

# DEEP LEARNING WEBINAR SERIES

In collaboration with IEEE and IES





# Content

1. What is Deep Learning?
2. Why is Deep Learning Taking off?
3. Neural Networks-An Overview
4. Basic Maths Fundamentals
5. Binary Classification
6. Gradient Descent
7. Cost and Loss function
8. Forward and Backward propagation
9. Activation Functions
10. NN architectures
11. Q and A



# What is Deep learning?

The basic idea behind a neural network is to simulate lots of densely interconnected brain cells inside a computer so you can get it to learn things, recognize patterns, and make decisions in a humanlike way. The amazing thing about a neural network is that you don't have to program it to learn explicitly: it learns all by itself, just like a brain!

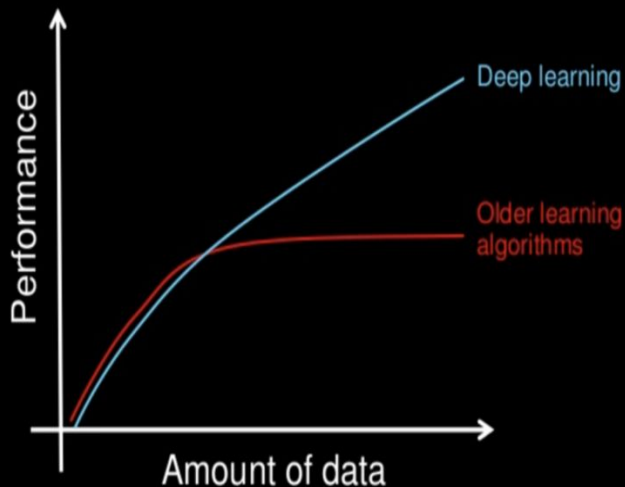
Deep learning is a subset of machine learning where artificial neural networks and algorithms inspired by the human brain, learn from large amounts of data.





# Why is Deep Learning taking off?

## Why deep learning



“Three main forces which improve Neural Networks are: Data, Computation and Algorithms.” -Andrew Ng

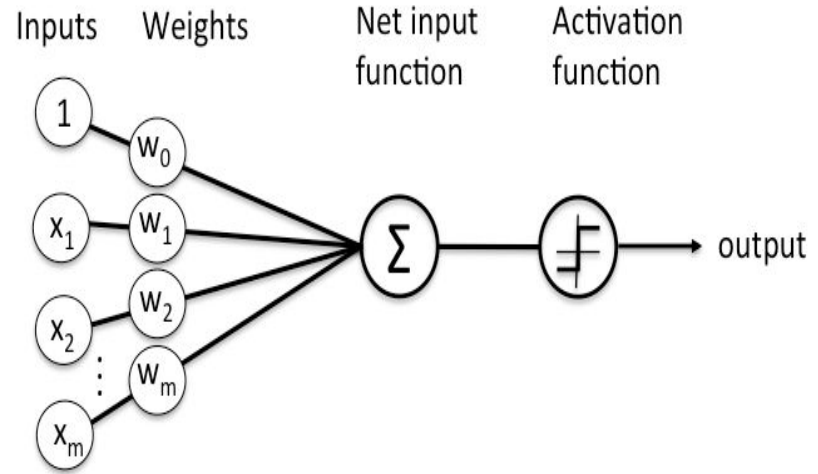
Algorithm improved a lot in NN area, such the introduce of ReLU activator. In the internet era, we have more data and the computational power has grown exponentially recently.

As a result, deep learning is so popular nowadays.



# Neural Networks- An overview

Deep learning is the name we use for “stacked neural networks”; that is, networks composed of several layers. Automatically apply RL to simulation use cases The layers are made of *nodes*. A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs with regard to the task the algorithm is trying to learn; Here’s a diagram of what one node might look like:\_\_\_\_\_

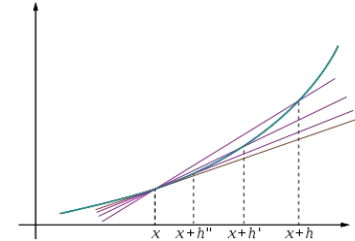


# Basic Maths Fundamentals

## 1. Calculus: Derivatives and Partial Derivatives

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}.$$

Derivative of  $f(x)$  is the slope of tangent at that point.

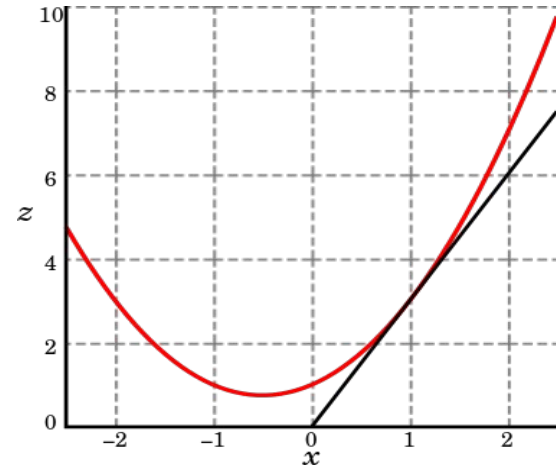
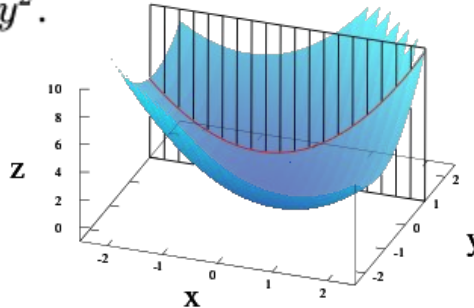



Partial derivatives is slope of tan. in a particular direction. (Other variables treated like constants)

For instance

$$z = f(x, y) = x^2 + xy + y^2.$$

$$\frac{\partial z}{\partial x} = 2x + y.$$





2. Matrix:  $\begin{bmatrix} 1 & 9 & -13 \\ 20 & 5 & -6 \end{bmatrix}$ .

Definition: a rectangular array of numbers, arranged in rows and columns.

An easy example is

$$\bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Famous operation of matrix includes transpose:

$$\bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

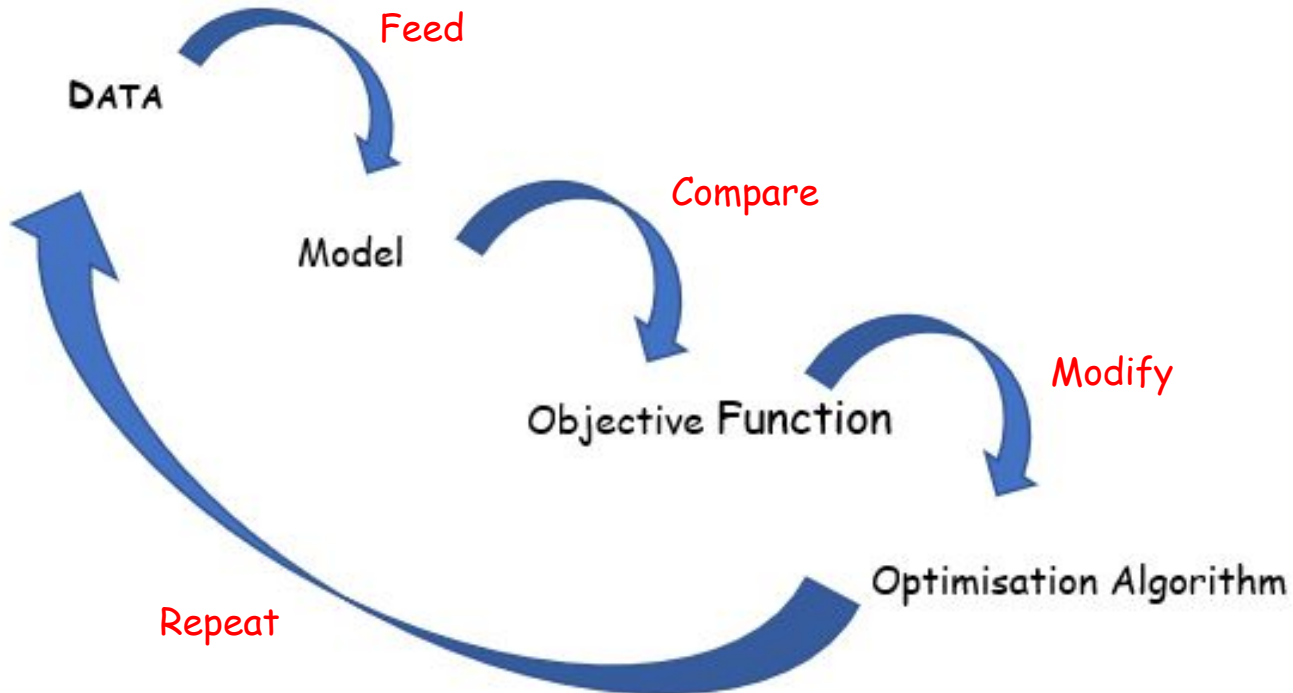
Another important operation is dot product :

For n-vectors

$a=[a_1, a_2, \dots, a_n]$ ,  $b=[b_1, b_2, \dots, b_n]$   $a \cdot b = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$

$$\begin{pmatrix} 4 & -1 \\ 0 & 5 \end{pmatrix} \begin{pmatrix} 1 & 8 & 0 \\ 6 & -2 & 3 \end{pmatrix} = \begin{pmatrix} -2 & 34 & -3 \\ 30 & -10 & 15 \end{pmatrix}$$

# Main Components of a Neural Net







# Binary Classification

It is the task of classifying the given input into 2 classes on the basis of classification rule(s).

A few examples:

- Differentiating b/w Humans and Horses
- Differentiating b/w Cats and Dogs
- Presence / absence of diff types of Cancers



# How ?

Binary Classification —————> Logistic Regression

Logistic Regression is a statistical model that is used to model a binary dependent variable using the logistic function. The logistic model is one that is used to model the probability of a certain class or event existing.

Since, the output can be one of two classes, we assign the label 1 to the output we are interested in and the label 0 to the other class

Note: One of the major assumptions in a logistic regression is that we assume that there is a linear relationship between the variables.



# The Goal

To find  $\hat{y}$ , the probability that  $y$ , the output, is equal to 1 given the input vector  $x$ .

Mathematically:  $\hat{y} = P(y = 1 \mid x)$

Since logistic regression assumes a linear relationship between the variables, we generalise the equation to be of the form

$$\hat{y} = w^T x + b$$

where  $w$  is the matrix of weights and  $b$  is the bias. The bias and the weights for the best possible results are determined by the computer as we implement the Gradient Descent Algorithm (more on that later)



# Using the Sigmoid Function

$$\hat{y} = w^T x + b$$

While the above equation can be used to accurately model the linear relationship between variables, the range of such a function is  $[-\infty, \infty]$ .

Since probability of any event has to lie b/w 0 and 1, we modify the above equation as such by using the sigmoid function

$$\hat{y} = \sigma(w^T x + b)$$

$$\text{where } \sigma(z) = 1 / (1 + e^{-z})$$

The sigmoid function can be used to regulate the outputs as

- A high value of  $z$  causes  $y$  to tend towards 1
- A low value of  $z$  causes  $y$  to tend towards 0



# Loss Function

The Loss Function helps us evaluate how far off our prediction is from the given output. Hence we want the loss function to be as small as possible. It is also known as the error function and is denoted by  $\mathcal{L}(\hat{y}, y)$ .

$$\mathcal{L}(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

Why does the above function make sense ?

- If  $y = 1$ ,  $\mathcal{L}(\hat{y}, y) = -\log(\hat{y})$ , and since we want the loss to be as small as possible, we need  $\log(\hat{y})$  to be as large as possible (due to the -ve sign) which can be achieved when  $\hat{y}$  is as big as possible and due to the sigmoid constraint, this means that  $\hat{y}$  is as close to 1 as possible (which minimises the loss)
- A similar analogy can be used when  $y = 0$

Note: This is just one of many possible error functions.



# Cost Function

While the Loss (error) function is for a single example in the training set, the cost function is the average loss for all the examples in the training set. It is denoted by  $J(w,b)$  where  $w$  (the weights) and  $b$  (the bias) are the 2 parameters on which the cost function depends.

$$J(w,b) = (1/m) \sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Where  $m$  is the number of training examples

Since minimising the cost function improves the accuracy of the regression, we use gradient descent in order to obtain the optimal weight vector and the bias that result in the least possible magnitude of the cost function.

# Gradient Descent

$$J(w, b) = - (1/m) \sum_i (y^{(i)} \cdot \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \hat{y}^{(i)}))$$

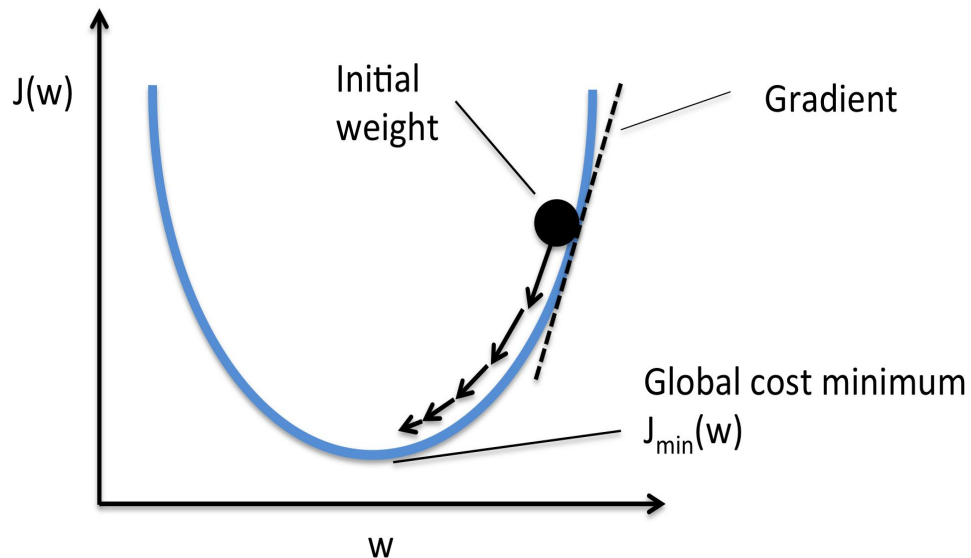
Where  $\hat{y}^{(i)} = w^T x^{(i)} + b$

Repeat {

$$w := w - \alpha \frac{\partial J(w, b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b}$$

}

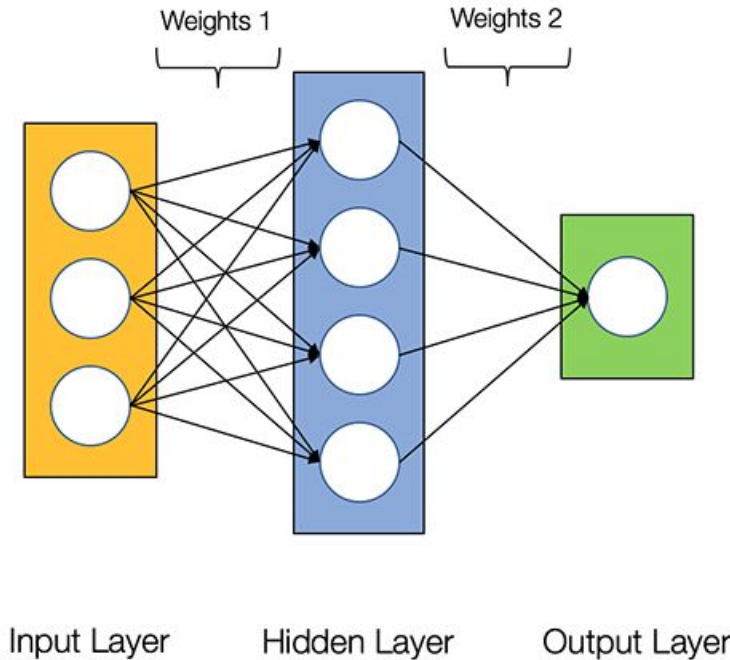


# Google Collab Notebook First-Part





# Forward Propagation



$$Z^{[1]}_1 = (W^{[1]}_1)^T X + b^{[1]}_1$$

$$Z^{[1]}_2 = (W^{[1]}_2)^T X + b^{[1]}_2$$

$$Z^{[1]}_3 = (W^{[1]}_3)^T X + b^{[1]}_3$$

$$Z^{[1]}_4 = (W^{[1]}_4)^T X + b^{[1]}_4$$

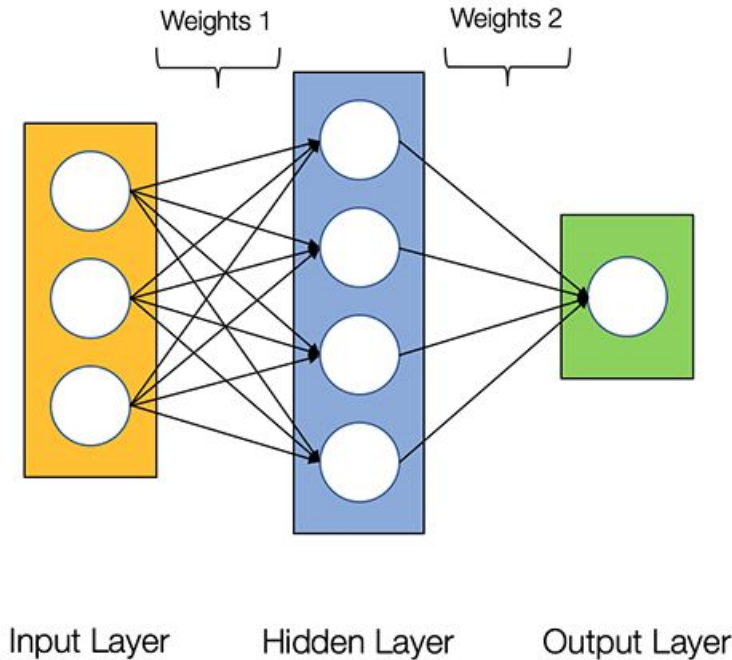
$$Z^{[1]} = [Z^{[1]}_1 \ Z^{[1]}_2 \ Z^{[1]}_3 \ Z^{[1]}_4]^T$$

$$a^{[1]}_1 = \sigma(Z^{[1]}_1) ; \text{ Similarly, } a^{[layer]}_{\text{node in layer}} = \sigma(Z^{[layer]}_{\text{node in layer}})$$

$$a^{[1]} = [a^{[1]}_1 \ a^{[1]}_2 \ a^{[1]}_3 \ a^{[1]}_4]^T$$

And so on, as it propagates forward. This is the jist of forward propagation.

# Back Propagation



$y$  = Target output

$\hat{y}$  = Predicted output

$$L = \sum (0.5)(y - \hat{y})^2$$

$dL = \partial L / \partial W$  We use sum rule, chain rule and product rule in order to evaluate this derivative

So, doing that, we get:  $\partial L / \partial W^{[1]} = -(y - \hat{y}) \cdot \sigma'(Z^{[2]}) \cdot \partial Z^{[2]} / \partial W^{[1]}$ ,

Following similarly for the other layers.

Then, we update the weights by gradient descent.

$$W^{[1]} = W^{[1]} - \alpha \cdot \partial L / \partial W^{[1]}$$

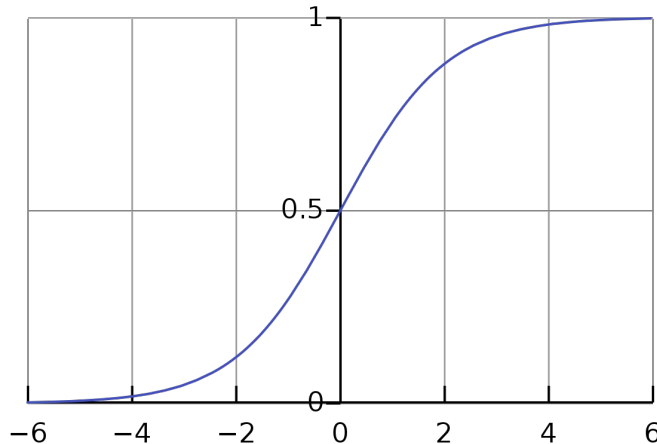
Similarly, we update all weights as such, and then forward propagate. Then we backward propagate again until we have achieved the target value, or until the error function is minimised.



# Activation Functions

Activation function is like a mathematical gate that feeds the current neuron and the output into the next. There are a variety of activation functions in use, and we will discuss a few of the here.

1. Sigmoid:  $f(z) = (1 + e^{-z})^{-1} \Rightarrow$  It has a range of (0,1). So, it is extremely useful for probability prediction, or for simple classification problems. Problems with this function is that the output of the sigmoid function saturates at higher values, and also because it is not zero-centered.

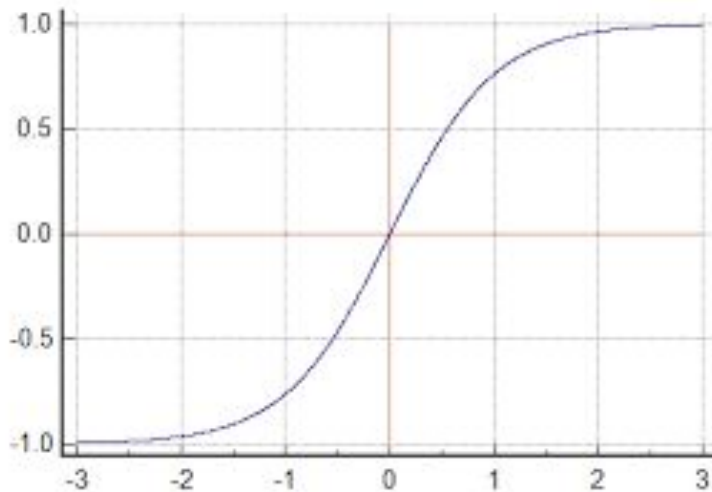


Fires neurons that are more positive in value. The function value gets closer to zero as the input gets more negative, and gets closer to 1 as the input gets more positive.

The sigmoid function is monotonic, but its derivative is not.

# Activation functions

2.  $\tanh(x)$  Range of  $\tanh(x)$  function is  $(-1,1)$ . It is centered at 0. Its is also sigmoid in shape(S-shaped). More negative inputs are mapped to a negative value, and positive inputs are mapped to a positive value. It is usually used for classification problems.



Like the classic sigmoid function, the hyperbolic tangent function is also monotonic. However, its derivative isn't. Therefore, utilizing this activation function also runs into problems such as the vanishing gradient problem.



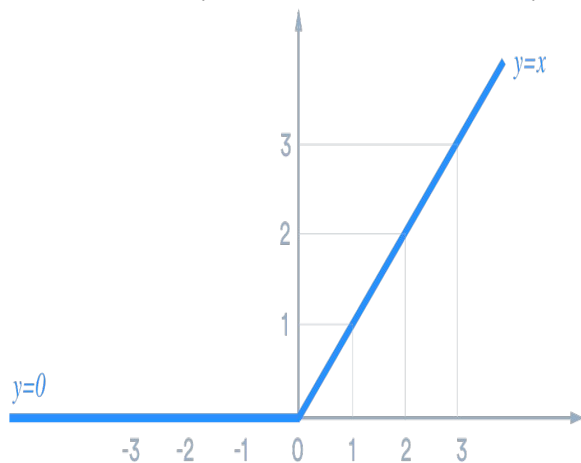
# Activation functions

3. Softmax function:  $f(\vec{a}): \exp(a_i) / \sum_k \exp(a_k) \Rightarrow$  Essentially, this function converts the given inputs of the array into a probability distribution function. That way, the sum of the outputs =1, and represents the probabilities of the outputs in some sense. It is used frequently in multiclass classification problems. Therefore, its graph varies depending on the input fed in.

Problems with the softmax function are commonly rounding errors that are occurred with the code. Therefore, this leads to modelling uncertainty while using the softmax function.

# Activation Functions

3. ReLU:  $f(x) = \max(0, x)$  => Probably the most popular activation function in use now. Used actively in CNNs. The biggest advantage of ReLU is that it has unsaturated gradient, and therefore, does not run into familiar problems of the previous activation functions. It also does not utilise exponentiation, like the previous functions discussed.



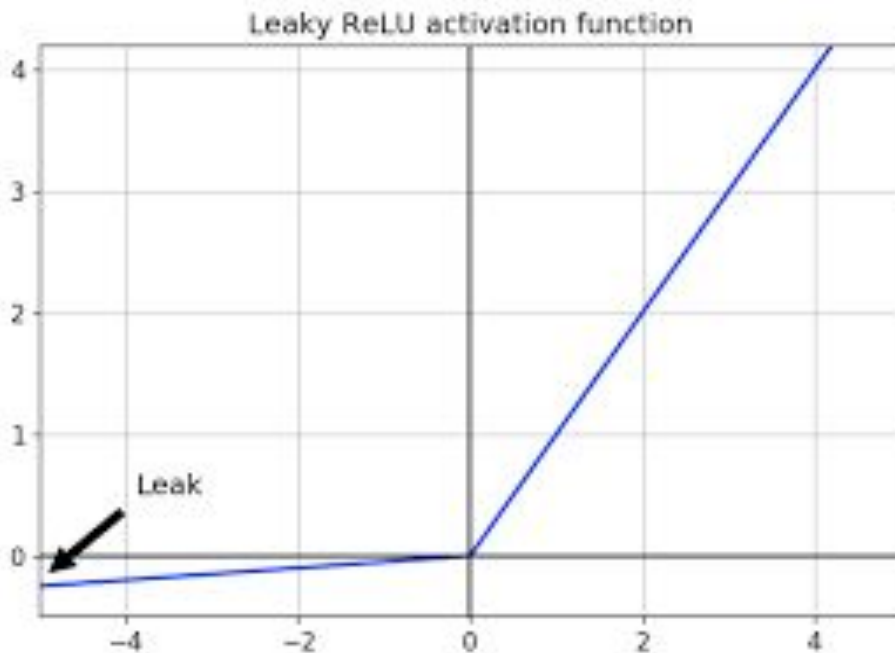
The ReLU function outputs 0 for all negative inputs received, immediately silencing them. This is the major problem of ReLU, that runs into issues whilst backpropagating.

Many activation functions have been built over the ReLU functions such as noisy ReLU, leaky ReLU, etc.



# Activation Functions

4. Leaky ReLU:  $f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{if } x < 0 \end{cases}$   $\Rightarrow$  Similar to the classic ReLU function, however, this does not instantly silence the negative inputs. Therefore, it performs better than the ReLU functions in most cases.





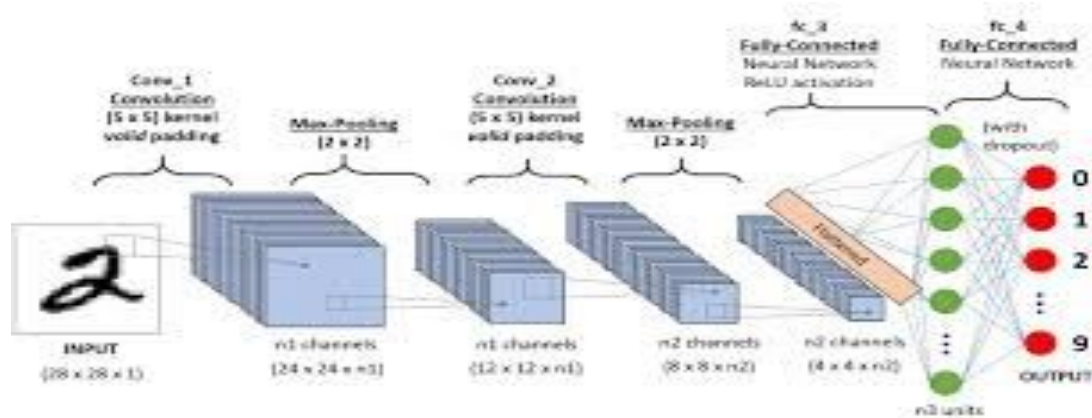
# Google Collab Notebook



# Neural Network architectures

With the wide popularity of deep learning, and the onset of the AI revolution, Neural Networks are used in several areas. Some famous neural network architectures will be discussed in very brief detail:

CNN: Convolutional Neural Networks form the backbone of Computer Vision. The neural network is unique, and has a convoluting step, giving this architecture its name. This architecture is very effective in identifying images, processing pixelated inputs, and has active uses in face recognition, NLP, among various other fields.

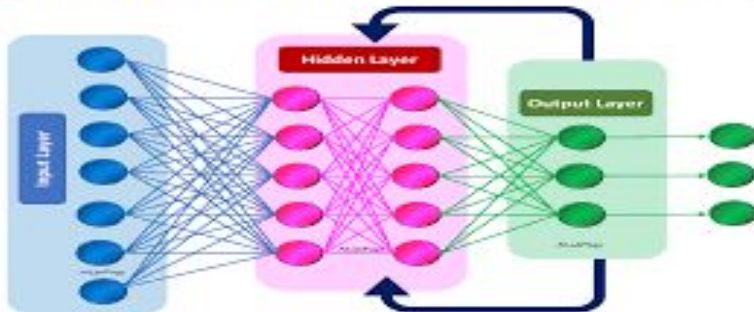


# Neural Network Architectures

RNN: Recurrent Neural Networks are utilised in several areas such as Sequence Classification, Sequence Generation, and Sequence Labelling. RNNs are commonly used in NLP, and are utilised for sentiment analysis, speech tagging, machine translation, etc.

GAN: Generative Adversarial Networks are used in the fields of Computer Vision. These networks are commonly used to identify if an image is natural or whether it has been duplicated, or re-made. GAN is nowadays commonly used in drug discovery, 3D shape estimation, robotics etc.

## Recurrent Neural Networks



## Generative Adversarial Network

