



PYTHON PROGRAMMING & DATA SCIENCE

UNIT - V

SYLLABUS



Introduction to Deep Learning: Multilayer perceptron. Back propagation. Loss functions. Hyper parameter tuning, Overview of RNN, CNN and LSTM.

Overview of Data Science Models: Applications to text, images, videos, recommender systems, image classification, Social network graphs.

Deep Learning



Deep Learning

Deep learning is a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.

➤ Deep learning models are capable enough to focus on the accurate features themselves by requiring a little guidance from the programmer and are very helpful in solving out the problem of dimensionality.

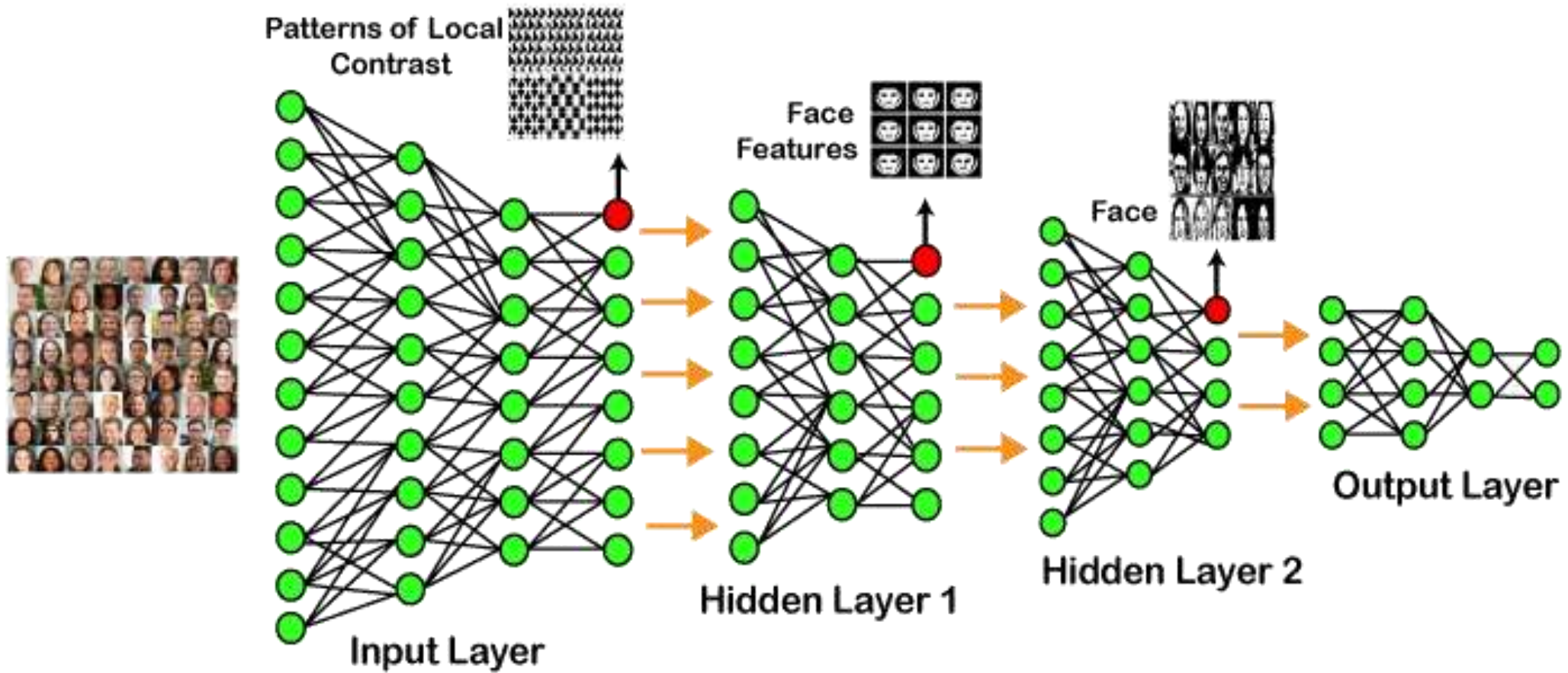
➤ Deep learning algorithms are used, especially when we have a huge no of inputs and outputs

Deep Learning



- Deep learning is implemented with the help of Neural Networks, and the idea behind the motivation of Neural Network is the biological neurons, which is nothing but a brain cell.
- “Deep learning is a collection of statistical techniques of machine learning for learning feature hierarchies that are actually based on artificial neural networks.”

Example of Deep Learning



Example of Deep Learning



In the example given above, we provide the raw data of images to the first layer of the input layer.

After then, these input layer will determine the patterns of local contrast that means it will differentiate on the basis of colors, luminosity, etc.

Then the 1st hidden layer will determine the face feature, i.e., it will fixate on eyes, nose, and lips, etc. And then, it will fixate those face features on the correct face template.

So, in the 2nd hidden layer, it will actually determine the correct face here as it can be seen in the above image, after which it will be sent to the output layer.

Likewise, more hidden layers can be added to solve more complex problems, for example, if you want to find out a particular kind of face having large or light complexions.

So, as and when the hidden layers increase, we are able to solve complex problems

Types of Deep Learning Networks



Types of Deep Learning Networks

- ▶ Feed Forward Neural Network
- ▶ Recurrent Neural Network
- ▶ Convolutional Neural Network
- ▶ Restricted Boltzmann Machine
- ▶ Autoencoders



Deep learning applications

Self-Driving Cars

In self-driven cars, it is able to capture the images around it by processing a huge amount of data, and then it will decide which actions should be incorporated to take a left or right or should it stop. So, accordingly, it will decide what actions it should take, which will further reduce the accidents that happen every year.

Voice Controlled Assistance

When we talk about voice control assistance, then **Siri** is the one thing that comes into our mind. So, we can tell Siri whatever we want it to do, it will search it and display it for us.



Deep learning applications

Automatic Image Caption Generation

Whatever image that you upload, the algorithm will work in such a way that it will generate caption accordingly. If you say blue colored eye, it will display a blue-colored eye with a caption at the bottom of the image.

Automatic Machine Translation

With the help of automatic machine translation, we are able to convert one language into another with the help of deep learning.

Limitations



- ❖ It only learns through the observations.
- ❖ It comprises of biases issues.

Advantages



- It lessens the need for feature engineering.
- It eradicates all those costs that are needless.
- It easily identifies difficult defects.
- It results in the best-in-class performance on problems.

Disadvantages



- It requires an ample amount of data.
- It is quite expensive to train.
- It does not have strong theoretical groundwork.

Differences between machine learning and deep learning



While there are many differences between these two subsets of artificial intelligence, here are five of the most important:

1. Human Intervention

Machine learning requires more ongoing human intervention to get results. Deep learning is more complex to set up but requires minimal intervention thereafter.

Differences Contd...



2. Hardware

Machine learning programs tend to be less complex than deep learning algorithms and can often run on conventional computers, but deep learning systems require far more powerful hardware and resources.

3. Time

Machine learning systems can be set up and operate quickly but may be limited in the power of their results. Deep learning systems take more time to set up but can generate results instantaneously (although the quality is likely to improve over time as more data becomes available).

Differences Contd...



4. Approach

Machine learning tends to require structured data and uses traditional algorithms like linear regression. Deep learning employs neural networks and is built to accommodate large volumes of unstructured data.

5. Applications

Machine learning is already in use in your email inbox, bank, and doctor's office. Deep learning technology enables more complex and autonomous programs, like self-driving cars or robots that perform advanced surgery.

Below is a table of differences between Machine Learning and Deep Learning:



S.No.	Machine Learning	Deep Learning
1.	Machine Learning is a superset of Deep Learning	Deep Learning is a subset of Machine Learning
2.	The data represented in Machine Learning is quite different as compared to Deep Learning as it uses structured data	The data representation is used in Deep Learning is quite different as it uses neural networks(ANN).
3.	Machine Learning is an evolution of AI	Deep Learning is an evolution to Machine Learning. Basically it is how deep is the machine learning.
4.	Machine learning consists of thousands of data points.	Big Data: Millions of data points.
5.	Outputs: Numerical Value, like classification of score	Anything from numerical values to free-form elements, such as free text and sound.
6.	Uses various types of automated algorithms that turn to model functions and predict future action from data.	Uses neural network that passes data through processing layers to the interpret data features and relations.
7.	Algorithms are detected by data analysts to examine specific variables in data sets.	Algorithms are largely self-depicted on data analysis once they're put into production.
8.	Machine Learning is highly used to stay in the competition and learn new things.	Deep Learning solves complex machine learning issues.

Perceptron



Perceptron:

- Perceptron is an **artificial neural network** unit that does calculations to understand the data better.
- A **group of artificial neurons interconnected** with each other through **synaptic connections** is known as a neural network
- An **artificial neuron is a complex mathematical function**, which **takes input and weights separately**, merge them together and **pass it through the mathematical function to produce output**.
- By means of the **synapse**, a neuron can transmit signals or information to **another neuron nearby**. Process it and signal the next one. The process continues, until an output signal is produced.



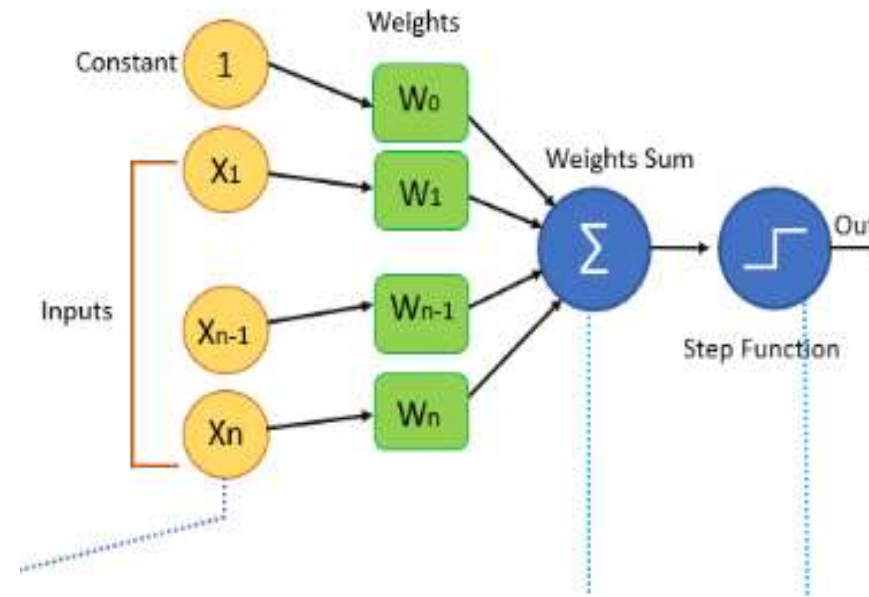
Perceptron

Perceptron Learning Algorithm

- In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers.
- A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.
- It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

Perceptron

Perceptron Algorithm Block Diagram





Perceptron

Perceptron Algorithm Block Diagram

➤ The terminology of the above diagram are:

1. Input:

All the features of the model we want to train the neural network will be passed as the input to it, Like the set of features $[X_1, X_2, X_3, \dots, X_n]$. Where n represents the total number of features and X represents the value of the feature.

2. Weights:

Initially, we have to pass some random values as values to the weights and these values get automatically updated after each training error that is the values are generated during the training of the model. In some cases, weights can also be called as weight coefficients.



Perceptron

Perceptron Algorithm Block Diagram

3. Weights Sum:

Each input value will be first multiplied with the weight assigned to it and the sum of all the multiplied values is known as a weighted sum.

$$\text{Weights sum} = \sum W_i * X_i \text{ (from } i=1 \text{ to } i=n) + (W_0 * 1)$$



Perceptron

Step or Activation Function

- Activation function **applies step rule which converts the numerical value to 0 or 1** so that it will be easy for data set to classify.
- Based on the type of value we need as output we can change the activation function.

1. Sigmoid function:

if we want **values to be between 0 and 1** we can use a sigmoid function that has a smooth gradient as well.

2. Sign function:

if we want **values to be +1 and -1** then we can use sign function.

3. hyperbolic tangent function:

The hyperbolic tangent function is a **zero centered function** making it easy for the multilayer neural networks.

Perceptron



Step or Activation Function

4. Relu function:

- Relu function is highly computational but it cannot process input values that approach zero.
- It is good for the values that are both greater than and less than a Zero.



Perceptron

Bias

- It is a special type of classifier which is also known as **synapse**. Even it helps input vectors to get the correct answer
- In the diagram, we have passed value one as input in the starting and W_0 in the weights section .
- W_0 is an element that adjusts the boundary away from origin to move the activation function left, right, up or down.
- since we want this to be independent of the input features, we add constant one in the statement so the features will not get affected by this and this value is known as **Bias**.



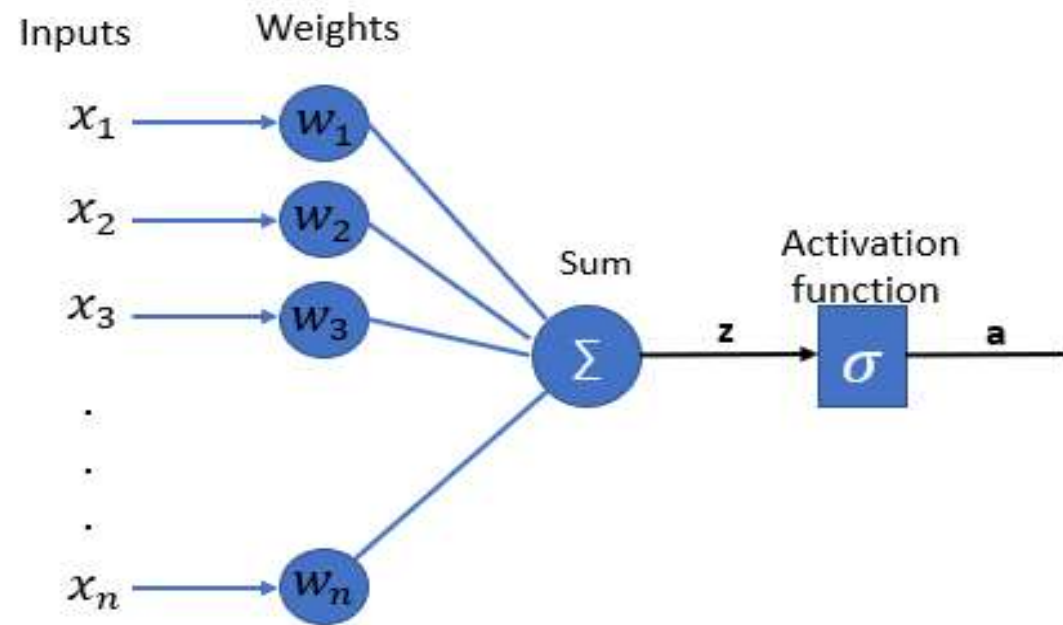
Perceptron

Types of Perceptron Algorithm:

- Perceptron algorithms can be divided into two types . They are
 1. [single layer perceptrons](#) and
 2. multi-layer perceptron's.
- In single-layer perceptron's neurons are organized in one layer.
- In a multilayer perceptron's a group of neurons will be organized in multiple layers.
- Every single neuron present in the first layer will take the input signal and send a response to the neurons in the second layer and so on.

Perceptron

Types of Perceptron Algorithm: Single Layer Perceptron

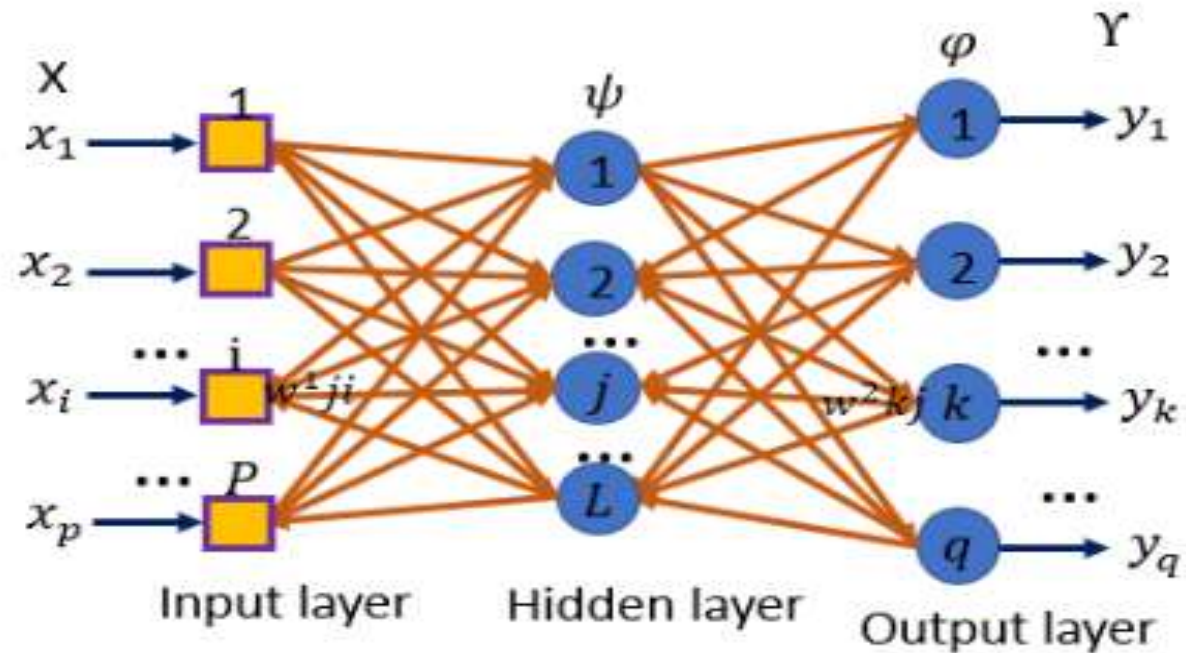




Perceptron

Types of Perceptron Algorithm:

Multi-Layer Perceptron





Perceptron

Perceptron Learning Steps

1. Features of the model we want to train should be passed as input to the perceptrons in the first layer.
2. These inputs will be multiplied by the weights or weight coefficients and the production values from all perceptrons will be added .
3. Adds the Bias value, to move the output function away from the origin.
4. This computed value will be fed to the activation function.
5. result value from the activation function is the output value.

Multi Layer Perceptron



Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially **formed from multiple layers of the perceptron.**

MLPs are the base of deep learning technology. It belongs to a **class of feed-forward neural networks** having various layers of **perceptrons**. These perceptrons have **various activation functions** in them.

MLPs also have connected input and output layers and their number is the same. Also, there's a layer that remains hidden amidst these two layers.

MLPs are mostly **used to build image and speech recognition** systems or some other types of the **translation software.**

Multi Layer Perceptron



The working of MLPs starts by feeding the data in the input layer. The neurons present in the **layer form a graph to establish a connection that passes in one direction.**

The weight of this input data is found to exist between the hidden layer and the input layer. MLPs use activation functions to determine which nodes are ready to fire. These activation functions include **tanh** function, **sigmoid** and **ReLU**s.

MLPs are mainly used to train the models to understand what kind of co-relation the layers are serving to achieve the desired output from the given data set.

See the below image to understand better....

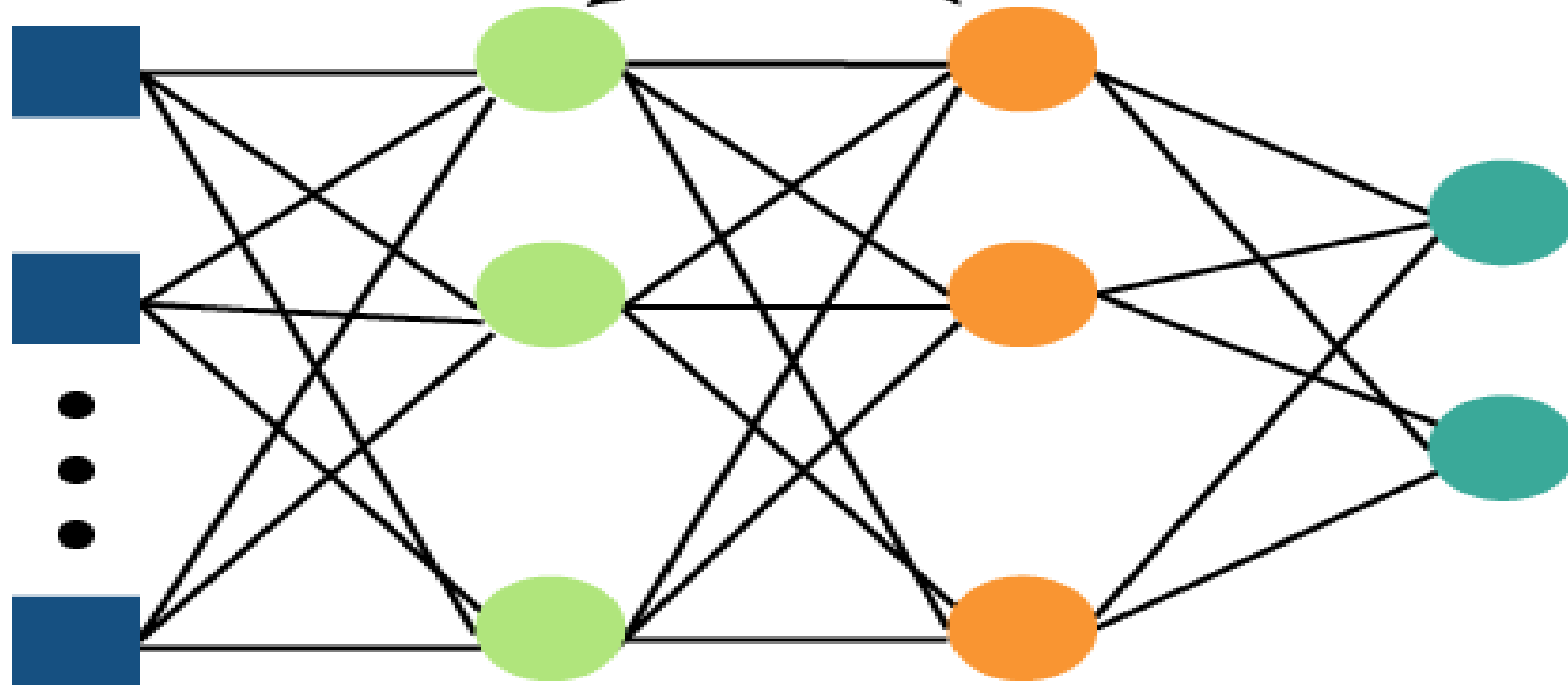
Multi Layer Perceptron



Input Layer

Hidden Layers

Output Layer



Multi Layer Perceptron



The algorithm for the MLP is as follows:

1. Just as with the perceptron, the inputs are pushed forward through the MLP by taking the dot product of the input with the weights that exist between the input layer and the hidden layer ($W-H$). This dot product yields a value at the hidden layer. We do not push this value forward as we would with a perceptron though.
2. MLPs utilize [activation functions](#) at each of their calculated layers. There are many activation functions to discuss: [rectified linear units \(ReLU\)](#), [sigmoid function](#), tanh. Push the calculated output at the current layer through any of these activation functions.

Multi Layer Perceptron



3. Once the calculated output at the hidden layer has been pushed through the activation function, push it to the next layer in the MLP by taking the dot product with the corresponding weights.
4. Repeat steps two and three until the output layer is reached.

At the output layer, the calculations will either be used for a [backpropagation](#) algorithm that corresponds to the activation function that was selected for the MLP (in the case of training) or a decision will be made based on the output (in the case of testing)

Multi Layer Perceptron



In the world of deep learning, TensorFlow, Keras, Microsoft Cognitive Toolkit (CNTK), and PyTorch are very popular.

Most of us may not realise that the very popular machine learning library **Scikit-learn** is also capable of a basic deep learning modelling.

Multi Layer Perceptron



Salient points of Multilayer Perceptron (MLP) in Scikit-learn

- There is no activation function in the output layer.
- For regression scenarios, the square error is the loss function, and cross-entropy is the loss function for the classification
- It can work with single as well as multiple target values regression.
- Unlike other popular packages, like Keras the implementation of MLP in Scikit doesn't support GPU.
- We cannot fine-tune the parameters like different activation functions, weight initializers etc. for each layer.

MLP Example - *Classification Example*



In Scikit-learn “MLPClassifier” is available for Multilayer Perceptron (MLP) classification scenarios.

Step1: Like always first we will import the modules which we will use in the example. We will use the Iris database and MLPClassifier from for the classification example.

```
from sklearn.datasets import load_iris
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
```




MLP Example

➤ **Step 2:** In separate data frames “X” and “y”, the values of the independent and dependent features are stored.

```
iris_data = load_iris()  
X = pd.DataFrame(iris_data.data,  
columns=iris_data.feature_names)  
y = iris_data.target
```



MLP Example

➤ **Step 3:** Similar to the regression example above we will split the dataset into train and test dataset. We have reserved 20% of the dataset for checking the accuracy of the trained model. Independent train and test dataset are further scaled to make sure that the input data is standard normally distributed and are centred around zero and have variance in the same order.

➤ `X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=1,
test_size=0.2)`

`sc_X = StandardScaler()`

`X_train_scaled=sc_X.fit_transform(X_train)`

`X_test_scaled=sc_X.transform(X_test)`



MLP Example

Step 4: In the below code, we have modelled four hidden layers with different neurons in each layer. Considering the input and output layer, we have a total of 6 layers in the model. In case any optimiser is not mentioned then “Adam” is the default optimiser.

```
clf = MLPClassifier(hidden_layer_sizes=(256,128,64,32),activation="relu",  
                    random_state=1).fit(X_trainscaled, y_train)  
y_pred=clf.predict(X_testscaled)  
print(clf.score(X_testscaled, y_test))
```

MLP Example



The classifier shows quite a high score for the test data. It is important to understand the areas in which the classification model is making an error to make a full sense of model accuracy.

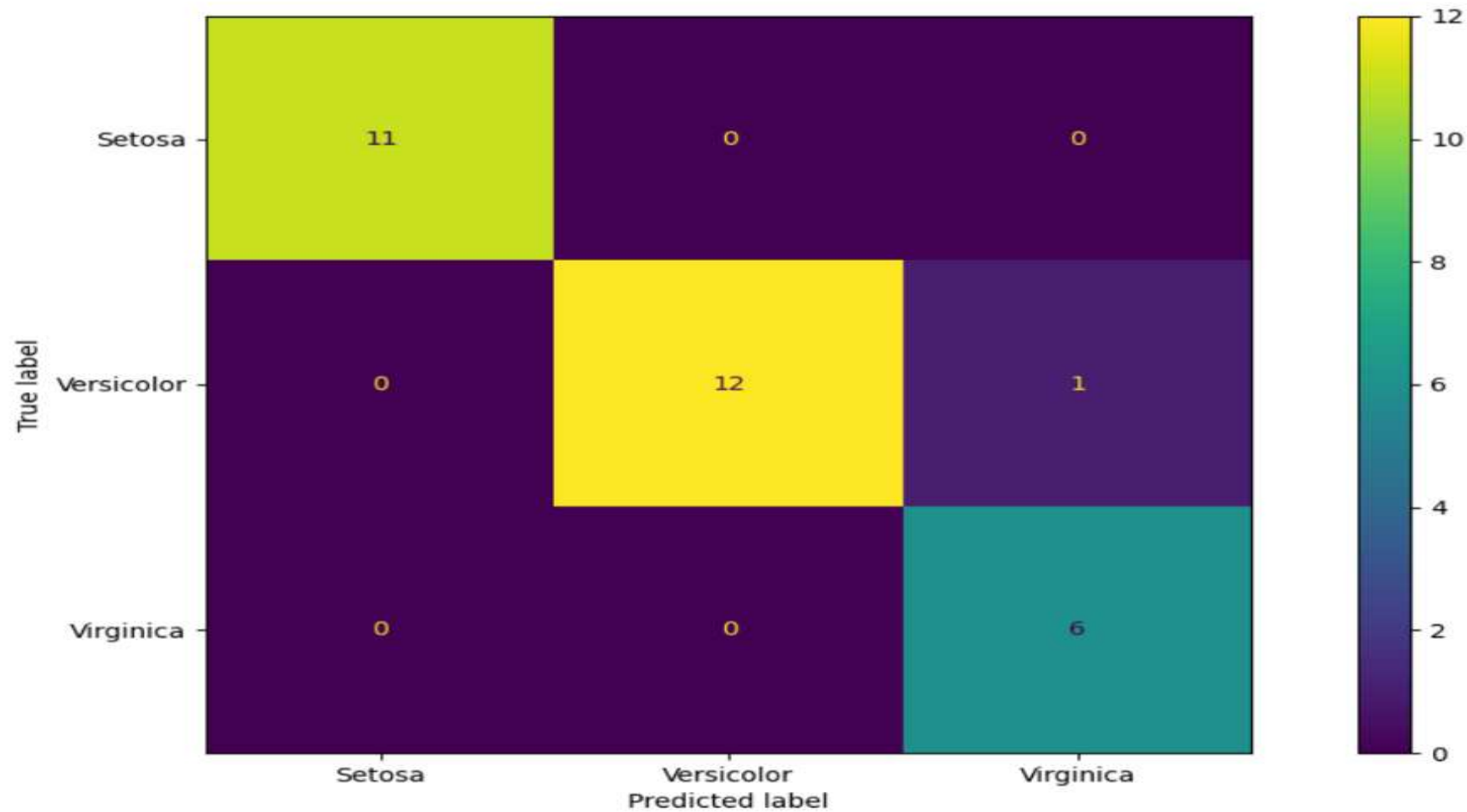
Step 5: We will draw a confusion matrix to understand the classifications which are made incorrect by the model.

```
fig=plot_confusion_matrix(clf, X_testscaled,  
y_test,display_labels=["Setosa","Versicolor","Virginica"])  
fig.figure_.suptitle("Confusion Matrix for Iris Dataset")  
plt.show()
```

MLP Example



Confusion Matrix for Iris Dataset



Backpropagation



Why We Need Backpropagation?

- While designing a Neural Network, in the beginning, we initialize weights with some random values.
- So, 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.
- To reduce the error we need to somehow explain the model to change the parameters (weights), such that error becomes minimum.

Backpropagation



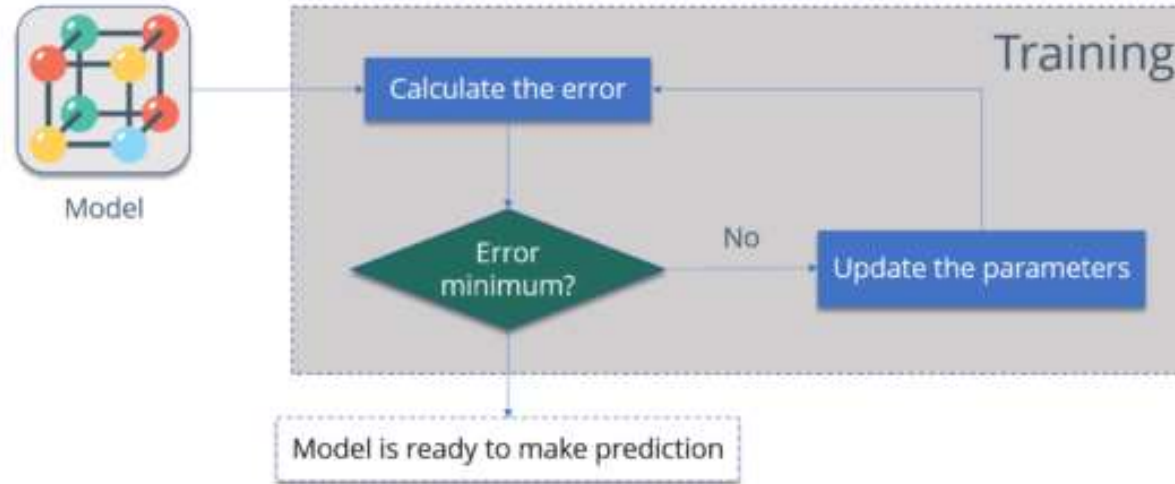
What is backpropagation?

- It was first introduced in the 1960s and 30 years later it was popularized by David Rumelhart, Geoffrey Hinton, and Ronald Williams in the famous 1986 paper.
- Neural network training happens through backpropagation. By this approach, we fine-tune the weights of a neural net based on the error rate obtained in the previous run.
- The right manner of applying this technique reduces error rates and makes the model more reliable.
- Backpropagation is used to train the neural network of the chain rule method. In simple terms, after each feed-forward passes through a network, this algorithm does the backward pass to adjust the model's parameters based on weights and biases.
- Backpropagation works with a multi-layered neural network and learns internal representations of input to output mapping.

Backpropagation



Consider the diagram below:



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.

Backpropagation



How does backpropagation work?

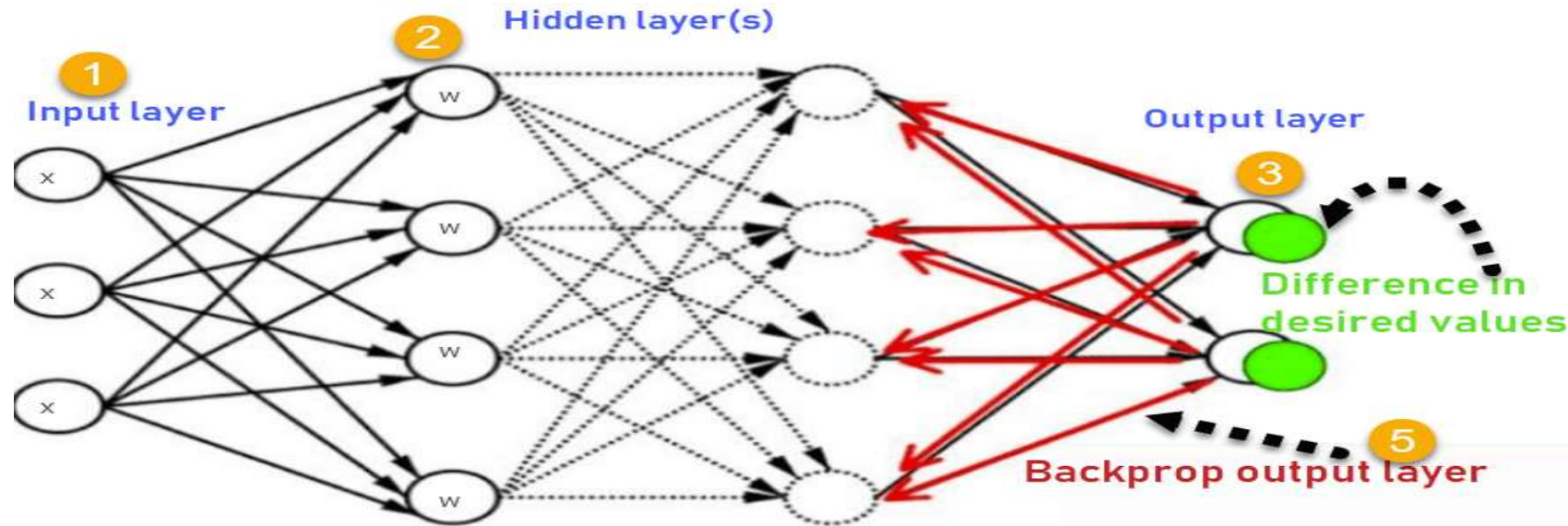
Let us take a look at how backpropagation works. It has four layers: input layer, hidden layer, hidden layer II and final output layer

So, the main three layers are:

- 1. Input layer**
- 2. Hidden layer**
- 3. Output layer**

Each layer has its own way of working and its own way to take action such that we are able to get the desired results and correlate these scenarios to our conditions. Let us discuss other details needed to help summarizing this algorithm.

Backpropagation



This image summarizes the functioning of the backpropagation approach.

1. Input layer receives x
2. Input is modeled using weights w
3. Each hidden layer calculates the output and data is ready at the output layer
4. Difference between actual output and desired output is known as the error
5. **Go back to the hidden layers and adjust the weights so that this error is reduced in future runs**

This process is repeated till we get the desired output. The training phase is done with supervision. Once the model is stable, it is used in production.



Backpropagation

Types of backpropagation

There are two types of backpropagation networks.

- Static backpropagation
- Recurrent backpropagation

➤ Static backpropagation

In this network, mapping of a static input generates static output. Static classification problems like optical character recognition will be a suitable domain for static backpropagation.

➤ Recurrent backpropagation

Recurrent backpropagation is conducted until a certain threshold is met. After the threshold, the error is calculated and propagated backward.

The difference between these two approaches is that static backpropagation is as fast as the mapping is static.

Backpropagation



Advantages:

Backpropagation has many advantages, some of the important ones are listed below-

- Backpropagation is fast, simple and easy to implement
- There are no parameters to be tuned
- Prior knowledge about the network is not needed thus becoming a flexible method
- This approach works very well in most cases
- The model need not learn the features of the function

Backpropagation



Disadvantages of backpropagation

- Input data holds the key to the overall performance
- Noisy data can lead to inaccurate results
- Matrix based approach is preferred over a mini-batch

Backpropagation



Example:

Consider Iris data which contains features such as length and width of sepals and petals. With the help of those, we need to identify the species of a plant.

For this, we will build a multilayered neural network and will use the sigmoid function as it is a classification problem.

Backpropagation



Let us read the libraries required and read the data..

```
import numpy as np
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
```

Backpropagation



To ignore warnings, we will import another library called warnings.

```
import warnings
warnings.simplefilter(action='ignore',
category=FutureWarning)
```


Backpropagation



Let us now read the data

```
iris = pd.read_csv("iris.csv")
```

```
iris.head()
```

1	Sepal.Len	Sepal.Wid	Petal.Len	Petal.Wid	Species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa
12	5.4	3.7	1.5	0.2	setosa

Backpropagation



Now we will put labels to the class as 0,1 and 2.

```
iris['Species'].replace(['setosa', 'virginica', 'versicolor'],  
[0, 1, 2], inplace=True)
```

We will now define functions which will do the following.

Perform one hot encoding to the output.

Perform sigmoid function

Normalize the features.

Backpropagation



For one hot encoding, we define the following function.

```
def to_one_hot(Y):  
    n_col = np.amax(Y) + 1  
    binarized = np.zeros((len(Y), n_col))  
    for i in range(len(Y)):  
        binarized[i, Y[i]] = 1.  
    return binarized
```

Let us now define a sigmoid function

```
def sigmoid_func(x):  
    return 1/(1+np.exp(-x))  
def sigmoid_derivative(x):  
    return sigmoid_func(x)*(1 - sigmoid_func(x))
```

Backpropagation



Now we will define a function for normalization

```
def normalize (X, axis=-1, order=2):  
    l2 = np. atleast_1d (np.linalg.norm(X, order, axis))  
    l2[l2 == 0] = 1  
    return X / np.expand_dims(l2, axis)
```

Backpropagation



Now we will apply normalization to the features and one hot encoding to the output

```
columns = ['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']
```

```
x = pd.DataFrame(iris, columns=columns)
```

```
x = normalize(x.as_matrix())
```

```
columns = ['Species']
```

```
y = pd.DataFrame(iris, columns=columns)
```

```
y = y.as_matrix()
```

```
y = y.flatten()
```

```
y = to_one_hot(y)
```

Backpropagation



Now it's time to apply back propagation. To do that, we need to define weights and a learning rate. Let us do that. But before that we need to split the data for training and testing.

#Split data to training and validation data

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33)
```

#Weights

```
w0 = 2*np.random.random((4, 5)) - 1 #for input - 4 inputs, 3 outputs
```

```
w1 = 2*np.random.random((5, 3)) - 1 #for layer 1 - 5 inputs, 3 outputs
```

#learning rate

```
n = 0.1
```

Backpropagation



We will set a list for errors and see how the change in training decreases the error via visualization.

`errors = []`

Let us perform the feed forward and back propagation network.

For backpropagation, we will use gradient descent.

Backpropagation



```
for i in range (100000):
```

```
    Feed forward network
```

```
        layer0 = X_train
```

```
        layer1 = sigmoid_func(np.dot(layer0, w0))
```

```
        layer2 = sigmoid_func(np.dot(layer1, w1))
```

```
        Back propagation using gradient descent
```

```
        layer2_error = y_train - layer2
```

```
        layer2_delta = layer2_error * sigmoid_derivative(layer2)
```

```
        layer1_error = layer2_delta.dot(w1.T)
```

```
        layer1_delta = layer1_error * sigmoid_derivative(layer1)
```

```
        w1 += layer1.T.dot(layer2_delta) * n
```

```
        w0 += layer0.T.dot(layer1_delta) * n
```

```
        error = np.mean(np.abs(layer2_error))
```

```
        errors.append(error)
```


Backpropagation

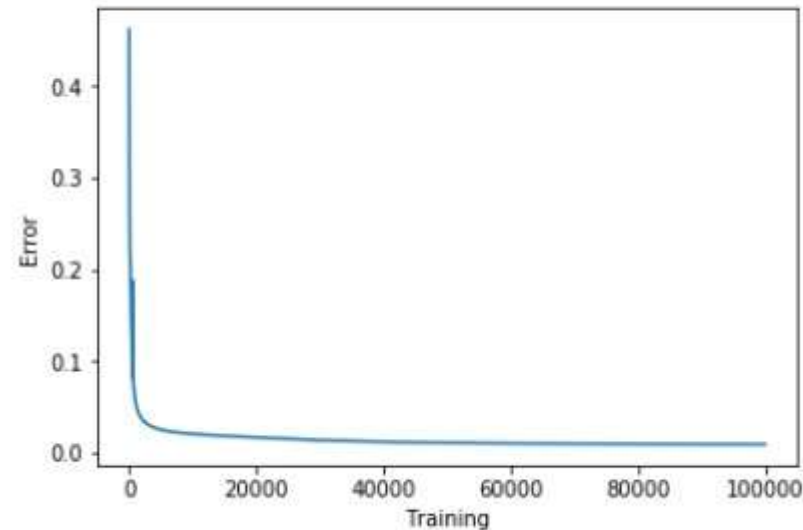


Accuracy will be gathered and visualized by subtracting the error from the training data

$\text{accuracy_training} = (1 - \text{error}) * 100$

Now let us visualize how accuracy increases by decreasing the error

```
plt.plot(errors)
plt.xlabel('Training')
plt.ylabel('Error')
plt.show()
```



Backpropagation



Let us look at the accuracy now

```
print ("Training Accuracy of the model " + str  
(round(accuracy_training,2)) + "%")
```

Output: Training Accuracy of the model 99.04%

Backpropagation



Our training model is performing really well. Now let us see the validation accuracy.

```
#Validate
```

```
layer0 = X_test
```

```
layer1 = sigmoid_func(np.dot(layer0, w0))
```

```
layer2 = sigmoid_func(np.dot(layer1, w1))
```

```
layer2_error = y_test - layer2
```

```
error = np.mean(np.abs(layer2_error))
```

```
accuracy_validation = (1 - error) * 100
```

```
print ("Validation Accuracy of the model "+ str(round(accuracy_validation,2)) + "%")
```

Output: Validation Accuracy 92.86%

The performance was as expected.



Loss Functions

Loss Functions

- Loss functions are used to **determine the error (“the loss”) between the output of our algorithms and the given target value.**
- The loss function (J) can be defined as a function which takes in **two parameters**:
 1. Predicted Output
 2. True Output
- There are multiple ways to determine loss.
- These are divided into **two categories**
 1. Regression loss and
 2. Classification Loss



Loss Functions

Regression Loss Function:

- Regression Loss is used when we are **predicting continuous values** like the price of a house or sales of a company.
- Some of the important Regression Loss Functions are:
 1. Mean Squared Error
 2. Mean Squared Logarithmic Error Loss
 3. Mean Absolute Error Loss

1. Mean Squared Error

Mean Squared Error is the **mean of squared differences between the actual and predicted value.**

If the difference is large the model will penalize it as we are computing the squared difference.



Loss Functions

2. Mean Squared Logarithmic Error Loss

- Suppose we want to reduce the difference between the actual and predicted variable we can take the **natural logarithm of the predicted variable** then take the mean squared error.
- This will overcome the problem possessed by the Mean Square Error Method.

3. Mean Absolute Error Loss

- Sometimes there may be some data points which far away from rest of the points i.e outliers, in case of cases Mean Absolute Error Loss will be appropriate to use as it **calculates the average of the absolute difference between the actual and predicted values.**



Loss Functions

Binary Classification Loss Function:

- Suppose we are dealing with a **Yes/No situation** like “a person has diabetes or not”, in this kind of scenario Binary Classification Loss Function is used.
- Some of the important Binary Classification Loss Function are:

- 1.Binary Cross Entropy Loss
- 2.Hinge Loss

1.Binary Cross Entropy Loss

- It gives the probability value between **0 and 1 for a classification task**. Cross-Entropy calculates the **average difference between the predicted and actual probabilities**.

2.Hinge Loss

- This type of loss is used when the **target variable has 1 or -1 as class labels**. It penalizes the model when there is a difference in the sign between the actual and predicted class values.

These are particularly used in SVM models.



Loss Functions

Multi-Class Classification Loss Function:

➤ If we take a dataset like Iris where we need to predict the three-class labels: Setosa, Versicolor and Virginia, in such cases where the target variable has more than two classes Multi-Class Classification Loss function is used.

➤ Some of the important Multi-Class Classification Loss Function are:

1. Categorical Cross Entropy Loss
2. Kullback Leibler Divergence Loss

1. Categorical Cross Entropy Loss

These are similar to binary classification cross-entropy, used for multi-class classification problems.

Loss Functions



2. Kullback Leibler Divergence Loss

- Kullback Leibler Divergence Loss calculates how much a **given distribution is away from the true distribution**.
- These are used to carry out complex operations like autoencoder where there is a need to learn the dense feature representation.



Loss Functions

Applications of Loss Functions

1. Loss functions are used in optimization problems with the goal of minimizing the loss.
2. Loss functions are used in regression when finding a line of best fit by minimizing the overall loss of all the points with the prediction from the line.
3. Loss functions are used while training [perceptrons](#) and [neural networks](#) by influencing how their weights are updated.

Hyper Parameter Tuning



Difference between parameter and hyperparameter:

Model parameters: These are the parameters that are estimated by the model from the given data. For example the weights of a deep neural network.

Model hyper parameters: These are the parameters that cannot be estimated by the model from the given data. These parameters are used to estimate the model parameters. For example, the learning rate in deep neural networks.

Hyper Parameter Tuning



What is hyperparameter tuning ?

Hyperparameter tuning is the process of **determining the right combination of hyperparameters** that allows the model to maximize model performance. Setting the correct combination of hyperparameters is the only way to extract the maximum performance out of models.

Hyper Parameter Tuning



Choosing good hyperparameters:

Choosing the right combination of hyperparameters is not an easy task.

There are two ways to set them.

Manual hyperparameter tuning: In this method, different combinations of hyperparameters are set (and experimented with) manually. This is a tedious process and cannot be practical in cases where there are many hyperparameters to try.

Automated hyperparameter tuning: In this method, optimal hyperparameters are found using an algorithm that automates and optimizes the process.

Hyper Parameter Tuning



Hyperparameter tuning methods

In this section, I will introduce all of the hyperparameter tuning methods that are popular today.

Random Search

In the random search method, we create a grid of possible values for hyperparameters. Each iteration tries a **random combination of hyperparameters** from this grid, records the performance, and lastly returns the combination of hyperparameters which provided the best performance.

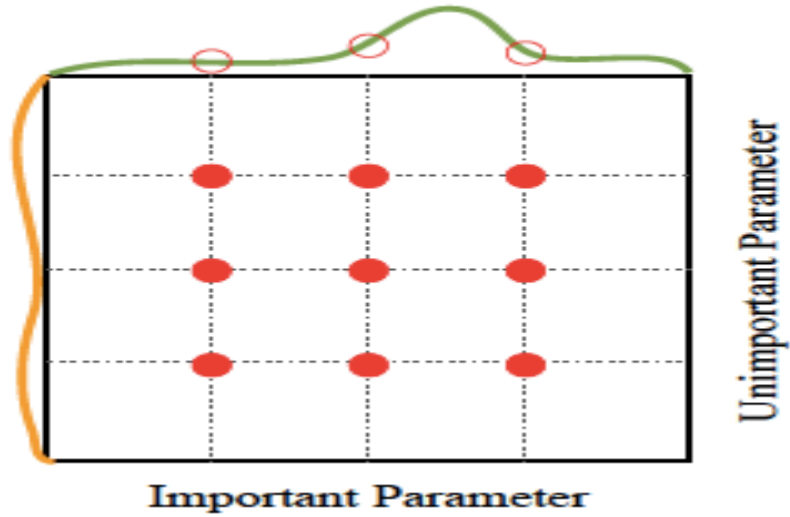
Grid Search

In the grid search method, we create a grid of possible values for hyperparameters. Each iteration tries a combination of hyperparameters in a specific order. It fits the model on each and **every combination of hyperparameter** possible and records the model performance. Finally, it returns the best model with the best hyperparameters.

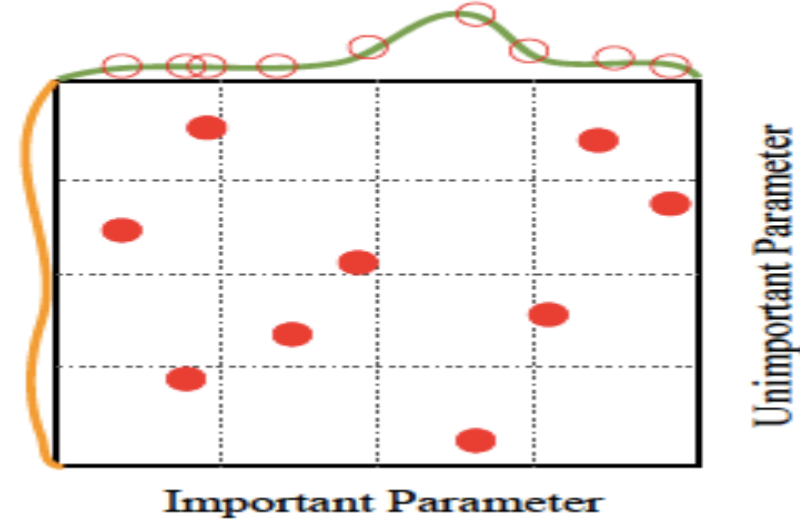
Hyper Parameter Tuning



Grid Layout



Random Layout



Hyper Parameter Tuning



Bayesian Optimization

Tuning and finding the right hyperparameters for your model is an **optimization problem**. We want to **minimize the loss function** of our model **by changing model parameters**. Bayesian optimization helps us find the minimal point in the minimum number of steps. [Bayesian optimization](#) also uses an *acquisition function* that directs sampling to areas where an improvement over the current best observation is likely.

Tree-structured Parzen estimators (TPE)

The idea of [Tree-based Parzen optimization](#) is similar to Bayesian optimization. Instead of finding the values of $p(y|x)$ where y is the function to be minimized (e.g., validation loss) and x is the value of hyperparameter the TPE models $P(x|y)$ and $P(y)$. One of the great drawbacks of tree-structured Parzen estimators is that they do not model interactions between the hyper-parameters. That said TPE works extremely well in practice and was battle-tested across most domains.

Hyper Parameter Tuning



Hyperparameter tuning algorithms

These are the algorithms developed specifically for doing hyperparameter tuning.

Hyperband

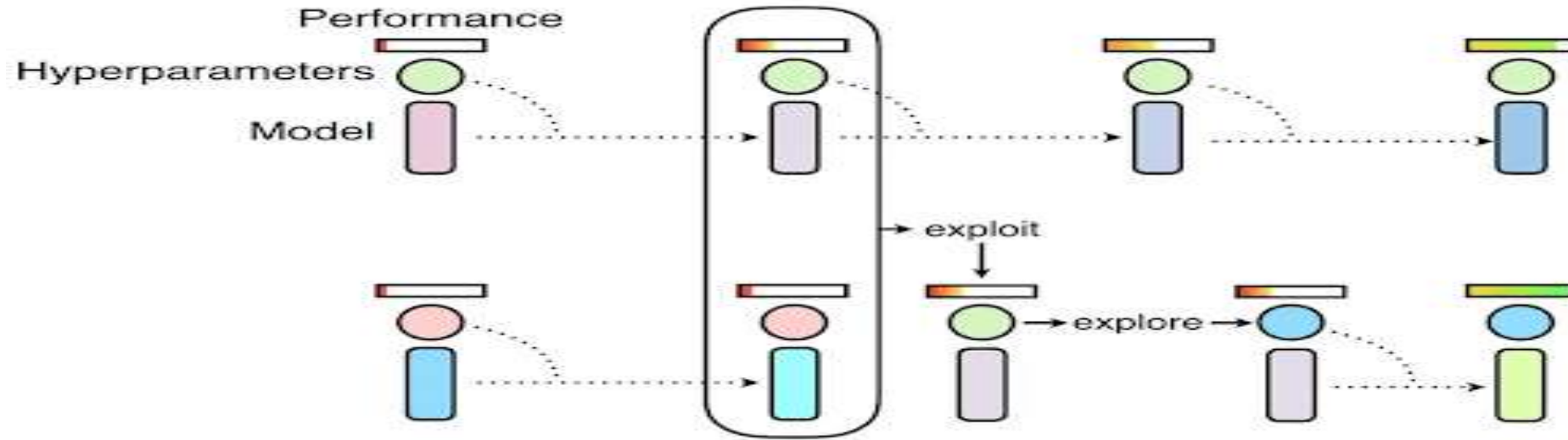
Hyperband is a variation of random search, but with some [explore-exploit](#) theory to find the best time allocation for each of the configurations. You can check this [research paper](#) for further references.

Population-based training (PBT)

This technique is a hybrid of two most commonly used search techniques: Random Search and manual tuning applied to Neural Network models.

PBT starts by training many neural networks in parallel with random hyperparameters. But these networks aren't fully independent of each other. It uses information from the rest of the population to refine the hyperparameters and determine the value of hyperparameter to try. You can check this [article](#) for more information on PBT.

Hyper Parameter Tuning



BOHB

BOHB (Bayesian Optimization and HyperBand) mixes the Hyperband algorithm and Bayesian optimization. You can check this [article](#) for further reference.

Hyper Parameter Tuning



Tools for hyperparameter optimization

Now that you know what are the methods and algorithms let's talk about tools, and there are a lot of those out there.

Some of the best Hyperparameter Optimization libraries are:

[Scikit-learn](#) (grid search, random search)

[Hyperopt](#)

[Scikit-Optimize](#)

[Optuna](#)

[Ray.tune](#)

Hyper Parameter Tuning



Scikit learn

Scikit-learn has implementations for grid search and random search and is a good place to start if you are building models with sklearn.

For both of those methods, scikit-learn trains and evaluates a model in a k fold cross-validation setting over various parameter choices and returns the best model.

Specifically:

Random search: with [randomsearchcv](#) runs the search over some number of random parameter combinations

Grid search: [gridsearchcv](#) runs the search over all parameter sets in the grid

Tuning models with scikit-learn is a good start but there are better options out there and they often have random search strategy anyway.

Hyper Parameter Tuning



Scikit-optimize

Scikit-optimize uses Sequential model-based optimization algorithm to find optimal solutions for hyperparameter search problems in less time.

Scikit-optimize provides many features other than hyperparameter optimization such as:
store and load optimization results,
convergence plots,
comparing surrogate models

Hyper Parameter Tuning



Hyperparameter tuning resources and examples

In this section, I will share some hyperparameter tuning examples implemented for different ML and DL frameworks.

Random forest

- [Understanding Random forest hyperparameters](#)
- [Bayesian hyperparameter tuning for random forest](#)
- [Random forest tuning using grid search](#)



Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN)

- A recurrent neural network (RNN) is a type of artificial neural network which **uses sequential data or time series data**.
- Deep learning algorithms are commonly **used for ordinal or temporal problems**, such as **language translation, natural language processing (nlp), speech recognition, and image captioning**;
- RNN are incorporated into **popular applications** such as **Siri, voice search, and Google Translate**.
- RNNs are a powerful and robust type of neural network, and belong to the most promising algorithms in use because it is the only one with **an internal memory**.



Recurrent Neural Networks (RNN)

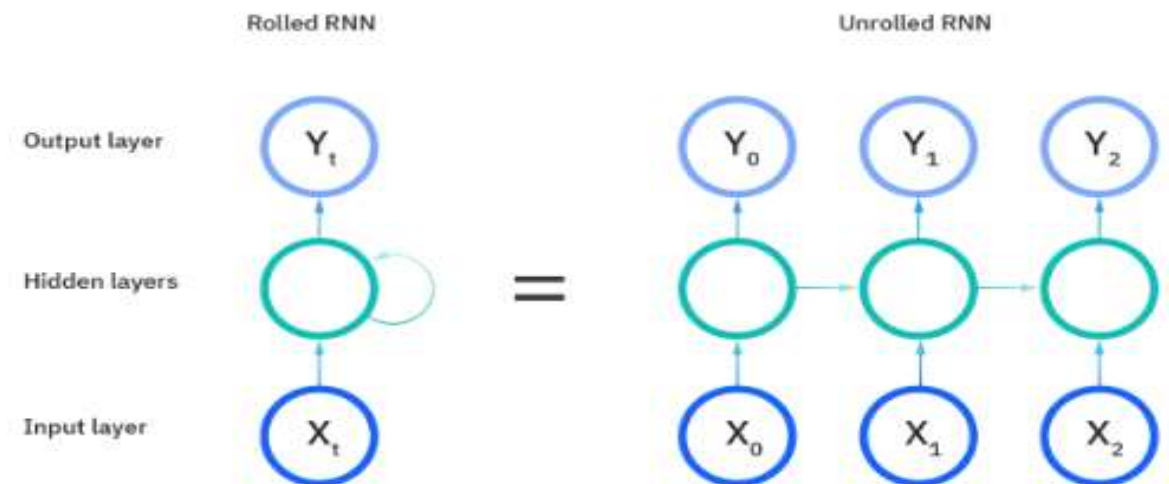
Recurrent Neural Networks (RNN)

- Recurrent neural networks **utilize training data to learn**.
- They are distinguished by their “memory” as they **take information from prior inputs to influence the current input and output**.

Architecture of a traditional RNN:

Recurrent neural networks are a class of **neural networks that allow previous outputs to be used as inputs while having hidden states**.

- They are typically as follows:





Recurrent Neural Networks (RNN)

How recurrent neural network works

➤ The working of an RNN can be described with the help of the following example.

Example:

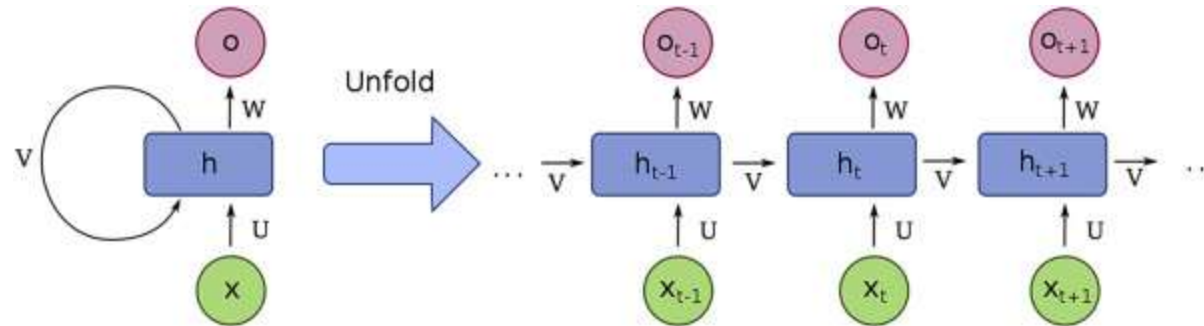
Suppose a deeper network consists of **one input layer, three hidden layers, and one output layer**. Then not like other neural networks, **each hidden layer will have its own set of weights and their biases**. The value for hidden layer is 1; then the weights and biases are w_1 and b_1 , w_2 and b_2 for second hidden layer, and w_3 and b_3 for third hidden layer. This means that each of these layers is independent of each other, i.e., they do not memorize any other previous outputs.



Recurrent Neural Networks (RNN)

How recurrent neural network works

➤ The process of RNN with input and output layer is as follows.



The RNN will do the following:

➤ RNN resolves the independent activations into dependent activations by presuming the same weights and biases to all the layers, thus minimizing the complexity of ascending parameters and memorizing each previous output by giving each output as input to the next hidden .



Recurrent Neural Networks (RNN)

How recurrent neural network works

- Three layers can be merged together such that the weights and bias of all the hidden layers are not different.
- Formula for calculating **current state** is defined as:

$$h_t = f(h_{t-1}, x_t)$$

where

h_t denotes current state

h_{t-1} denotes previous state

x_t denotes input state



Recurrent Neural Networks (RNN)

How recurrent neural network works

- Formula for applying **activation function (tanh)** is defined as:

$$h_t = \tanh(w_{hh} h_{t-1} + w_{xh} x_t)$$

Where,

w_{hh} is weight at recurrent neuron

w_{xh} is weight at input neuron

h_{t-1} denotes previous state

x_t denotes input state

- Formula for calculating **output** is :

$$Y_t = W_{hy} h_t$$

Y_t is output

W_{hy} is weight at output layer

h_t denotes current state



Recurrent Neural Networks (RNN)

Types of recurrent neural networks

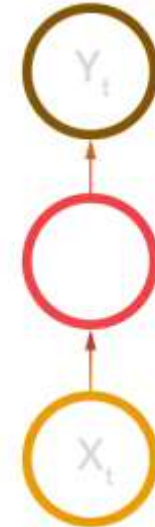
➤ There are mainly 4 types of RNNs. Those are :

1. One-to-one
2. One-to-many
3. Many-to-one
4. Many-to-many

One-to-one:

Example:

Traditional neural network



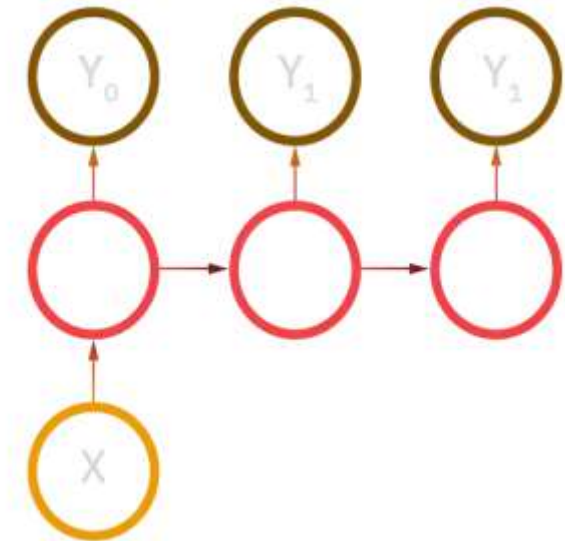


Recurrent Neural Networks (RNN)

One-to-many:

Example:

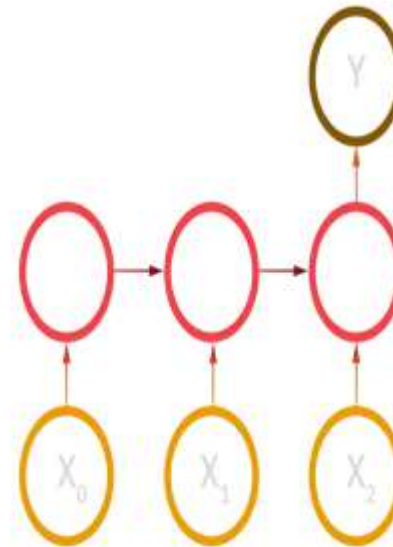
Music generation



Many-to-one:

Example:

Sentiment classification





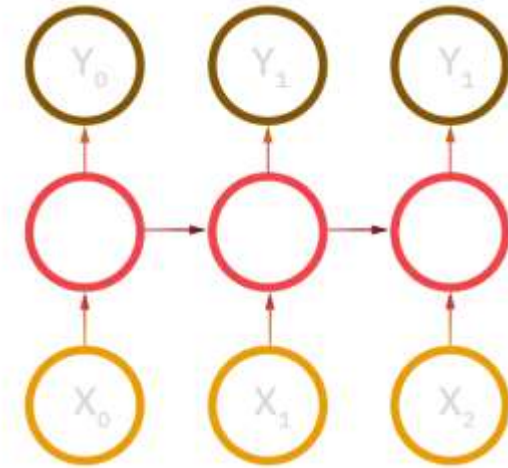
Recurrent Neural Networks (RNN)

Many-to-many:

i)

Example:

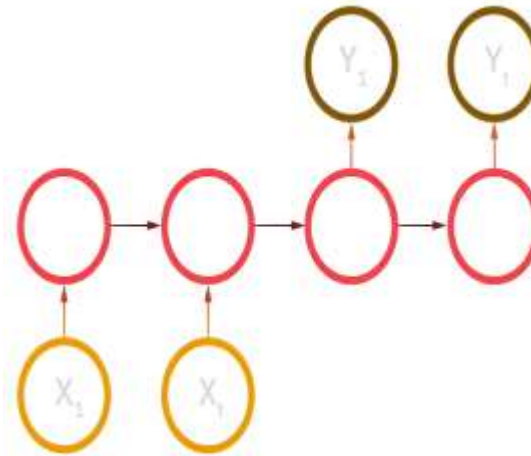
Name entity recognition



ii)

Example:

Machine translation





Recurrent Neural Networks (RNN)

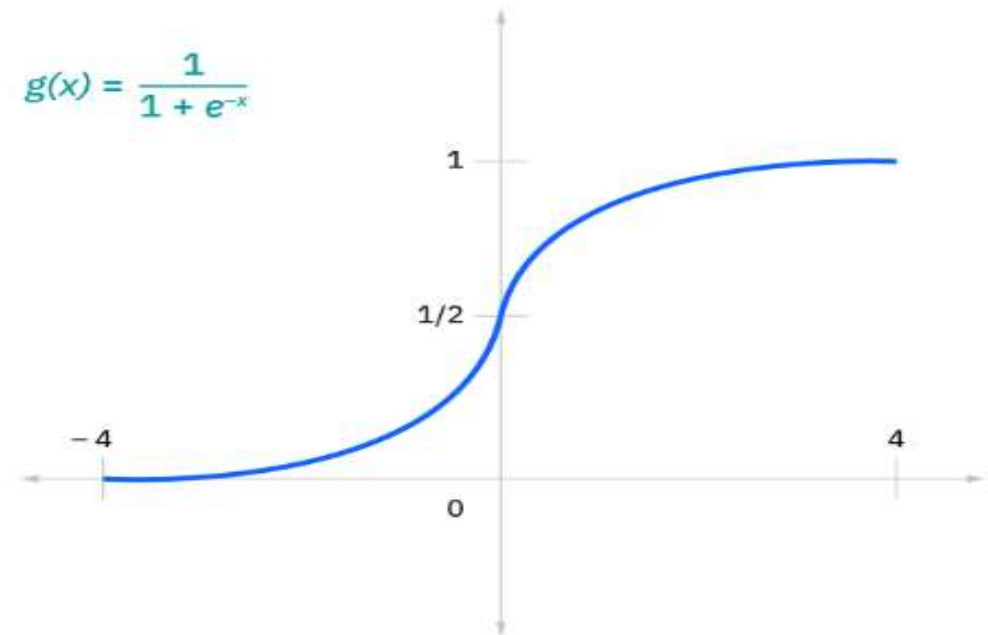
Common activation functions

➤ Some of the most commonly used functions are :

1. Sigmoid
2. Tanh
3. Relu

Sigmoid:

➤ This is represented with the formula
 $g(x) = 1/(1 + e^{-x})$.

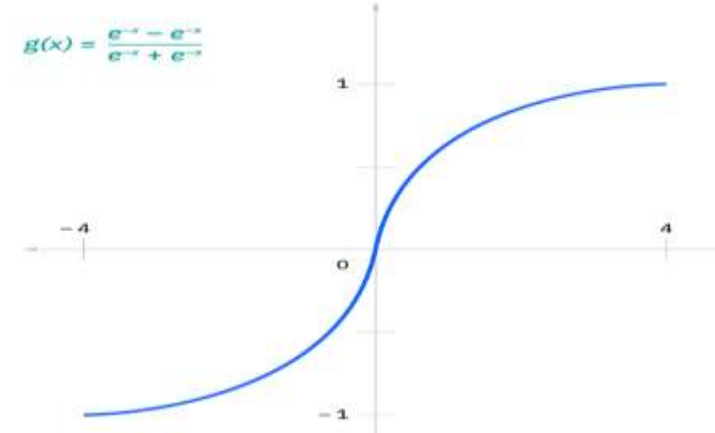




Recurrent Neural Networks (RNN)

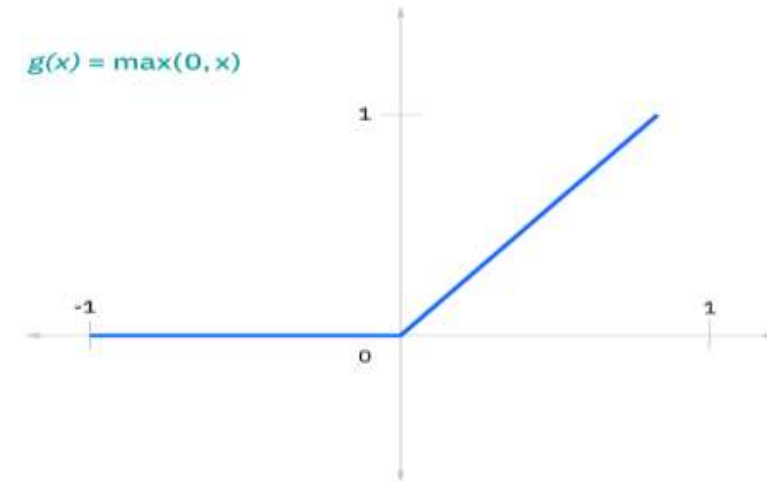
Tanh:

- This is represented with the formula
$$g(x) = (e^x - e^{-x}) / (e^x + e^{-x})$$



Relu:

- This is represented with the formula
$$g(x) = \max(0, x)$$





Recurrent Neural Networks (RNN)

Variant RNN architectures:

➤ Variant RNN architectures are:

1. **Bidirectional recurrent neural networks (BRNN)**
2. **Long short-term memory (LSTM)**
3. **Gated recurrent units (GRUs)**

Bidirectional recurrent neural networks (BRNN):

➤ Bidirectional recurrent neural networks (BRNNs) are another type of RNN that simultaneously **learn the forward and backward directions of information flow.**

➤ The process of both directions being learned simultaneously is known as bidirectional information flow.



Recurrent Neural Networks (RNN)

Long short-term memory (LSTM):

- This is a popular RNN architecture, which was introduced by Sepp Hochreiter and Juergen Schmidhuber as a **solution to vanishing gradient problem**.
- LSTMs have “**cells**” in the hidden layers of the neural network, which have **three gates**—an input gate, an output gate, and a forget gate. These gates control the flow of information which is needed to predict the output in the network.



Recurrent Neural Networks (RNN)

Gated recurrent units (GRUs):

➤ This RNN variant is similar to the LSTMs as it also works to address the short-term memory problem of RNN models. **Instead of using a “cell state” regulate information, it uses hidden states, and instead of three gates, it has two—a reset gate and an update gate.** Similar to the gates within LSTMs, the reset and update gates control how much and which information to retain.



Recurrent Neural Networks (RNN)

Advantages of RNN:

1. An RNN remembers that each and **every information depends on time**. It is useful in **time series prediction** only because of the highlight point to remember previous inputs as well. This is known as long short-term memory.
2. RNNs are often used with convolution layers to elongate the effective pixel neighborhood.

Disadvantages of RNN:

1. Gradient vanishing problems and exploding problems.
2. Training an RNN for a computational problem is a very tedious task.
3. It **cannot execute very long sequences if tan h** is used as an activation function.

Convolutional Neural Networks (CNN)



Convolutional Neural Networks (CNN)

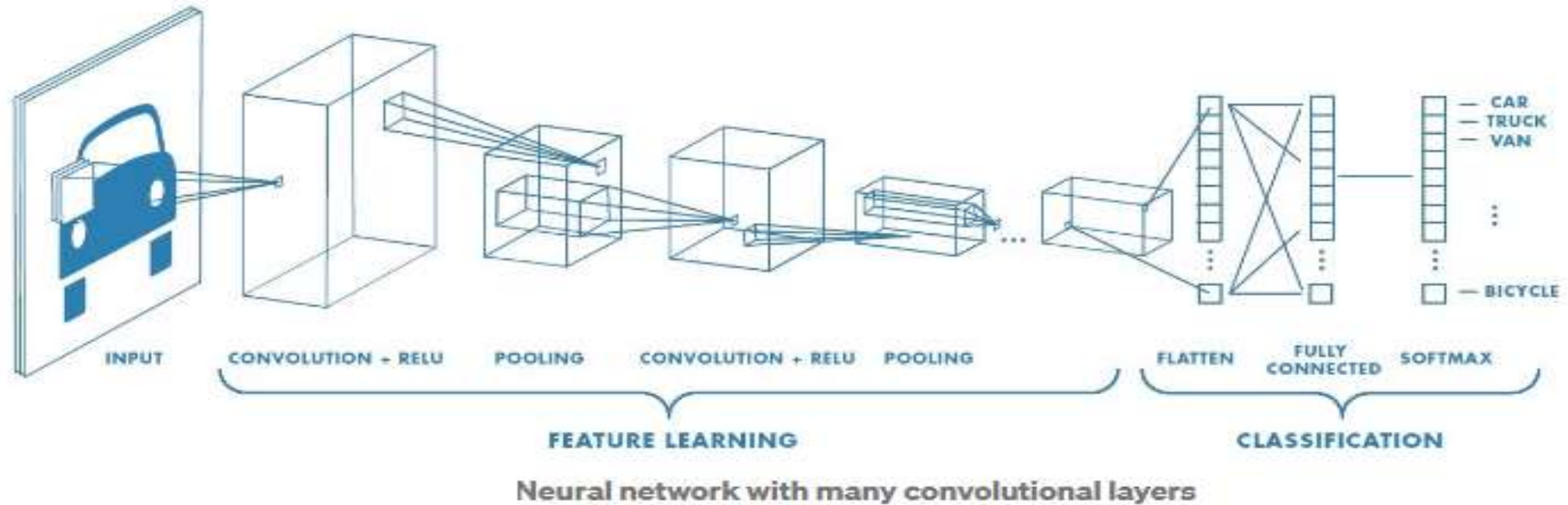
- A Convolutional Neural Network or CNN is a type of artificial neural network, which is widely **used for image/object recognition and classification**.
- Deep Learning recognizes objects in an image by using a CNN.
- CNN image classifications **takes an input image, process it and classify it under certain categories** (Eg., Dog, Cat, Tiger, Lion).
- Deep learning **CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1.**

Convolutional Neural Networks (CNN)



Convolutional Neural Networks (CNN)

➤ The complete flow of CNN to process an input image and classifies the objects based on values is as follows.



Convolutional Neural Networks (CNN)



Convolution Layer:

- Convolution is the first layer to extract features from an input image.
- Convolution preserves the relationship between pixels by learning image features using small squares of input data.
- It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.
- CNNs perform convolutions on the image data using a filter or kernel, then produce a feature map.

Convolutional Neural Networks (CNN)



Convolution Layer:

In generally,

- An image matrix (volume) of dimension **$(h \times w \times d)$**
- A filter **$(f_h \times f_w \times d)$**
- Outputs a volume dimension **$(h - f_h + 1) \times (w - f_w + 1) \times 1$**

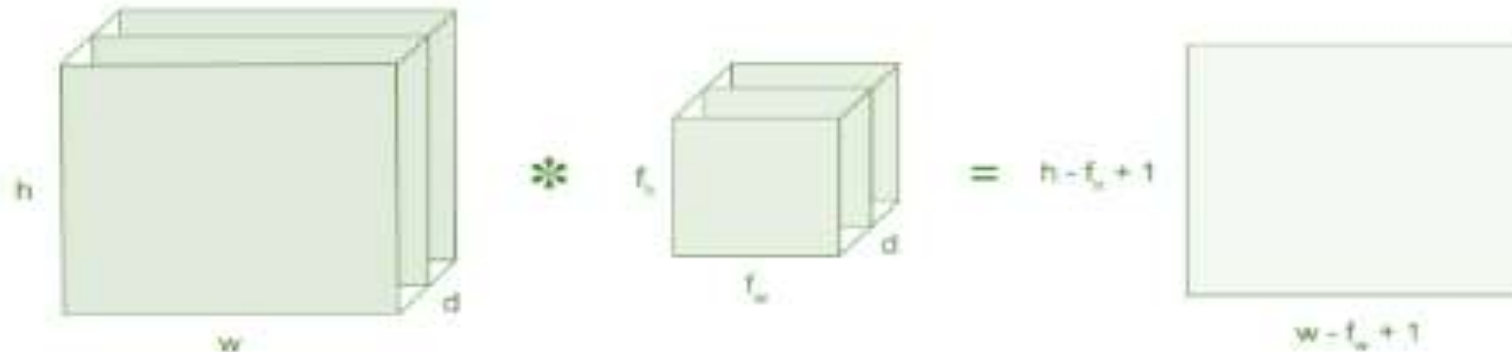


Image matrix multiplies kernel or filter matrix

Convolutional Neural Networks (CNN)



Convolution Layer:

➤ Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below

➤ The convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “Feature Map” as output shown in below

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

*

1	0	1
0	1	0
1	0	1

Image Matrix

3 x 3 - Filter Matrix

Image matrix multiplies kernel or filter matrix

1 _{x1}	1 _{x0}	1 _{x1}	
0 _{x0}	1 _{x1}	1 _{x0}	
0 _{x1}	0 _{x0}	1 _{x1}	

Image

4		

Convolved
Feature

Convolutional Neural Networks (CNN)



➤ Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

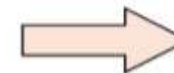
Strides:

➤ Stride is the number of pixels shifts over the input matrix.

➤ When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

1	2	3	4	5	6	7
11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57
61	62	63	64	65	66	67
71	72	73	74	75	76	77

Convolve with 3x3
filters filled with ones



108	126	
288	306	

Stride of 2 pixels

Convolutional Neural Networks (CNN)



Padding:

- Sometimes filter does not fit perfectly fit the input image.
- We have two options:
 1. Pad the picture with zeros (zero-padding) so that it fits.
 2. Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

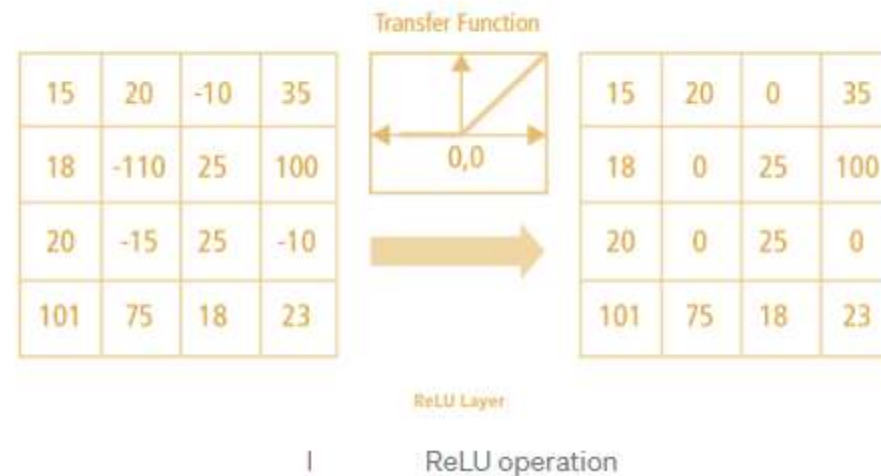
Non Linearity (ReLU):

- ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0, x)$.

Convolutional Neural Networks (CNN)



➤ ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.



➤ There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU.

➤ Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

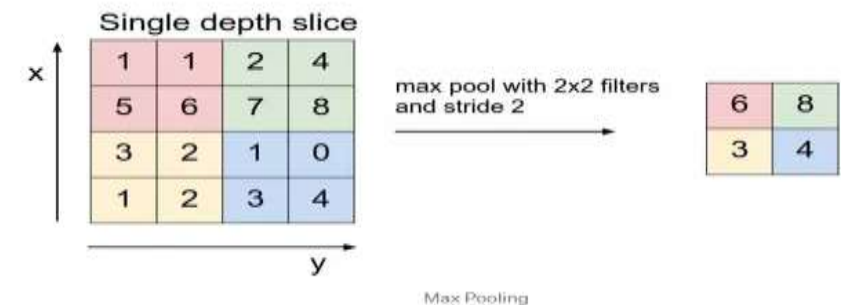
Convolutional Neural Networks (CNN)



Pooling Layer:

➤ Pooling layers section would **reduce the number of parameters when the images are too large**. Spatial pooling also called **subsampling or downsampling** which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

1. Max Pooling
2. Average Pooling
3. Sum Pooling



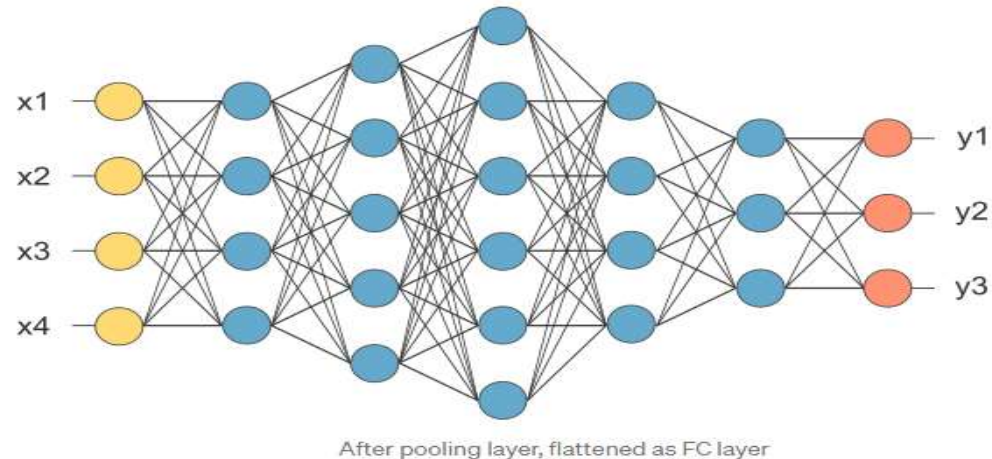
➤ Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

Convolutional Neural Networks (CNN)



Fully Connected Layer:

➤ It is called as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

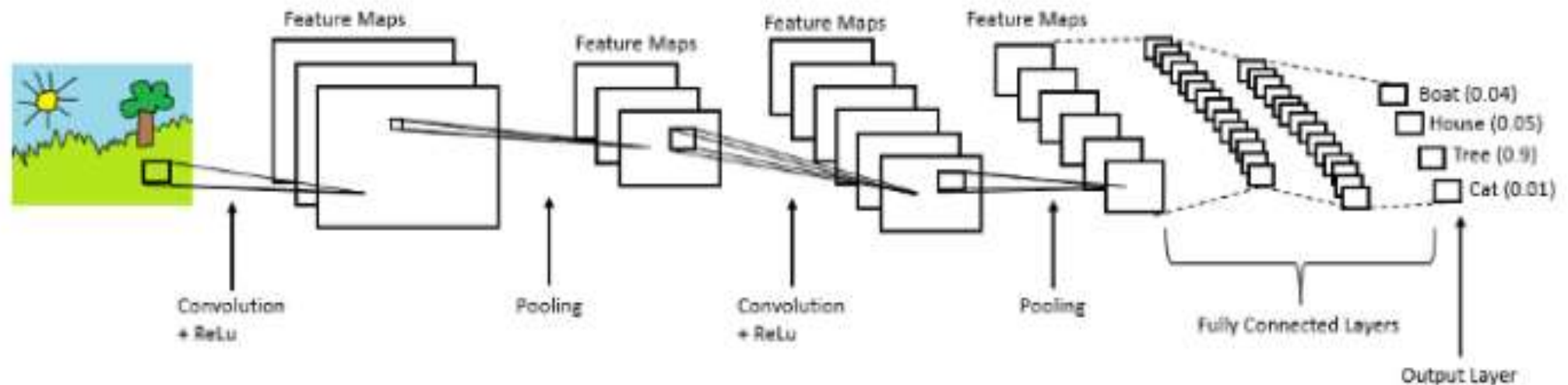


➤ the feature map matrix will be converted as vector (x1, x2, x3, ...). With the fully connected layers, we combined these features together to create a model.

Convolutional Neural Networks (CNN)



➤ Finally, we have an **activation function** such as softmax or sigmoid to classify the **outputs** as cat, dog, car, truck etc.,



Complete CNN architecture

Convolutional Neural Networks (CNN)



➤ some popular CNN architectures are AlexNet, VGGNet, GoogLeNet, and ResNet.

Procedure to implement CNN :

1. Provide input image into convolution layer
2. Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
3. Perform pooling to reduce dimensionality size
4. Add as many convolutional layers until satisfied
5. Flatten the output and feed into a fully connected layer (FC Layer)
6. Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

Long Short-Term Memory Networks(LSTM)



Long Short-Term Memory Networks:

➤ The most popular and efficient way to deal with gradient problems, i.e., Long Short-Term Memory Network (LSTMs).

Long-Term Dependencies:

➤ Suppose we want to predict the last word in the text:

“The clouds are in the _____.”

The most obvious answer to this is the “sky.”

➤ Consider this sentence:

“I have been staying in Spain for the last 10 years...I can speak fluent _____.”

The word we predict will depend on the previous few words in context.

Here we need the context of Spain to predict the last word in the text, and the most suitable answer to this sentence is “Spanish.”

Long Short-Term Memory Networks(LSTM)



➤ The gap between the relevant information and the point where it's needed may have become very large. LSTMs help us solve this problem.

Long Short-Term Memory Networks:

➤ LSTMs are a special kind of Recurrent Neural Network — capable of learning long-term dependencies by remembering information for long periods is the default behavior.

➤ All recurrent neural networks are in the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

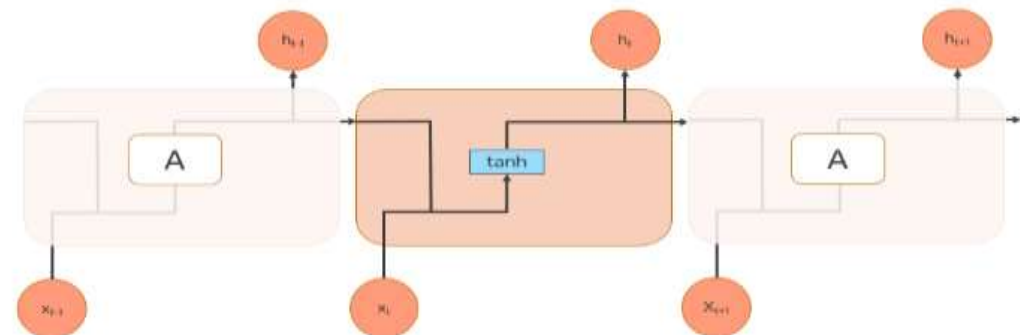
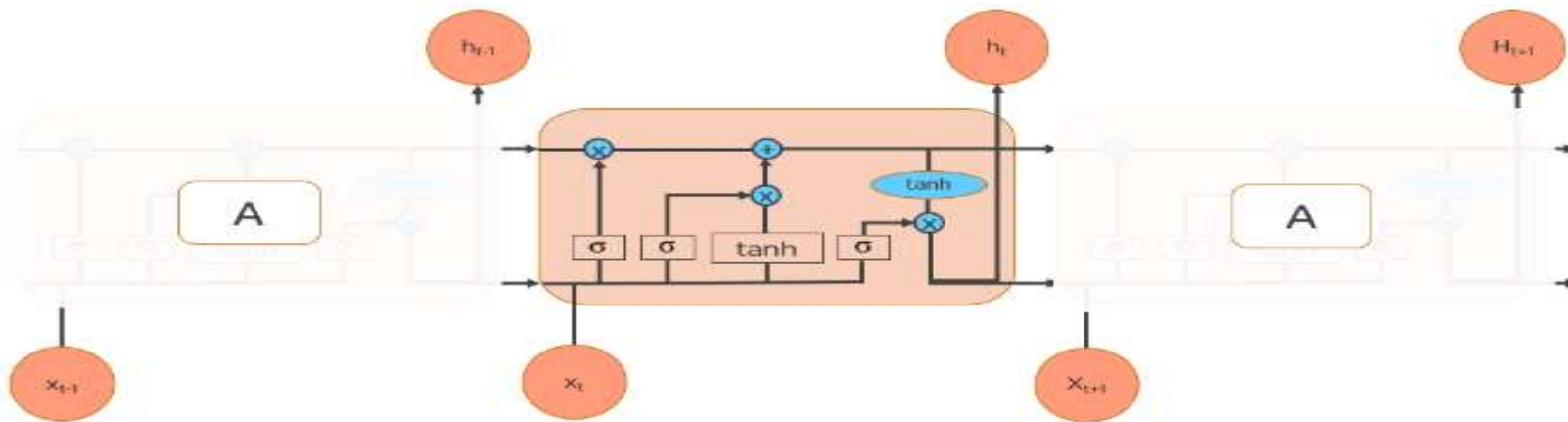


Fig: Long Short Term Memory Networks

Long Short-Term Memory Networks(LSTM)



➤ LSTMs also have a chain-like structure, but the repeating module is a bit different structure. Instead of having a single neural network layer, **four interacting layers are communicating extraordinarily.**

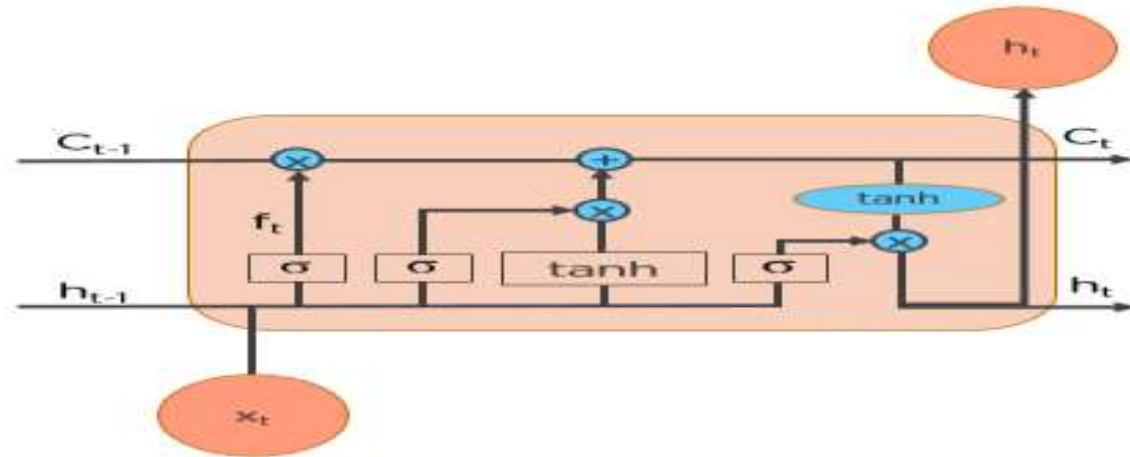


Long Short-Term Memory Networks(LSTM)



Workings of LSTMs:

➤ The diagrammatical represent of working of an LSTM are:



➤ LSTMs work in a **3-step process**.

Long Short-Term Memory Networks(LSTM)



Step 1: **Decide how much past data it should remember**

- The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step.
- The sigmoid function determines this.
- It looks at the previous state (h_{t-1}) along with the current input x_t and computes the function.

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

f_t = forget gate
Decides which information to
delete that is not important
from previous time step

Long Short-Term Memory Networks(LSTM)



Step 2: **Decide how much this unit adds to the current state**

- In the second layer, there **are two parts**. One is the **sigmoid function**, and the other is **the tanh function**.
- In the *sigmoid* function, it decides which values to let through (0 or 1).
- *tanh* function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

i_t = input gate
Determines which information to let through based on its significance in the current time step

Long Short-Term Memory Networks(LSTM)



Step 3: **Decide what part of the current cell state makes it to the output**

- The third step is to decide what the output will be.
- First, we run a sigmoid layer, which decides what parts of the cell state make it to the output.
- Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

o_t = output gate
Allows the passed in information to impact the output in the current time step

Long Short-Term Memory Networks(LSTM)



Applications of LSTM:

➤ **Some of the famous applications of LSTM includes:**

1. Language Modelling
2. Machine Translation
3. Image Captioning
4. Handwriting generation
5. Question Answering Chatbots



Applications to text

Applications to text

- One of the important application to text is Text Classification.

Text Classification Process

- The process of text classification starts with reading the document into the code.
- text classification is followed by series of data pre processing steps chosen based on business problem.
- Some of the important data preprocessing steps are:
 1. Tokenization
 2. Text normalization
 3. Feature Selection
 4. Feature Extraction
 5. Tagging of the data to predefined categories

Applications to text



Text Classification Process

1. Tokenization:

- It **breaks down** longer string of text into smaller pieces.

for example:

“This sentence needs to be tokenized”

- will be broken to subsequent words:

{‘This’, ‘sentence’, ‘needs’, ‘to’, ‘be’, ‘tokenized’}

2. Text normalization:

- It aims **to bring all the text content to same level.**

- Some of the Text normalization options are :

- I. Basic Normalization steps
- II. Stemming
- III. Lemmatization
- IV. Stop Word Removal
- V. Vectorization

Applications to text



Text Classification Process

Basic Normalization steps:

➤ Lowercasing text, Removal of Punctuations/Tags/whitespaces.

Stemming :

➤ Stemming **removes affixes** (prefix, suffix, infixes, circumfixes) **from the word**.

➤ **for example:**

1. studies will become studi and
2. studying will become study.



Applications to text

Text Classification Process

Lemmatization:

- Obtains the **canonical/dictionary** form of the word.
- For example:
 studies and studying will be converted to **study**.
- It is **useful** in context **where words need to retain meaning after pre-processing**.

Stop Word Removal:

- **Removal of common words** such as the, and, is, a that provide no value to the overall sentence.

Vectorization:

- **The text sequences are transformed into numerical features** that can be used in the model. TF-IDF, Count vectorizer are some of the commonly used approaches.



Applications to text

Text Classification Process

3.Feature Selection

- This techniques **select a subset of the features** based on their importance.
- **Document Frequency** is one of the common **Feature Selection** methods used, where words/features present below certain frequency in the document are filtered out.

4. Feature Extraction

- It is an optional step in some business scenarios, where **additional features are created from pre-existing features.**
- **Clustering methods** is one technique used to add new features.

Applications to text



Text Classification Process

5. Tagging of the data to predefined categories

- The final step in this process is the tagging of the data to predefined categories using one of the following methods:
 - a) Manual tagging
 - b) Rule Based Filtering or string-matching algorithms such as fuzzy matching.
 - c) Learning Algorithms such as **Neural Networks** that can utilize several hundred features to tag the Text content.
- The Learning Algorithms can be classified into **two approaches**:
 - I. **Unsupervised Learning**
 - II. **Supervised Learning**



Applications to text

Text Classification Process

Unsupervised Learning:

- It is applied where there is lack of previously tagged data.
- Techniques like clustering and associating rule-based algorithms can be applied to group together similar text.
- An example is segmentation of customers into groups based on their details, purchase history and behavior.
- These groups can then be further analyzed to identify patterns that allows for customizing customer approach.
- The unsupervised approach to text classification looks for similar patterns and structures between text to group them together.
- It finds application in real world where the data volume is too large to be completely classified or the labels are not predefined.



Applications to text

Text Classification Process

Unsupervised Learning:

Example :

CRM (Customer Relationship Management) Automation – Speeding up response time to customer complaints.

- Any product-based company needs to act quickly on customer complaints to provide efficient customer service.
- The task of reading every customer complaint/ feedback can be both time consuming and error prone.
- **By application of unsupervised learning methods, the complaints can be divided into main topics such as Technical Issues, Subscription related queries etc that can be assigned to respective teams to work upon.**



Applications to text

Text Classification Process

Unsupervised Learning:

➤ the Process for CRM **Automation** using unsupervised Learning is:

1. **Data Extraction and Pre-Processing**
2. **Clustering**
3. **Topic Modelling**

1. Data Extraction and Pre-Processing:

➤ Data is extracted from the **mails of customers or the CRM database.**

➤ This data is then tokenized, normalized and **TF-IDF algorithm(Term Frequency — Inverse Document Frequency)** is used to extract relevant features from the data.

➤ Once the data has been cleaned and features are extracted it is ready for the next step.



Applications to text

Text Classification Process

Unsupervised Learning

2. Clustering:

- The clustering approach aims at **grouping together similar unlabeled texts in same group (clusters)**.
- The DBSCAN approach of clustering is applied to group together complaints with similar text content.
- This method **doesn't require us to define the number of clusters needed**, as it gains that information based on initial parameters setps (the min distance between two points to be part of same cluster) , minPoints(minimum number of points that can form a dense region).



Applications to text

Text Classification Process

Unsupervised Learning

2. Clustering:

➤ The clusters give us an idea on the **number of complaint types that exist**. However, to get any meaningful insights from these clusters the complaint types need to be identified and named. This takes us to another technique in unsupervised text **classification**.



Applications to text

Text Classification Process

Unsupervised Learning:

3. Topic Modeling:

- The topic model **discovers abstract topics from a collection of documents.**
- Examples of topic models are:
 - I. Latent Dirichlet Allocation (LDA),
 - II. TextRank,
 - III. Latent Semantic Analysis(LSA).
- **The LDA topic model takes input of the customer complaints and assigns topic to them.** The basis it functions on is that every document- the complaint here, is a mixture of small topics. It does so by calculating the probability of the words in the complaint occurring in given topics.



Applications to text

Text Classification Process

Supervised Learning:

- It is applied where there is enough volume of accurately categorized data available.
- The ML algorithms learn the mapping function between the text and the tags based on already categorized data.
- Algorithms such as SVM, Neural Networks, Random Forest are commonly used for text classification.
- The supervised approach to text classification works with training models with already classified/tagged text. This approach has better accuracy and scaling up possibilities than unsupervised learning techniques.



Applications to text

Text Classification Process

Supervised Learning:

Example : Deriving insights on new product from Social Media Posts

“Statistics show that 49% of world’s population, 3.8 Billion people today use social media.”

➤ At launch of a new product, companies often try to understand the public perception on it by analyzing the social media posts shared about their product. Since reading each and every post can be tiresome, they utilize Supervised Learning Techniques to derive insights.



Applications to text

Text Classification Process

Supervised Learning:

1. Topic Labelling:

- After data extraction from social media posts, pre-processing, the first step is to find out what product features are been discussed the most by using **Contextual Semantic Search(CSS)**.
- It takes concepts such as Price, feature, customer service, user accessibility as inputs and filters the posts to relevant concepts, respectively.
- Sentiment Analysis can then provide insights on the social sentiment across these topics.



Applications to text

Text Classification Process

Supervised Learning

2. Sentiment Analysis

- It can lead way in understanding if the social perception is positive, negative, or neutral towards the new product and its features.
- The algorithms applied for this can either be rule based (look for presence of certain words that are grouped as positive/negative/neutral) to decipher overall sentiment of the post or ML based that uses previously tagged posts to train upon and predict the final sentiment on posts.
- Sentiment Analysis on competitor's social post can be used to obtain a benchmark.



Applications to text

Text Classification Process

What is Topic Modeling?

- **Topic modeling** is a machine learning technique that **automatically analyzes text data to determine cluster words for a set of documents**. This is known as 'unsupervised' machine learning because it doesn't require a predefined list of tags or training data that's been previously classified by humans.
- **Topic classification** models **require training, they're known as 'supervised' machine learning techniques**. Topic classification needs to know the topics of a set of texts before analyzing them. Using these topics, data is tagged manually so that a topic classifier can learn and later make predictions by itself.
- **Note:** unsupervised techniques are a short-term or quick-fix solution, while supervised techniques are more of a long-term solution that will help your business grow.



Applications to text

Text Classification Process

How Does Topic Modeling Work?

- Topic modeling involves counting words and grouping similar word patterns to infer topics within unstructured data.
- *Topic Modeling* refers to the process of dividing a corpus of documents in two:
 1. A list of the topics covered by the documents in the corpus (a collection of written texts)
 2. Several sets of documents from the corpus grouped by the topics they cover.



Applications to text

Text Classification Process

- Two topic modeling methods, namely,
 - I. *Latent Semantic Analysis (LSA)* and
 - II. *Latent Dirichlet Allocation (LDA)*.

Latent Semantic Analysis (LSA)

- *Latent Semantic Analysis (LSA)* is one of the most frequent topic modeling methods analysts make use of.
- It is based on what is known as the [distributional hypothesis](#) which states that the semantics of words can be grasped by looking at the contexts the words appear in. In other words, under this hypothesis, the semantics of two words will be similar if they tend to occur in similar contexts.



Applications to text

Text Classification Process

Latent Dirichlet Allocation (LDA)

- *Latent Dirichlet Allocation (LDA)* and LSA are based on the same underlying assumptions: the distributional hypothesis, (i.e. similar topics make use of similar words) and the statistical mixture hypothesis (i.e. documents talk about several topics) for which a statistical distribution can be determined.
- The purpose of LDA is **mapping each document in our corpus to a set of topics which covers a good deal of the words in the document.**



Images

Understanding what an Image actually is?

- Image is basically a **two-dimensional signal**.
- The **signal function is $f(x,y)$** , where the value of x and y at a **point generates the pixel at the point**.
- Image is basically a two-dimensional **array consisting of numbers between 0 and 255**.
- Various factors are involved in Image Processing.

Image Processing help in :

1. Improvement in digital information stored by us.
2. Making working with images automated.
3. Better image optimization leading to efficient storage and transmission.

Images



Image Processing Uses :

- 1. Image Correction, Sharpening, and Resolution Correction**
- 2. Filters on Editing Apps and Social Media**
- 3. Medical Technology**
- 4. Computer / Machine Vision**
- 5. Pattern recognition**
- 6. Video Processing**



Images

Image Processing Uses :

1. Image Correction, Sharpening, and Resolution Correction

- **we could make old images better.** And that is possible nowadays. **Zooming, sharpening, edge detection, high dynamic range edits** all fall under this category. All these steps help in enhancing the image. Most editing software and Image correction code can do these things easily.

2. Filters on Editing Apps and Social Media

- Most editing apps and social media apps **provide filters** these days.



Original Image



Filtered Image



Images

- Filters make the **image** look more visually appealing.
- **Filters are** usually a set of **functions that change the colors and other aspects in an image that make the image look different.** Filters are an interesting application of Image processing.

3. Medical Technology :

- In the medical field, Image Processing is used for various tasks like **PET scan, X-Ray Imaging, Medical CT, MRI, UV imaging, Cancer Cell Image processing,** and much more. The introduction of Image Processing to the medical technology field has greatly **improved the diagnostic**

- The image on the left is the original image.
- The image on the right is the processed image.
- We can see that the processed image is far better and can be used for better diagnostics.






Images

Image Processing Uses :

4. Computer / Machine Vision :

➤ One of the most interesting and useful applications of Image Processing is in **Computer Vision**. Computer Vision is used to **make the computer see, identify things, and process the whole environment as a whole**. An important use of Computer Vision is **Self Driving cars, Drones etc**. CV helps in obstacle detection, path recognition, and understanding the environment.



➤ This is how typical Computer Vision works for Car Autopilots.

➤ The computer takes in live footage and analyses other cars, the road, and other obstacles.



Images



Image Processing Uses :

5. Pattern recognition:

➤ Pattern recognition is a part of Image Processing that involves AI and Machine Learning. Image processing is used **to find out various patterns and aspects in images**. Pattern Recognition is **used for Handwriting analysis, Image recognition, Computer-aided medical diagnosis**, and much more.

6. Video Processing:

➤ Video is basically a **fast movement of images**. Various image processing techniques are used in Video Processing. Some **methods of Video Processing are noise removal, image stabilization, frame rate conversion**, detail enhancement, and much more.



Images

Getting started with Image Processing in Python:

- Python Imaging Library (PIL) is used for various image processing tasks.

Installation:

```
pip install pillow
```

- With PIL installed, we can now move to the code. First, we work with some matplotlib functions.

```
import matplotlib.image as img
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

- The following image will be read.
It is named image1.jpg.





Images

Getting started with Image Processing in Python:

```
# reading jpg image
```

```
img = img.imread('image1.jpg')
```

```
plt.imshow(img)
```

```
# modifying the shape of the image
```

```
lum1 = img[:, :, 0]
```

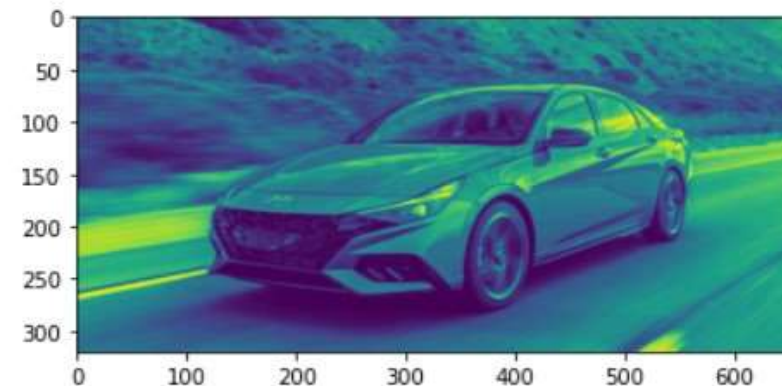
```
plt.imshow(lum1)
```

➤ Now the image shape is modified.

<matplotlib.image.AxesImage at 0x1e6d5ee3bc8>



<matplotlib.image.AxesImage at 0x1e6d5f9da08>



Images

Getting started with Image Processing in Python:

➤ Now we will change it into the “hot” colourmap.

```
plt.imshow(lum1, cmap='hot')
```

```
plt.colorbar()
```

Image output looks:

➤ Now we try a different colormap.

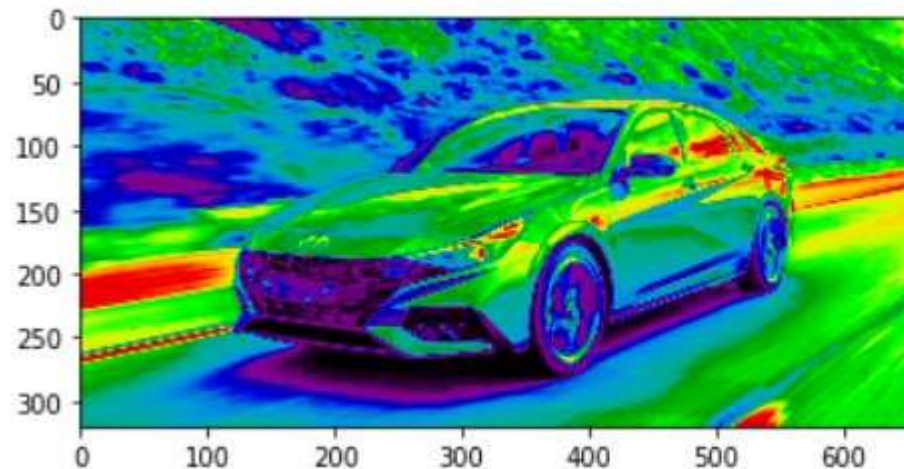
```
imgplot = plt.imshow(lum1)
```

```
imgplot.set_cmap('nipy_spectral')
```

Image output:



<matplotlib.colorbar.Colorbar at 0x1e6d6053d48>





Images

➤ Now let us have a look at why we called an image a 2D array.

#data type of lum1

```
print(type(lum1))
```

Output: <class 'numpy.ndarray'>

```
print(lum1)
```

```
[ [ 92 91 89 ... 169 168 169]
```

```
[110 110 110 ... 168 166 167]
```

```
[100 103 108 ... 164 163 164]
```

```
...
```

```
[ 97 96 95 ... 144 147 147]
```

```
[ 99 99 98 ... 145 139 138]
```

```
[102 102 103 ... 149 137 137]]
```

➤ The dots are just there to show that there are many more data points in between. But one thing is sure, is that it is all numeric data.



Images

➤ To find the size of the array.

```
len(lum1)
```

Output: 320

```
len(lum1[300])
```

Output: 658

➤ This gives us the number of pixels, and dimensions of the image: 320*658.

Images

work with PIL

from PIL import Image

We will use this image file, named as: people.jpg.

```
img2 = Image.open('people.jpg')
```

```
plt.imshow(img2)
```

<matplotlib.image.AxesImage at 0x1e6d614bf88>





Images

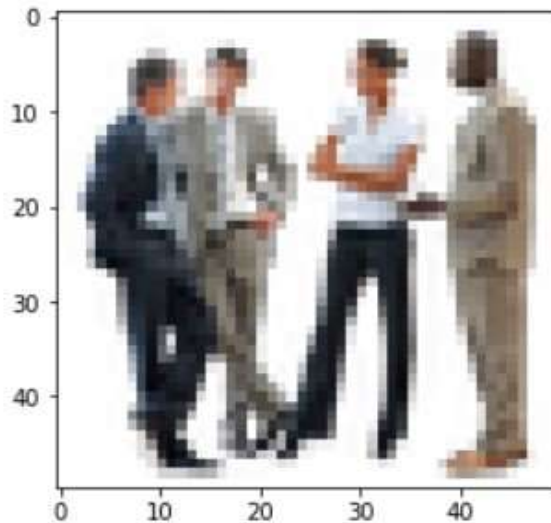
work with PIL

➤ we resize the image.

`img2.thumbnail((50, 50), Image.ANTIALIAS)` # resizes image in-place

`imgplot = plt.imshow(img2)`

`imgplot1 = plt.imshow(img2, interpolation="nearest")`



Images

work with PIL

```
imgplot2 = plt.imshow(img2, interpolation="bicubic")
```

- For Pattern Recognition and Computer Vision algorithms, it becomes difficult to process the images if they are very sharp. Thus blurring is done to make the images smooth.
- Blurring also makes the colour transition in an image, from one side to the other, a lot more smooth.





Images

work with PIL

➤ let us verify the dimensions of the car image, we worked on earlier.

```
file='image1.jpg'
```

```
with Image.open(file) as image:
```

```
    width, height = image.size
```

#Image width, height is be obtained

```
In [16]: width
```

```
Out[16]: 658
```

```
In [17]: height
```

```
Out[17]: 320
```

➤ These are the dimensions we got earlier as well. So we can conclude that the image is 320*658.



Images

work with PIL

➤ Let us also try **rotating and transposing the image.**

#Relative Path

```
img3 = Image.open("image1.jpg")
```

#Angle given

```
img_rot= img3.rotate(180)
```

#Saved in the same relative location

```
img_rot.save("rotated_picture.jpg")
```





Images

work with PIL

#transposing image

```
transposed_img = img3.transpose(Image.FLIP_LEFT_RIGHT)
```

#Saved in the same relative location

```
transposed_img.save("transposed_img.jpg")
```

transposed_img





Images

Learning on Python, Image processing

- The function **tensorflow.io.read_file** takes the file name as its required argument and returns the contents of the file as a tensor with type **tensorflow.string**. When the input file is an image, the output of **tensorflow.io.read_file** will be the raw byte data of the image file.
- implementation in Python using the soccer ball image.

```
import tensorflow as tf
values = tf.io.read_file('soccer_ball.jpg')
```
- The **decoding function** that we use depends on the **format of the image**.
- For generic decoding (i.e. decoding any image format), we use **tensorflow.image.decode_image** but if the input is a JPEG image we use **tensorflow.image.decode_jpeg**.



Images

Learning on Python, Image processing

Dataset

- Normally when we do **image related tasks** we're **dealing with a large amount of image data**. In this case, it's best to use a TensorFlow dataset, i.e. **tensorflow.data.Dataset**, to store all the images.
- We can create a dataset using the **from_tensor_slices** function.
- The **Dataset** class makes it easier and more efficient to perform tasks with all the image files. After we create a dataset with the image files, we will need to decode each file's contents into usable pixel data.
- Since the **decode_image** function works for single image files, we will need to use the dataset object's **map** function to apply **decode_image** to each image file in our dataset.



ImageClassification

What is Image Classification?

- Classification between objects is a fairly easy task for us, but it has proved to be a complex one for machines and therefore image classification has been an important task within the field of computer vision.
- Image classification refers to the labeling of images into one of a number of predefined classes. (OR)
- Image classification is a supervised learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos.
- There are potentially n number of classes in which a given image can be classified. Manually checking and classifying images could be a tedious task especially when they are massive in number (say 10,000) and therefore it will be very useful if we could automate this entire process using computer vision.



ImageClassification

Some examples of image classification include:

- Labeling an x-ray as cancer or not (binary classification).
- Classifying a handwritten digit (multiclass classification).
- Assigning a name to a photograph of a face (multiclass classification).

Structure of an Image Classification Task:

- Image classification task mainly consists of 4 steps.

- 1. Image Preprocessing**
- 2. Detection of an object**
- 3. Feature extraction and**
- 4. Classification of the object**



ImageClassification

Image Preprocessing :

The aim of this process is to **improve the image data(features) by suppressing unwanted distortions and enhancement of some important image features** so that our Computer Vision models can benefit from this improved data to work on.

Detection of an object:

Detection refers to the **localization of an object** which means the **segmentation of the image and identifying the position of the object** of interest.

Feature extraction and Training:

This is a crucial step wherein statistical or deep learning methods are used to identify the most interesting patterns of the image, features that might be **unique to a particular class and that will, later on, help the model to differentiate between different classes**. This process where the model learns the features from the dataset is called model training.

ImageClassification



Classification of the object :

This step categorizes **detected objects** into **predefined classes** by using a **suitable classification technique** that compares the image patterns with the target patterns.

ImageClassification



Steps for image pre-processing:

1. Read image
2. Resize image
3. Data Augmentation
 - I. Gray scaling of image
 - II. Reflection
 - III. Gaussian Blurring
 - IV. Histogram Equalization
 - V. Rotation
 - VI. Translation



ImageClassification

Step 1 : Reading Image

➤ In this step, we simply store the path to our image dataset into a variable and then we create a function to load folders containing images into arrays so that computers can deal with it.

Step 2 : Resize image

➤ Some images captured by a camera and fed to our AI algorithm vary in size, therefore, we should establish a base size for all images fed into our AI algorithms by resizing them.

➤ Sample code for resizing images into 229x229 dimensions:

```
img = cv2.resize(img, (229,229))
```



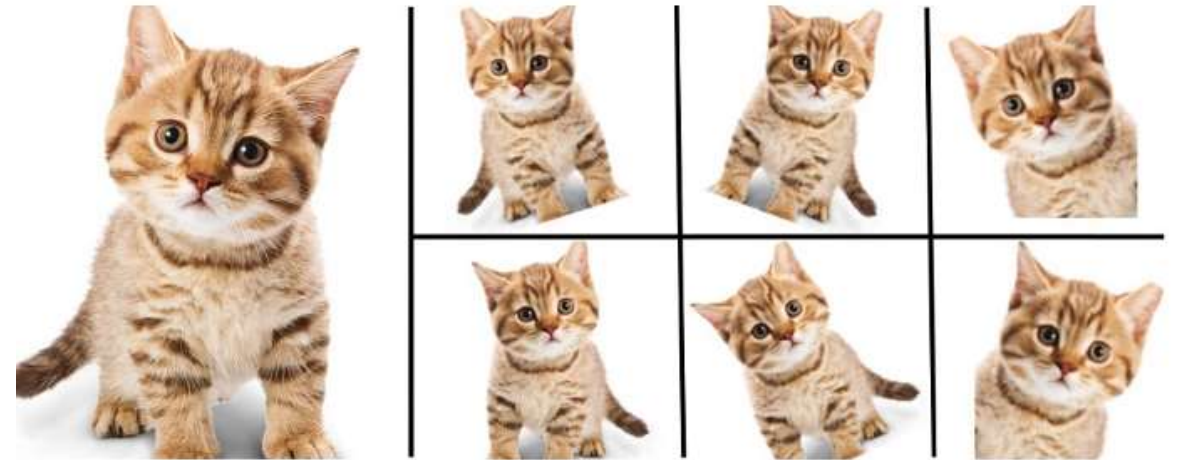
ImageClassification

Step 3 : Data Augmentation

➤ Data augmentation is a way of **creating new 'data' with different orientations**. The benefits of this are two-fold.

1. The ability to generate 'more data' from limited data.
2. It prevents over fitting.

Example:



Enlarge your Dataset



ImageClassification

Data Augmentation Techniques:

1. Gray Scaling:

- The image will be **converted to gray scale** (range of gray shades from white to black) the computer will assign each pixel a value based on how dark it is. All the numbers are put into an array and the computer does computations on that array.
- Sample code to convert an RGB(3 channels) image into a Gray scale image:

```
import cv2
```

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

RGB Image



Grayscale Image





ImageClassification

Data Augmentation Techniques:

2. Reflection/Flip:

- We can **flip images horizontally and vertically**. Some frameworks do not provide function for vertical flips. But, a vertical flip is equivalent to rotating an image by 180 degrees and then performing a horizontal flip.

Example:

horizontal flip

```
img = cv2.flip(img, 0)
```

vertical flip

```
img = cv2.flip(img, 1)
```



ImageClassification



Data Augmentation Techniques:

3. Gaussian Blurring:

➤ Gaussian blur (also known as Gaussian smoothing) is the **result of blurring an image by a Gaussian function**. It is a widely used effect in graphics software, typically **to reduce image noise**.

Example:

```
from scipy import ndimage
```

```
img = ndimage.gaussian_filter(img, sigma= 5.11)
```

Image with blur radius = 5.1



ImageClassification



Data Augmentation Techniques:

4. Histogram Equalization:

➤ Histogram equalization is another image processing technique **to increase global contrast of an image using the image intensity histogram**. This method needs no parameter, but it sometimes results in an unnatural looking image.

Example:

```
# histogram equalization function
```

```
def hist(img):
```

```
    img_to_yuv = cv2.cvtColor(img,cv2.COLOR_BGR2YUV)
```

```
    img_to_yuv[:, :, 0] = cv2.equalizeHist(img_to_yuv[:, :, 0])
```

```
    hist_equalization_result = cv2.cvtColor(img_to_yuv, cv2.COLOR_YUV2BGR)
```

```
➤ return hist_equalization_result
```

Before



After



ImageClassification



Data Augmentation Techniques:

5. Rotation:

➤ This is another image augmentation technique. Rotating an image might not preserve its original dimensions.

Example:

```
import random
```

```
# function for rotation
```

```
def rotation(img):
```

```
    rows,cols = img.shape[0],img.shape[1]
```

```
    randDeg = random.randint(-180, 180)
```

```
    matrix = cv2.getRotationMatrix2D((cols/2, rows/2), randDeg, 0.70)
```

```
    rotated = cv2.warpAffine(img, matrix, (rows, cols),
```

```
    borderMode=cv2.BORDER_CONSTANT,borderValue=(144, 159, 162))
```

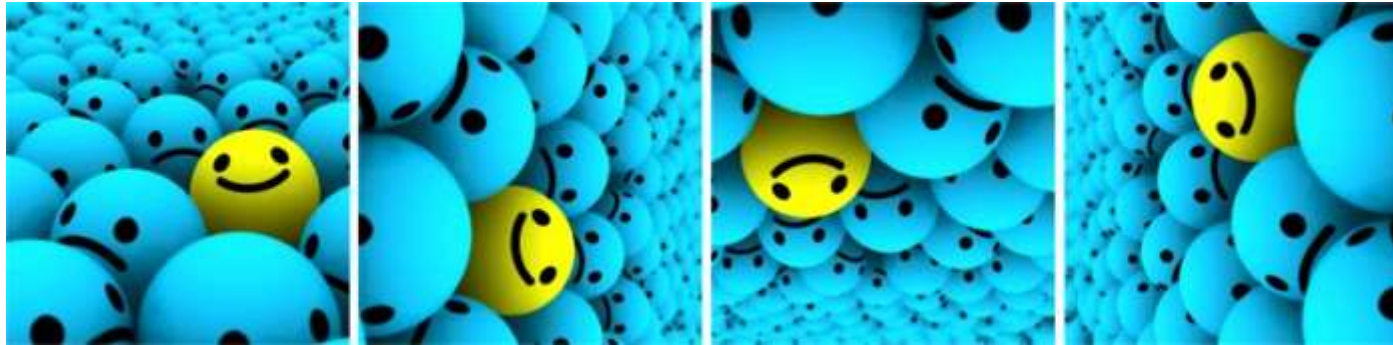
```
    return rotated
```



ImageClassification

Data Augmentation Techniques:

Rotation:



➤ The images are rotated by 90 degrees clockwise with respect to the previous one, as we move from left to right.

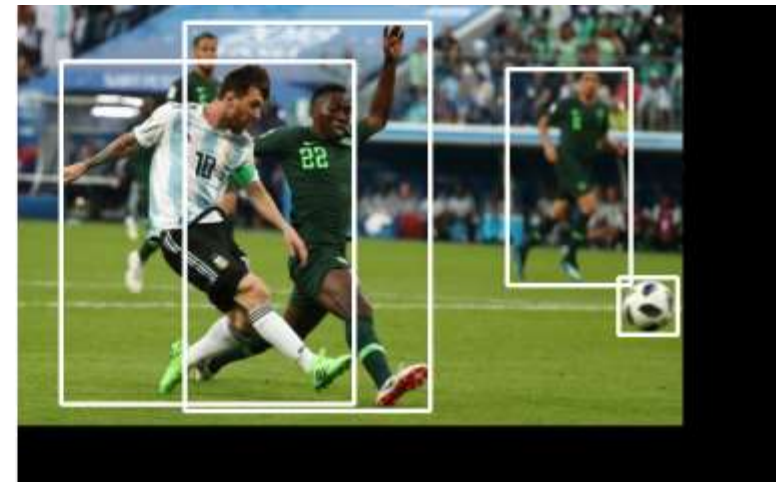
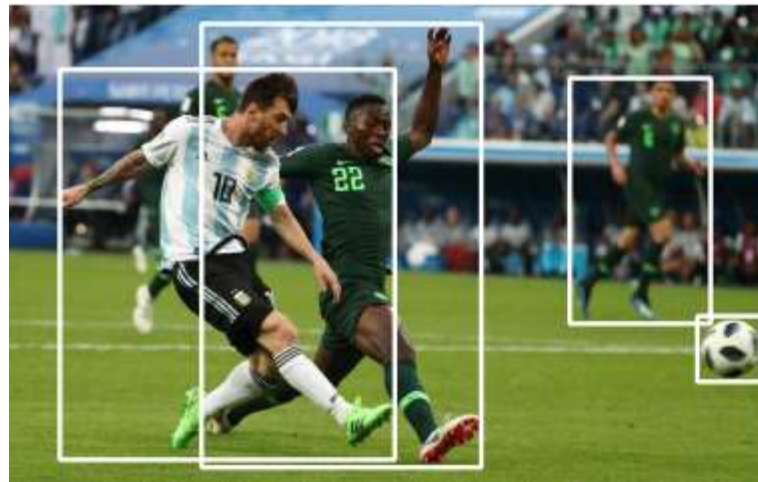


ImageClassification

Data Augmentation Techniques:

6. Translation:

- Translation just involves moving the image along the X or Y direction (or both).
- This method of augmentation is very useful as most objects can be located at almost anywhere in the image. This forces our feature extractor to look everywhere.





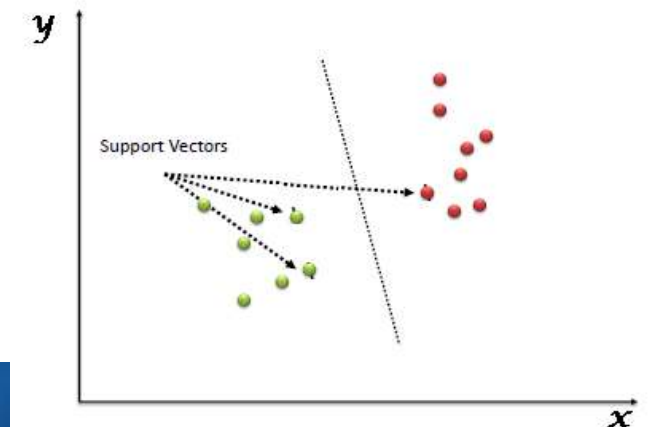
ImageClassification

Image Classification Techniques

➤ some statistical machine learning classifiers like Support Vector Machine and Decision Tree and deep learning architectures like Convolutional Neural Networks are used for image classification.

1. Support Vector Machines

- It is a supervised machine learning algorithm used for both regression and classification problems.
- When used for classification purposes, it separates the classes using a linear boundary.
- It builds a hyper-plane or a set of hyper-planes in a high dimensional space and good separation between the two classes is achieved by the hyperplane that has the largest distance to the nearest training data point of any class.





ImageClassification

1. Support Vector Machines

- The real power of this algorithm depends on the kernel function being used.
- The most commonly used kernels are:
 - I. Linear Kernel
 - II. Gaussian Kernel
 - III. Polynomial Kernel

2. Decision Trees

- Decision trees are based on a hierarchical rule-based method and permits the acceptance and rejection of class labels at each intermediary stage/level.

ImageClassification

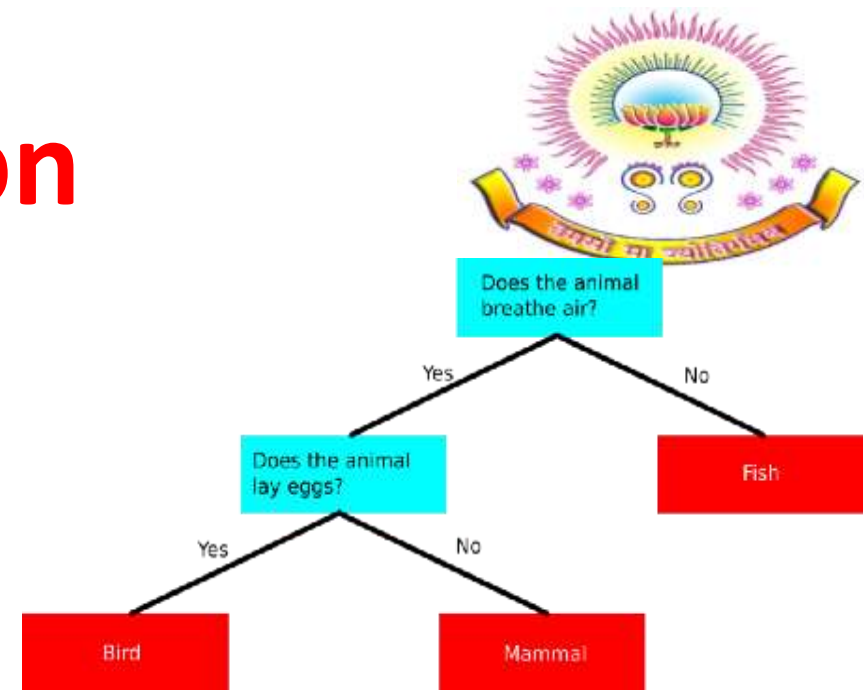
2. Decision Trees

➤ This method consists of 3 parts:

I. Partitioning the nodes

II. Finding the terminal nodes

III. Allocation of the class label to terminal node



3. K Nearest Neighbor

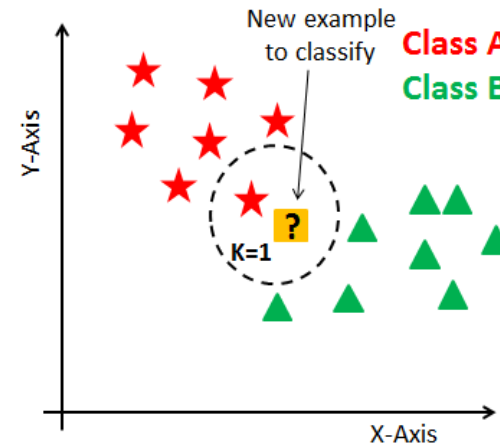
➤ The k-nearest neighbor is by far the most simple machine learning algorithm.

➤ This algorithm simply relies on the **distance between feature vectors and classifies unknown data points by finding the most common class among the k-closest examples.**

ImageClassification



3. K Nearest Neighbor



- there are two categories of images and that each of the data points within each respective category are grouped relatively close together in an n-dimensional space.
- In order to apply the k-nearest Neighbor classification, we need to define a distance metric or similarity function. Common choices include the [Euclidean distance](#) and [Manhattan distance](#)

ImageClassification



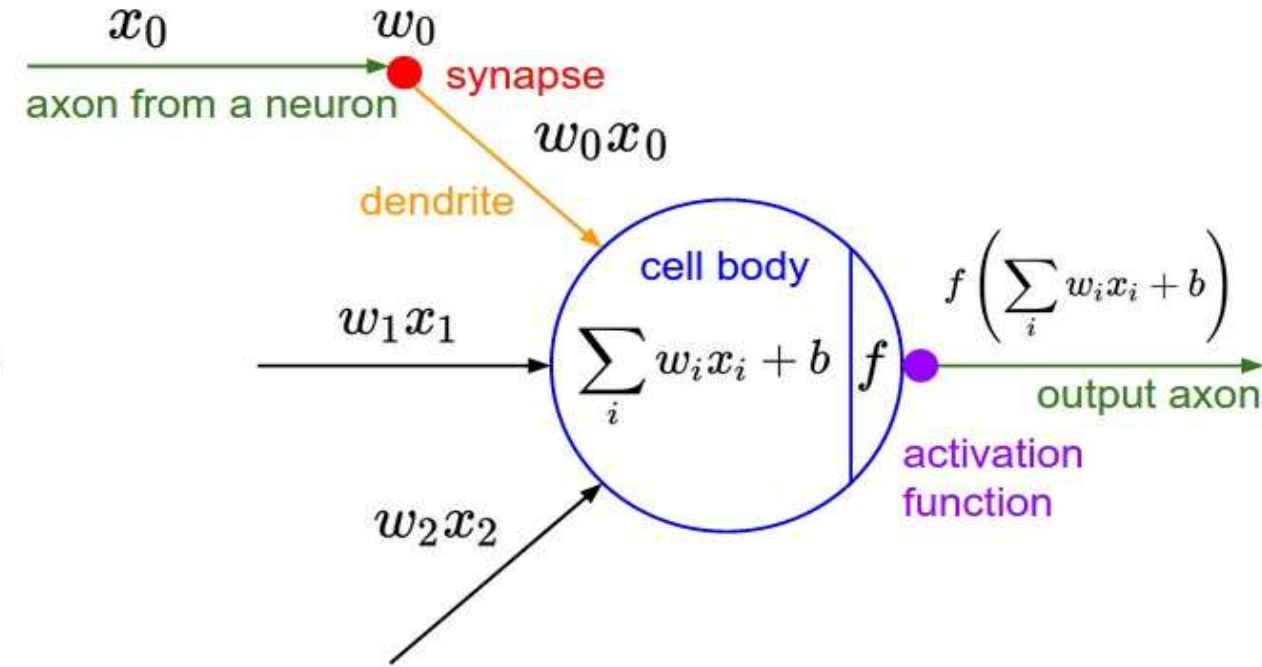
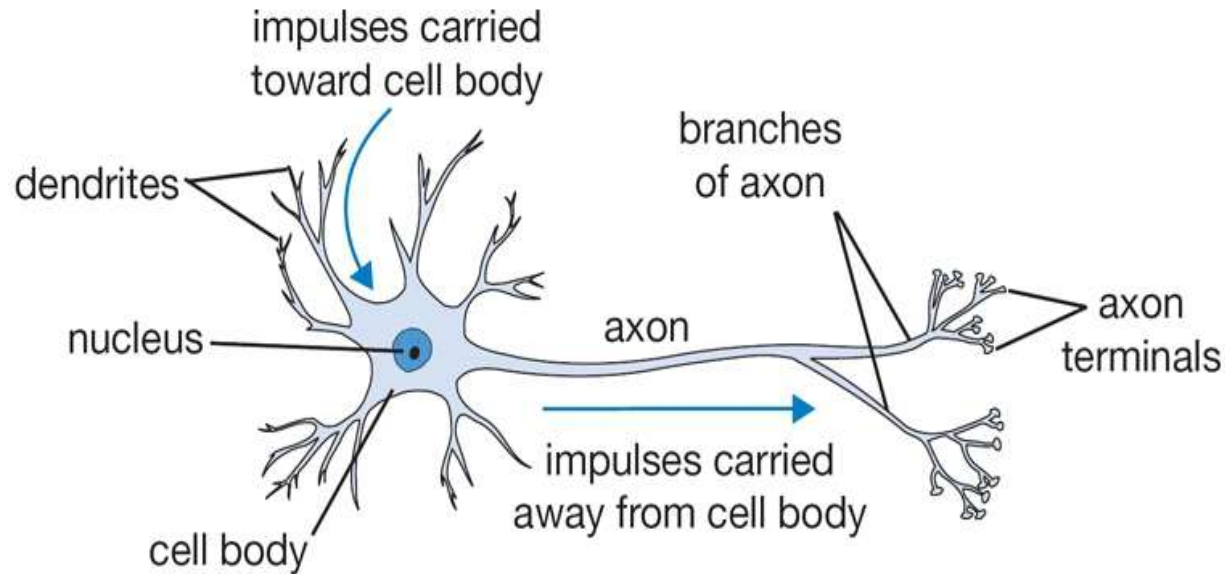
4. Artificial Neural Networks

- Inspired by the properties of biological neural networks, **Artificial Neural Networks are statistical learning algorithms** and are used for a variety of tasks, from relatively simple classification tasks to computer vision and speech recognition.
- ANNs are implemented as a **system of interconnected processing elements, called nodes, which are functionally analogous to biological neurons.** The connections between different nodes have numerical values, called weights, and by altering these values in a systematic way, the network is eventually able to approximate the desired function.

ImageClassification



4. Artificial Neural Networks



ImageClassification

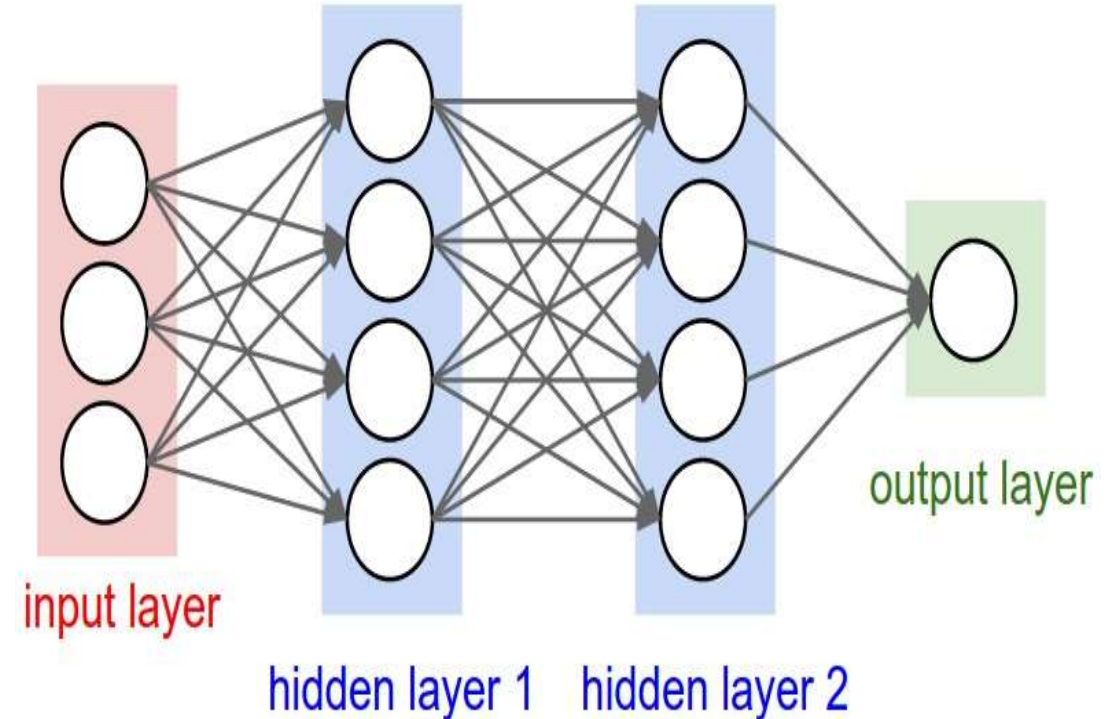


4. Artificial Neural Networks

➤ The **hidden layers** can be thought of as **individual feature detectors**, recognizing more and more complex patterns in the data as it is propagated throughout the network.

For example:

if the network is given a task to recognize a face, the first hidden layer might act as a line detector, the second hidden takes these lines as input and puts them together to form a nose, the third hidden layer takes the nose and matches it with an eye and so on, until finally the whole face is constructed. This hierarchy enables the network to eventually recognize very complex objects.



ImageClassification



5. Convolutional Neural Networks

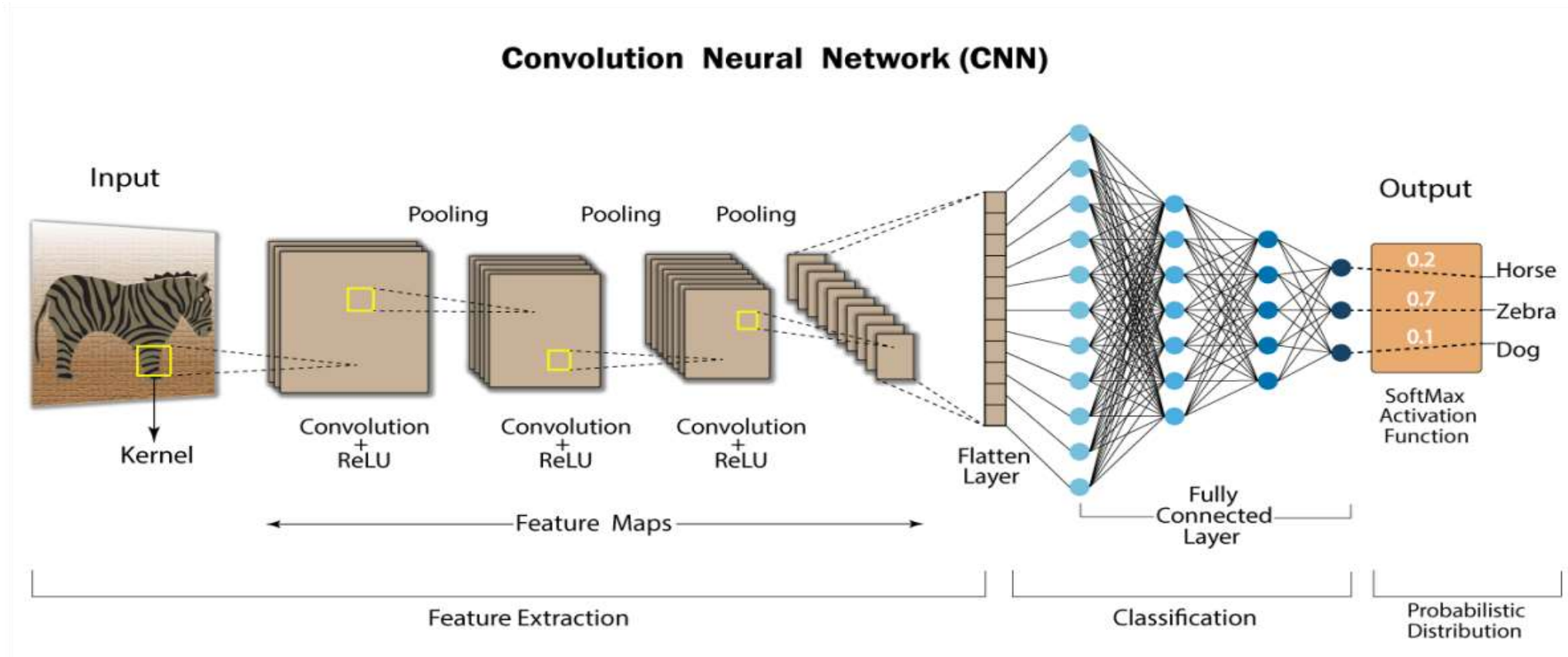
- Convolutional neural networks (**CNN**) is a special architecture of artificial neural networks. **CNNs uses some of its features of visual cortex and have therefore achieved state of the art results in computer vision tasks.**
- Convolutional neural networks are **comprised of two** very simple **elements**, namely **convolutional layers and pooling layers.**
- The challenging part of using convolutional neural networks in practice is how to **design model architectures that best use these simple elements.**



ImageClassification

5. Convolutional Neural Networks

Example:





ImageClassification

5. Convolutional Neural Networks

The **3 important elements** to understand from the CNN architecture.

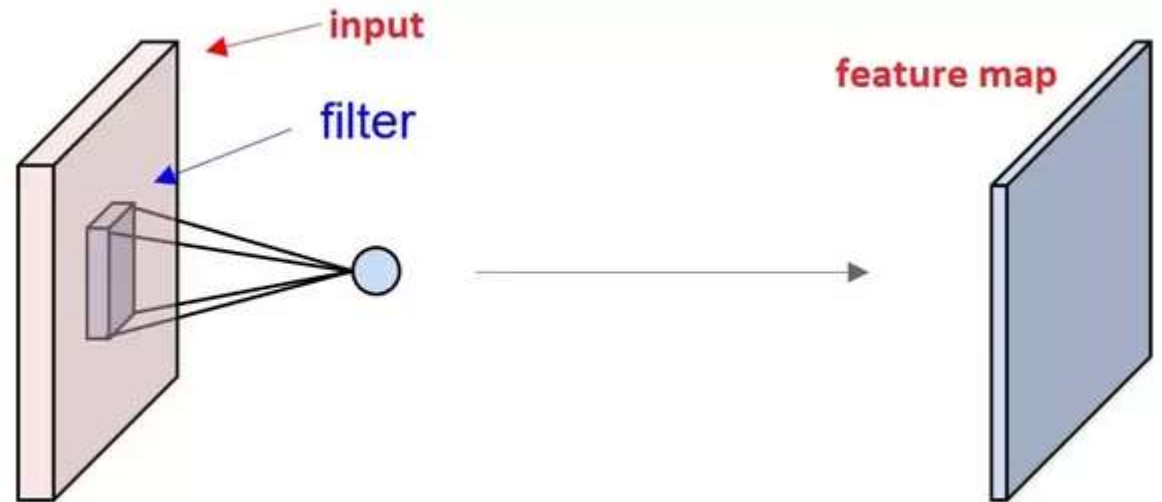
1. Convolutional Layers (Conv)

2. Max Pooling (MaxPool)

3. Fully Connected layers (Fc)

1. Convolutional Layers (Conv)

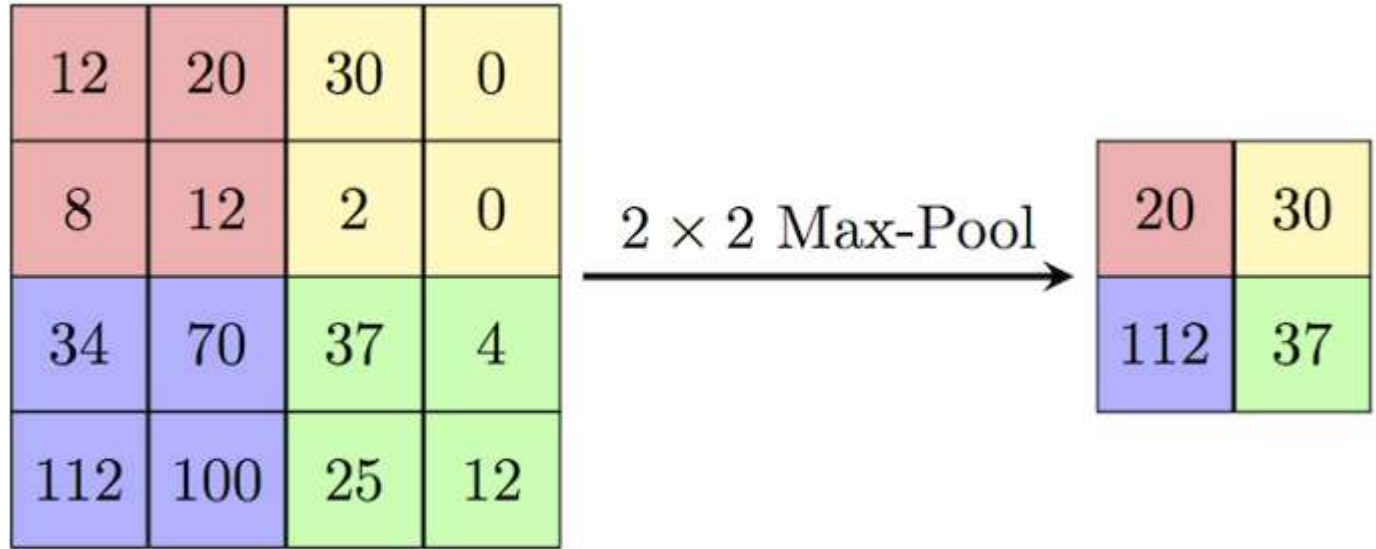
- This applies a 2D convolution over an input signal composed of several input planes.
- In other words, we turn input signals of several channels into [feature maps](#) or activation maps.
- [convolution](#) refers to the mathematical combination of two functions, thus producing a third function.
- CNNs perform convolutions on the image data using a filter or [kernel](#), then produce a feature map



ImageClassification



2. Max Pooling (MaxPool)



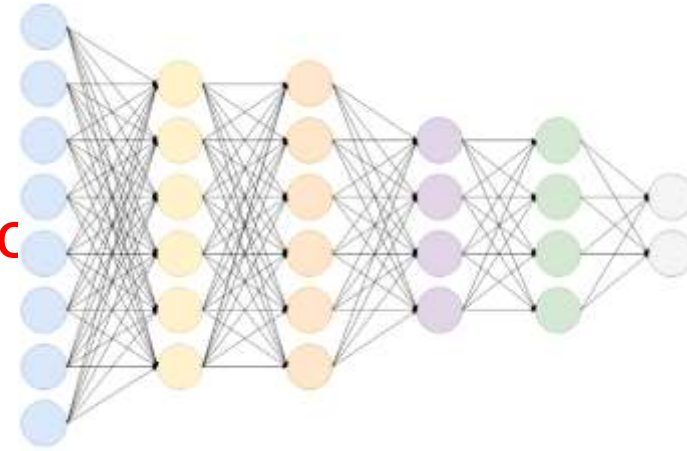
- The primary purpose of max pooling is to **down-sample the dimensions of our image to allow for assumptions to be made about the features in certain regions of the image.**
- Since we passed in (2,2) into MaxPool2d, this means we're turning our image into 2x2 dimensions while retaining "important" features.

ImageClassification



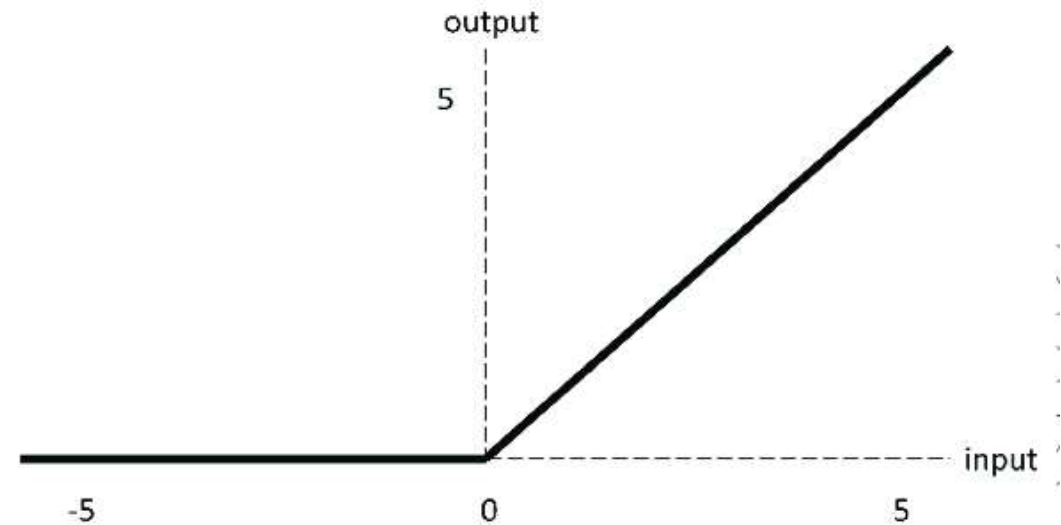
3. Fully Connected layers (Fc)

- Fully connected layers means that **every neuron from the previous layers connects to all neurons in the next**
- **Fully connected layers are the last few layers in the network**
- We use the Linear transformation in the fc layers, which are the same mathematic operation that happens in Artificial Neural Network.
- The operation follows the form of $g(Wx + b)$ where x is our input vectors, W is the weight matrix, b is the bias vector and g is an activation, which is **ReLU** in our case.
- A good way to think of fc layers is to use the concept of Principal Component Analysis PCA that selects the good features among the feature space created by the Conv and pool layers



ImageClassification

Rectified Linear Unit (ReLU)



- We use [ReLU](#) as an activation function in our Conv layers and fc layers.
- ReLU accomplishes is essentially it converts the sum of inputs into a single output.
- If the input is negative then it's zero, and if it's positive, it outputs the input.

ImageClassification



Top 4 Pre-Trained Models for Image Classification with Python Code

➤ Pre-Trained Models for Image Classification

1. VGG-16
2. ResNet50
3. Inceptionv3
4. EfficientNet



Social Network Graphs

Social Network Graphs

A social network graph is a **graph where the nodes represent people and the lines between nodes** called edges, represent social connections between them, such as friendship or working together on a project.

➤ These graphs can be either **undirected or directed**.

For Example:

1. **Facebook** can be described with an **undirected graph** since the friendship is bidirectional.

Example : Raju and Kiran being friends is the same as Kiran and Raju being friends.

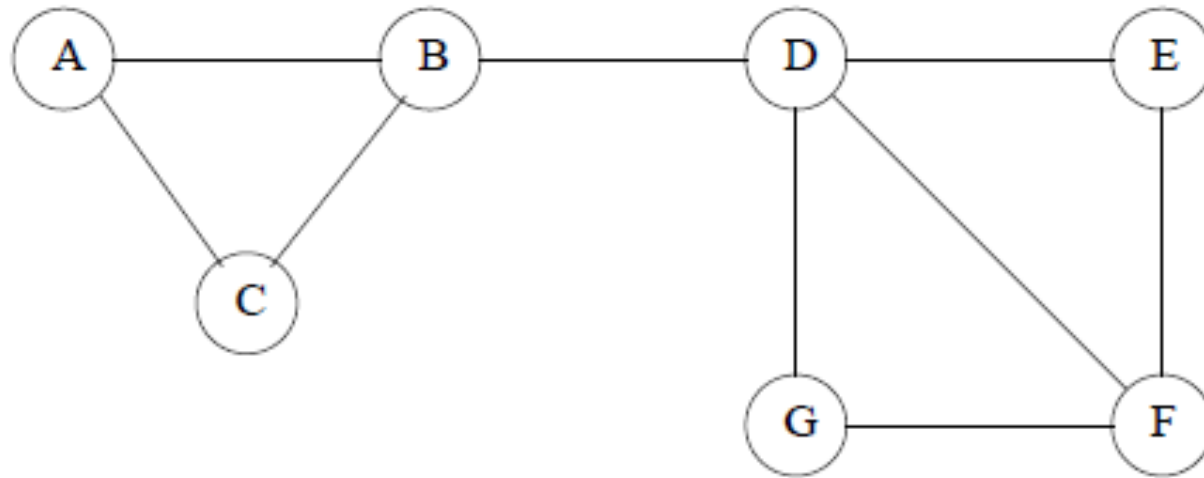
2. **Twitter** can be described with a **directed graph**:

Example: Raju can follow Kiran without Kiran following Raju.

Social Network Graphs



Example:



Social Network Graphs



Varieties of Social Networks

Some of the important variety of social networks are:

1. Telephone Networks
2. Email Networks
3. Collaboration Networks

Telephone Networks

- Here the **nodes represent phone numbers**, which are really individuals. There is an **edge between two nodes** if a call has been placed between those **phones** in some fixed period of time.
- Communities in a telephone network will form groups of people that communicate frequently: groups of friends, members of a club, or people working at the same company.

Social Network Graphs



Email Networks

- The **nodes represent email addresses**, which are again individuals.
- An **edge represents** the fact that there was **at least one email in at least one direction between the two addresses**.
- The communities seen in email networks come from the same sorts of groupings we mentioned in connection with telephone networks.

Collaboration Networks

- **Nodes represent individuals** who have **published research papers**.
- An **edge** between **two individuals** who **published one or more papers jointly**.



Social Network Graphs

Clustering of Social-Network Graphs

- clustering of the graph is a way to identify communities.

Distance Measures for Social-Network Graphs:

- To apply standard clustering techniques to a social-network graph, first step is to define a distance measure.
- When the edges of the graph have labels, these labels might be usable as a distance measure, depending on what they represented.
- when the edges are unlabeled we can't define a suitable distance.



Social Network Graphs

Applying Standard Clustering Methods

- There are **two general approaches to clustering**. Those are:
 1. Hierarchical (agglomerative) and
 2. point-assignment
- Hierarchical clustering of a social-network graph starts by **combining some two nodes that are connected by an edge**.
- a **point-assignment** approach to clustering social networks. The fact that all **edges are at the same distance will introduce a number of random factors that will lead to some nodes being assigned to the wrong cluster**.
- Since there are problems with standard clustering methods, several specialized clustering techniques have been developed to find communities in social net-works.



Social Network Graphs

- one of the simplest, based on finding the edges that are least likely to be inside a community is **Betweenness**.

Betweenness

The betweenness of an edge (a, b) to be the number of pairs of nodes x and y such that the edge (a, b) lies on the shortest path between x and y .

- The **betweenness scores for the edges of a graph** behave something **like a distance measure on the nodes of the graph**.
- Girvan-Newman Algorithm is used to calculate the betweenness.

Social Network Graphs



The Girvan-Newman Algorithm

- In order to exploit the betweenness of edges, we need to calculate the number of shortest paths going through each edge.
- Girvan-Newman (GN) Algorithm, which visits each node X once and computes the number of shortest paths from X to each of the other nodes that go through each of the edges.

Social Network Graphs



Procedure:

1. Apply breadth-first search (BFS) on the graph starting at the node X.
2. The second step of the GN algorithm is to label each node by the number of shortest paths that reach it from the root.
3. The third step is to calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root X to Y that go through edge.
4. After performing the credit calculation with each node as the root, we sum the credits for each edge.
5. To complete the betweenness calculation, we have to repeat this calculation for every node as the root and sum the contributions. Finally, we must divide by 2 to get the true betweenness, since every shortest path will be discovered twice, once for each of its endpoints.



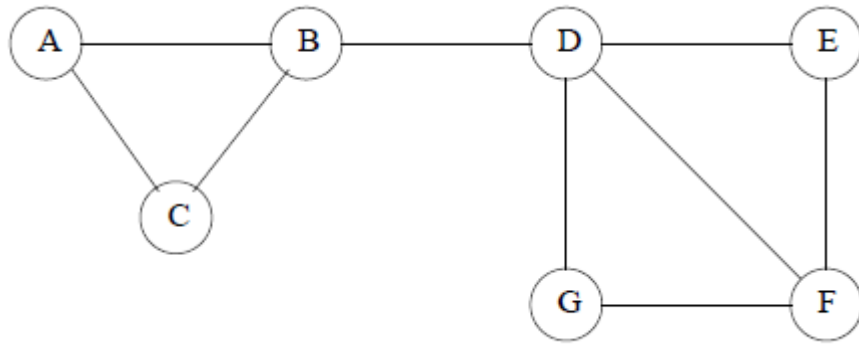
Social Network Graphs

Procedure with Example:

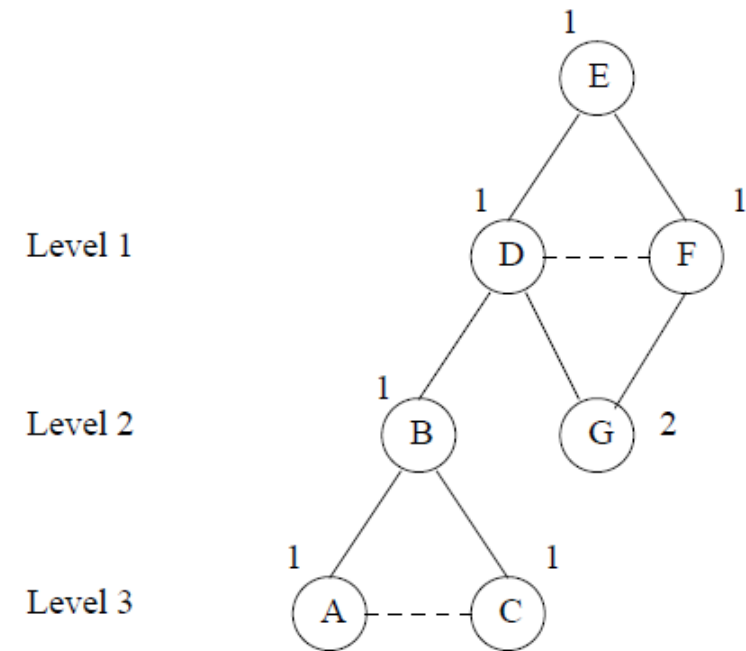
➤ Apply breadth-first search (BFS) on the graph starting at the node X.

Example:

Consider the graph



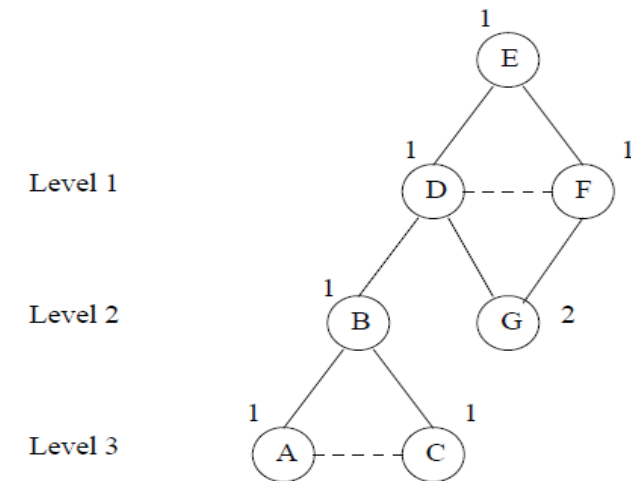
➤ breadth-first presentation for the graph starting at node E.



Social Network Graphs



- the level of each node in the BFS presentation is the length of the shortest path from X to that node. the edges that go between nodes at the same level can never be part of a shortest path from X.
- Edges between levels are called DAG edges (“DAG” stands for directed, acyclic graph). Each DAG edge will be part of at least one shortest path from root X.
- Solid edges are DAG edges and dashed edges connect nodes at the same level.



Social Network Graphs



➤ The second step of the GN algorithm is to label each node by the number of shortest paths that reach it from the root. Start by labeling the root 1. Then, from the top down, label each node Y by the sum of the labels of its parents.

Example:

➤ First, label the root E with 1. At level 1 are the nodes D and F . Each has only E as a parent, so they too are labeled 1. Nodes B and G are at level 2. B has only D as a parent, so B 's label is the same as the label of D , which is 1. However, G has parents D and F , so its label is the sum of their labels, or 2. Finally, at level 3, A and C each have only parent B , so their labels are the label of B , which is 1.

Social Network Graphs



➤ The third and final step is to calculate for each edge e the sum over all nodes Y of the fraction of shortest paths from the root X to Y that go through e .

This calculation involves computing this sum for both nodes and edges, from the bottom.

➤ The rules for the calculation are as follows:

1. Each leaf in the DAG (a leaf is a node with no DAG edges to nodes at levels below) gets a credit of 1.
2. Each node that is not a leaf gets a credit equal to 1 plus the sum of the credits of the DAG edges from that node to the level below.

Social Network Graphs



3. A DAG edge e entering node Z from the level above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e . Formally, let the parents of Z be Y_1, Y_2, \dots, Y_k . Let p_i be the number of shortest paths from the root to Y_i ; this number was computed in Step 2 and is illustrated by the labels in Figure. Then the credit for the edge (Y_i, Z) is the credit of Z times p_i divided

Social Network Graphs

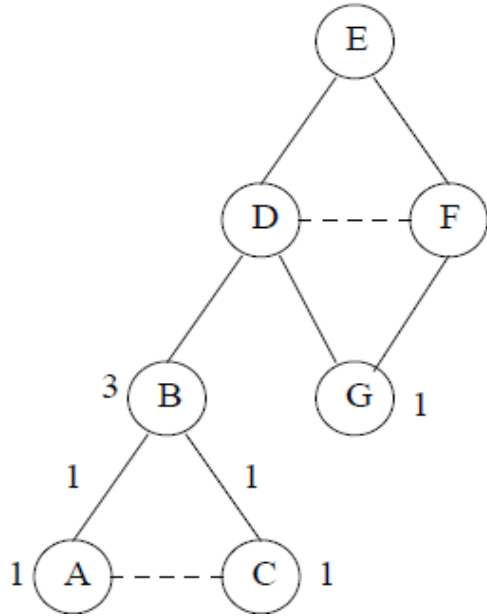


➤ After performing the credit calculation with each node as the root, we sum the credits for each edge. Then, since each shortest path will have been discovered twice – once when each of its endpoints is the root – we must divide the credit for each edge by 2

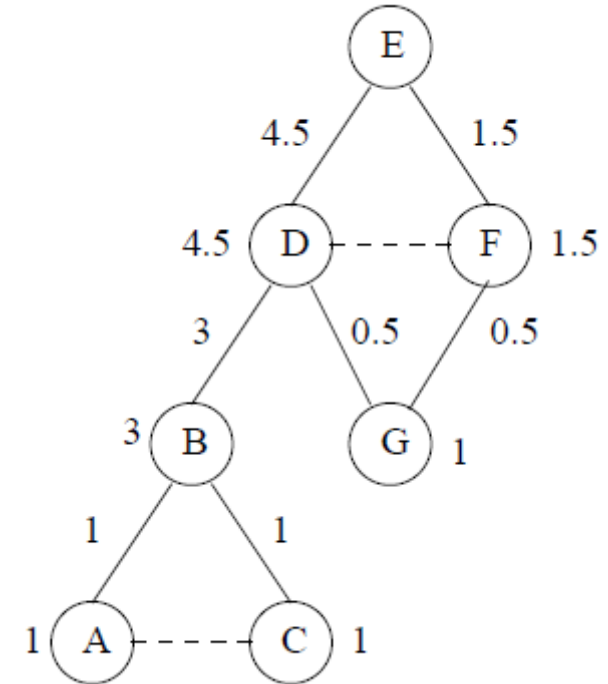
Example:

credit calculation for the BFS presentation. We shall start from level 3 and proceed upwards. First, A and C, being leaves, get credit 1. Each of these nodes have only one parent, so their credit is given to the edges (B,A) and (B,C), respectively.

Social Network Graphs



➤ The credit on each of the edges



➤ To complete the betweenness calculation, we have to repeat this calculation for every node as the root and sum the contributions. Finally, we must divide by 2 to get the true betweenness, since every shortest path will be discovered twice, once for each of its endpoints.

Social Network Graphs

Graph Traversal:

Mainly two types of Graph Traversal .Those are:

1. BFS(Breadth First Search)
2. DFS(Depth First Search)



THANK YOU