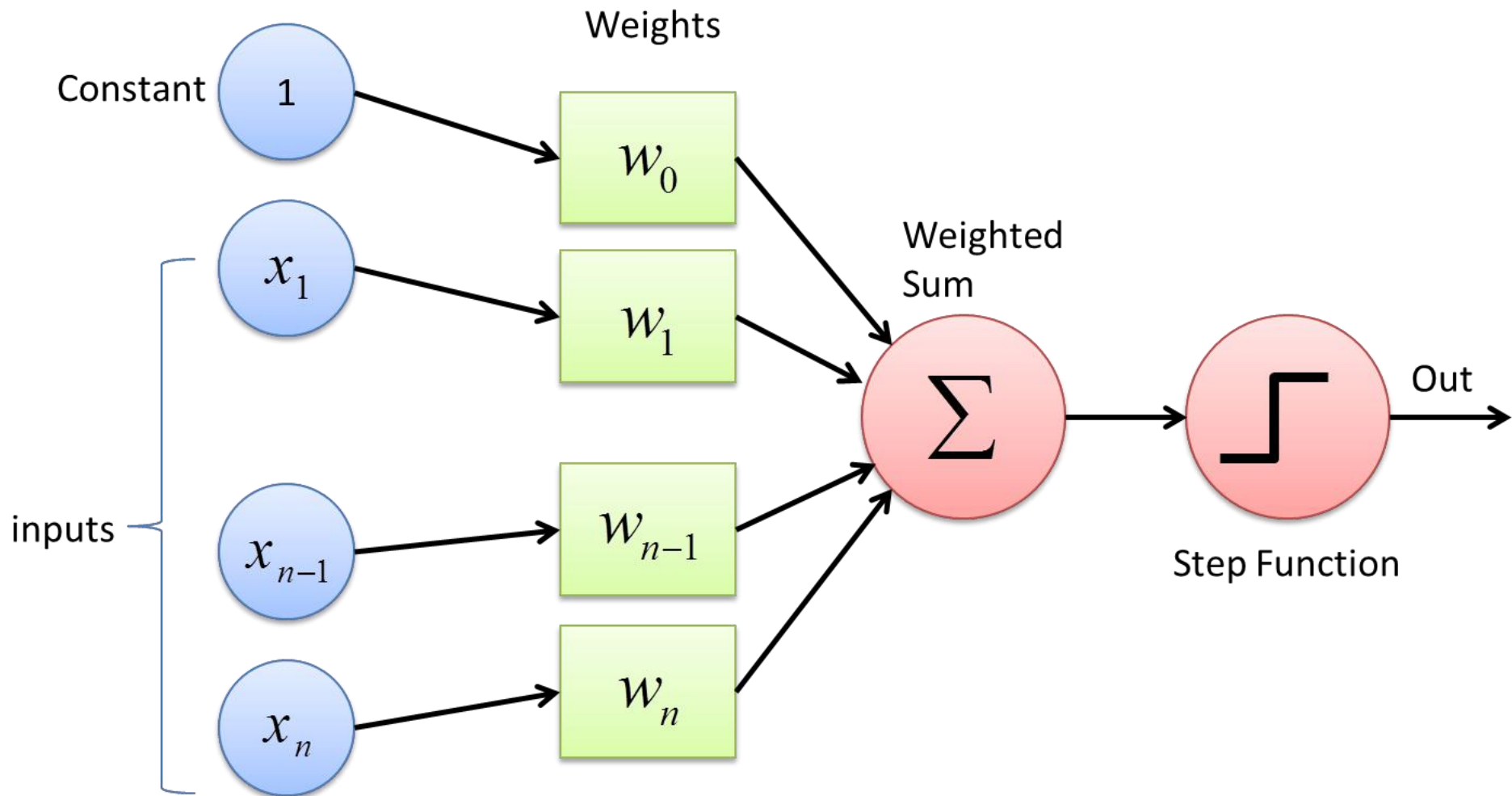
DOG



OUTPUT

# The perceptron consists of 4 parts

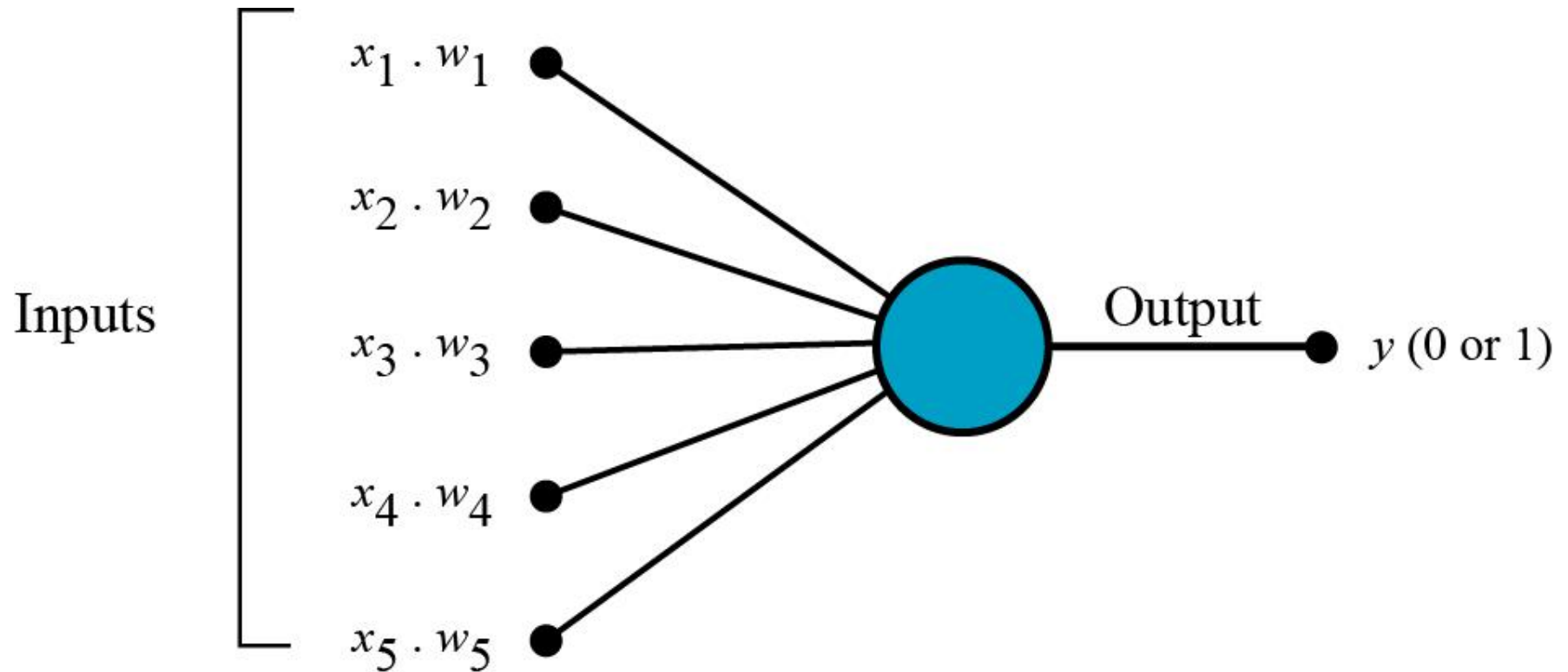
- Input values or One input layer
- Weights and Bias
- Net sum
- Activation Function



# The perceptron works on these simple steps



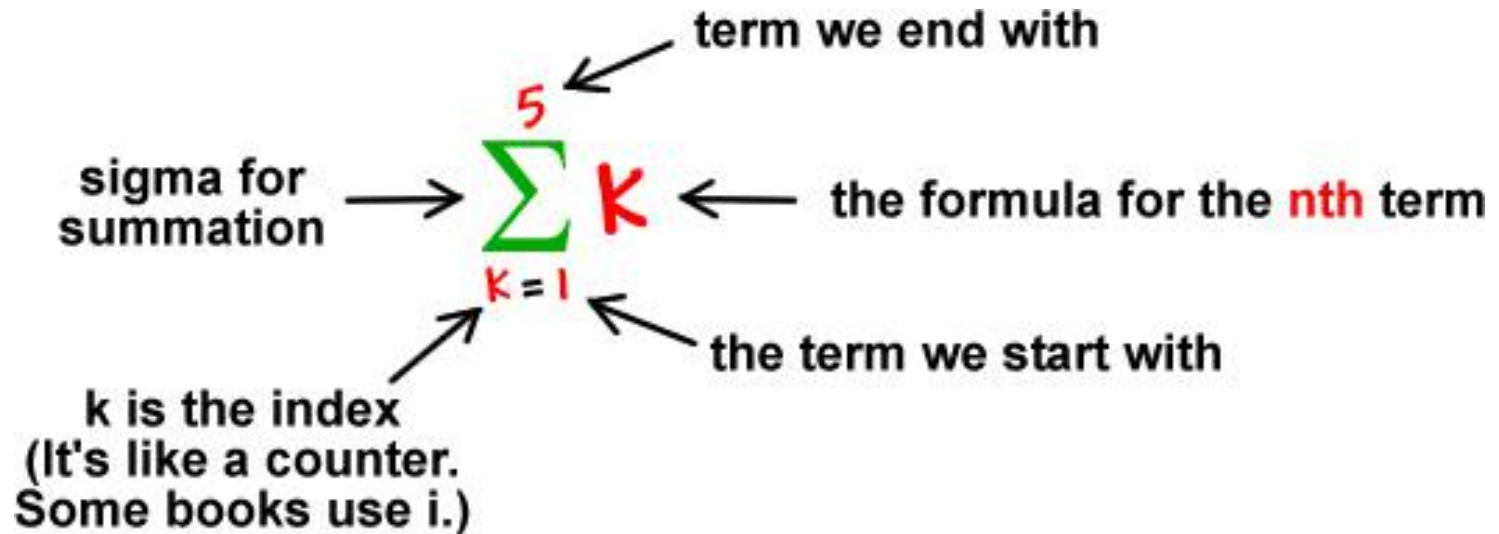
a. All the inputs  $x$  are multiplied with their weights  $w$ . Let's call it



Multiplying inputs with weights for 5 inputs

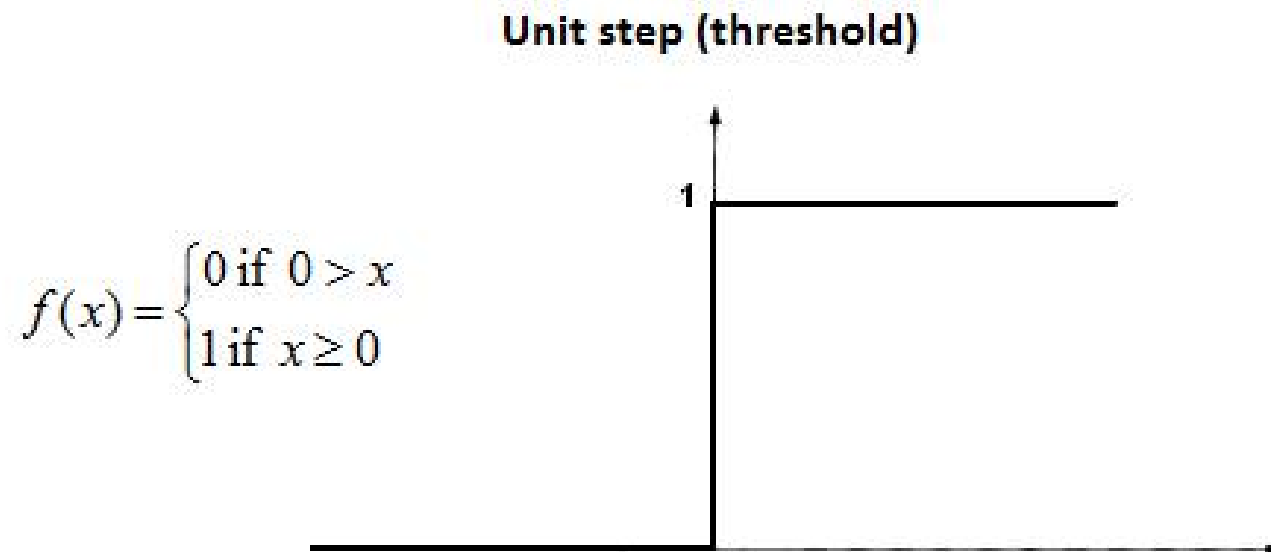
Krishna Modi, DCS

b. **Add** all the multiplied values and call them **Weighted Sum**.



c. Apply that weighted sum to the correct Activation Function.

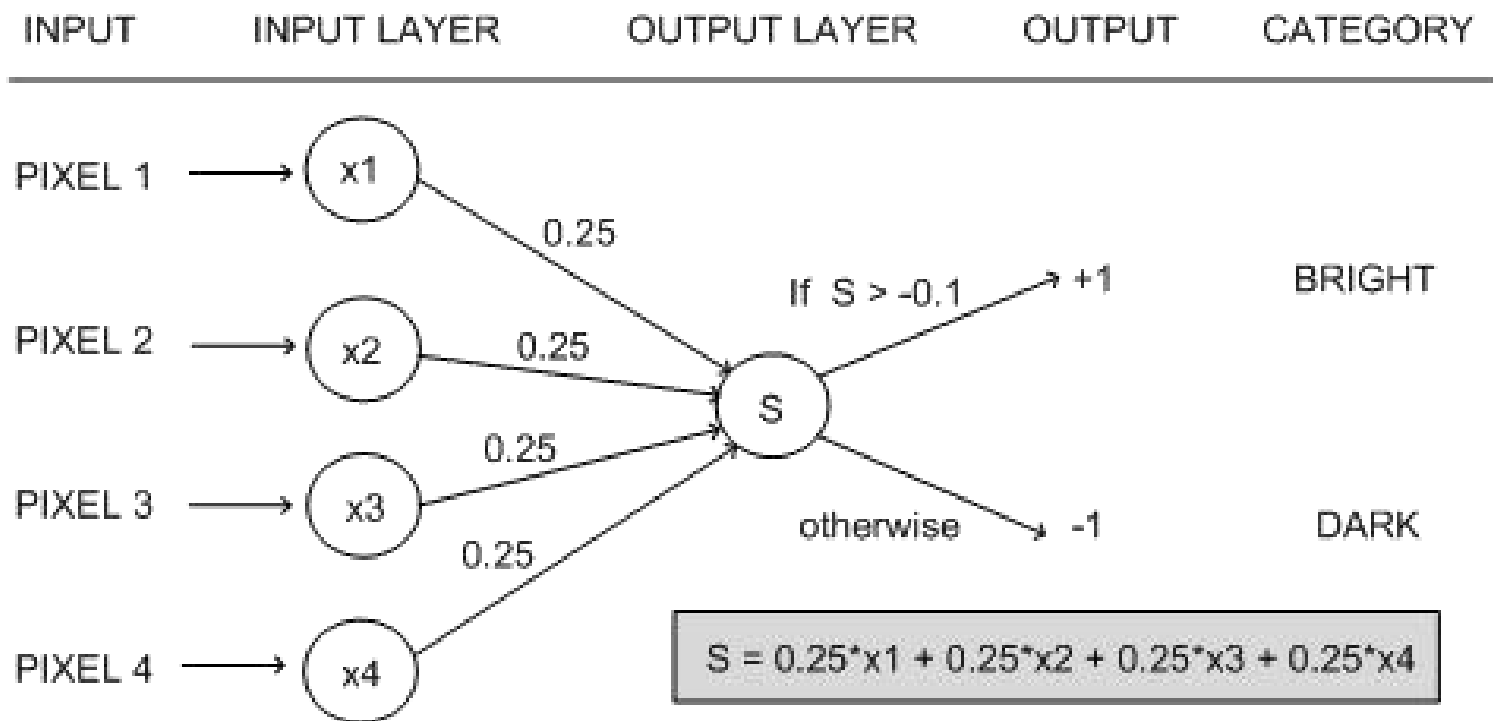
For Example : Unit Step Activation Function.



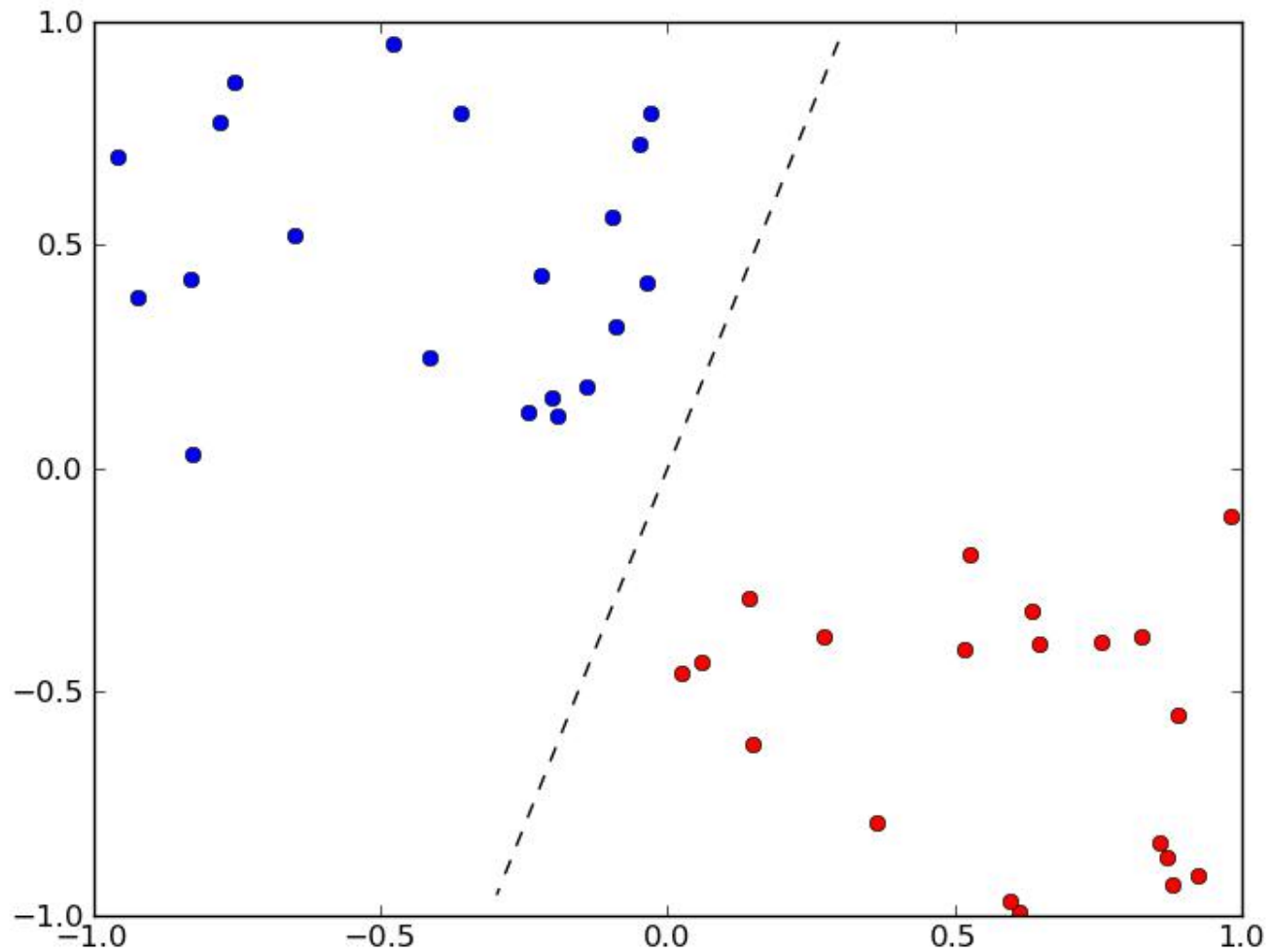
## Why do we need Weights and Bias?

Weights shows the strength of the particular node.

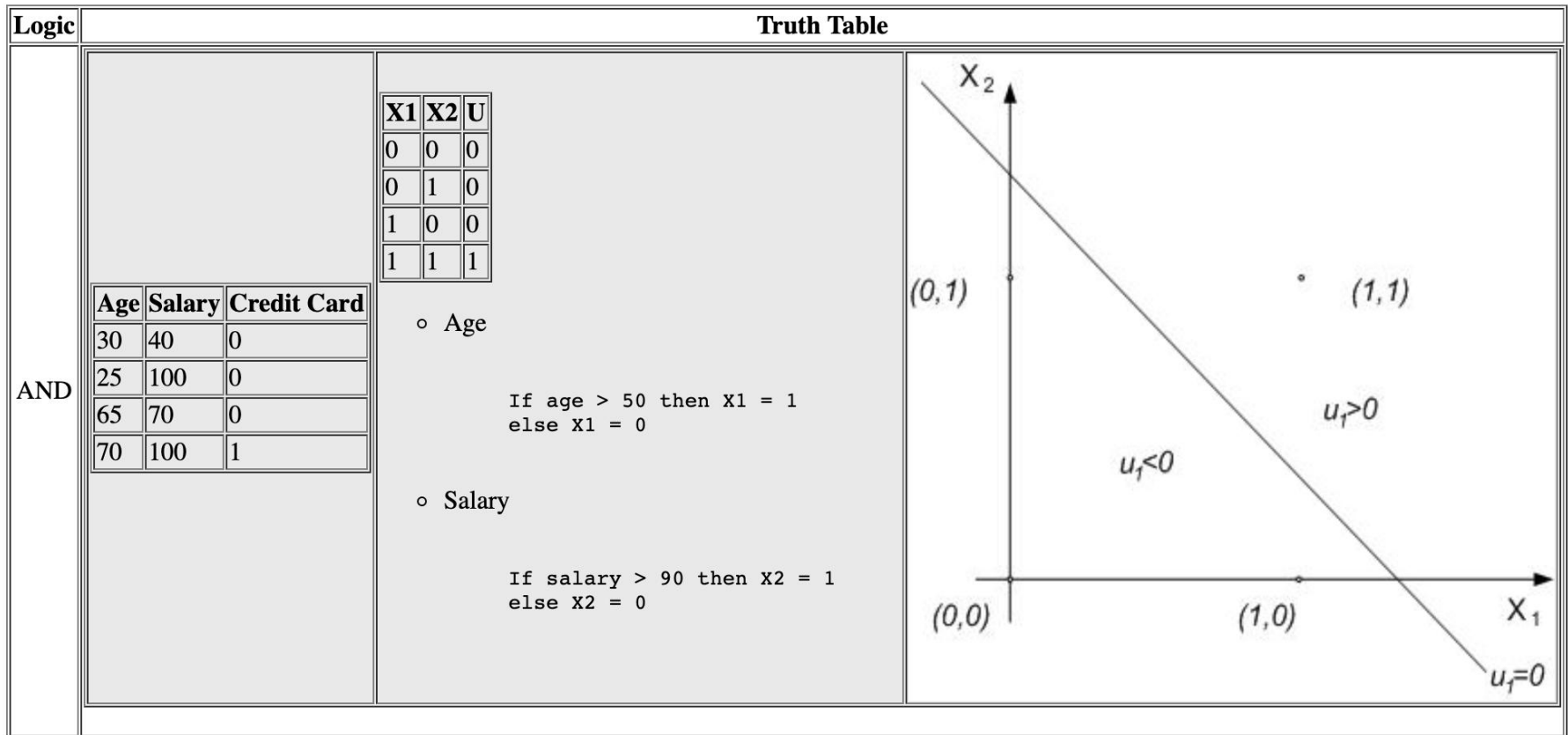
A bias value allows you to shift the activation function curve up or down.



Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.



# Single Layer Perceptron



## Single Perceptron Drawbacks

The single perceptron approach to deep learning has one major drawback: it can only learn linearly separable functions.

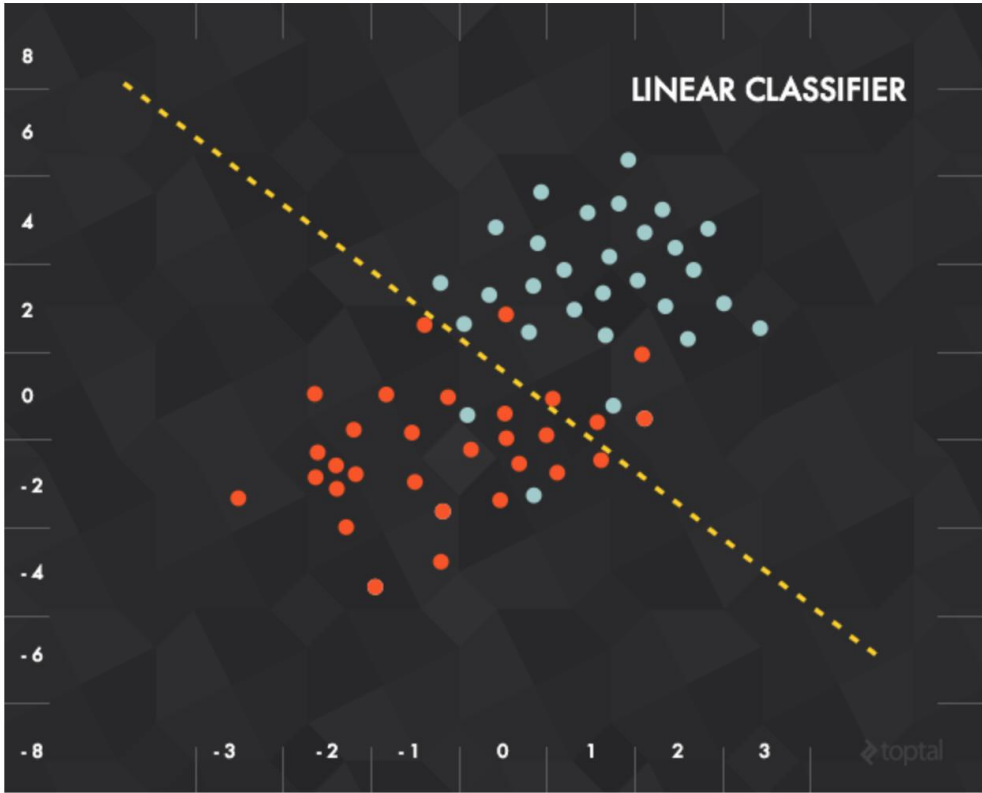
- Take XOR, a relatively simple function, and notice that it can't be classified by a linear separator (notice the failed attempt, below):
- To address this problem, we'll need to use a multilayer perceptron, also known as feedforward neural network
  - We'll compose a bunch of these perceptrons together to create a more powerful mechanism for learning.

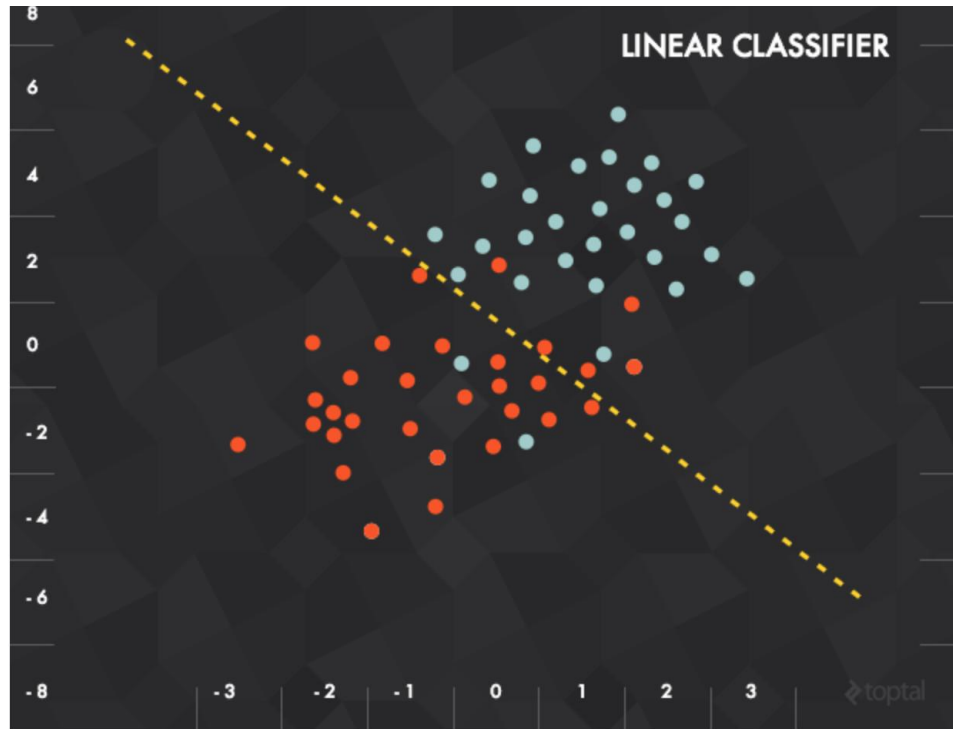
# Perceptrons: Early Deep Learning Algorithms

One of the earliest supervised training algorithms is that of the perceptron, a basic neural network building block.

Say we have  $n$  points in the plane, labeled '0' and '1'. We're given a new point and we want to guess its label (this is akin to the "Dog" and "Not dog" scenario above). How do we do it?

One approach might be to look at the closest neighbor and return that point's label. But a slightly more intelligent way of going about it would be to pick a line that best separates the labeled data and use that as your classifier.





In this case, each piece of input data would be represented as a vector  $\mathbf{x} = (x_1, x_2)$  and our function would be something like “0” if below the line, “1” if above”.

To represent this mathematically, let our separator be defined by a vector of weights  $\mathbf{w}$  and a vertical offset (or bias)  $b$ . Then, our function would combine the inputs and weights with a weighted sum transfer function:

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b$$

The result of this transfer function would then be fed into an activation function to produce a labeling. In the example above, our activation function was a threshold cutoff (e.g., 1 if greater than some value):

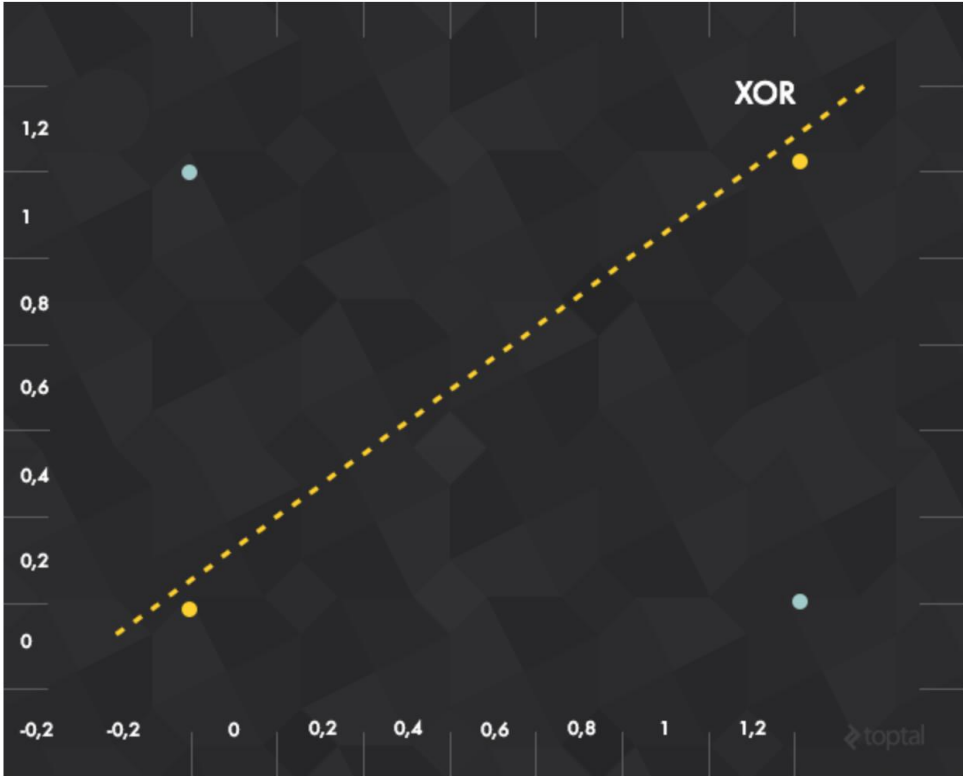
$$h(\mathbf{x}) = \begin{cases} 1 & : \text{ if } f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & : \text{ otherwise} \end{cases}$$

## Training the Perceptron

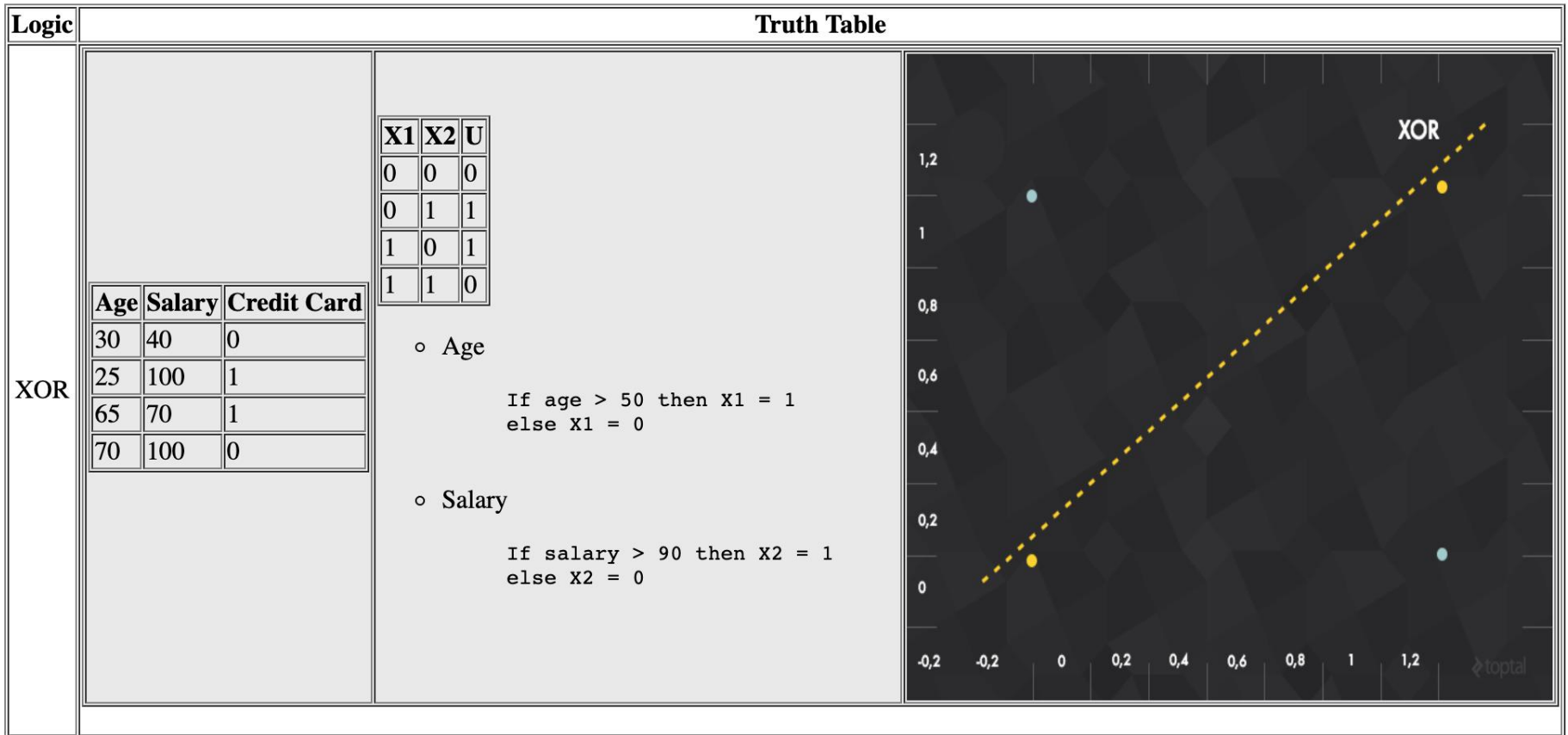
The training of the perceptron consists of feeding it multiple training samples and calculating the output for each of them. After each sample, the weights  $w$  are adjusted in such a way so as to minimize the *output error*, defined as the difference between the *desired* (target) and the *actual* outputs. There are other error functions, like the [mean square error](#), but the basic principle of training remains the same.

## Single Perceptron Drawbacks

The single perceptron approach to deep learning has one major drawback: it can only learn [linearly separable functions](#). How major is this drawback? Take XOR, a relatively simple function, and notice that it can't be classified by a linear separator (notice the failed attempt, below):



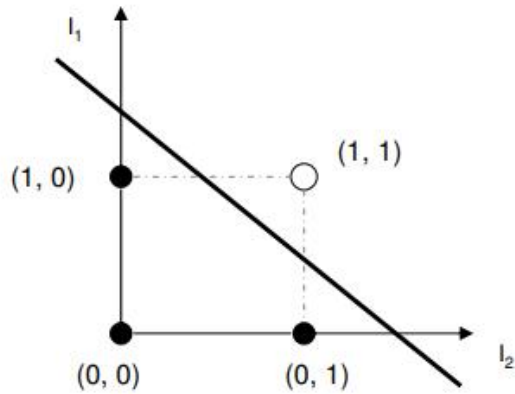
# Multilayer Perceptron



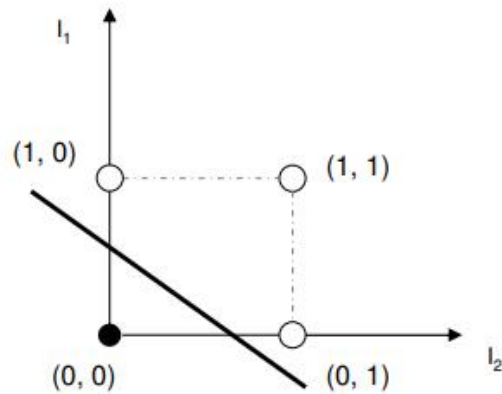
## Multilayer Perceptron

The multilayer perceptron consists of three or more layers (an input and an output layer with one or more hidden nonlinearly-activating nodes and is thus considered a deep neural network.

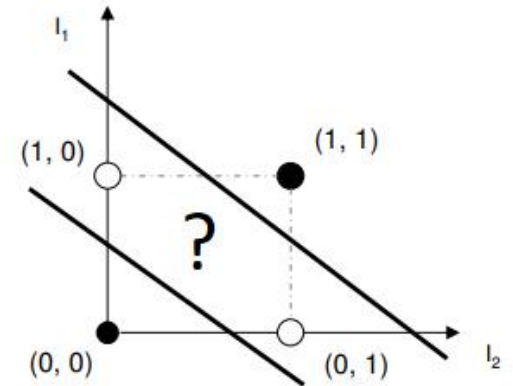
AND		
$I_1$	$I_2$	out
0	0	0
0	1	0
1	0	0
1	1	1



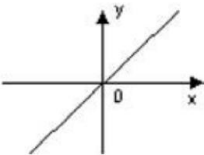
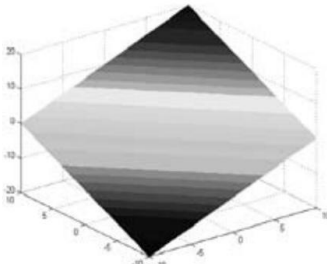
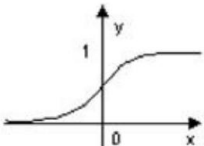
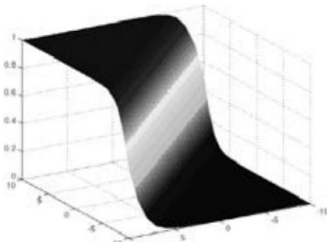
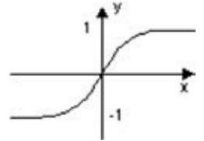
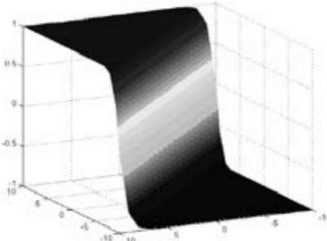
OR		
$I_1$	$I_2$	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
$I_1$	$I_2$	out
0	0	0
0	1	1
1	0	1
1	1	0



# Activation Function

Activation Function	Mathematical Equation	2D Graphical Representation	3D Graphical Representation
Linear	$y = x$		
Sigmoid (logistic)	$y = \frac{1}{1 + e^{-x}}$		
Hyperbolic tangent	$y = \frac{1 - e^{-2x}}{1 + e^{2x}}$		

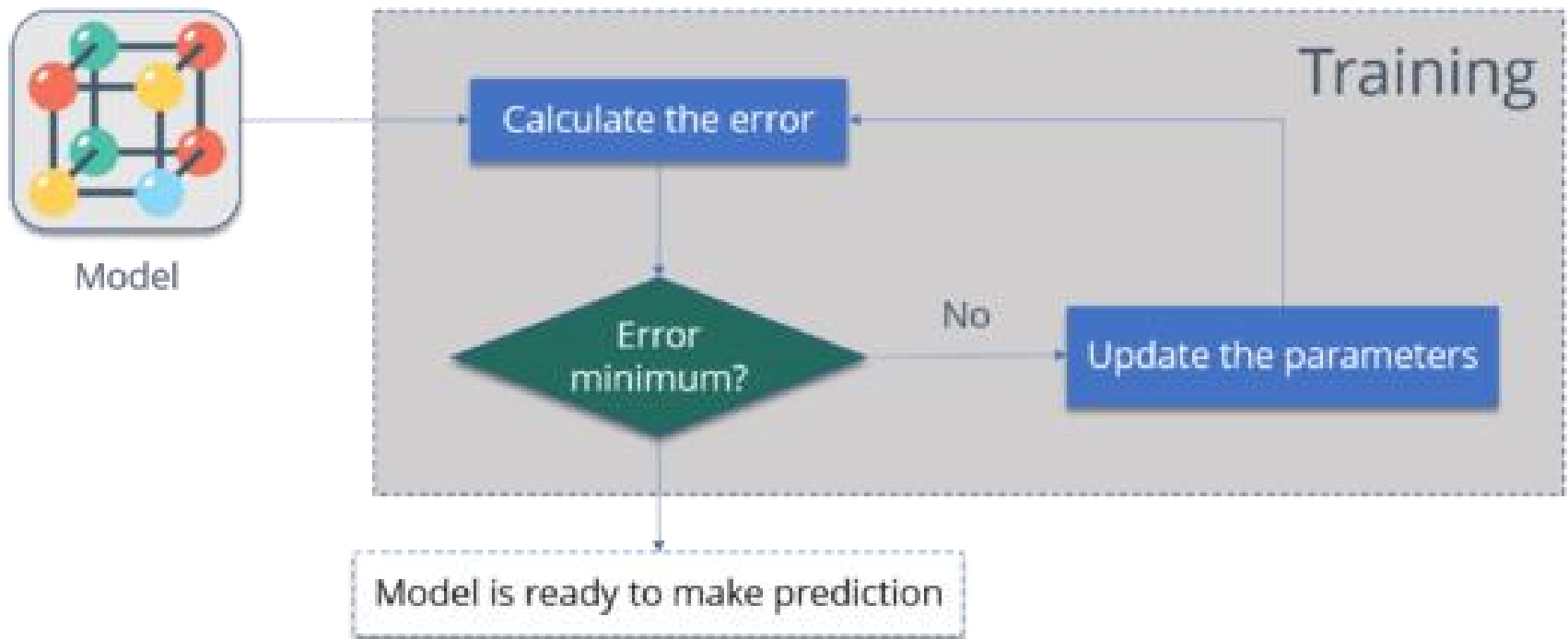
# Back Propagation Algorithm

- Algorithm used to efficiently train Artificial Neural Network.

# Why we need Back Propagation

- While designing a Neural Network, in the beginning, we initialize weights with some random values or any variable for that fact.
- Now obviously, it's not necessary that whatever weight values we have selected will be correct, or it fits our model the best.
- we have selected some weight values in the beginning, but our model output is way different than our actual output i.e. the error value is huge.
- Basically, what we need to do, we need to somehow explain the model to change the parameters (weights), such that error becomes minimum.

# Why we need Back Propagation



# Why we need Back Propagation

- **Calculate the error** – How far is your model output from the actual output.
- **Minimum Error** – Check whether the error is minimized or not.
- **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.
- **Model is ready to make a prediction** – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

# Example

input | output

-----

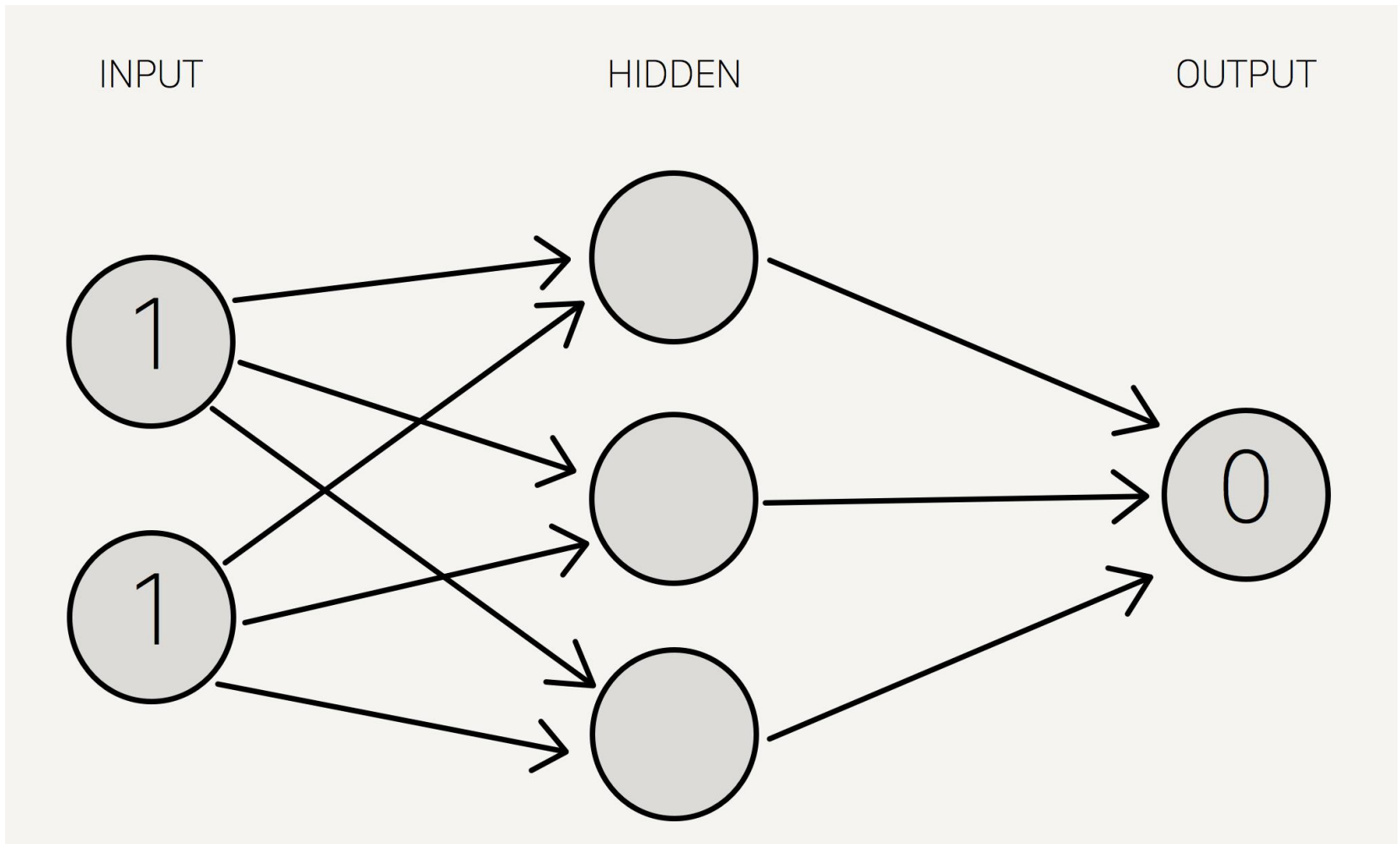
0, 0 | 0

0, 1 | 1

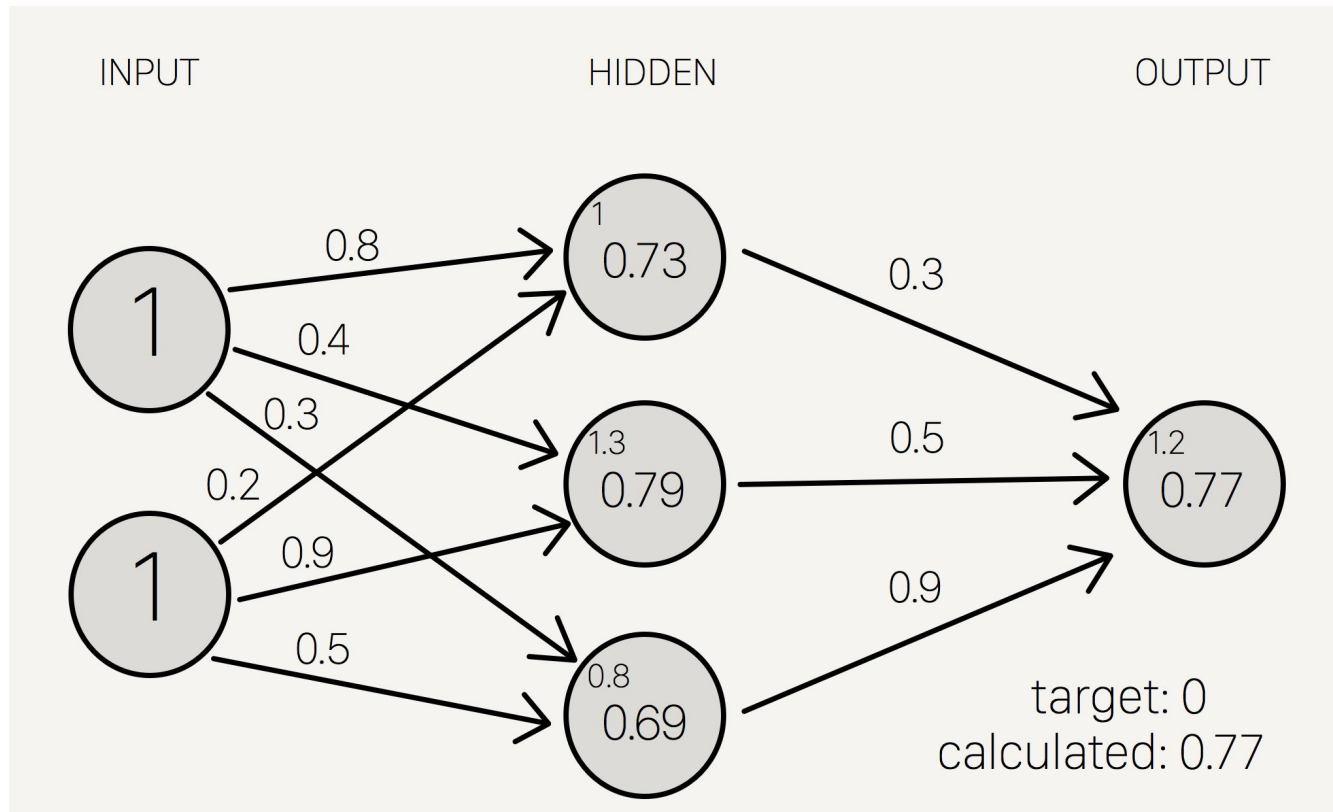
1, 0 | 1

1, 1 | 0

# Actual output



# Predicted output



# Example

- Target = 0
- Output sum margin of error = *target – calculated*
- target - calculated = -0.77

# Example

Delta output sum =  $S'(\text{sum}) * (\text{output sum margin of error})$

Here  $S'(\text{sum})$  is derivative of sigmoid function.

Delta output sum =  $S'(1.2) * (-0.77)$

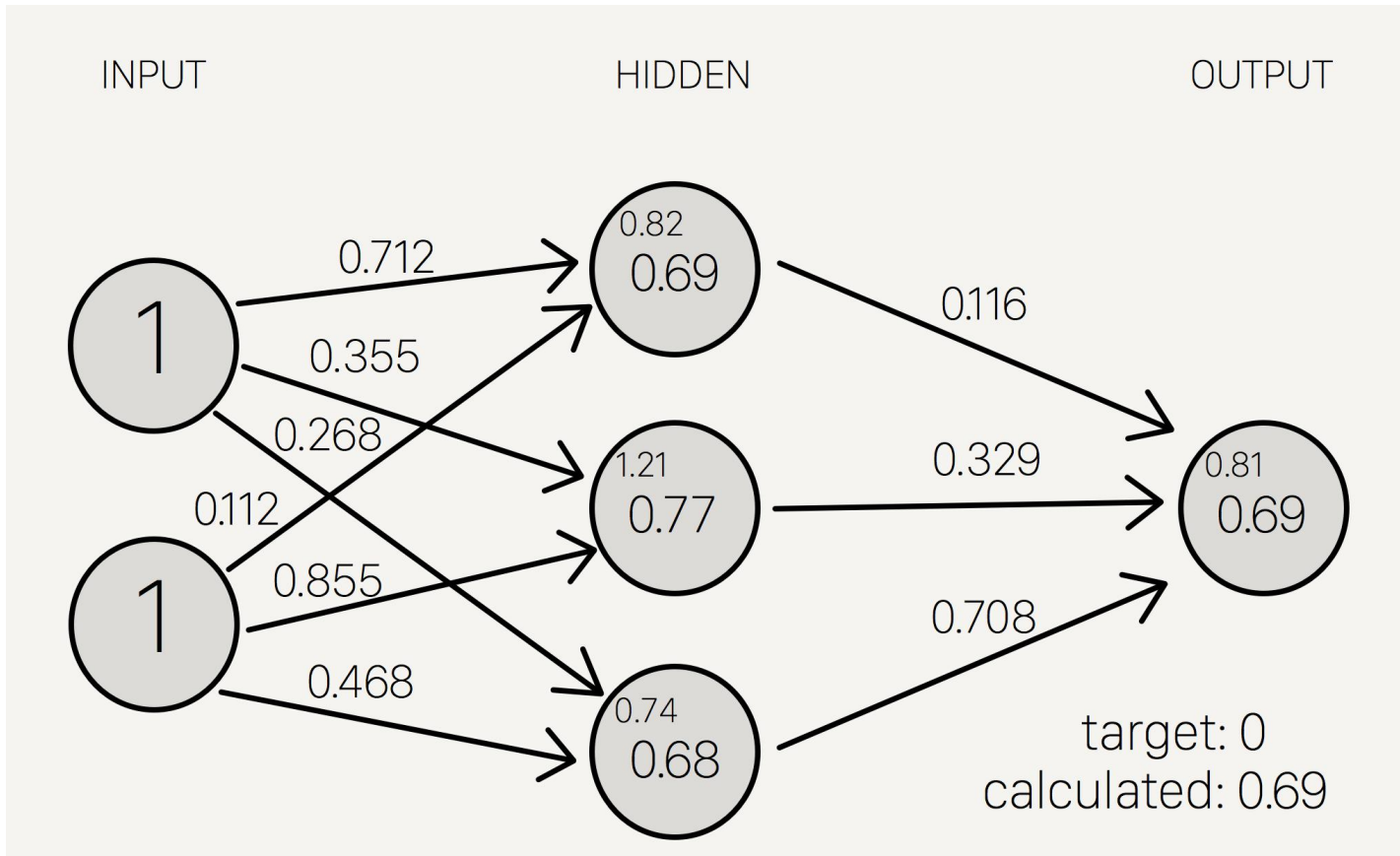
$S'(1.2) = S(1.2) * (1 - S(1.2)) =$   
 $0.77 * (1 - 0.77) = 0.1771$

Delta output sum = -0.13

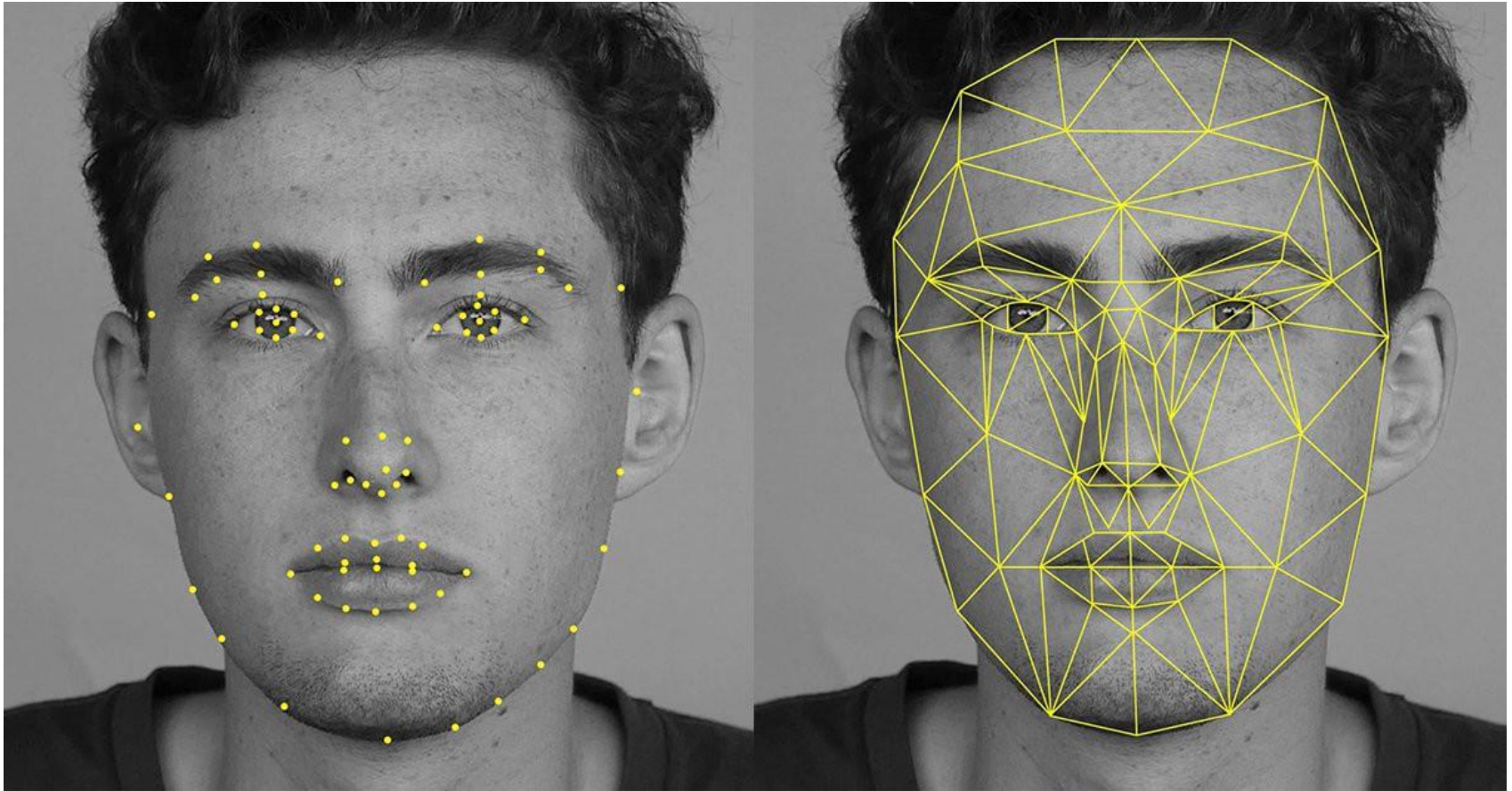
- hidden result 1 = 0.73
- Delta weights = delta output sum / hidden layer results
- Delta weights =  $-0.13 / 0.73 = -0.18$

Old weight : 0.3

New Weight :  $0.3 - 0.18 = 0.12$



# Face Recognition Method



# How to give input?

- we could preprocess the image to extract edges, regions of uniform intensity, or other local image features, then input these features to the network.
- All inputs were linearly scaled to range from 0 to 1 so that network inputs would have values in the same interval

- <http://www.cs.cmu.edu/-tomlmlbook.html>
- All image data and code used to produce the examples described in above site , along with complete documentation on how to use the code

# Case Based Reasoning

Solves new problems by adapting solutions that were used to solve old problems.

# Introduction

- In **case-based reasoning**, the training examples - the cases - are stored and accessed to solve a new problem.
- To get a prediction for a new example, those cases that are similar, or close to, the new example are used to predict the value of the target features of the new example.
- Case-based reasoning is a recent approach to problemsolving and learning

# What is Case Based reasoning?

- A methodology to model human reasoning.
- A methodology for building intelligent systems.
- CBR:
  - Store previous experience (cases) in memory
  - Solve new problems by
    - Retrieving**
    - Reusing**
    - Storing** new experience in memory, i.e. learn

# A Simple Example of case Diagnosis of Car Faults :

- Given: Symptoms  
e.g. engine doesn't start  
and measured values  
e.g. battery voltage = 6.3V
- Goal: Find cause for fault  
e.g. dead battery  
and repair strategy

# What is Case Based reasoning?

- Case-based reasoning can be seen as a cycle of the following four tasks.
- **Retrieve**: Given a new case, retrieve similar cases from the case base.
- **Reuse**: Adapt the retrieved cases to fit to the new case.
- **Revise**: Evaluate the solution and revise it based on how well it works.
- **Retain**: Decide whether to retain this new case in the case base.

# Example case

## Case 1

### Problem & Features

- Problem: Front light not working
- Car: VW Golf, 2.0L
- Year: 1999
- Battery voltage: 13.6V
- State of lights: OK
- State of light switch: OK

### Solution

- Diagnosis: Front light fuse defect
- Repair: Replace front light fuse

## Case 2

### Problem & Features

- Problem: Front light not working
- Car: Passat
- Year: 2000
- Battery voltage: 12.6V
- State of lights: surface damaged
- State of light switch: OK

### Solution

- Diagnosis: Bulb defect
- Repair: Replace front light

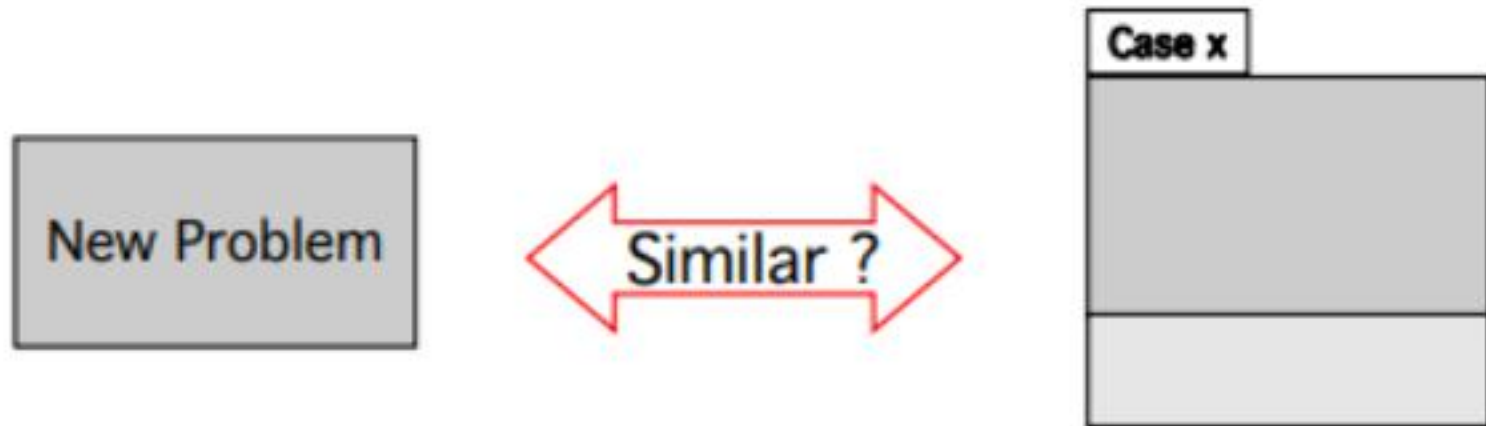
# New Problem

- Observations define a new problem
- Not all feature values may be known
- New problem = case without solution

## Problem & Features

- Problem: Brake light not working
- Car: Passat V6
- Year: 2002
- Battery voltage: 12.9V
- State of lights: OK
- State of light switch: ?

# Find similar case



- Compare similarity of each feature
- But some features may be more important

# Compare with Case 1

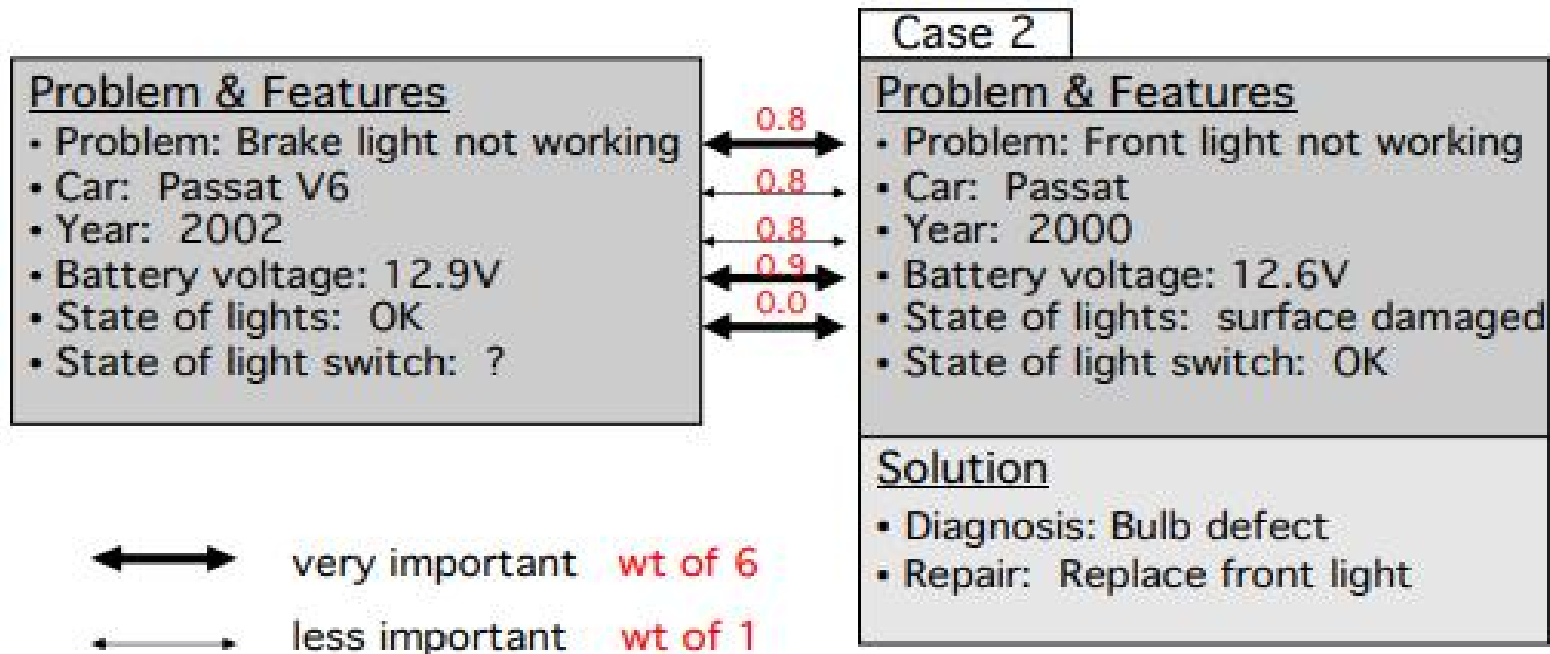
Case 1	
<u>Problem &amp; Features</u> <ul style="list-style-type: none"><li>• Problem: Brake light not working</li><li>• Car: Passat V6</li><li>• Year: 2002</li><li>• Battery voltage: 12.9V</li><li>• State of lights: OK</li><li>• State of light switch: ?</li></ul>	<u>Problem &amp; Features</u> <ul style="list-style-type: none"><li>• Problem: Front light not working</li><li>• Car: VW Golf, 2.0L</li><li>• Year: 1999</li><li>• Battery voltage: 13.6V</li><li>• State of lights: OK</li><li>• State of light switch: OK</li></ul>
<u>Solution</u> <ul style="list-style-type: none"><li>• Diagnosis: Front light fuse defect</li><li>• Repair: Replace front light fuse</li></ul>	

↔ very important wt of 6  
→ less important wt of 1

Similarity by wted avg =  $1/20 (6*0.8 + 1*0.4 + 1*0.7 + 6*0.9 + 6*1.0) = 0.87$

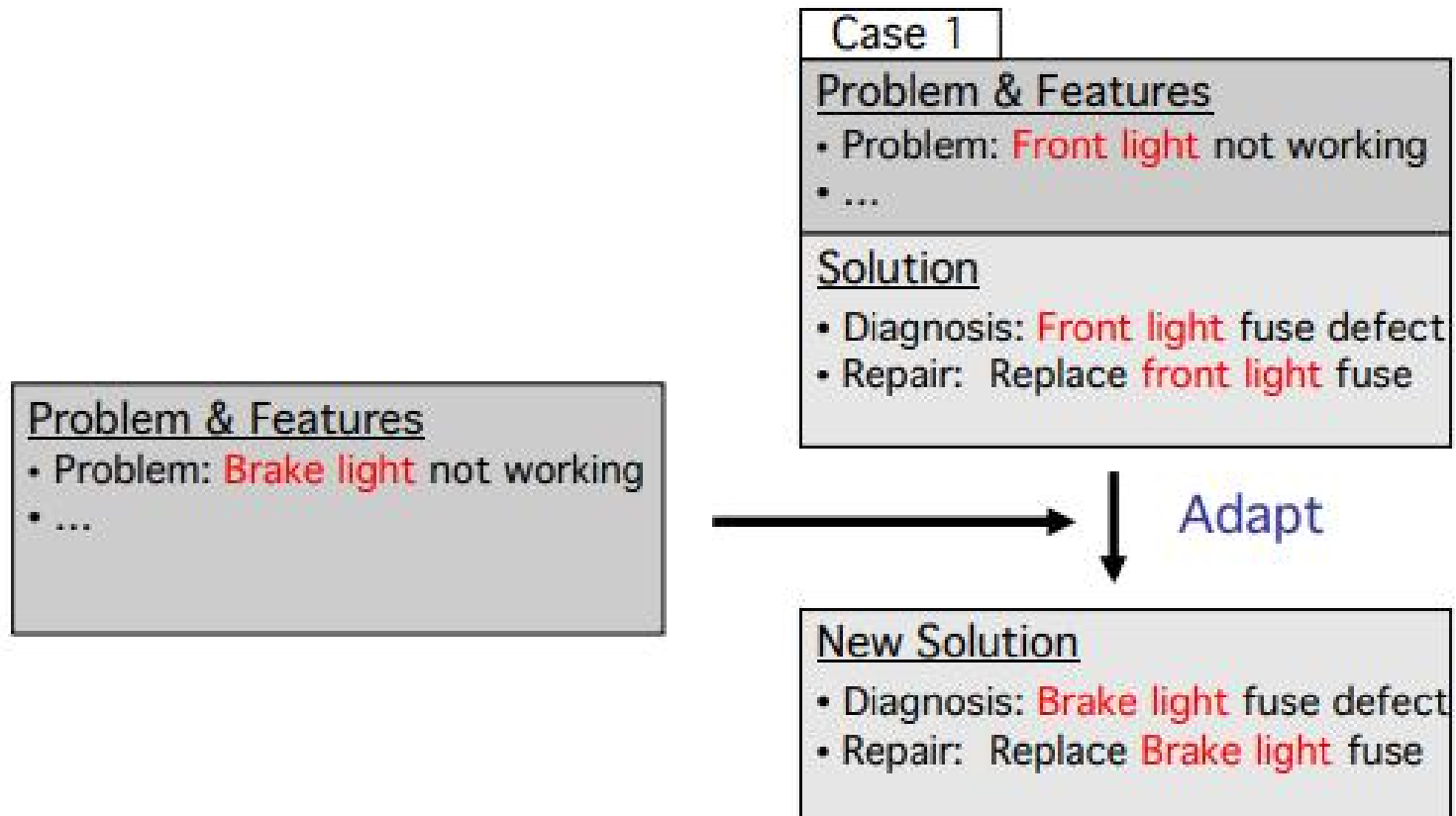
# Compare with Case 2

## Compare with Case 2



Similarity by wted avg =  $1/20 (6*0.8 + 1*0.8 + 1*0.8 + 6*0.9 + 6*0.0) = 0.59$  गशते जगत्

# Reuse case 1



# Store new case

## Case 3

### Problem & Features

- Problem: Brake light not working
- Car: Passat V6
- Year: 2002
- Battery voltage: 12.9V
- State of lights: OK
- State of light switch: OK

### Solution

- Diagnosis: Brake light fuse defect
- Repair: Replace brake light

# References

<https://courses.csail.mit.edu/6.871/lectures06/Lect17CBR.pdf>

# References

- <https://www.edureka.co/blog/backpropagation/>
- <https://stevenmiller888.github.io/mini-how-to-build-a-neural-network/>

# Thank You

Krishna Modi, DCS