

# RECAP

Topics Covered:

- Cost Function
- Gradient Descent
- Derivatives
- Implementation of logistic regression

# Today we will learn

- Neural Network representation and architecture
- Weight Initialisation and batch normalization
- Activation Functions
  - Definition
  - Types of activation function
- Optimisers

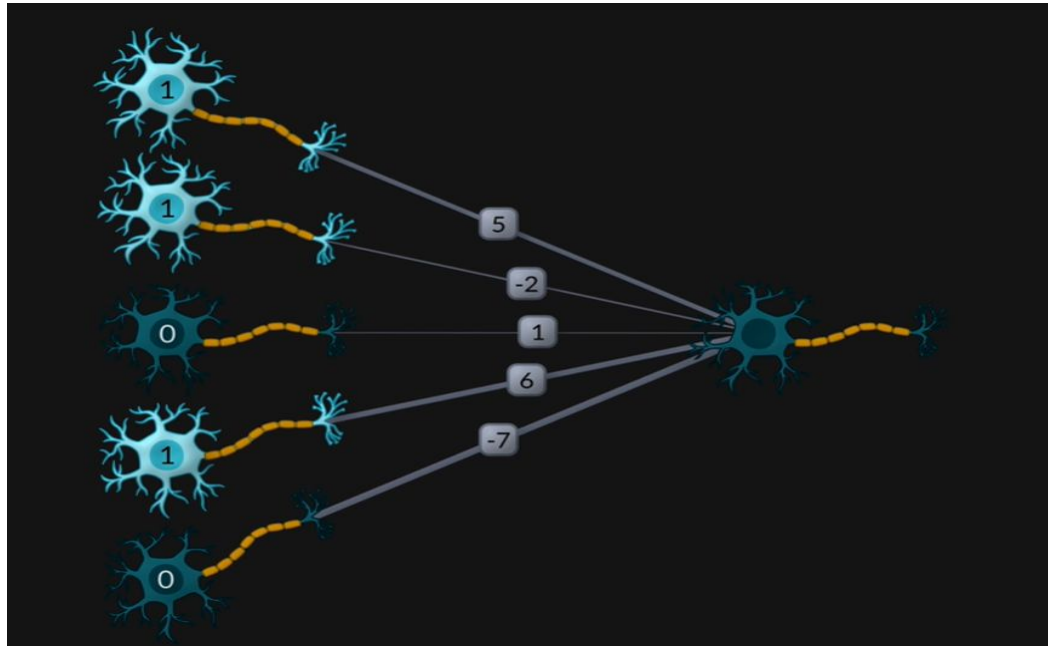
# History of AI - year 1956

When a psychologist made perceptron  
To mimic how neurons fire in our brain

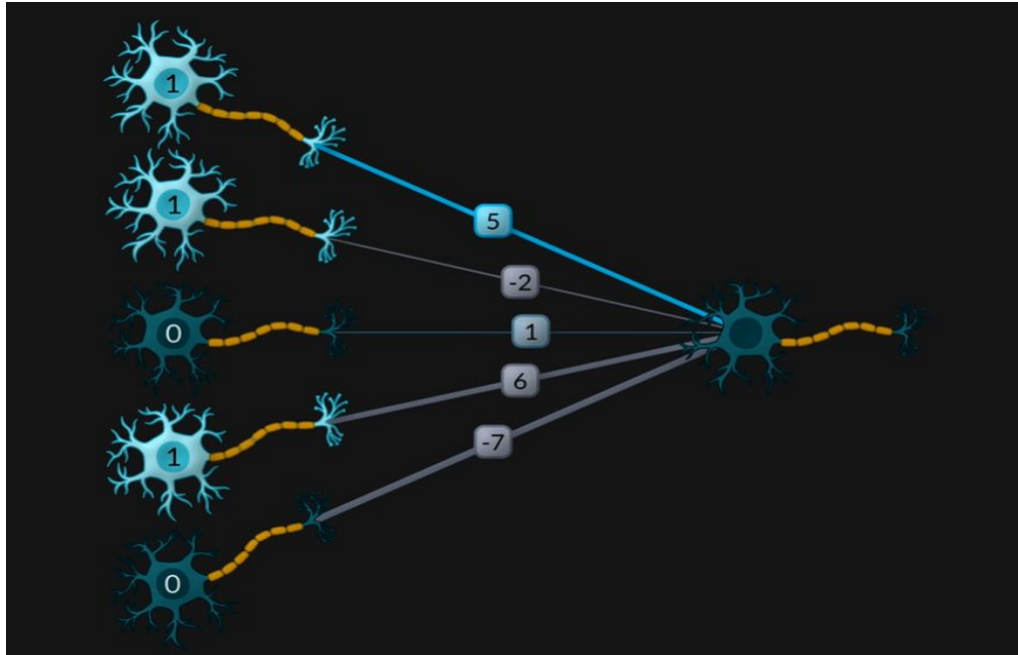


Frank Rose Frank Rosenblatt, built the perceptron.

Similar to human brain in a neural network input of one neuron is actually the output of other neuron



Each neuron also has a weight associated with the connection with other neuron



If the sum of multiplication of neuron and connection weight  $>$  threshold  
the neuron fires or passes the information further

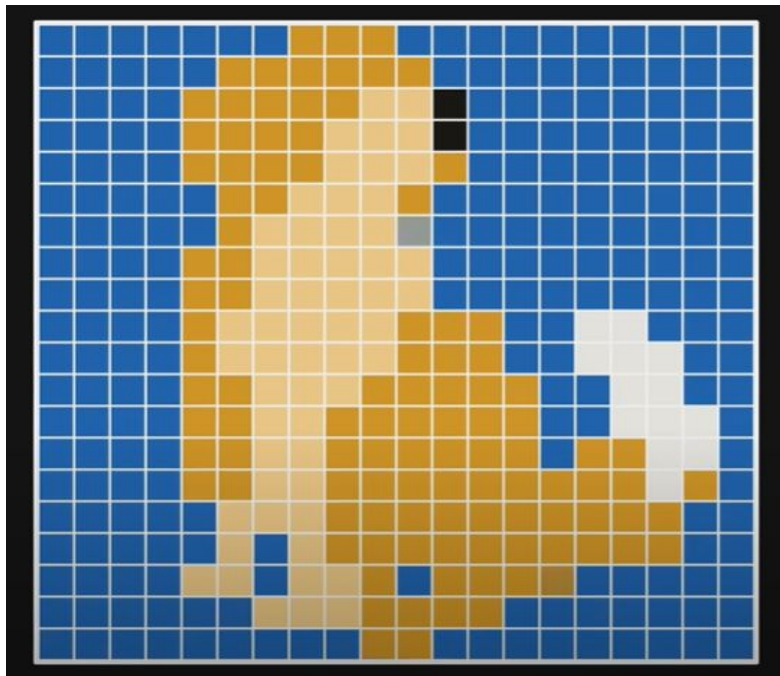
$$1 \times 5 + 1 \times -2 + 0 \times 1 + 1 \times 6 + 0 \times -7 = 9$$

$$9 > 6$$

# How a neural network works to decide if image is of dog or cat

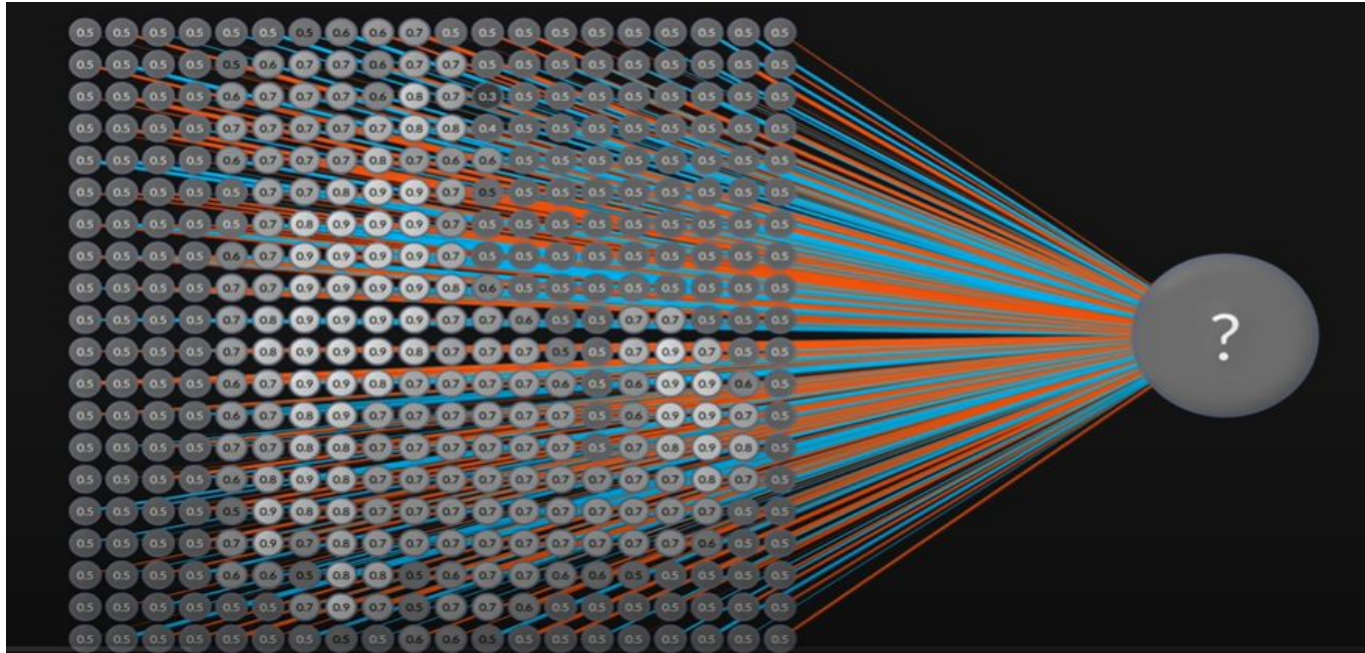


In general an image is arranged in a grid





Each pixel can be considered as a neuron with its value (between 0 and 1), wrt intensity at that pixel, and all neurons are connected to single output neuron

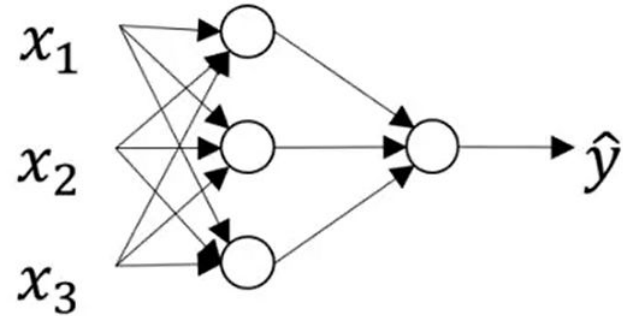
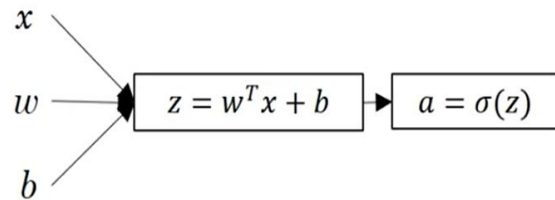
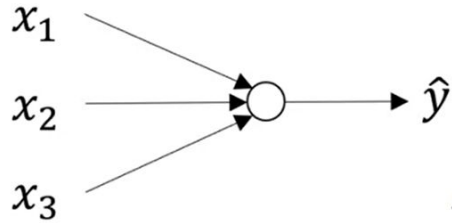


If the output (summation)  $>$  threshold then image belongs to class 1 else class 0

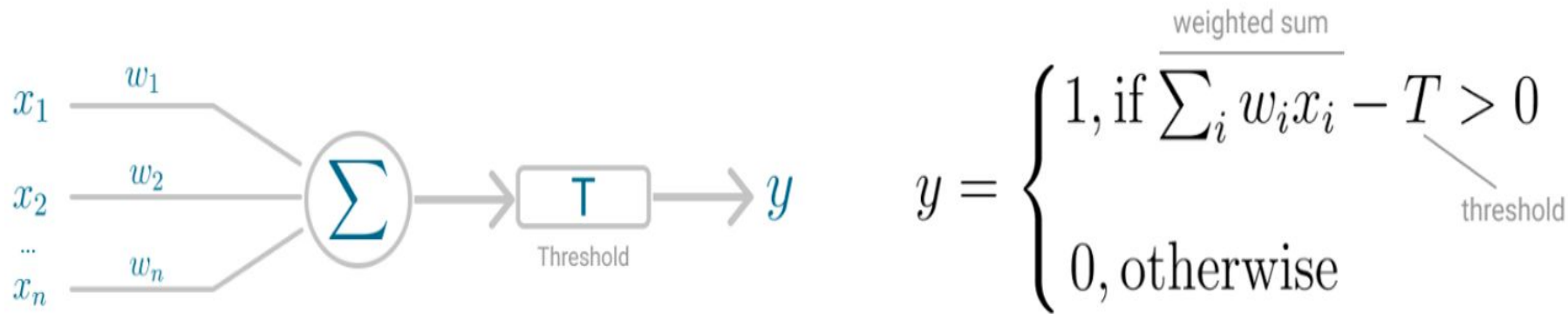
To achieve this the perceptron needs to be trained with training data images  
Further, each pixel has its own weight, usually initialising with 0

This explains a single layer perceptron

# NEURAL NETWORK REPRESENTATION



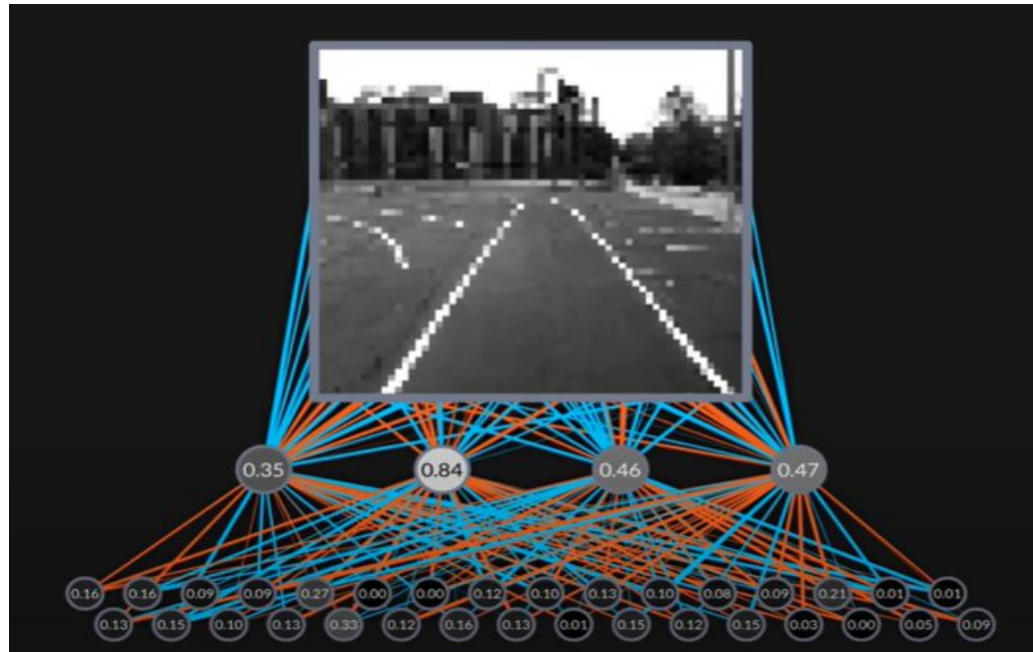
# NEURAL NETWORK REPRESENTATION



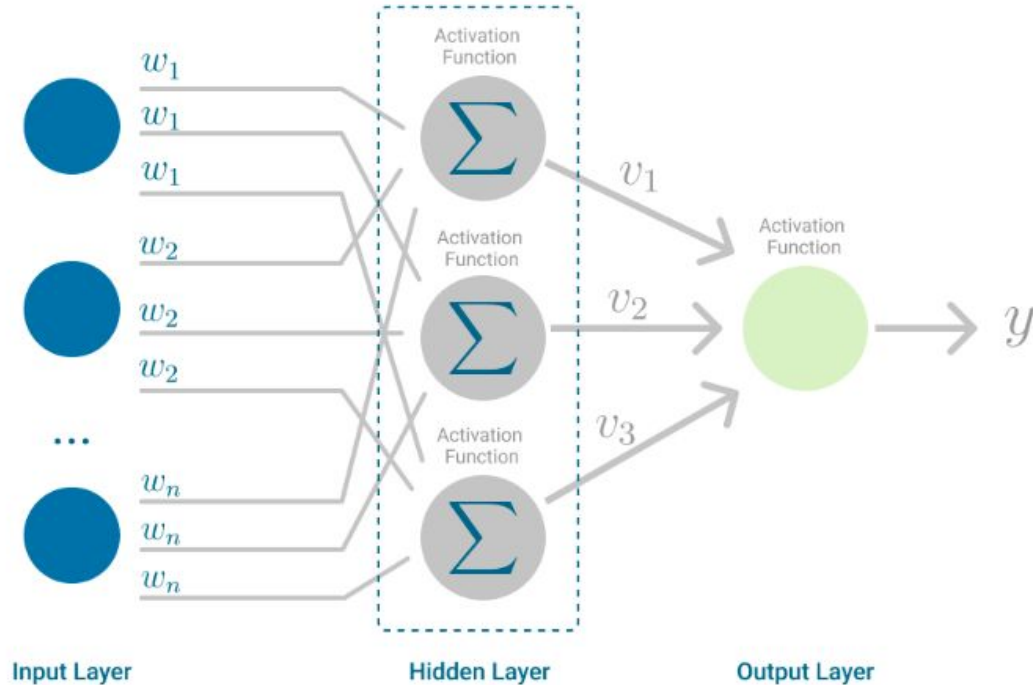
Reference:

Carolina Bento: <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>

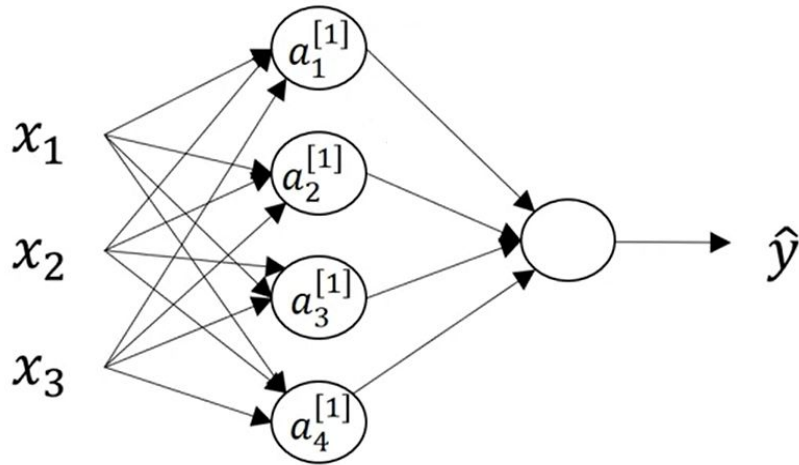
A multilayer perceptron looks something like this



A multilayer perceptron looks something like this



# NEURAL NETWORK REPRESENTATION



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

# Properties of MLP

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units
- Each unit is a perceptron
- Often include bias as an extra weight



## To improve the neural network Batch Normalization is used

- Before entering into Batch normalization let's understand the term "Normalization".
- Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.
- Generally, when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale. The reason we normalize is partly to ensure that our model can generalize appropriately.

# What is Batch Normalisation

Now coming back to Batch normalization, it is a process to make neural networks faster and more stable through adding extra layers in a deep neural network.

The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.

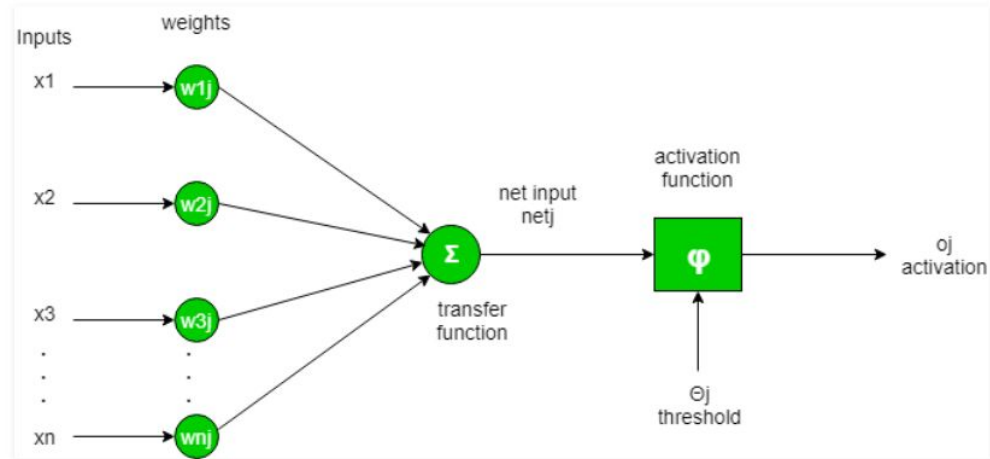
But what is the reason behind the term “Batch” in batch normalization?

A typical neural network is trained using a collected set of input data called batch.


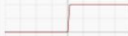







Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

# Activation function

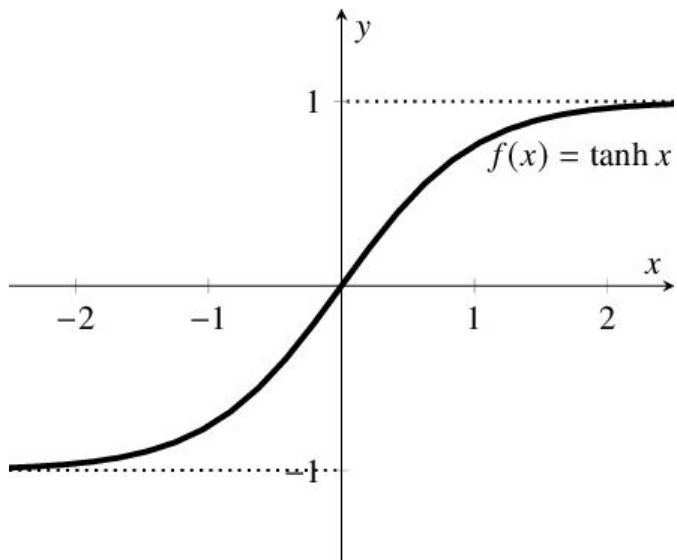
To put in simple terms, an artificial neuron calculates the 'weighted sum' of its inputs and adds a bias, as shown in the figure below by the net input.



# Types of activation function

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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

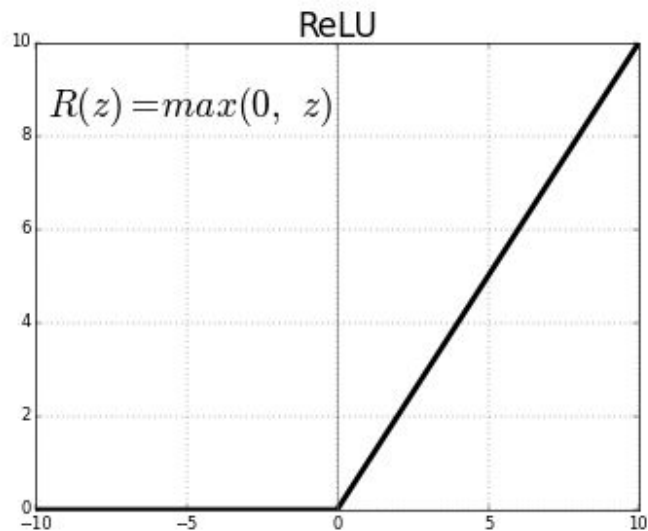
# ACTIVATION FUNCTIONS



TanH

$$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

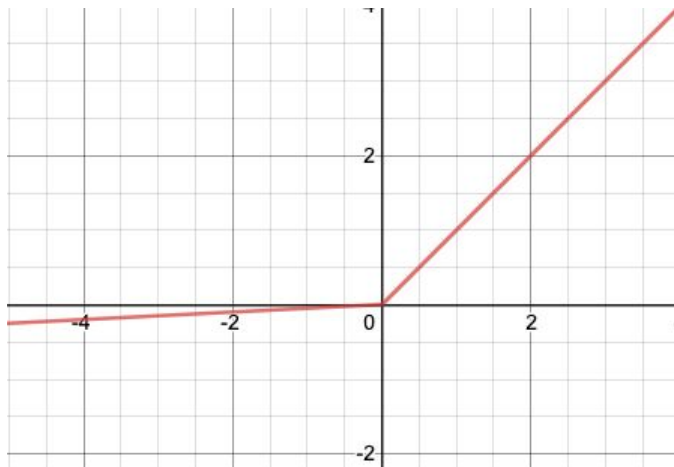
# ACTIVATION FUNCTIONS



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

# ACTIVATION FUNCTIONS

## Leaky ReLU



$$a = \text{relu}(z) = \max(0.01 * z, z)$$



# How do we improve MLP / NN

We use optimisation methods to arrive at the target or optimum weight value

# Gradient descent

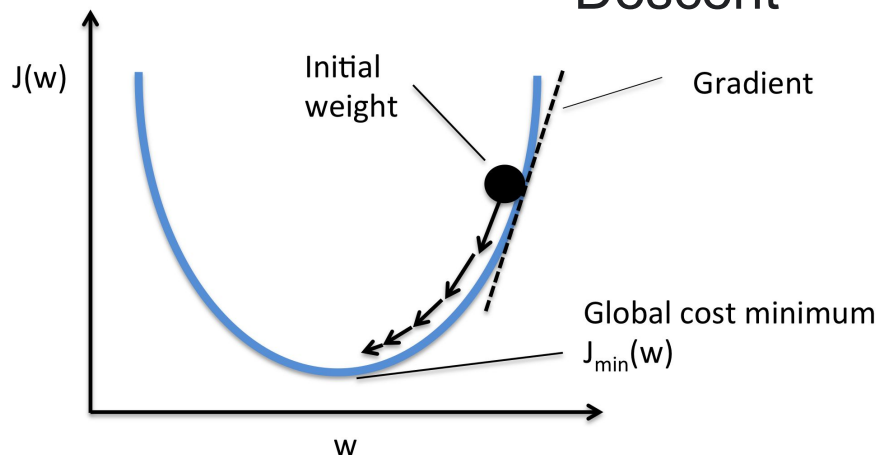
The gradient descent method is the most popular optimisation method.

The idea of this method is to update the variables iteratively in the (opposite) direction of the gradients of the objective function.

With every update, this method guides the model to find the target and gradually converge to the optimal value of the objective function.

# OPTIMIZERS

## Stochastic Gradient Descent



$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

# OPTIMIZERS

Ada

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t$$

# Neural Networks: Feedforward and Backpropagation & Optimization

Neural networks consists of neurons, connections between these neurons called weights and some biases connected to each neuron.

We distinguish between input, hidden and output layers, where we hope each layer helps us towards solving our problem.

To move forward through the network, called a forward pass, we iteratively use a formula to calculate each neuron in the next layer.

This takes us forward, until we get an output.



## Continued....

We measure how good this output, by a cost function and the result we wanted in the output layer

And we do this for every example

This one is commonly called mean squared error (MSE)

## Continued...

Given the first result, we go back and adjust the weights and biases, so that we optimize the cost function — called a backwards pass.

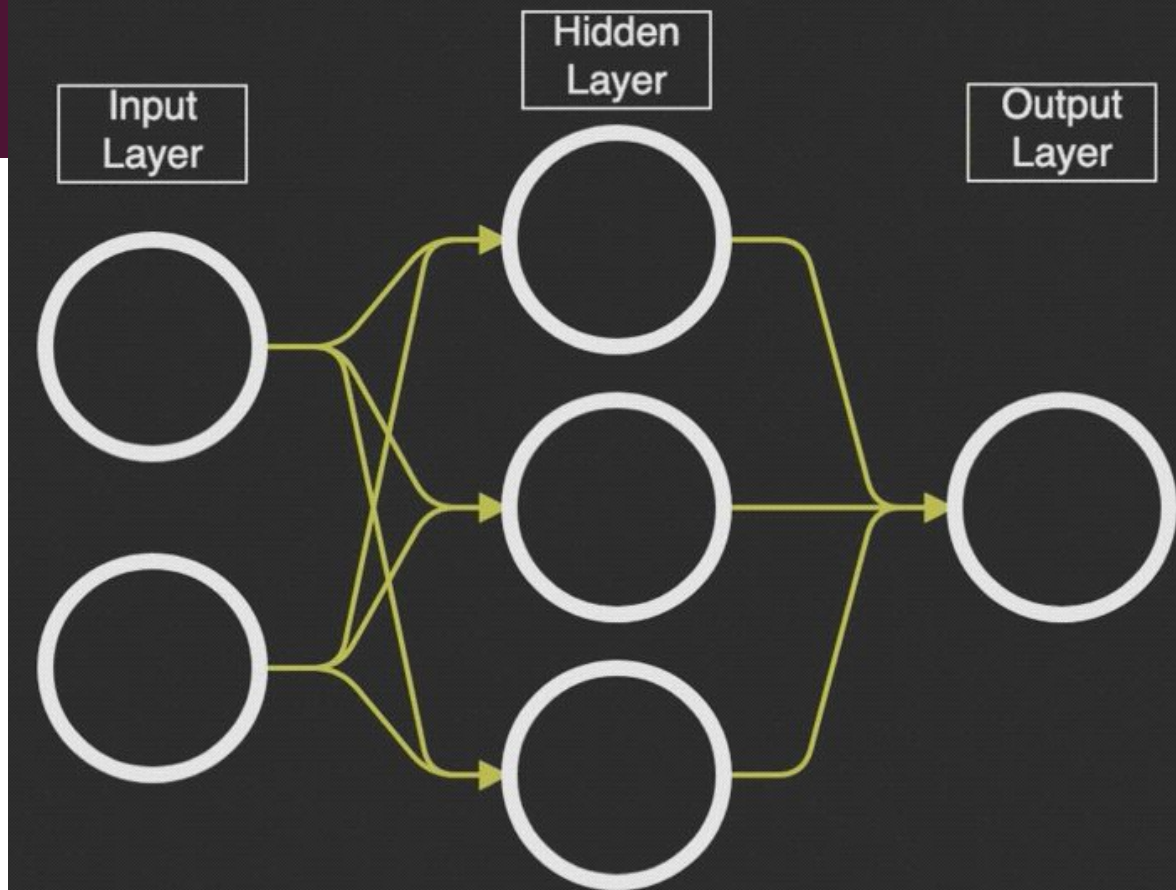
We essentially try to adjust the whole neural network, so that the output value is optimized.

In a sense, this is how we tell the algorithm that it performed poorly or good.

We keep trying to optimize the cost function by running through new observations from our dataset.

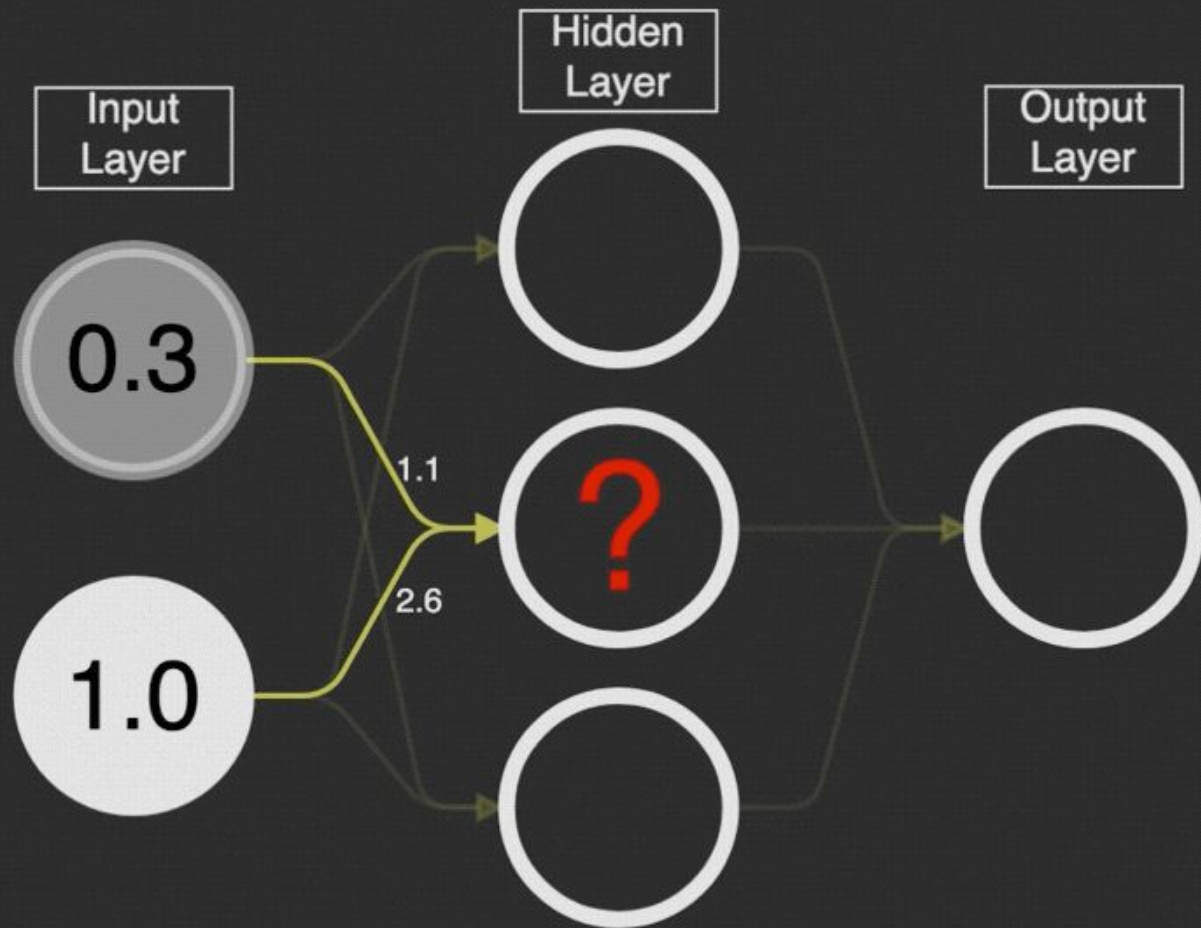
To update the network, we calculate so called gradients, which is small nudges (updates) to individual weights in each layer.

## Step 1

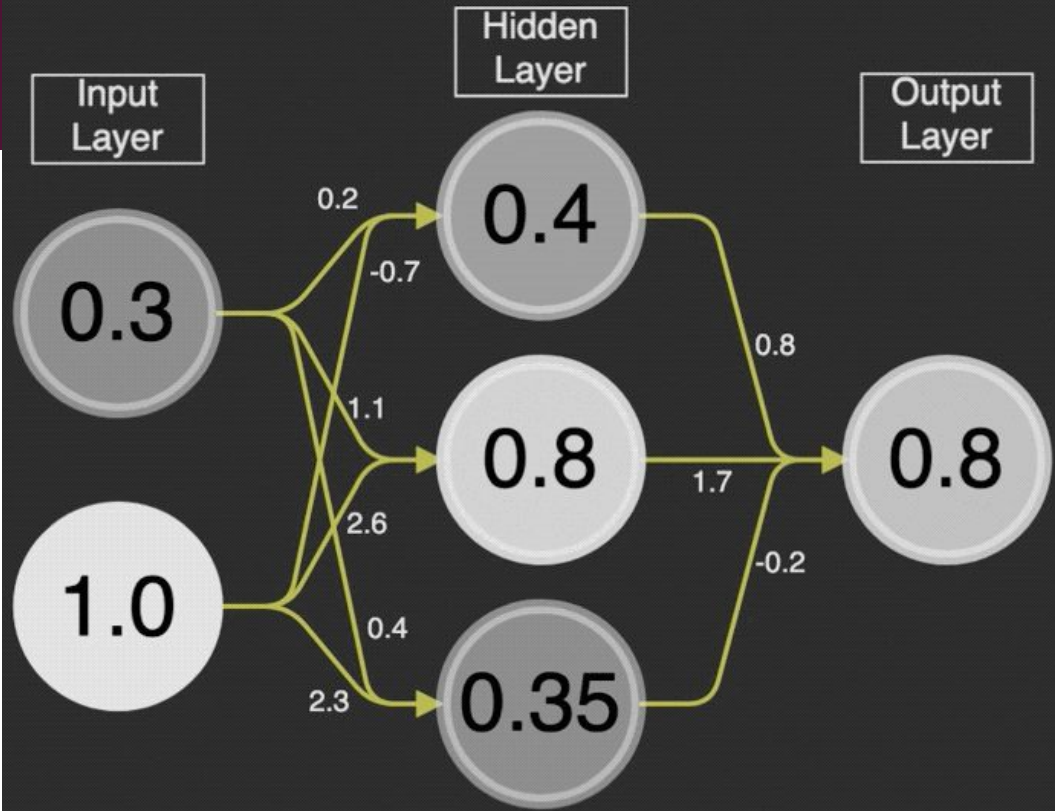




## Step 2

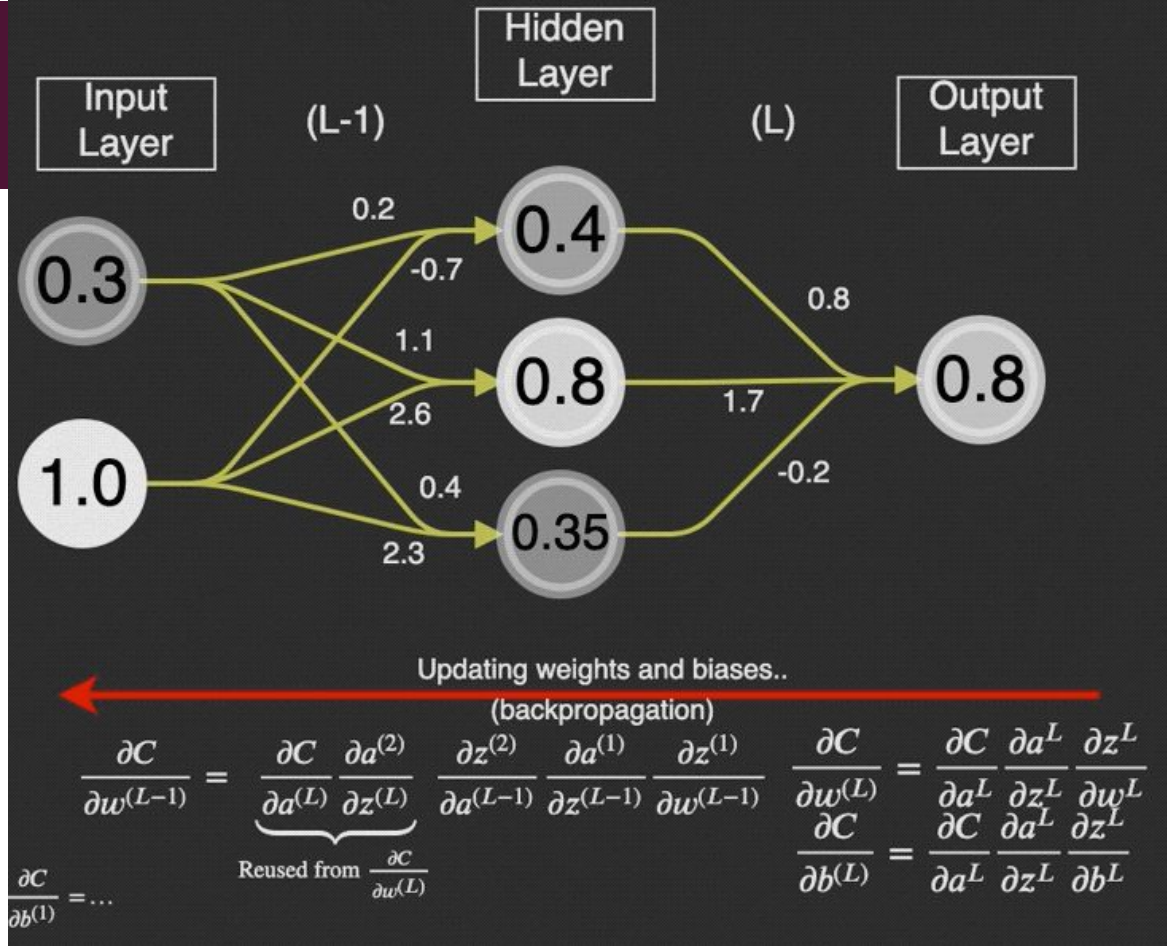


## Step 3



$$a^{(l)} = \sigma(Wa^{l-1} + b)$$

## Step 4



# References

- <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>
- <https://www.ibm.com/in-en/cloud/learn/neural-networks#toc-how-do-neu-vMq6OP-P>

# NEXT Session

To be Covered:

- Neural Network - Python Implementation on kaggle Notebook