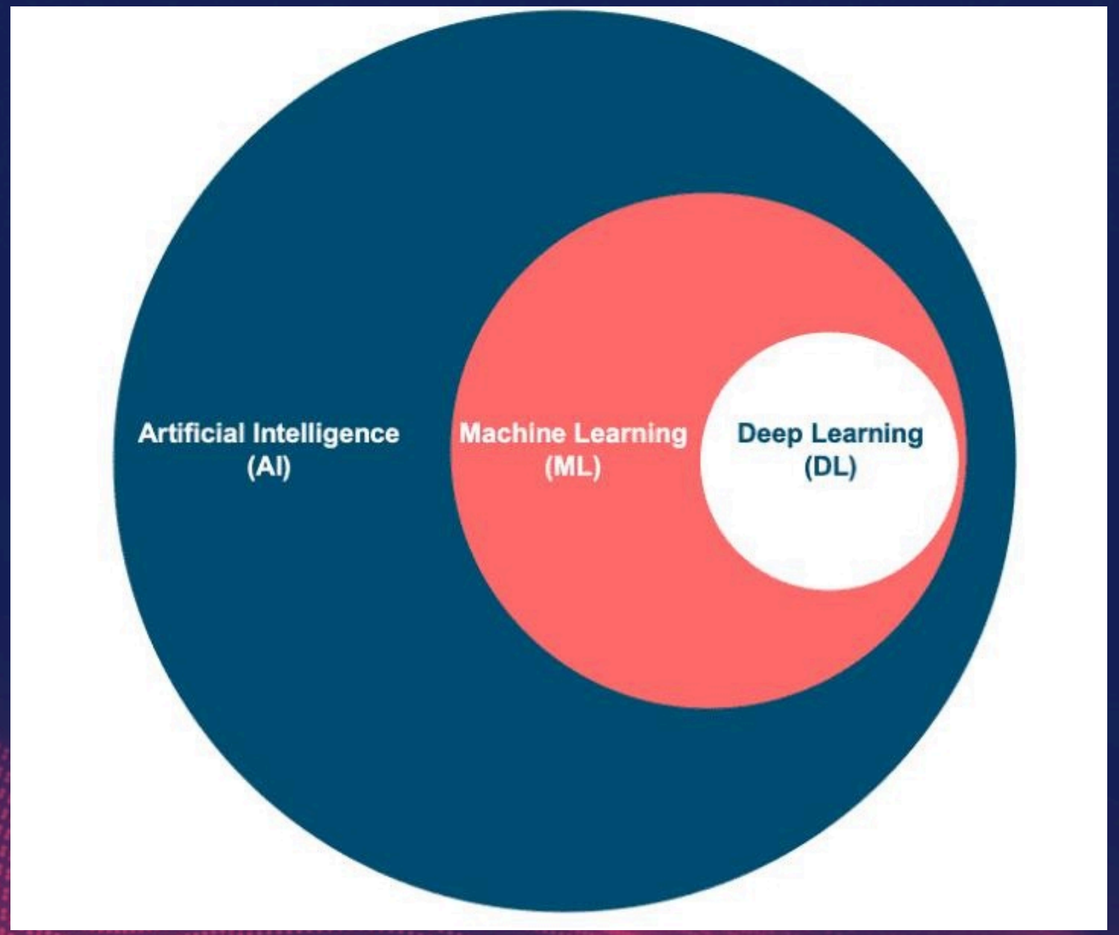
DEEP LEARNING AND PYTORCH

Table of CONTENTS

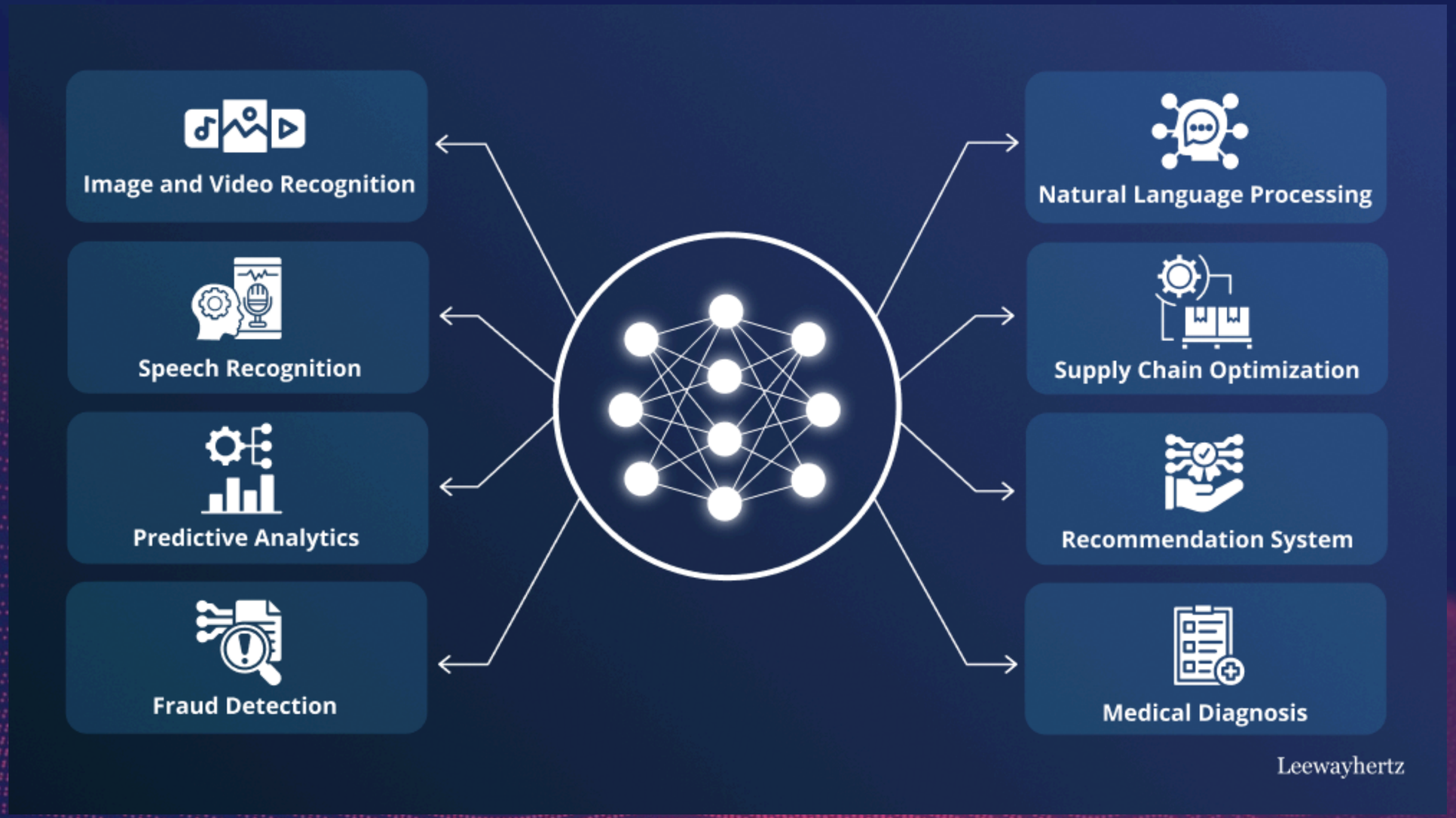
What is Deep Learning?

Deep Learning

- Deep learning is a powerful subset of machine learning known for its ability to process unlabeled data and recognize patterns with exceptional accuracy
- The entire topic of Deep Learning is based of the concept of Neural Networks.



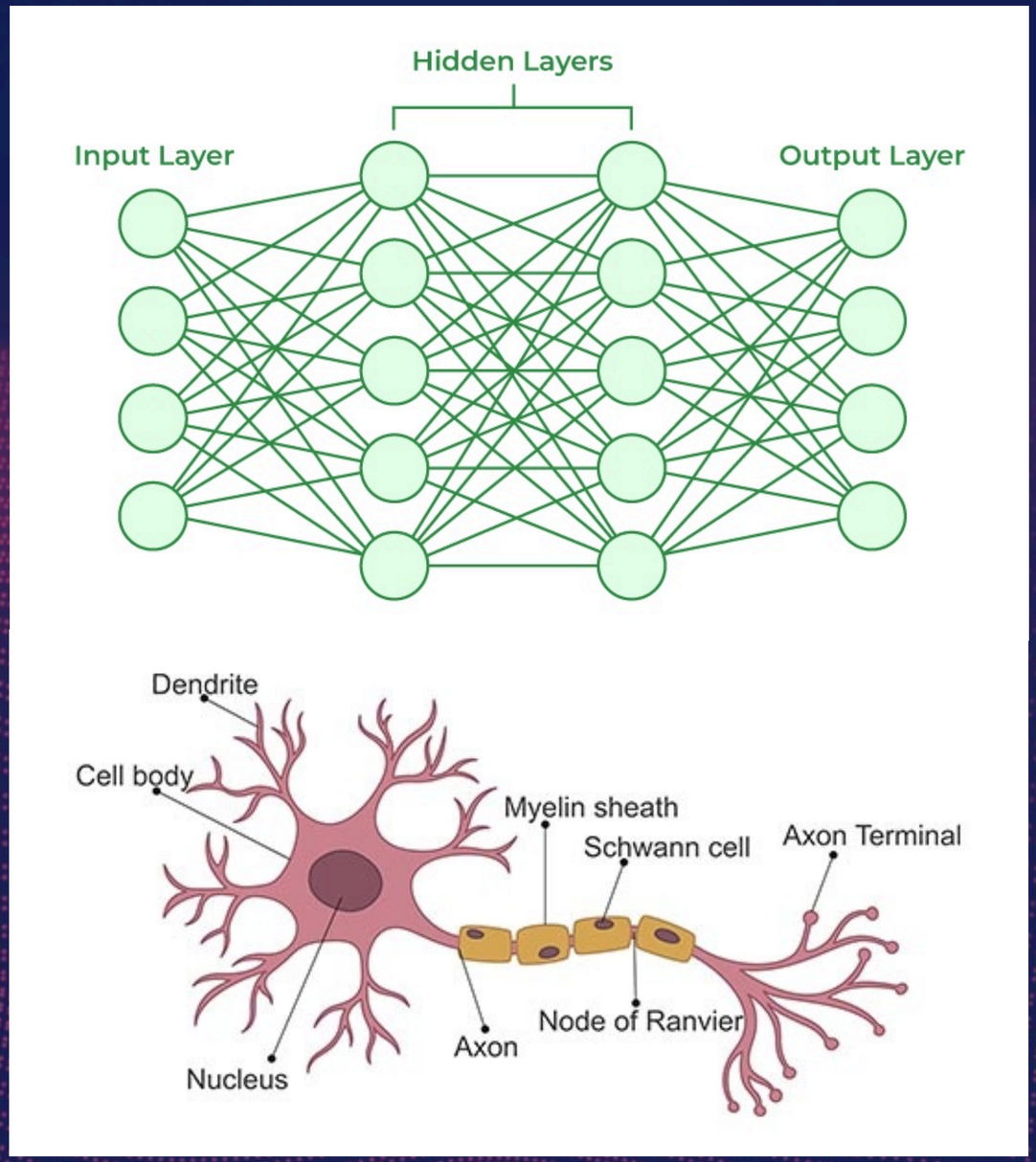
Use cases of deep learning



What is a Neural Network?

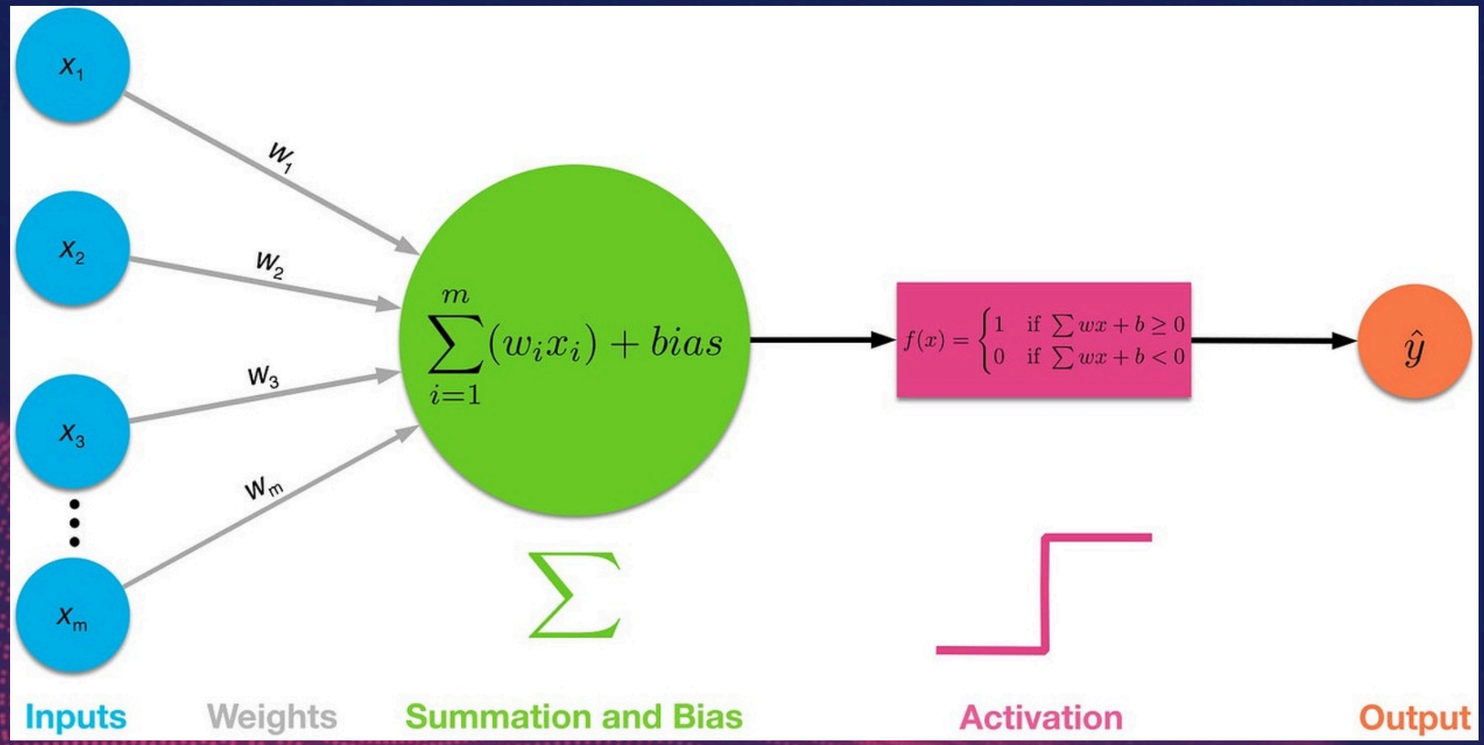
Neural Network

- Neural networks serve as the foundational architecture of Deep Learning.
- Their structure and operation are inspired by the biological neurons found in the human brain, hence the term 'neural'.
- This interconnected structure allows neural networks to learn complex patterns and relationships in data



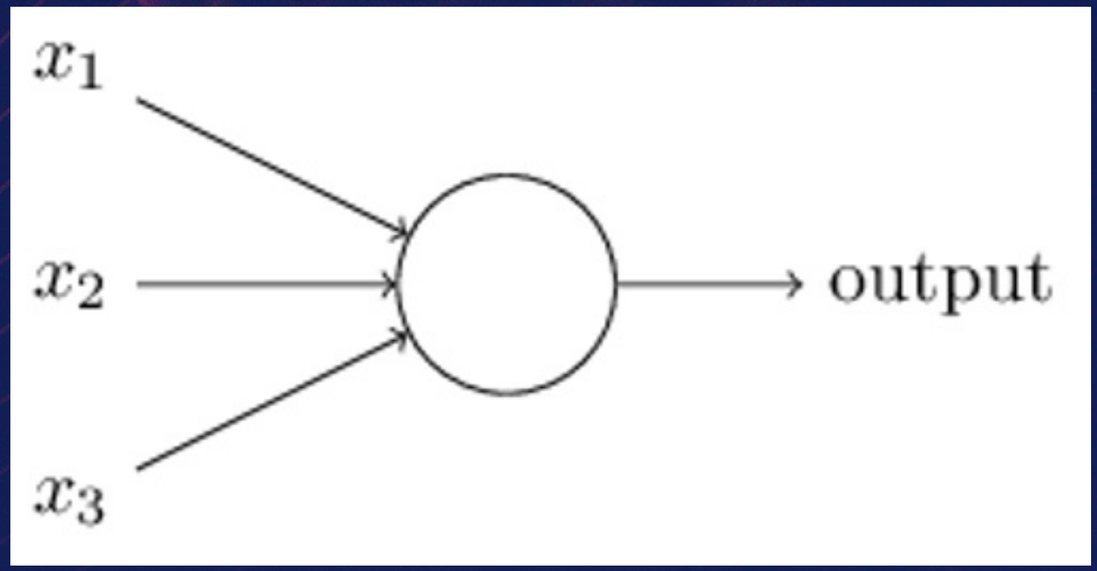
Components of a Neural Network

- Neurons (Nodes)
- Weights
- Biases
- Activation Function
- Loss Function
- Optimiser
- Learning Rate



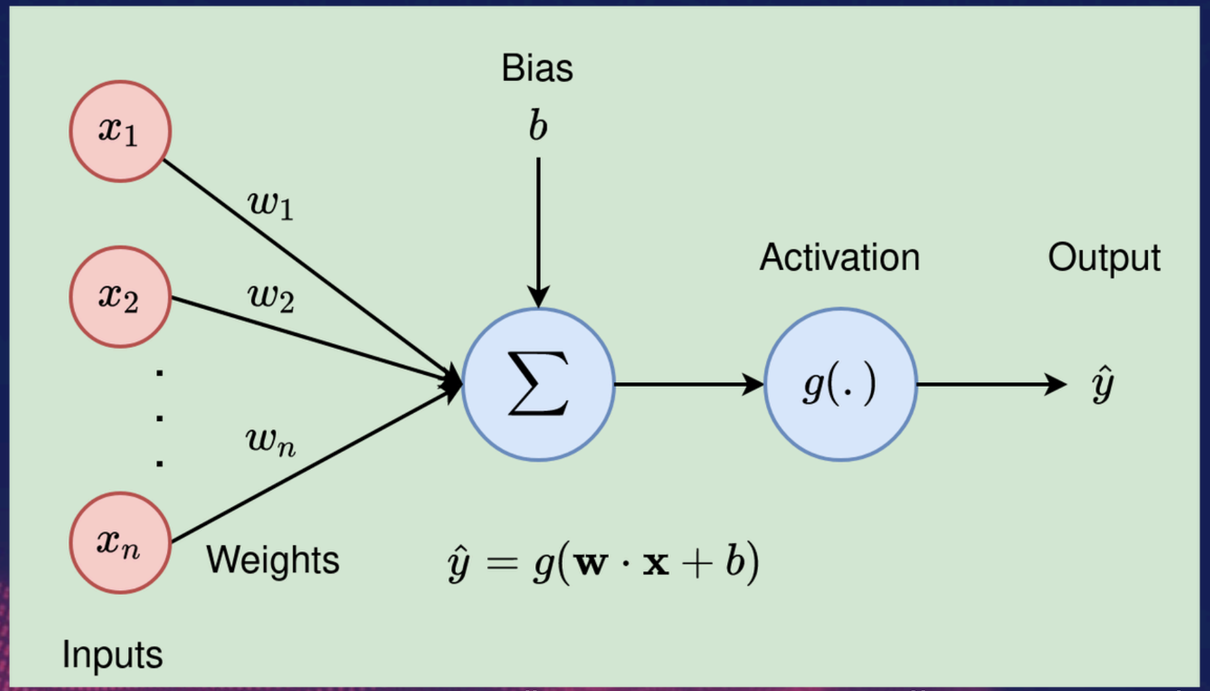
Neurons

- Neurons serve as the foundational unit of Neural Networks.
- Their structure and operation are inspired by the biological neurons found in the human brain, hence the term 'neural'.
- This interconnected structure allows neural networks to learn complex patterns and relationships in data



Weights and Bias

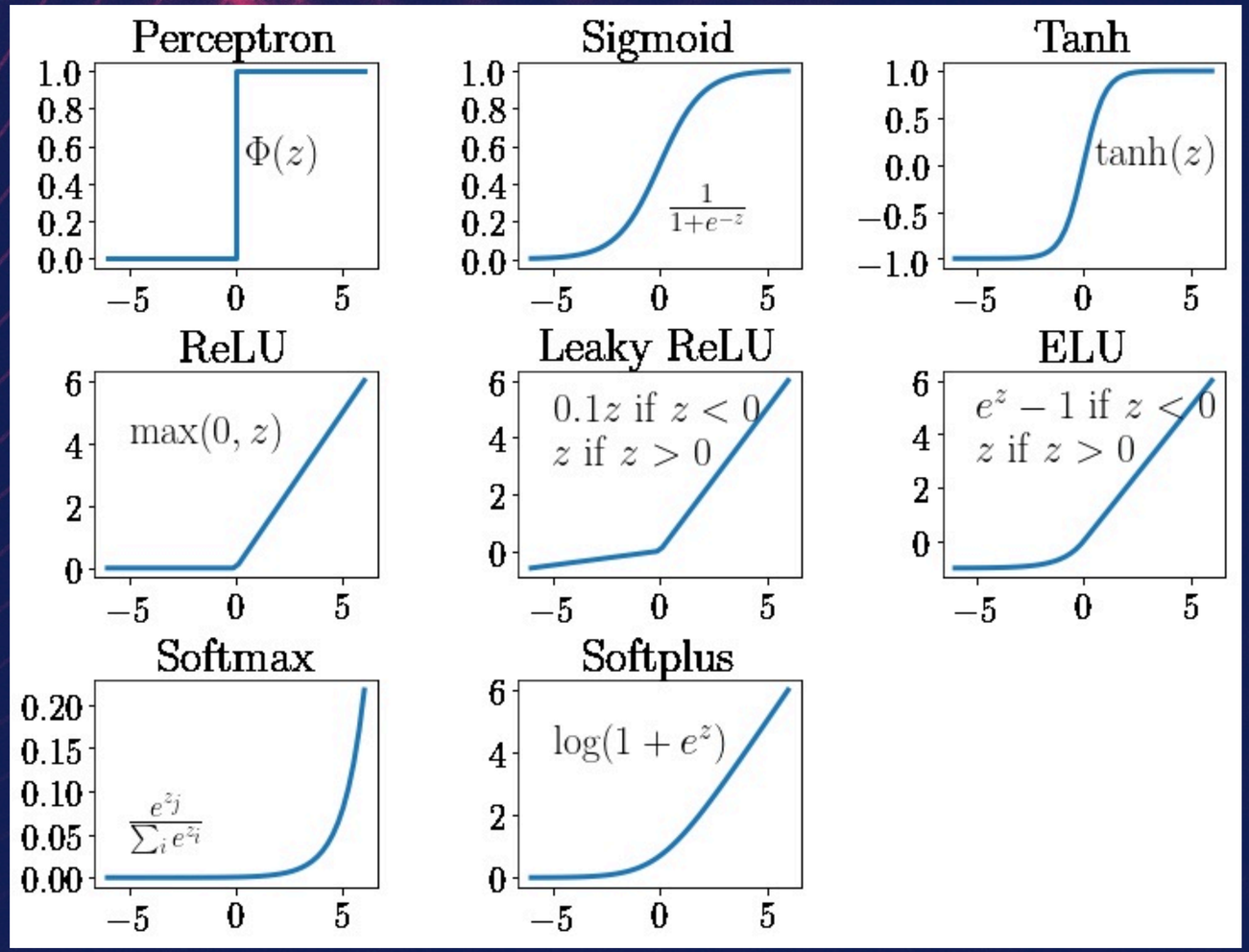
- Weights in neural networks are adjustable parameters that determine the relation between neurons
- A positive weight contributes towards the activation of the associated neuron and vice versa.
- Biases on the other hand provide each neuron with a trainable constant value that helps adjust the output along with the weighted sum of inputs.



*Weights and biases are typically positive or negative floating point values

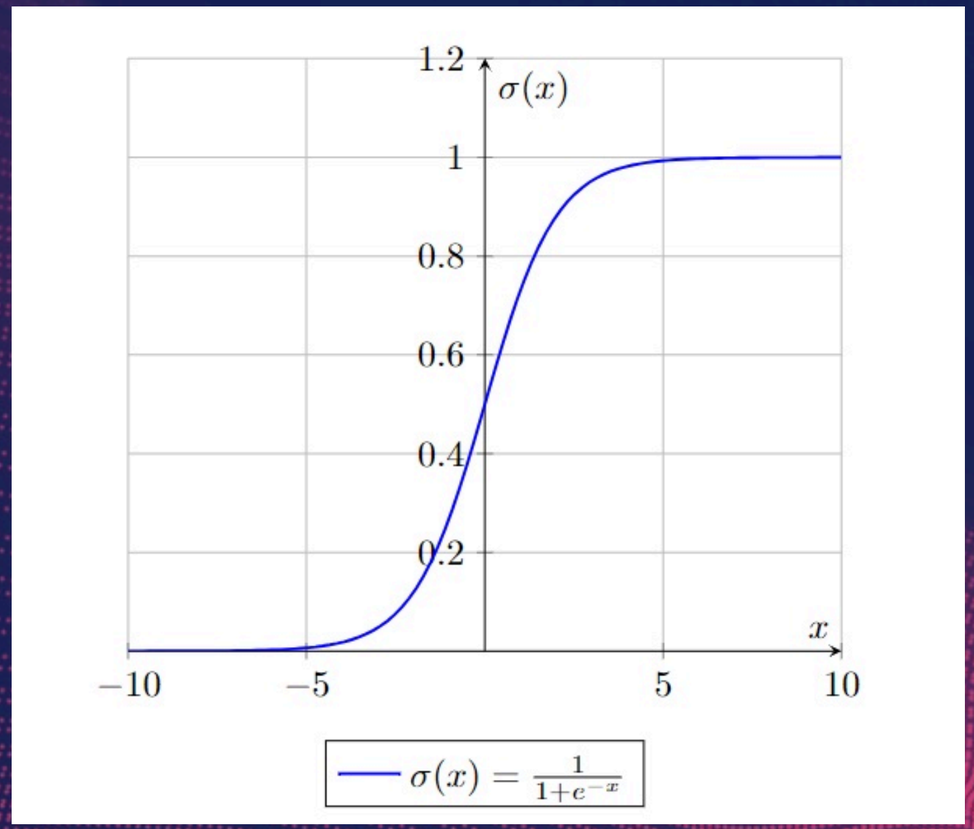
Activation function

- An activation function in a neural network serves as control function that determines whether and to what extent a neuron should be activated.
- It helps in ensuring non-linearity which is essential for modeling complex relationships in the data

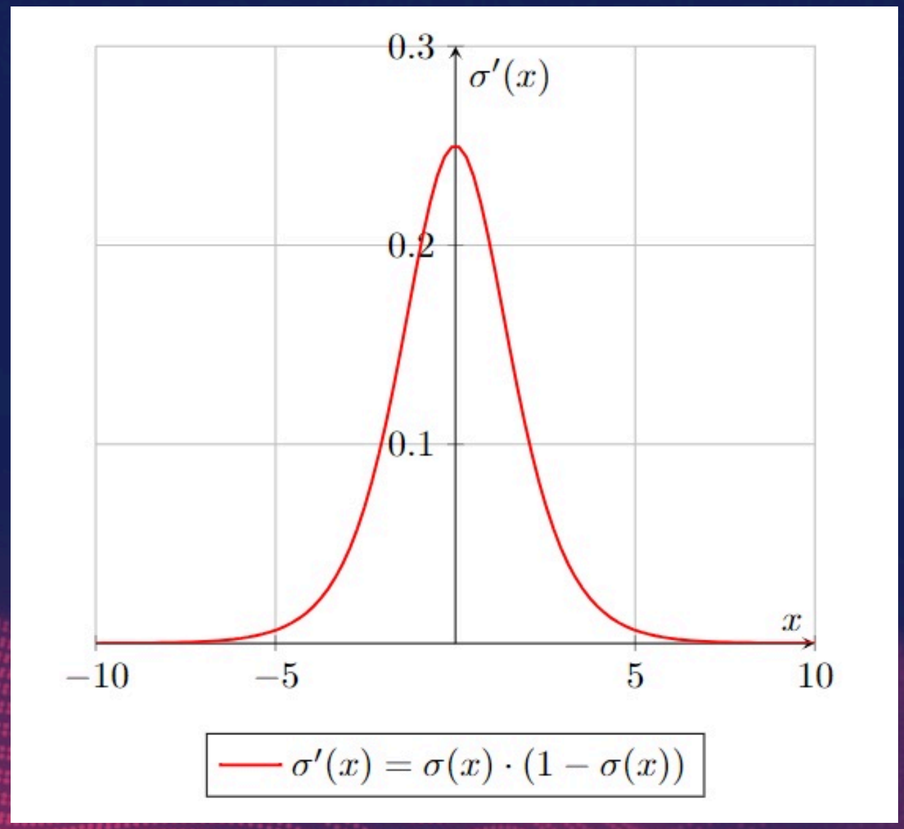


Common examples of Activation functions

1. Sigmoid function



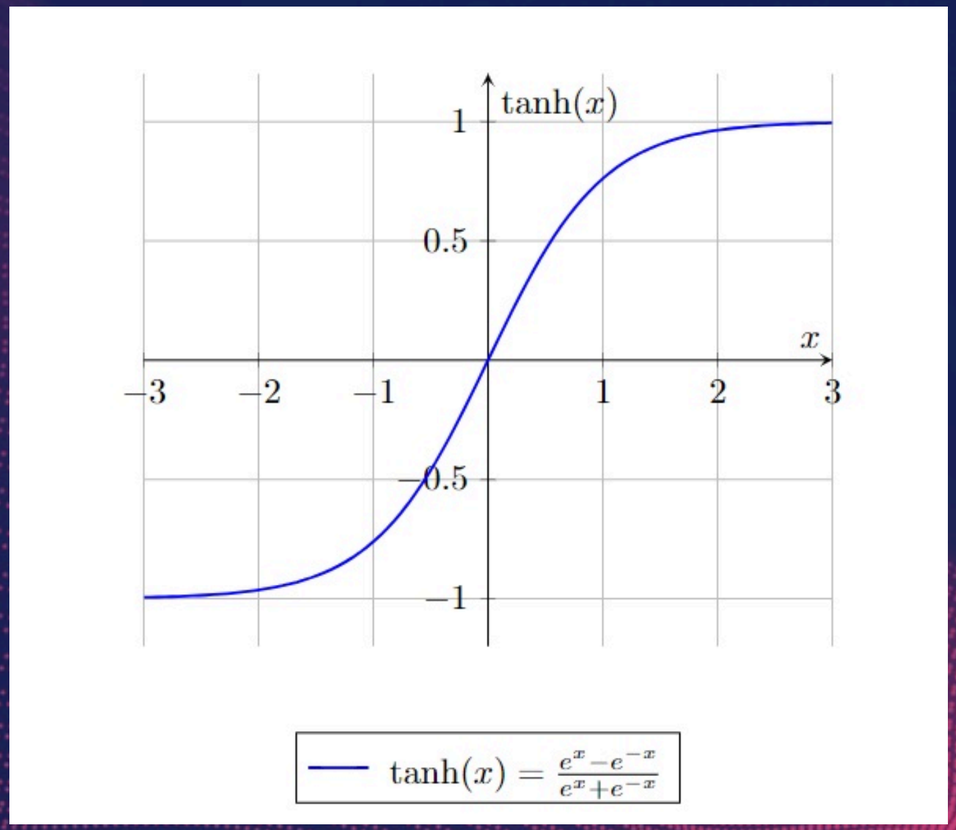
Sigmoid function



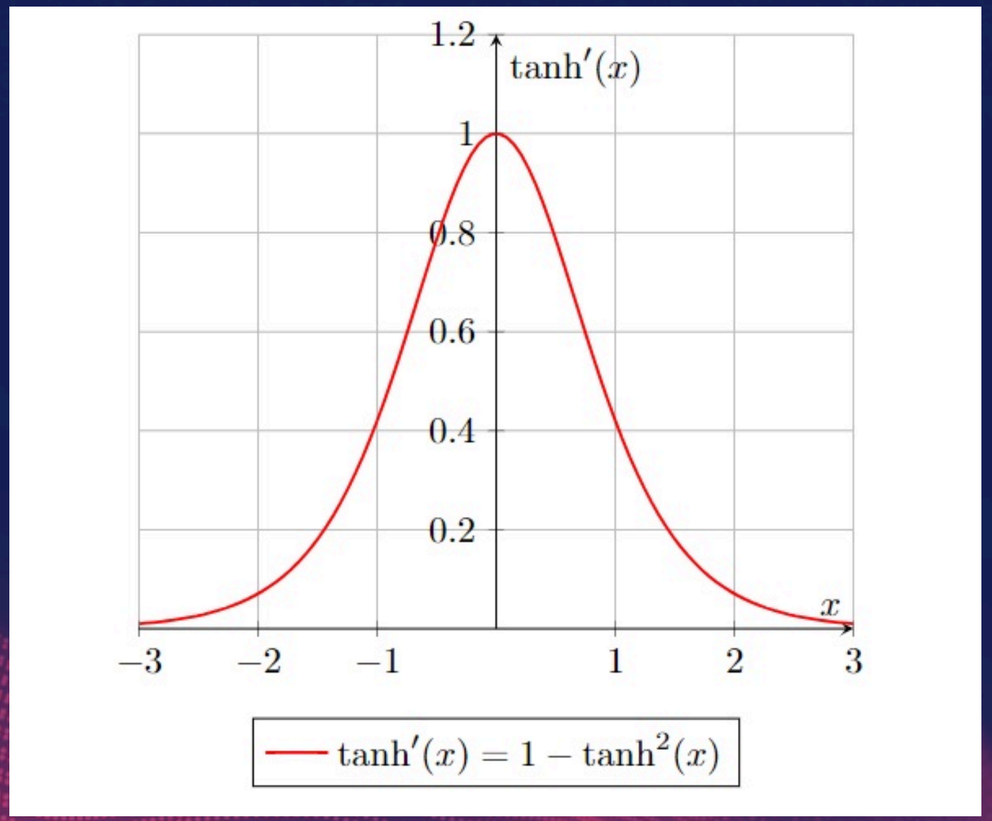
Derivative of Sigmoid

Common examples of Activation functions

2. Tanh function



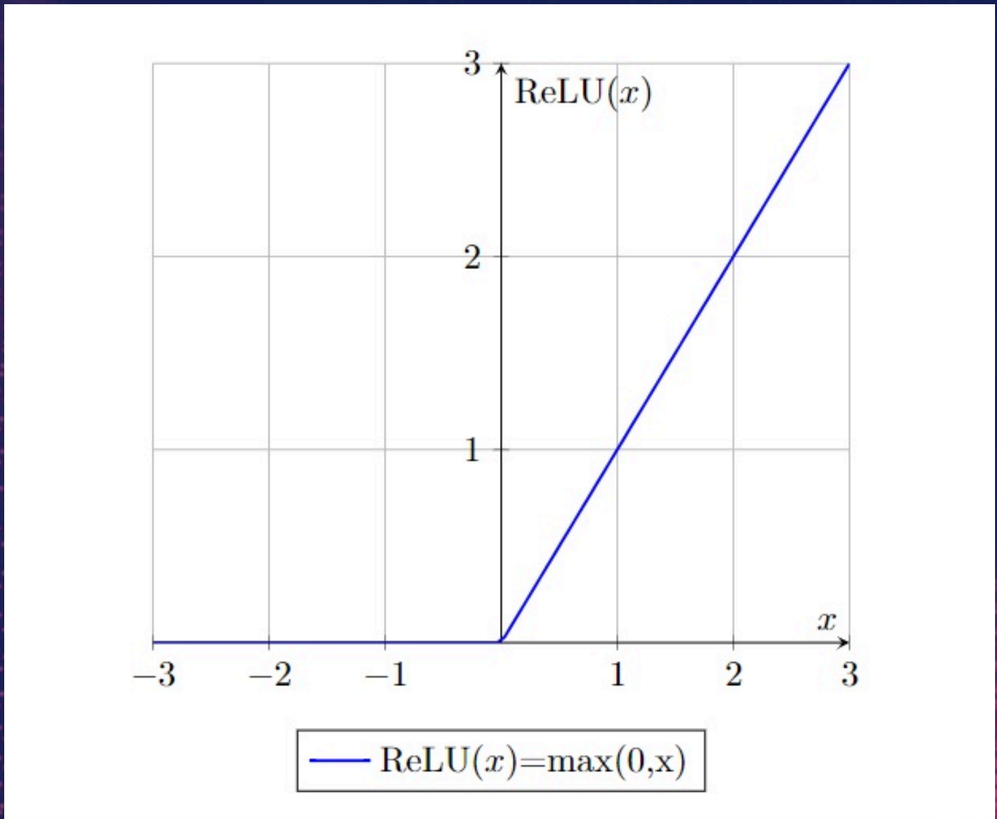
tanh function



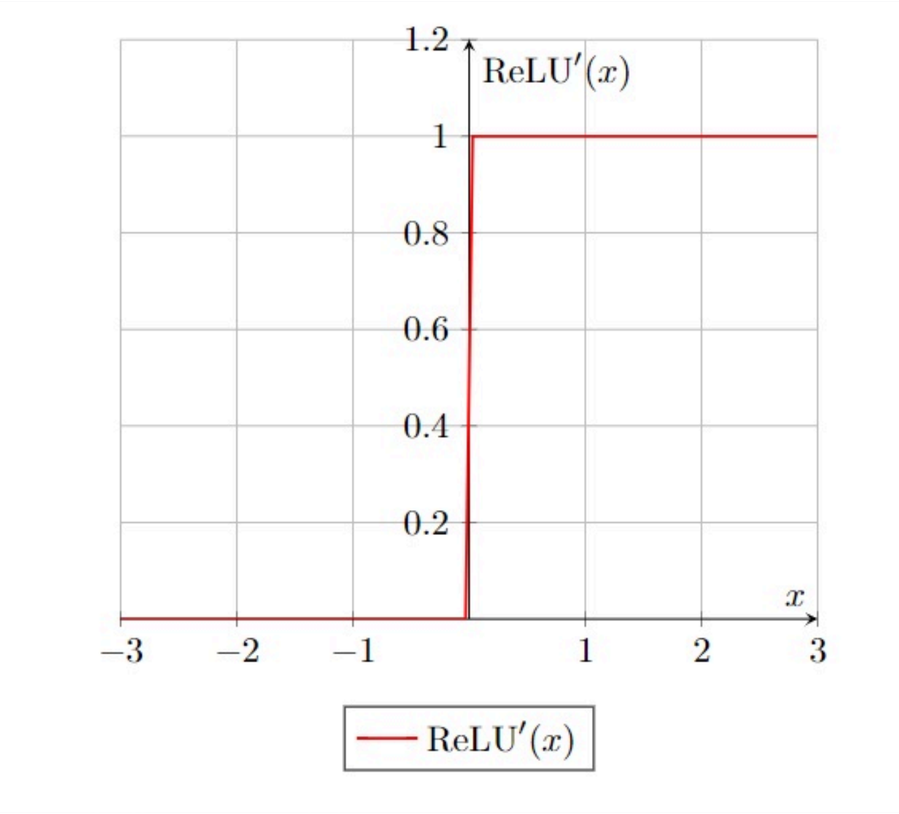
Derivative of tanh

Common examples of Activation functions

3. ReLU Function



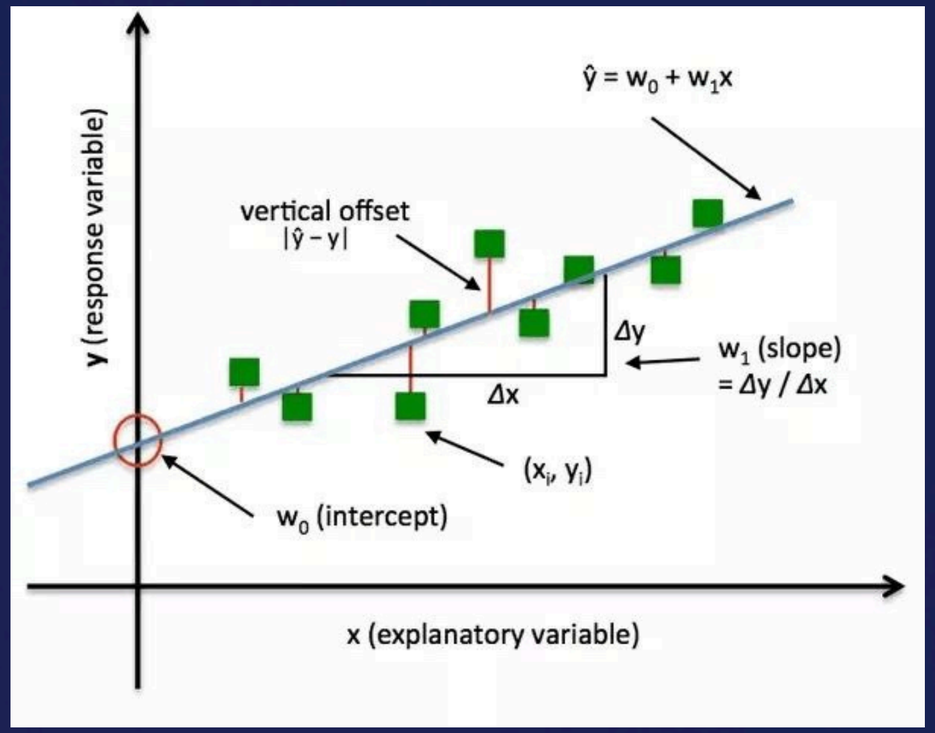
ReLU Function



Derivative of ReLU

Loss function

- A loss function, also known as a cost function or objective function, measures the difference between the predicted outputs of a neural network and the actual target values.
- It is essentially a quantification of the error in value estimation. Minimising the loss function results in obtaining a high prediction accuracy

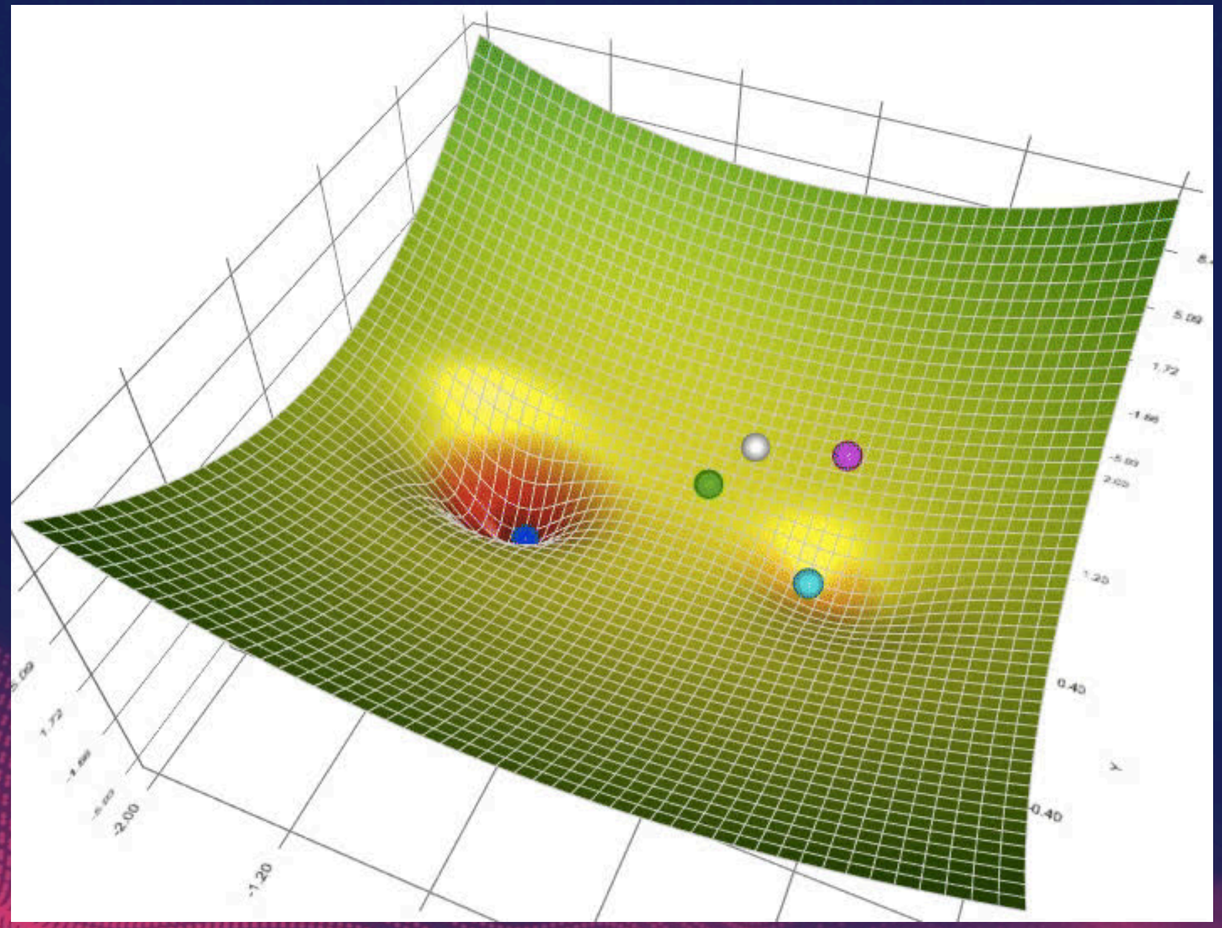


Here, If The total summed value of $|\hat{y}-y|$ is less, the line could be said to be the best fit.

We can expand this logic beyond merely one variable(x) to n variables.

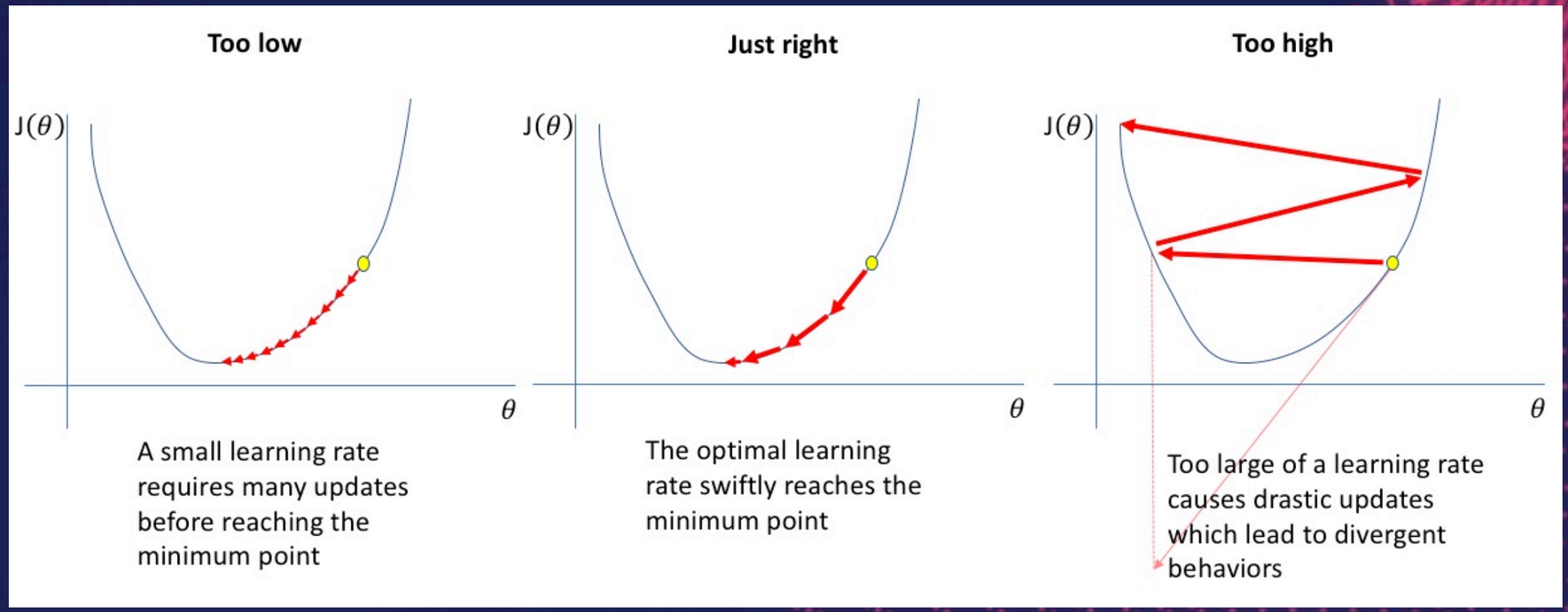
Optimiser

- An optimizer refers to an algorithm that adjusts the weights and biases of the neural network during training in order to minimize the loss function.
- The main goal of an optimizer is to find the optimal set of parameters that allow the neural network to make accurate predictions.



Learning Rate

- The learning rate (often denoted as η or α) is a scalar value that controls the step size at each iteration while moving towards a minimum of a loss function.
- In the context of neural networks, it dictates how much the model's weights should be adjusted with respect to the loss gradient.



How do Neural Networks learn?

Steps to train a neural network

- Initialisation
- Forward propogation
- Loss Computation
- Back propogation
- Gradient Descent

SLIDO QUIZ

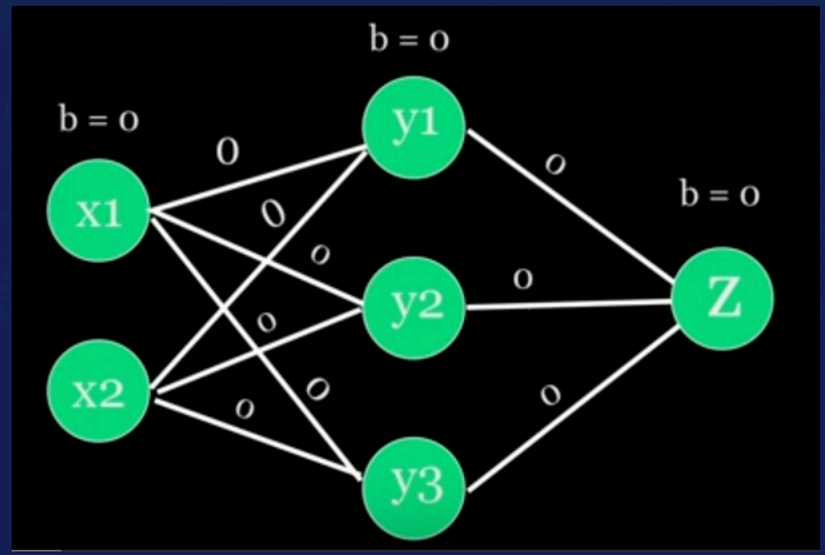
Test your understanding of weights and biases

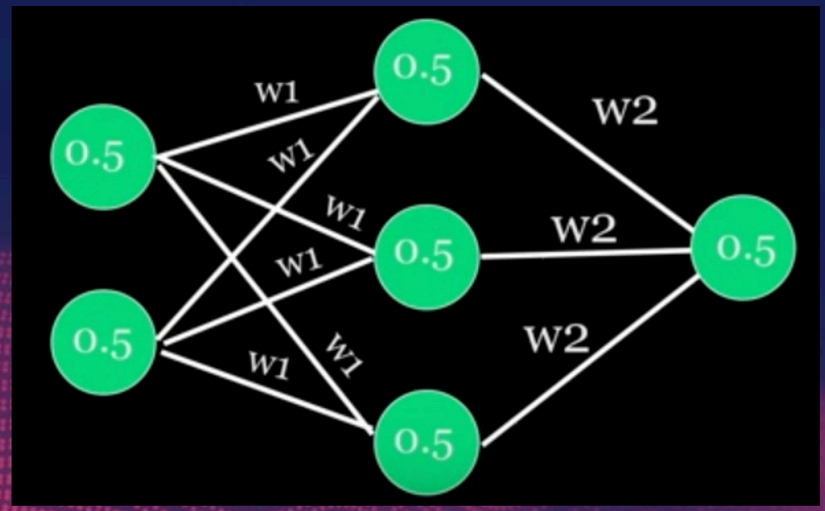
#8245 526

What if weights are not different?

- While initialising, Weights must NEVER be all equal.
- If they are equal then a form of symmetry happens in the network. This essentially prevents the neural network from identifying crucial patterns
- It is thus a standard practice to randomise all weights initially.

Note that bias does not face this issue and can have any value at the time of initialisation





Initialisation

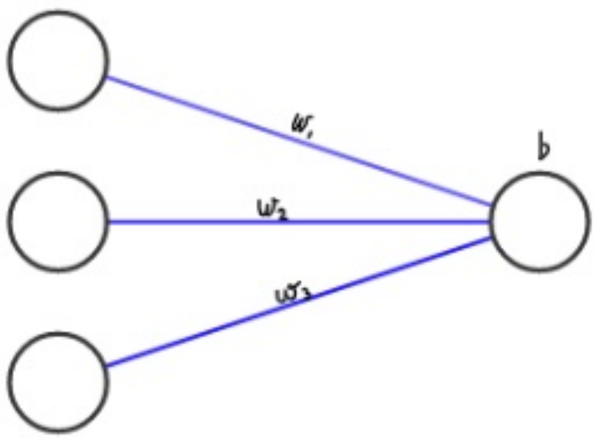
- The initial values of weights and biases can significantly impact training so we may begin by initialising them.
- Common practices include initializing weights randomly (e.g., from a normal distribution) to break symmetry and prevent neurons from learning the same features.

78

If all weights are initialized to be zero no learning can take place.

Forward propogation

- During forward propagation, input data is fed into the network. Each layer performs a weighted sum of inputs followed by an activation function, producing an output that serves as input to the next layer. The final layer's output represents the network's prediction.



Input Layer $\in \mathbb{R}^3$ Output Layer $\in \mathbb{R}^1$

Step 1:

$$y = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + bias$$

$$y = \sum_{i=1}^{n} \{(\omega_i * x_i) + b\}$$

Step 2:

$$z = y_{pred} = \frac{1}{1 + e^{-(\omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + b)}}$$

Loss Computation

- Here we calculate how far the network's prediction deviates from the actual target.
- This is achieved by comparing the actual value (y) with our current prediction (ŷ). We usually do this using mean absolute error or mean square error.
- If we manage to minimise this value over time, our model will gradually attain a decent level of accuracy.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

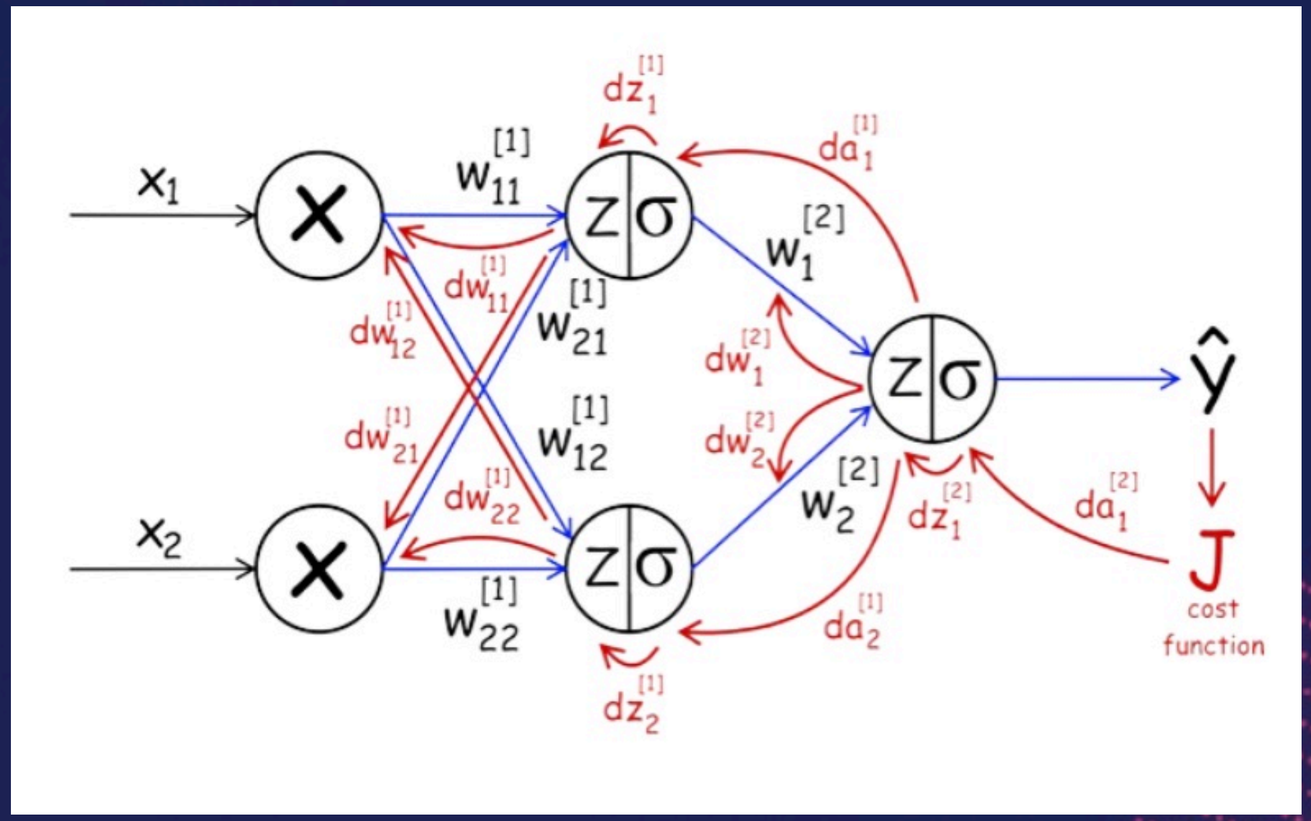
Where,

$\hat{y}$ – *predicted value of* y

$\bar{y}$ – *mean value of* y

Back Propogation

- We can observe and understand the error in the model from loss computation. The objective of back propogation is to change the weights in the direction that would lead to greatest decrease in error.
- We do it as shown below -

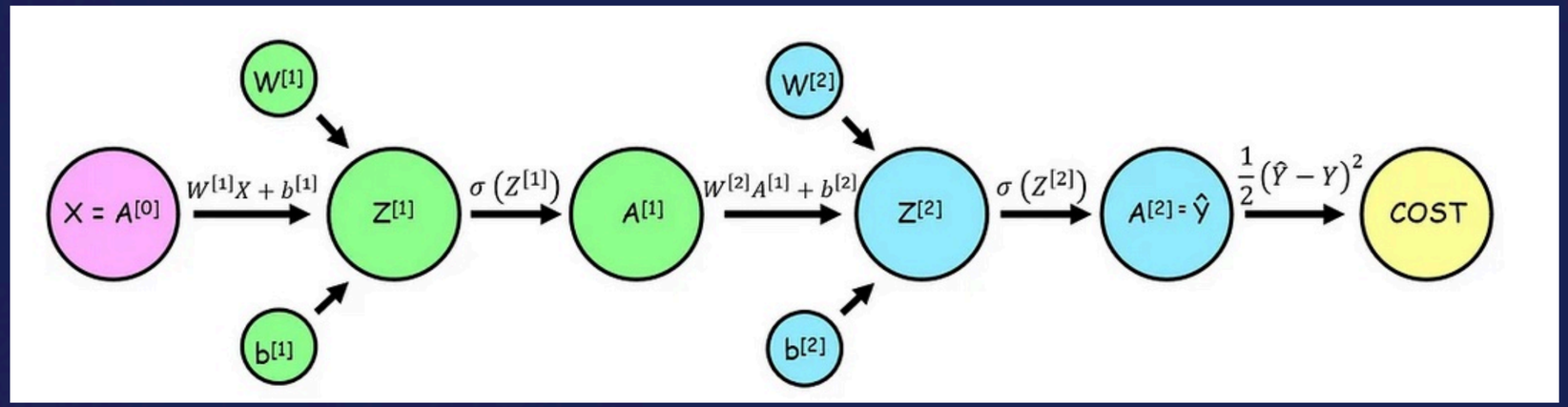


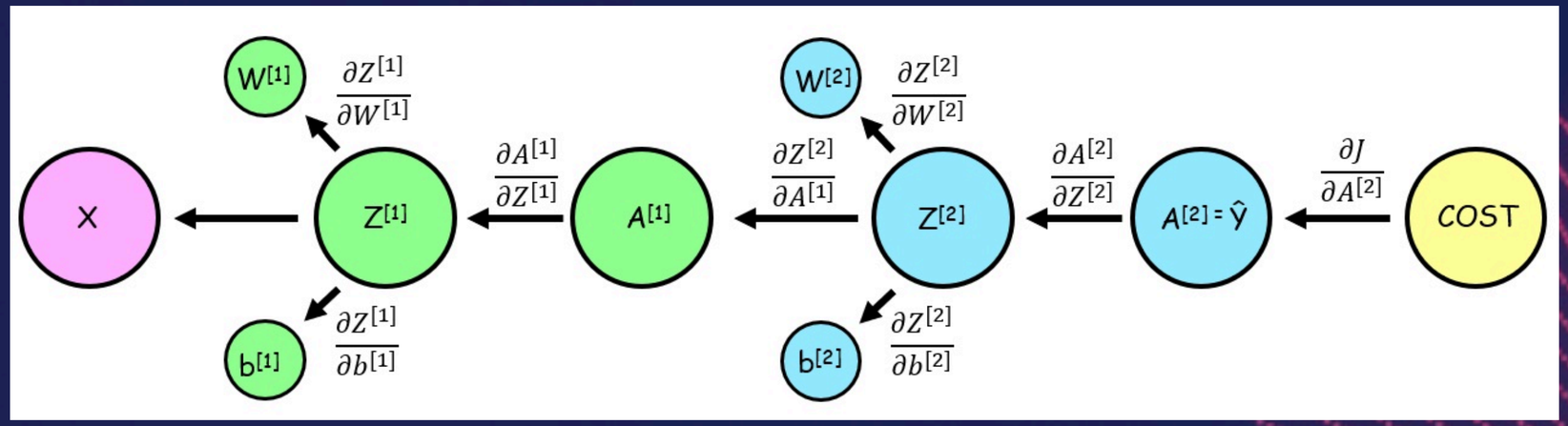
The idea is to walk backwards along each path to identify the change in weight/ bias required.

We use the concept of partial derivatives to acheive the same.

Back Propogation

Lets understand with an example





Here we are going to use sigmoid for the activation function and MSE to calculate the cost

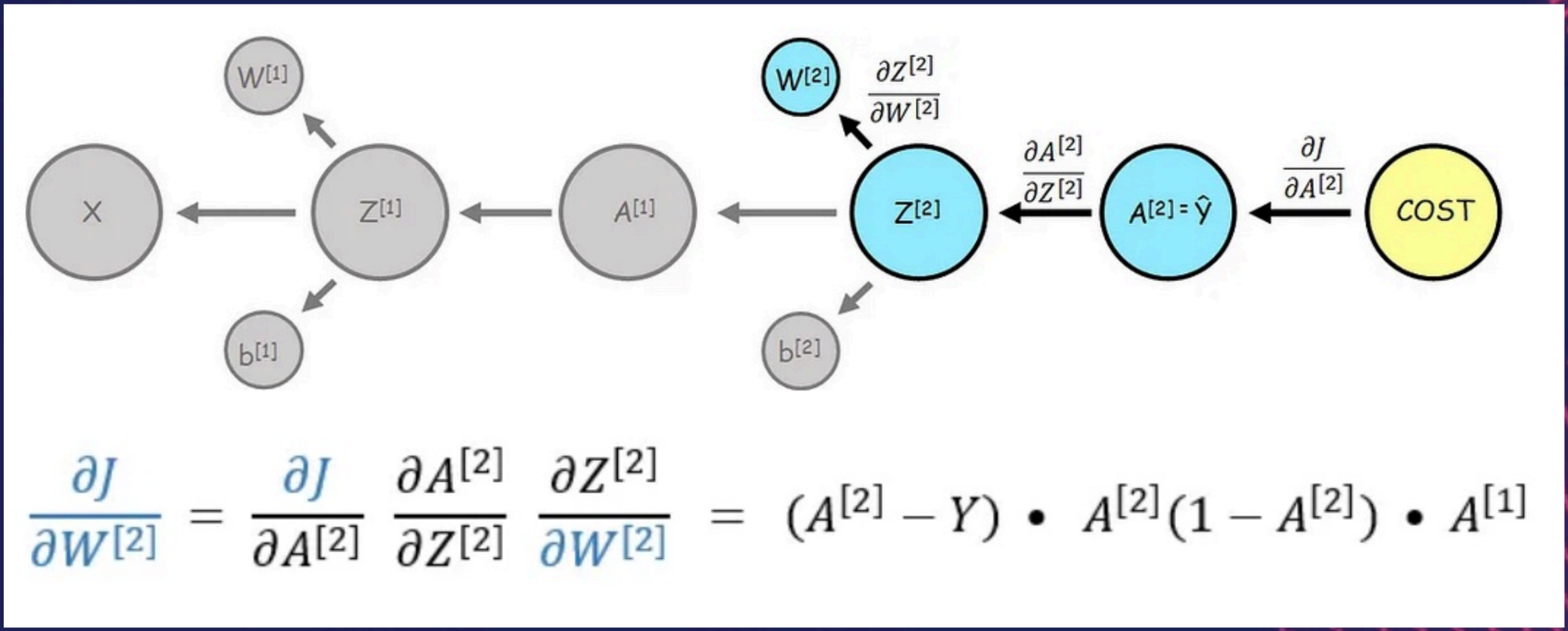
Back Propogation

First find partial derivatives and keep them ready for use

Original function:	Partial derivative:
$J = \frac{1}{2}\left(A^{[2]} - Y\right)^2$	$\frac{\partial J}{\partial A^{[2]}} = A^{[2]} - Y$
$A^{[2]} = \sigma\left(Z^{[2]}\right) = \frac{1}{1+e^{-Z^{[2]}}}$	$\frac{\partial A^{[2]}}{\partial Z^{[2]}} = A^{[2]}\left(1 - A^{[2]}\right)$
$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$	$\frac{\partial Z^{[2]}}{\partial W^{[2]}} = A^{[1]}$

Back Propogation

Corresponding to the path chosen find the partial derivatives and make use of the previous findings as need be

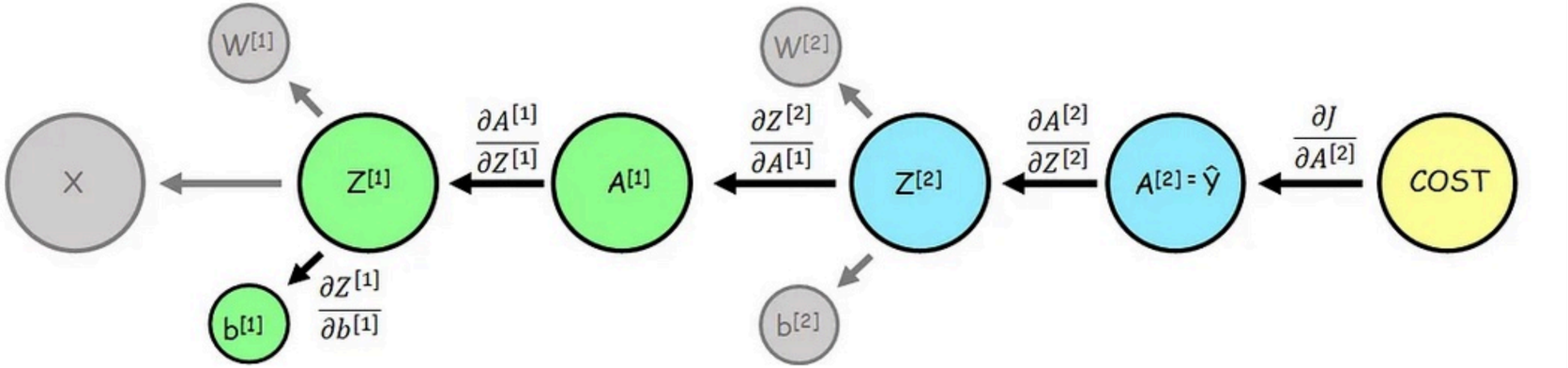


$$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial W^{[2]}} = (A^{[2]} - Y) \bullet A^{[2]}(1 - A^{[2]}) \bullet A^{[1]}$$

On solving this we will get the required correction in W^2.

Back Propogation

Corresponding to the path chosen find the partial derivatives and make use of the previous findings as need be



$$\frac{\partial J}{\partial b^{[1]}} = \underbrace{\frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}}}_{\text{Previously calculated}} \frac{\partial Z^{[2]}}{\partial A^{[1]}} \frac{\partial A^{[1]}}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial b^{[1]}} = (A^{[2]} - Y) \bullet A^{[2]}(1 - A^{[2]}) \bullet W^{[2]} \bullet A^{[1]}(1 - A^{[1]}) \bullet 1$$

On solving this we will get the required correction in b^1

Gradient Descent

- Using the gradients computed during backpropagation, the optimizer adjusts the weights and biases in the direction that reduces the loss.
- On calculating the gradient associated with all weights and biases, we simply need to begin updating their values

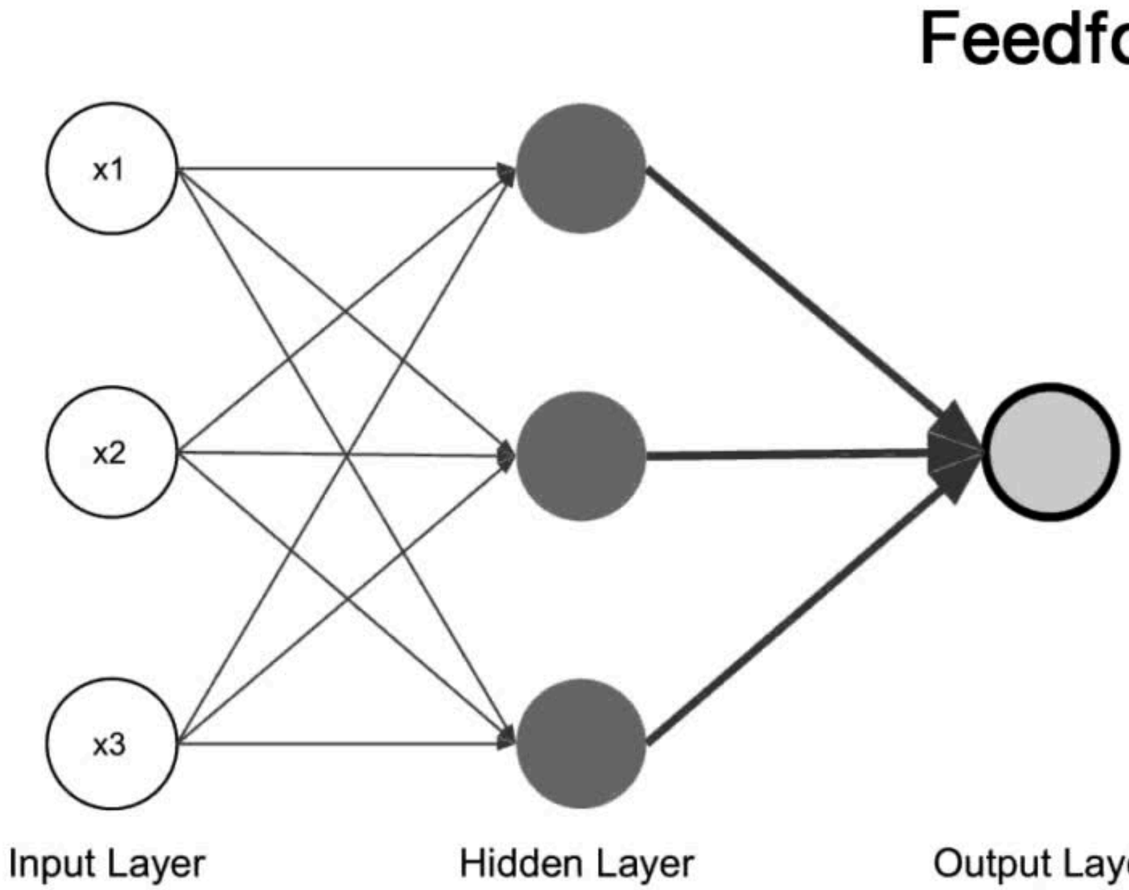
Taking the previous example we can find new weights and biases as-

$$W_{new}^{[1]} = W_{old}^{[1]} - \alpha \frac{dJ}{dW^{[1]}} \quad | \quad b_{new}^{[1]} = b_{old}^{[1]} - \alpha \frac{dJ}{db^{[1]}}$$

$$W_{new}^{[2]} = W_{old}^{[2]} - \alpha \frac{dJ}{dW^{[2]}} \quad | \quad b_{new}^{[2]} = b_{old}^{[2]} - \alpha \frac{dJ}{db^{[2]}}$$

We can now update these values

Summary



Batching

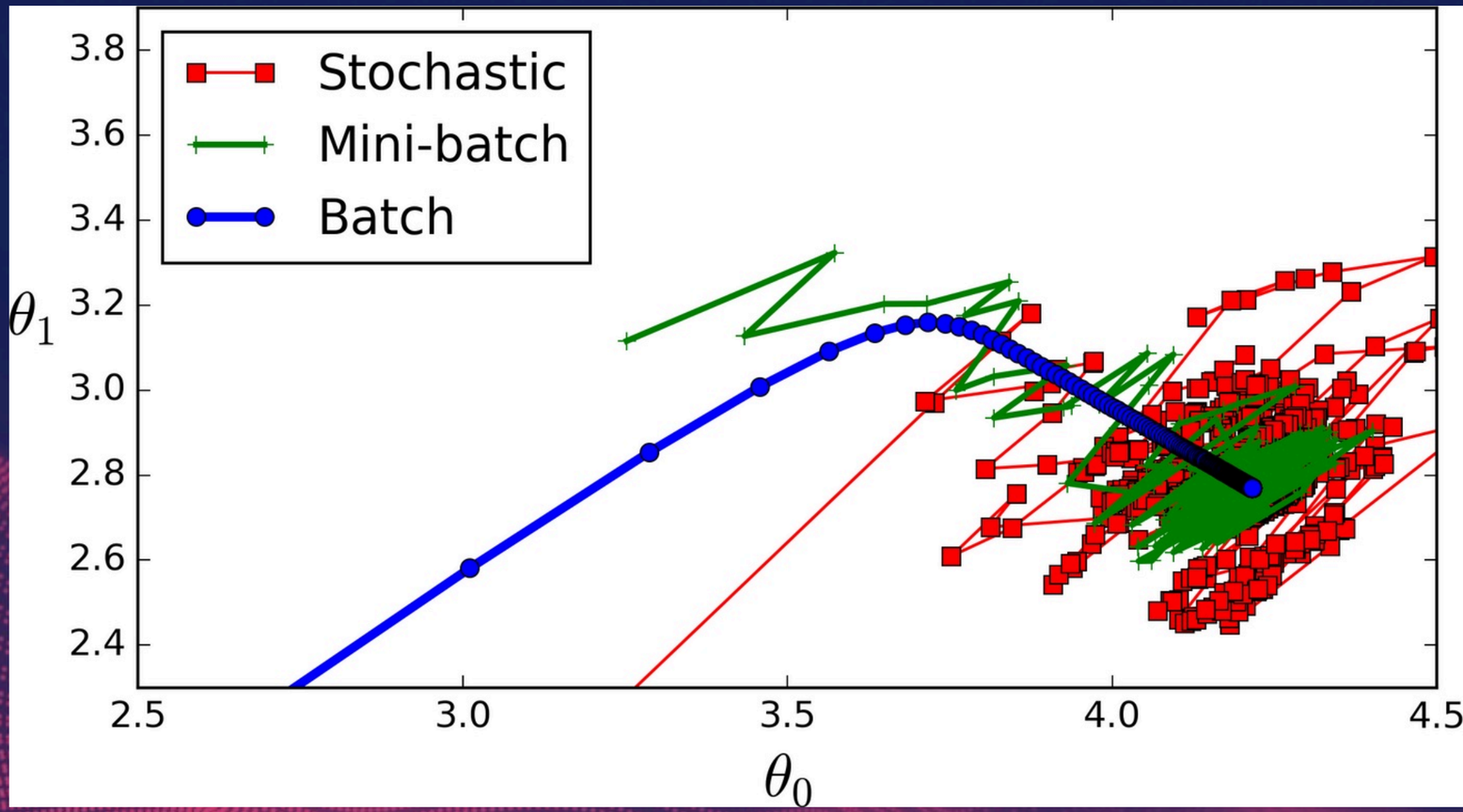
- A batch is a subset of the training data processed together by the neural network.
- Batching is crucial for efficient training of neural networks, especially with large datasets.

Example-
Lets say we are given 1000 training samples. Let us create batches of size 100.

We could thus make groups of 100 samples and use them to train the network. We could then go to the next 100 and so on. We are therefore not working with a full batch (batch size = 1000) or stochastic case (batch size =1).

- Creating batches helps us avoid the ‘chaos’ associated with purely Stochastic Gradient descent. It helps deal with extreme data points that may exist within the dataset

- Creating batches is also found to be useful from a computational standpoint when it comes to speed and memory management
- However, The most important use of batching is faster convergence that can be achieved using it



Code Implementation

Pytorch Fundamentals

What is Pytorch?

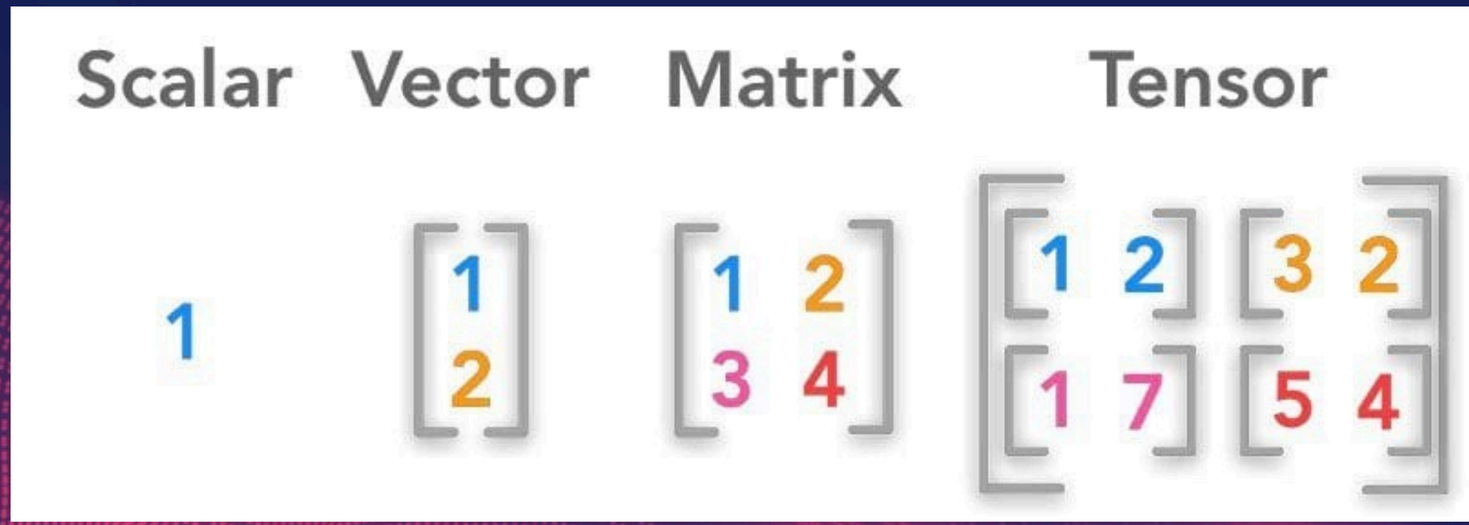
- PyTorch is a machine learning library created with the objective of simplifying the implementation and training of ML models.
- It is an open source framework i.e its source code is open to public.
- It was created by Facebook's AI Research lab, led by Yann LeCun

Why Pytorch?

- PyTorch is based of python, making it easy to learn and use.
- It supports dynamic computational graphs, allowing for changes in network behavior at runtime, which adds flexibility.
- PyTorch offers well-organized, open-source documentation.

Tensors

- The central component of PyTorch is the tensor data structure.
- Tensors are similar to NumPy n-dimensional arrays but are capable of running on hardware accelerators like GPUs (CUDA-capable).
- Tensors are optimized for automatic differentiation, which is essential for backpropagation.



nn.module

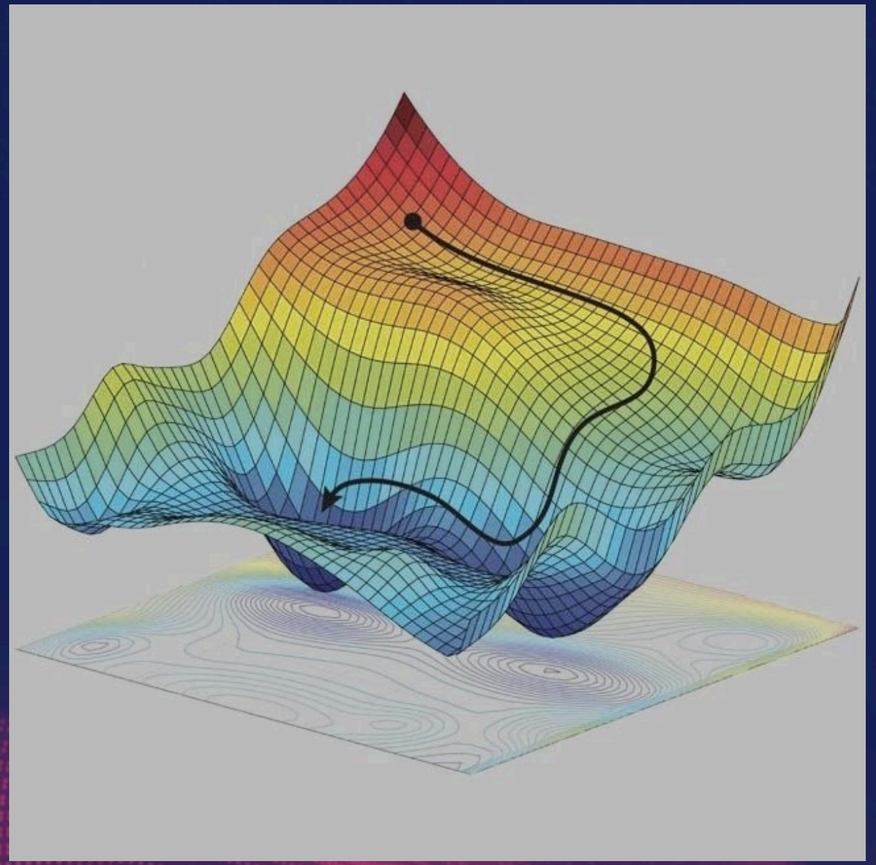
- This is the core of PyTorch.
- It serves as the base class for developing all neural network models.
- It enables easy creation of linear neural networks, convolutional neural networks (CNNs), and more complex architectures by combining different types of networks.

Torch.device()

- torch.device() specifies the device (CPU or GPU) where tensors and models will be allocated.
- You can specify a device using a string, such as 'cpu' or 'cuda' (for GPU).
- This helps in leveraging GPU acceleration for faster computations in deep learning tasks.

Torch.optim()

- torch.optim() is a module in PyTorch that provides optimization algorithms for training neural networks.
- You can create an optimizer object by passing parameters like model parameters and learning rate.
- The optimizer updates the parameters of the model during training based on the computed gradients.



Code Implementation

ATTENDANCE

Convolutional Neural Networks

PIXELS

Pixel, Pel or Picture Element is defined as a minute area of illumination on a display screen, one of many from which an image is composed . A pixel can be regarded as the “atom” of an image.

Each Pixel in an image has a numerical value / values associated with it, and its colour is determined by this value / values (In case of colour images).

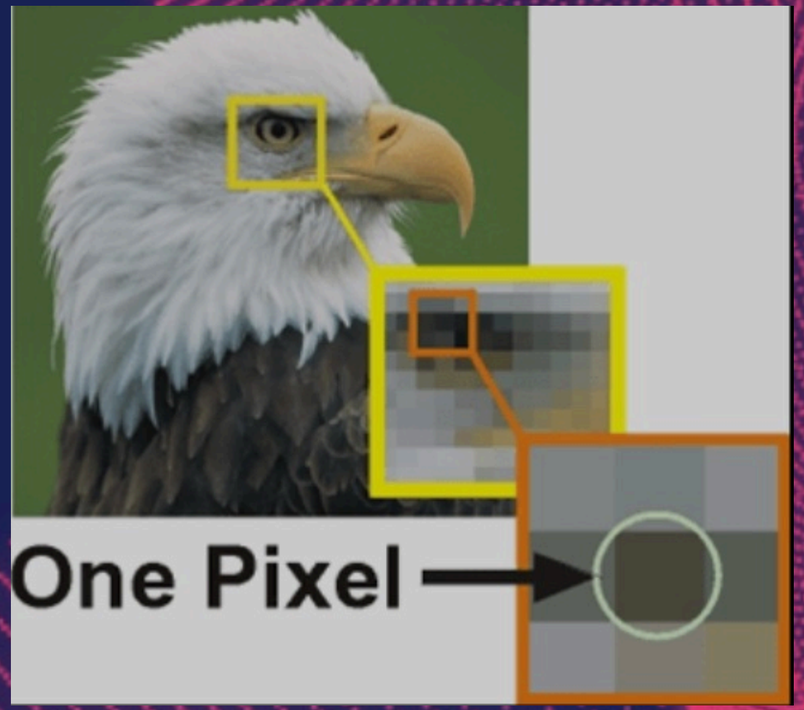


Image Data

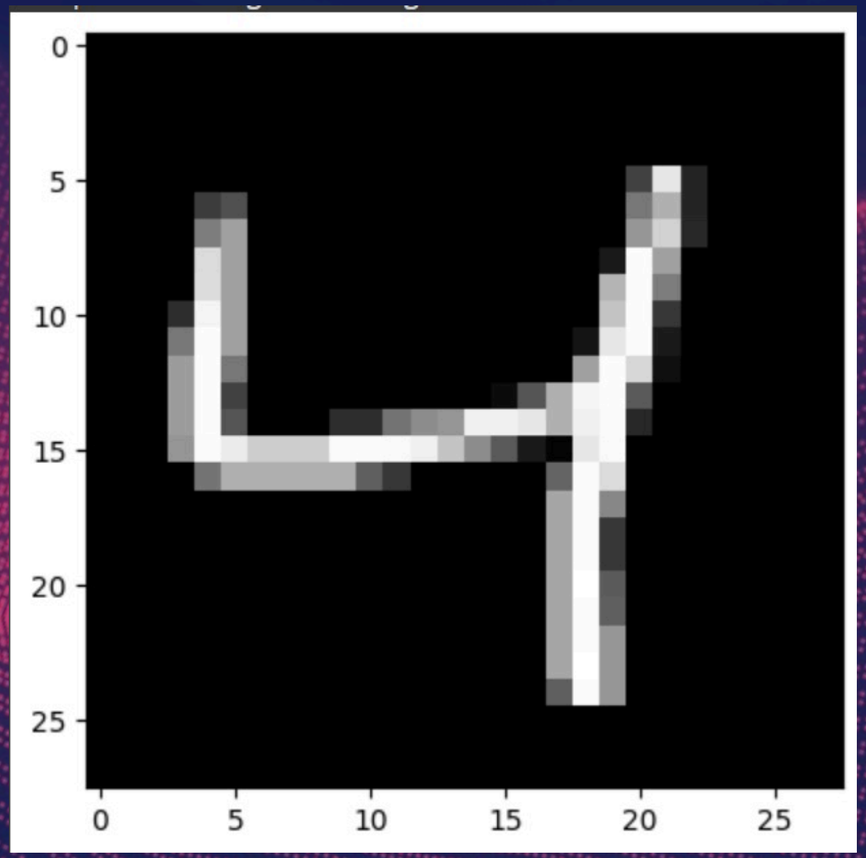
Image data in python is represented by arrays or matrices of pixel values.

Image data can be broadly categorised into two main types :

1) Gray Scale Images

2) RGB or Colour Images

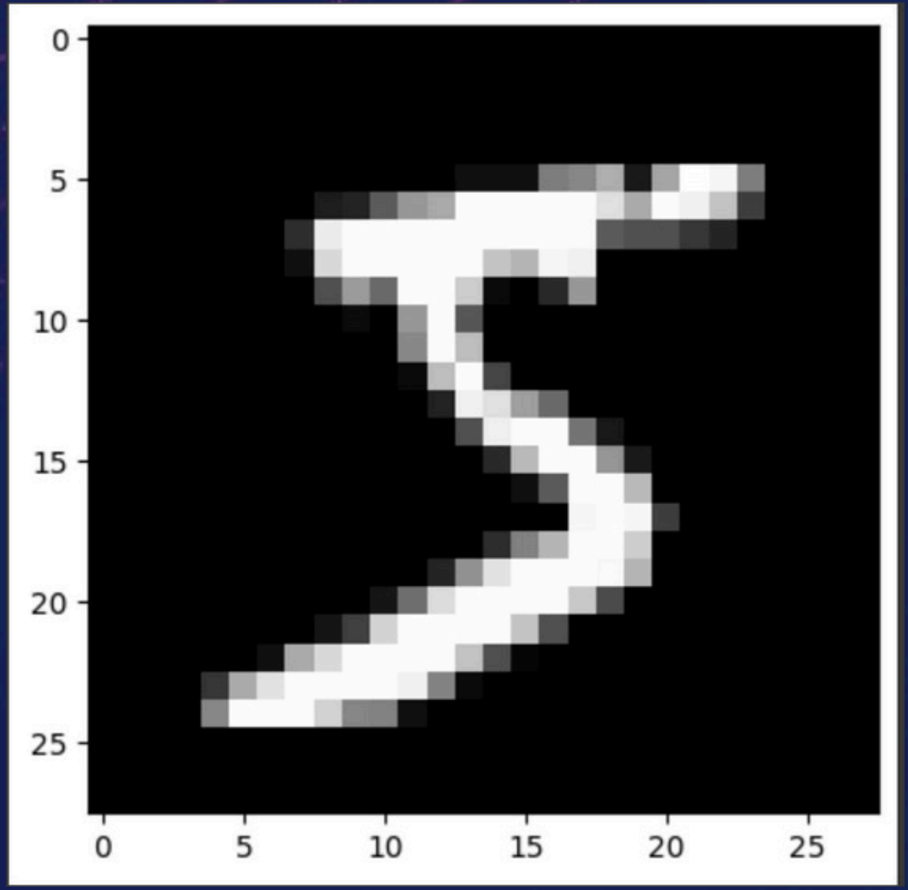
GRAY SCALE IMAGES



A Gray Scale Image is basically a Black and White Image or a Gray Monochrome, which is composed of different shades of Gray.
In such images, the colour of a pixel is determined by a single pixel value, which is an integer in the range [0, 255] (8 - bit grayscale image) .

GRAY SCALE IMAGES

The pixel value of ‘0’ denotes Black colour whereas a pixel value of ‘255’ denotes white. The intermediate Integers are different shades of Gray (ranging from darkGray to light Gray).

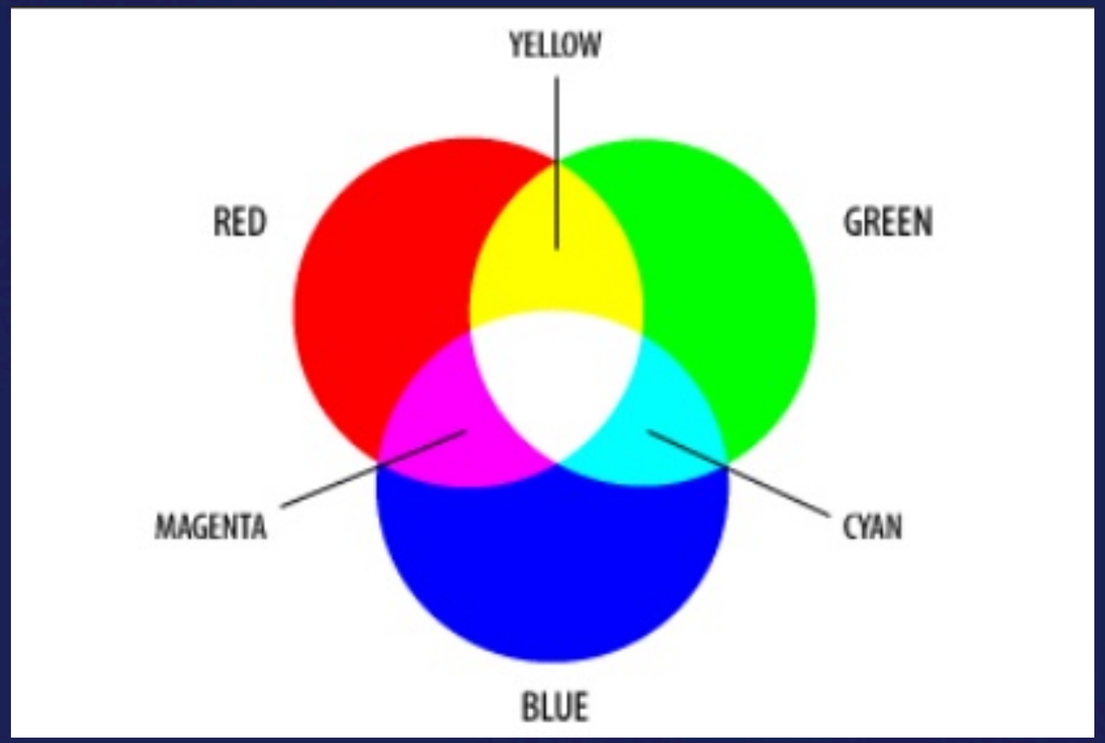



```
tensor([[  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           3.,  18.,  18.,  18., 126., 136., 175.,  26., 166., 255., 247., 127.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  30.,  36.,  94., 154.,
         170., 253., 253., 253., 253., 253., 225., 172., 253., 242., 195.,  64.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,  49., 238., 253., 253., 253.,
         253., 253., 253., 253., 253., 251.,  93.,  82.,  82.,  56.,  39.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,  18., 219., 253., 253., 253.,
         253., 253., 198., 182., 247., 241.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  80., 156., 107., 253.,
         253., 205.,  11.,   0.,  43., 154.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  14.,   1., 154.,
         253.,  90.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 139.,
         253., 190.,   2.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  11.,
         190., 253.,  70.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          35., 241., 225., 160., 108.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,  81., 240., 253., 253., 119.,  25.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
           0.,   0.,  45., 186., 253., 253., 150.,  27.,   0.,   0.,   0.,   0.,
           0.,   0.,   0.,   0.],
        [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
```

COLOUR IMAGES

Colour Images are images which contain multiple colour channels per pixel. Unlike Grayscale images that have a single channel typically representing intensity (varying from Black to White), colour images represent each pixel with a combination of colours.

COLOUR IMAGES

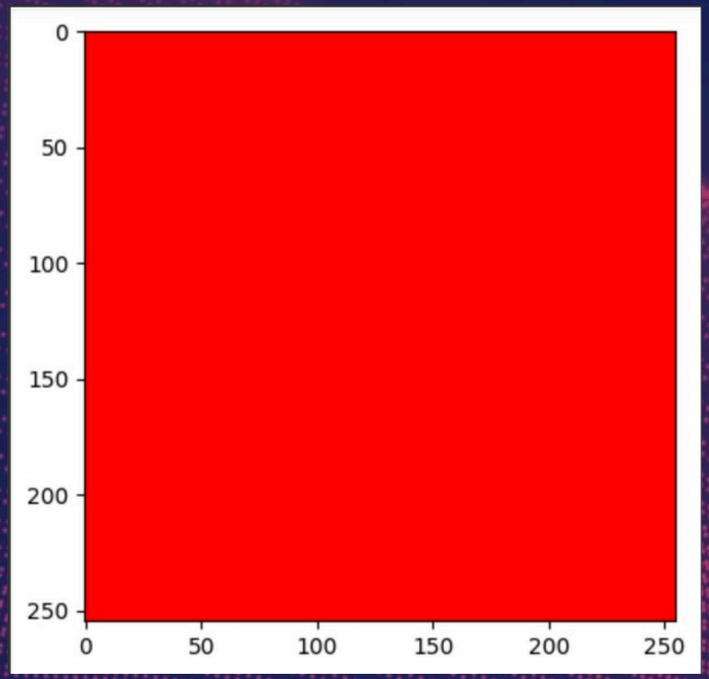


Key Characteristics :
1) Channels - Colour images are primarily composed of three colour channels, Red (R), Green (G) and Blue(B). All other colours are produced as a combination of these three colours.
2) Each pixel in the colour image is expressed as a combination of intensities from these three channels. The intensity values typically range from 0 - 255 for each channel.

COLOUR IMAGES

To reiterate the previous slide :

Image :



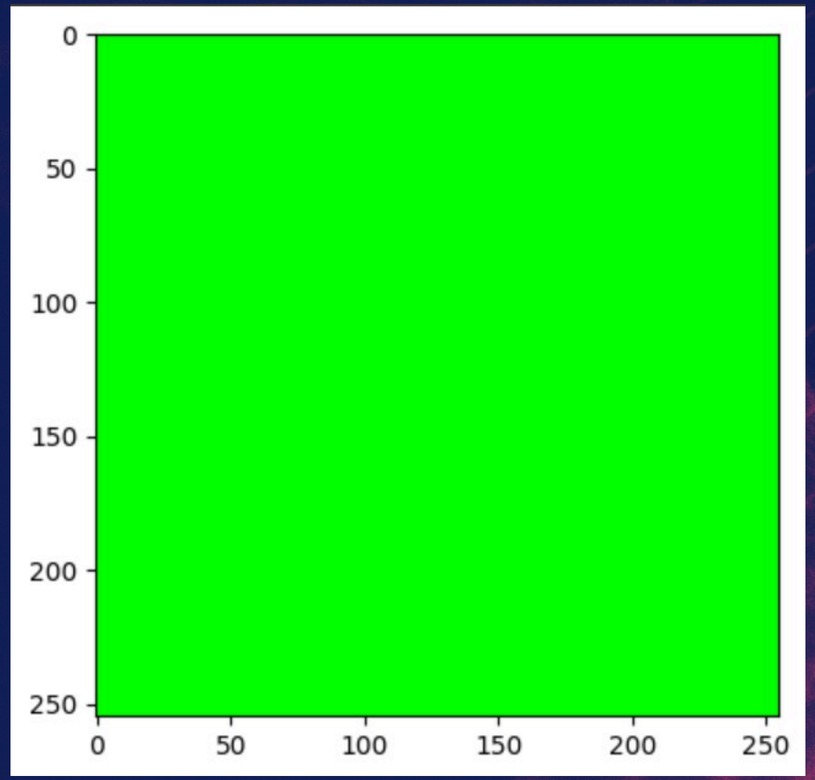
Pixel - Representation

```
(3, 255, 255)
[[[255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]
  ...
  [255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]]

 [[  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  ...
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]]

 [[  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  ...
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]]]
```

COLOUR IMAGES

Another Example :

Image :

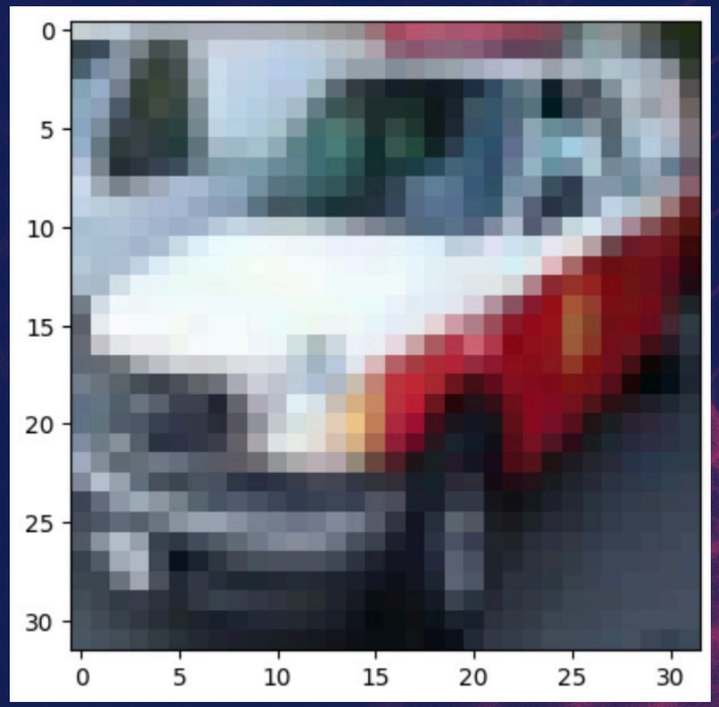


Pixel - Representation

```
(3, 255, 255)
[[[  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  ...
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]]

 [[255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]
  ...
  [255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]
  [255. 255. 255. ... 255. 255. 255.]]

 [[  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  ...
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]
  [  1.   1.   1. ...   1.   1.   1.]]]
```

COLOUR IMAGES

The colour of each pixel in the image is produced by a combination of ‘Red’ , ‘Green’ and ‘Blue’ according to their respective intensity values which lie in the set {0, 1, 2, 3 ..... 255} .




```
torch.Size([3, 32, 32])
tensor([[[201.0000, 191.0000, 194.0000,  ...,  83.0000,  39.0000,  36.0000],
         [ 57.0000,  62.0000, 134.0000,  ...,  79.0000,  48.0000,  35.0000],
         [ 74.0000, 123.0000, 138.0000,  ..., 162.0000, 132.0000,  56.0000],
         ...,
         [ 67.0000,  62.0000,  55.0000,  ...,  69.0000,  72.0000,  72.0000],
         [ 73.0000,  67.0000,  59.0000,  ...,  72.0000,  72.0000,  71.0000],
         [ 74.0000,  71.0000,  67.0000,  ...,  61.0000,  58.0000,  63.0000]],

        [[209.0000, 204.0000, 207.0000,  ...,  88.0000,  48.0000,  47.0000],
         [ 73.0000,  78.0000, 148.0000,  ...,  91.0000,  59.0000,  45.0000],
         [ 99.0000, 142.0000, 153.0000,  ..., 175.0000, 139.0000,  61.0000],
         ...,
         [ 80.0000,  72.0000,  62.0000,  ...,  78.0000,  80.0000,  82.0000],
         [ 87.0000,  77.0000,  67.0000,  ...,  81.0000,  81.0000,  81.0000],
         [ 89.0000,  84.0000,  79.0000,  ...,  73.0000,  69.0000,  75.0000]],

        [[211.0000, 210.0000, 216.0000,  ...,  82.0000,  33.0000,  24.0000],
         [ 79.0000,  92.0000, 161.0000,  ...,  96.0000,  57.0000,  32.0000],
         [110.0000, 165.0000, 169.0000,  ..., 186.0000, 145.0000,  56.0000],
         ...,
         [ 89.0000,  82.0000,  73.0000,  ...,  93.0000,  95.0000,  98.0000],
         [ 95.0000,  87.0000,  77.0000,  ...,  96.0000,  96.0000,  96.0000],
         [100.0000,  95.0000,  89.0000,  ...,  88.0000,  85.0000,  91.0000]]])
```

WHY CNNS ?

IMAGE CLASSIFICATION

Q) What is the need for CNNs (Convolutional Neural Networks) ? Why not just flatten the image at hand and pass it through a neural network ?

A) With today's level of technology, an image of resolution '256 * 256' is also considered as low resolution. The flattened vector for such an image will contain '65,536' elements in case of Gray Scale images and will have '1,96,608' elements in case of a colour image. It is not suitable to have such a high dimensional vector as input to our Neural Network.

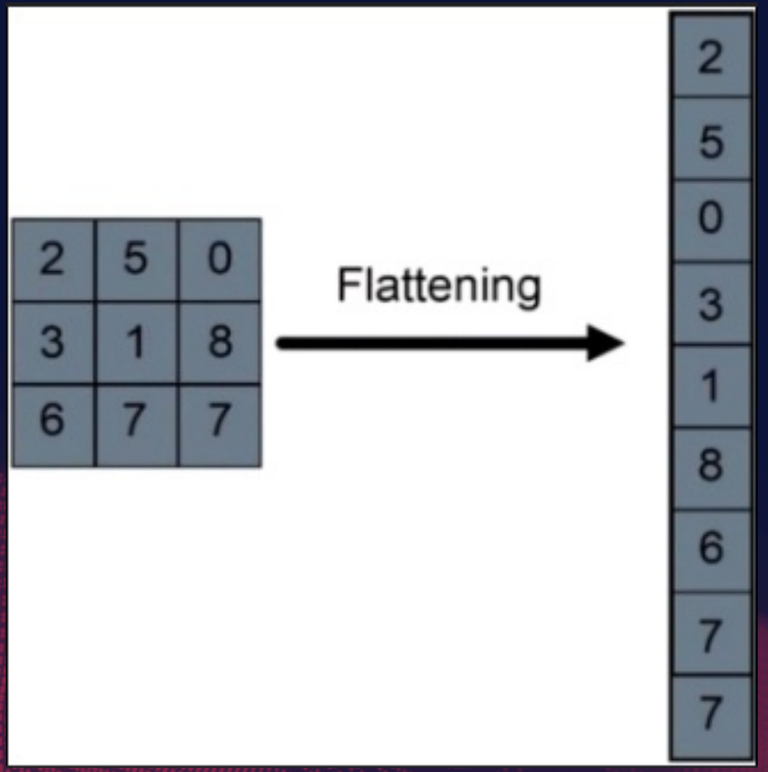


IMAGE CLASSIFICATION

Q) Why are high - dimensional vectors not suitable to be inputs to Neural Networks ?

A) 1. High dimensional inputs increase the number of neurons in the input layer.

2. This leads to an increase in the number of parameters in the Neural Network.

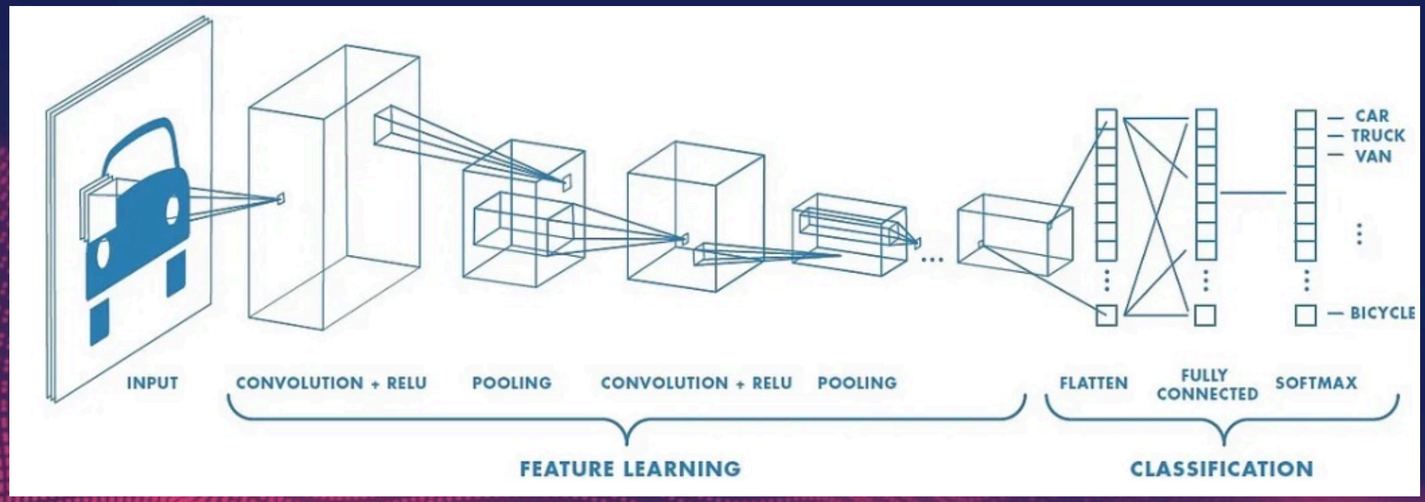
3. An increase in the number of parameters of the Neural Network can cause Overfitting and will make the training of the Network computationally inefficient .

CNN

Convolutional Neural Network is a model architecture developed for tasks related to image data such as image classification.

The basic units of a CNN are the following :

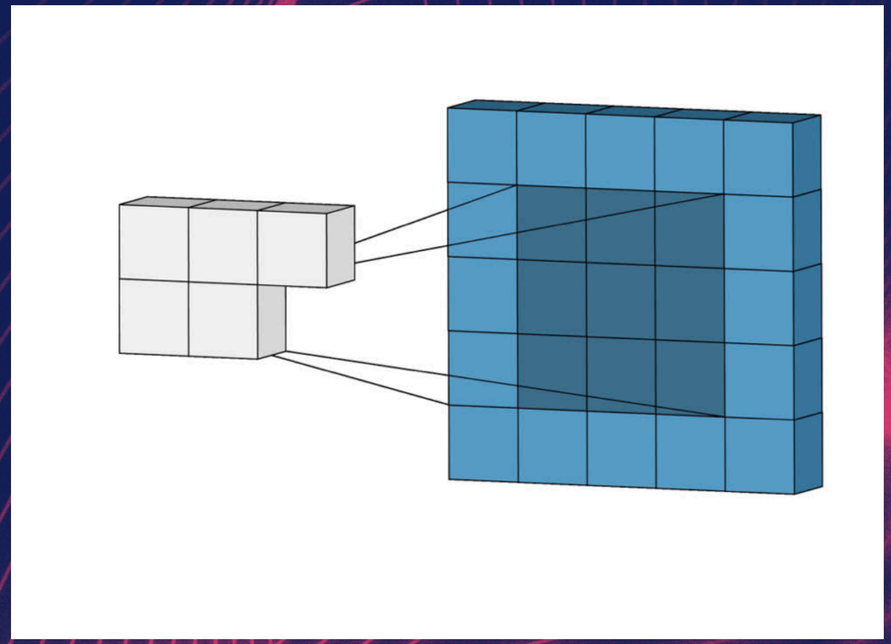
1) Convolutional Layers
2) Activation Functions
3) Pooling Layers
4) Fully Connected Layers



CONVOLUTIONAL LAYERS

The Convolution operation mainly involves the sliding of another matrix over a channel of our input image. At each point, a dot product operation is performed between the sliding matrix and the area of the input channel which the matrix covers.

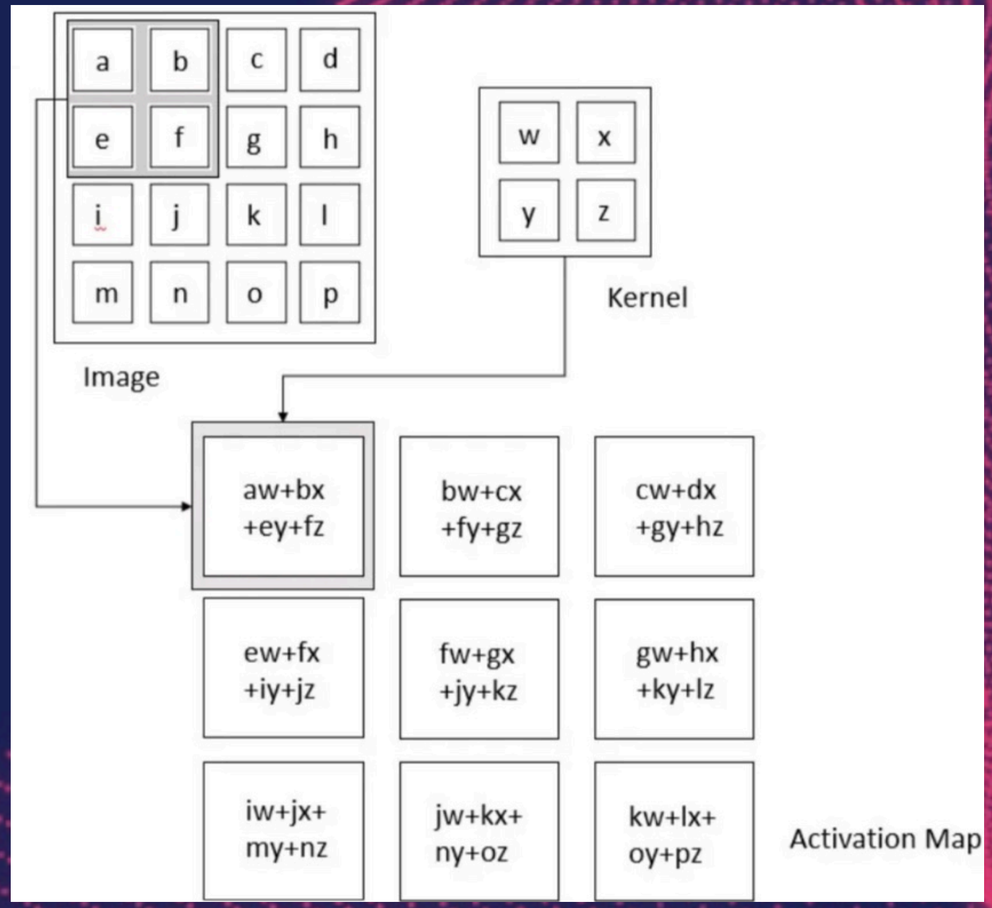
We shall now see some of the related terminology.



CONVOLUTIONAL LAYERS

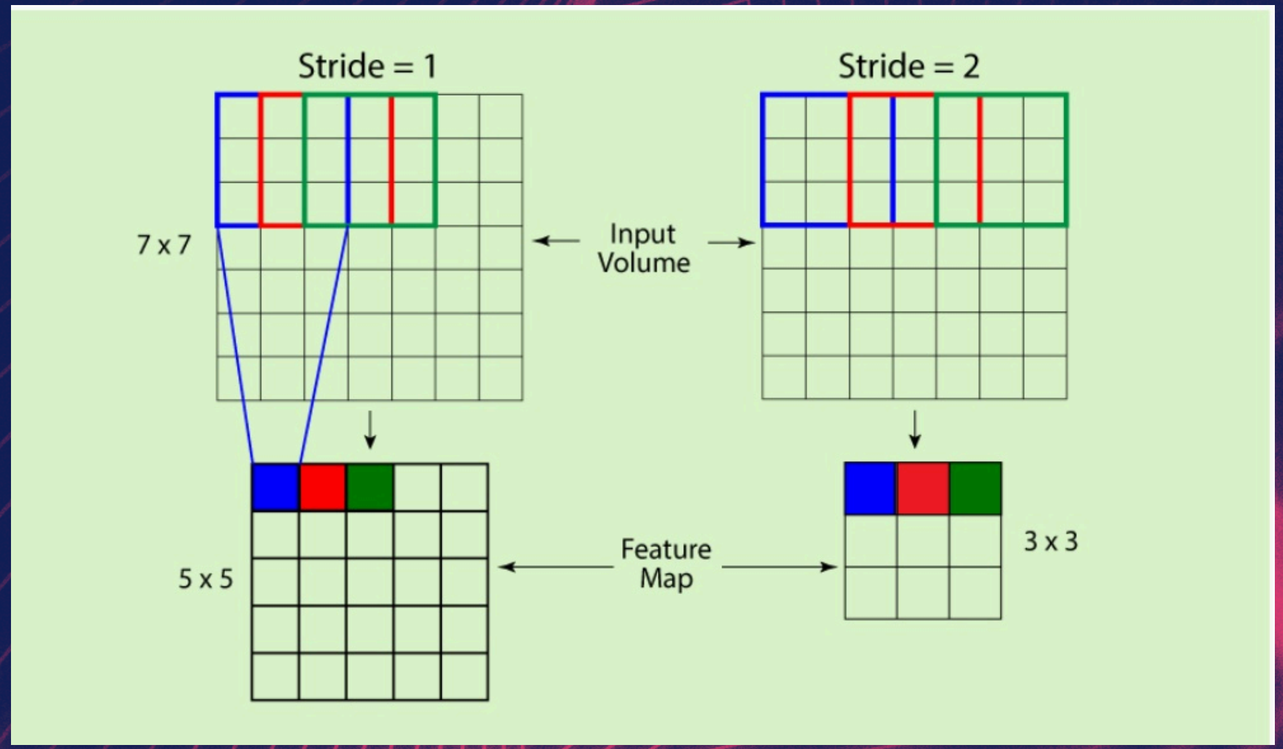
1) Kernel : This is the smaller matrix which is slid over the larger image channel. The dot - product operation is performed between this smaller matrix and the pixels of the image channel that it covers.

2) Activation Map : The two dimensional representation of the image obtained after the Kernel is done sliding across the height and width of the image is the Activation Map.

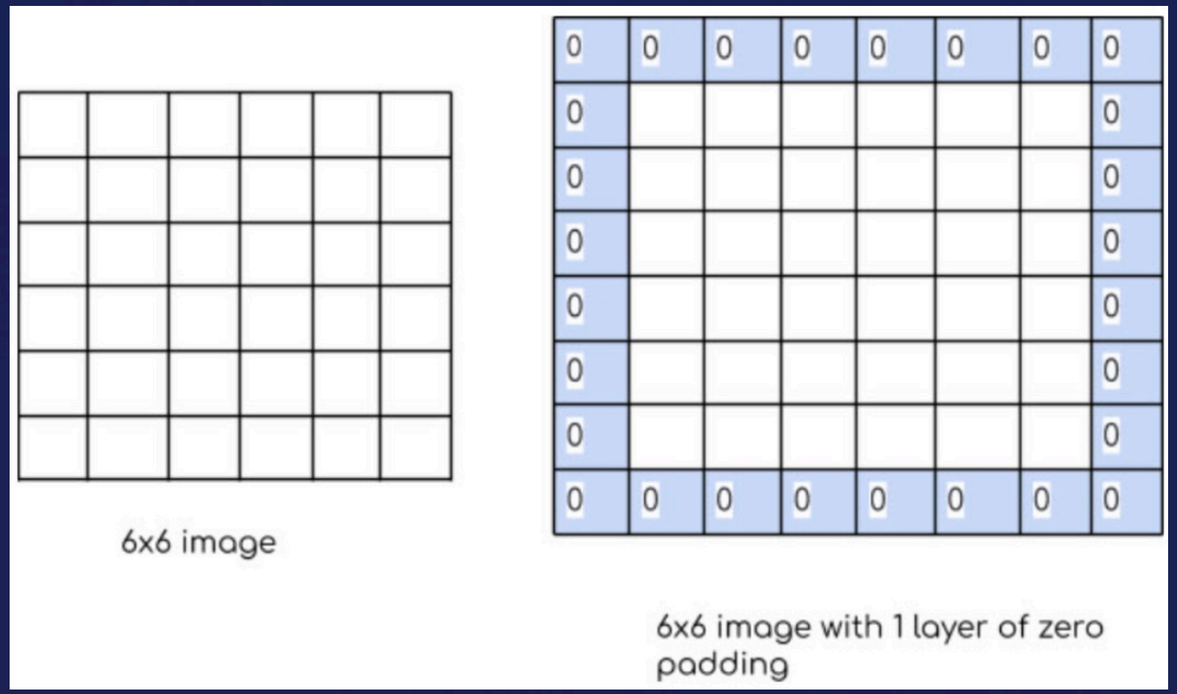


CONVOLUTIONAL LAYERS

3) Stride : This parameter determines the number of pixels by which the kernel moves or slides. It determines how much the filter shifts or moves after each dot - product operation.



CONVOLUTIONAL LAYERS



4) Padding : The most common type of padding is zero - padding, in which layers of pixels containing only zeros are added to the border of the image. This is done to control the size of the Activation Map produced.

QUESTION

#3925 093

CONVOLUTIONAL LAYERS

There is a simple formula which relates the dimensions of the output Activation Map, as a function of the input channel dimension, stride, padding and kernel size.

It is given as follows :

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1$$

Where:

N out = Activation Map's dimension

N in = input channel dimension

p= padding

s = stride

CONVOLUTIONAL LAYERS

When we initialise a Convolutional Layer in Python, apart from Kernel size, padding, stride etc, we have to pass in two more arguments which are :

1) in_channels 2) out_channels

```
layer_1 = nn.Conv2d (in_channels = 3, out_channels = 6, kernel_size = 3, padding = 1, stride = 1)
```

CONVOLUTIONAL LAYERS

5) in_channels :

Common Misconception -> Only a single image channel can be passed to a Convolutional Layer.

Reality -> Many image channels can be passed at once to a Convolutional layer.

Eg : when we pass in a colour image, in_channels = 3

Another misconception is that each Convolutional layer has only a single Kernel. Actually, there are as many Kernels as the number of input channels, for each output channel.

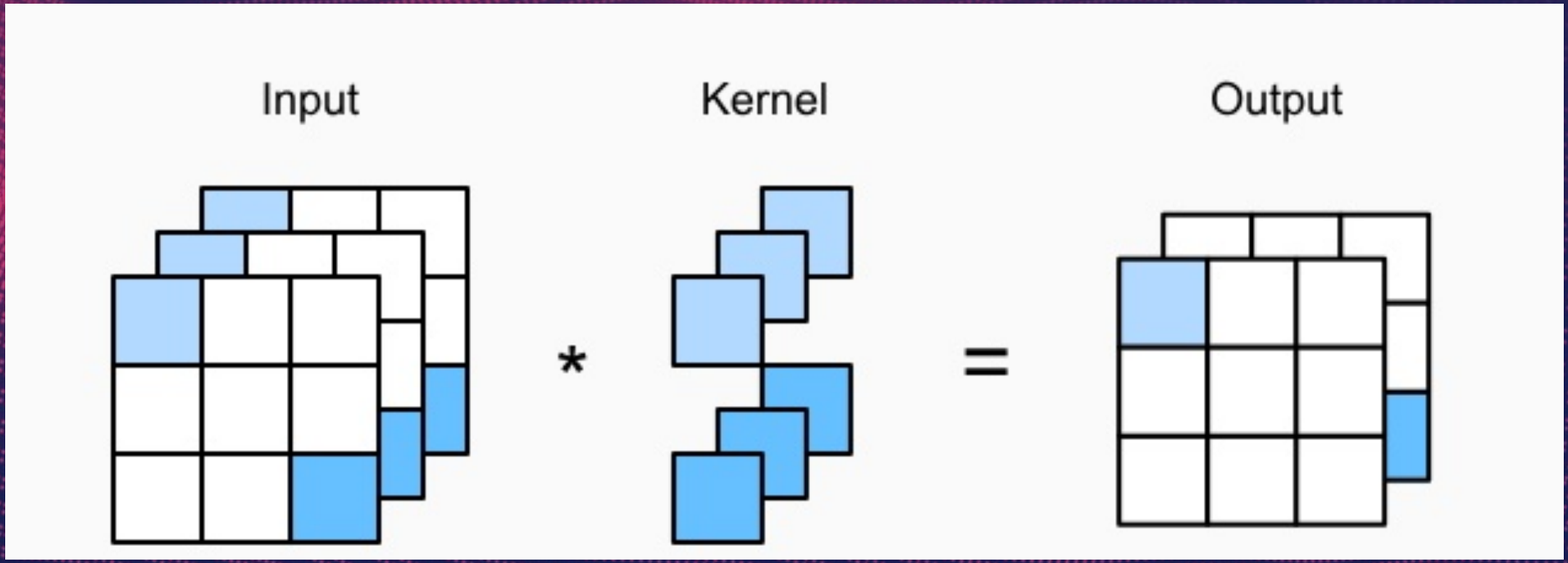
CONVOLUTIONAL LAYERS

6) out_channels :
The number of Activation Maps or channels that are to be outputted by the Convolutional Layer.

This helps in determining the total number of Kernels as well as the number of bias terms contained in the Convolutional Layer.

CONVOLUTIONAL LAYERS

Pictorial representation of in_channels and out_channels :



CONVOLUTIONAL LAYERS

How the output feature channels are produced in case of multiple in and out channels :

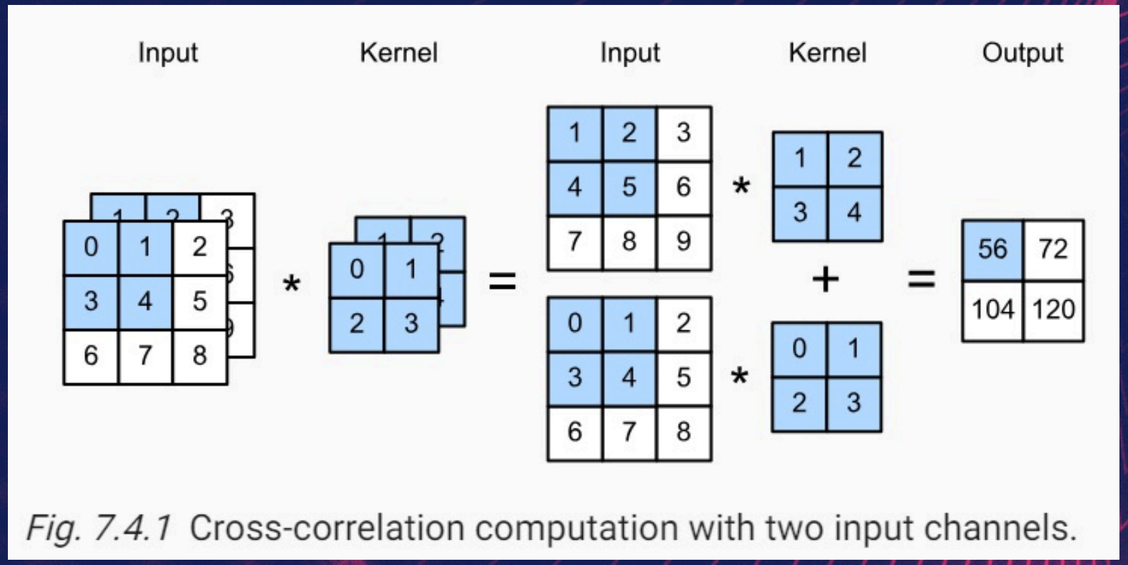


Fig. 7.4.1 Cross-correlation computation with two input channels.

The above image shows the calculation of a single output actiavtion map when the number of input channels = 2.

CONVOLUTIONAL LAYERS

The example on the previous slide reiterated our statement that, the number of Kernels in a Convolutional layer for each output channel

= the number of input channels

Hence :

total no of Kernels in a layer = in_channels * out_channels

q) What is the point of so many Kernels ?

A) Each Kernel serves as a means of feature extraction, with there being specific Kernels for edge detection, smoothening of images etc.

CONVOLUTIONAL LAYERS

There is also a Bias parameter associated with each output channel.

No of bias parameters = out_channels

The Bias parameter associated with an output channel is basically added to each element of that channel.

This helps to produce non - zero outputs in cases where the region of the kernel in the input channel is filled with ‘0’ pixels .

ACTIVTION FUNCTIONS

Activation Functions are applied to the Activation Maps outputted by the Convolutional layer. This just means that, each ‘entry’ of each Activation map is replaced by ‘f(entry)’, where ‘f’ denotes the Activation function used.

The Activation Functions used here are the same as the ones used in Artificial Neural Networks.

Their use here also is to help the model learn non - Linear and complicated relations.

POOLING LAYERS

Pooling Layers in a CNN perform dimensionality reduction, while preserving as much information as possible.

There are different types of Pooling such as -

1) Max - Pooling

2) Average - Pooling

Max - Pooling is the more popularly used of the two.

MAX - POOLING

Max - Pooling involves a Kernel of a certain dimension which slides over the image channel.

Hence, for a Max - Pool layer also, we need to specify the parameters :

1) Kernel Size

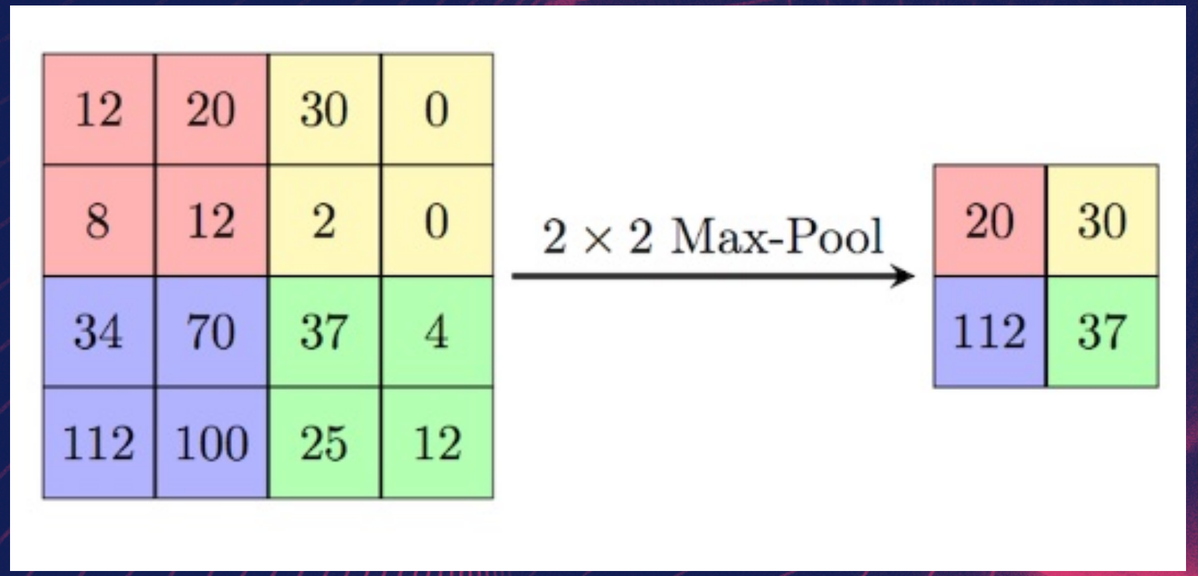
2) Stride

MAX - POOLING

How Max - Pooling happens :

1) The Kernel of specified size slides over the image channels.

2) The maximum of all the elements in the image channel, present in the area covered by the Kernel, is an entry for the output channel, given after the Max - Pooling layer.



A Max-Pooling operation with a Kernel size of 2*2 and stride = 2

FULLY CONNECTED LAYERS

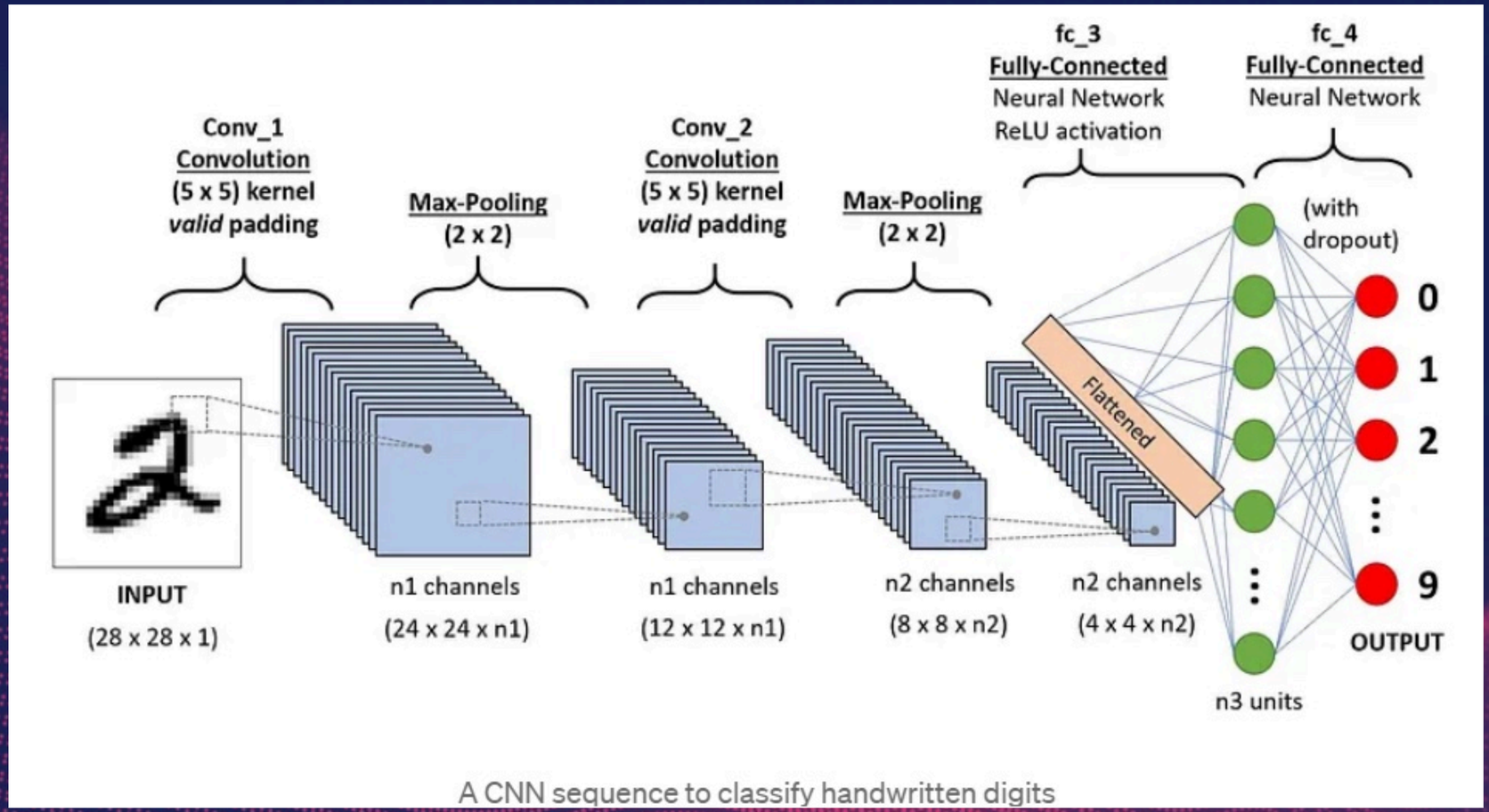
At the end of a Convolutional Neural Network, after all the Convolutional and Max - Pooling layers, fully connected layers or an Artificial Neural Network is present .

The content of all the channels that we have are flattened and joined to form a 1-D vector, which is then passed into the Neural Network.

In case of Classification tasks, the final layer of the ANN will have as many nodes as the number of classes and will have a Softmax Activation function to output probabilities.

CNN

A pictorial representation of all the components of a CNN :



A CNN sequence to classify handwritten digits

Code Implementation

ATTENDANCE