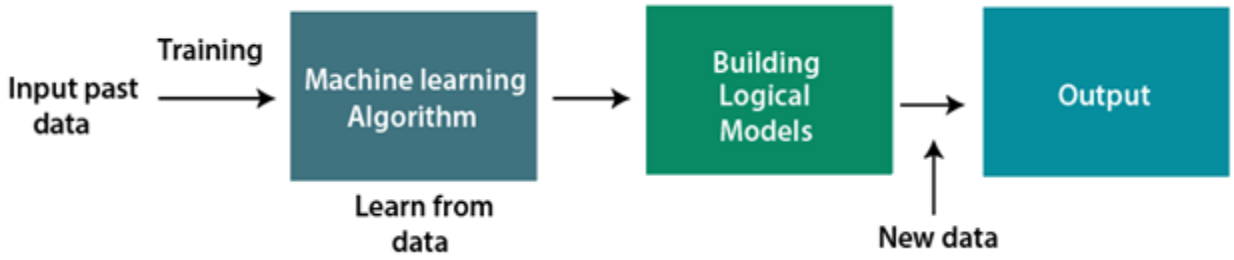
The Machine Learning Tutorial covers both the fundamentals and more complex ideas of machine learning. Students and professionals in the workforce can benefit from our machine learning tutorial.

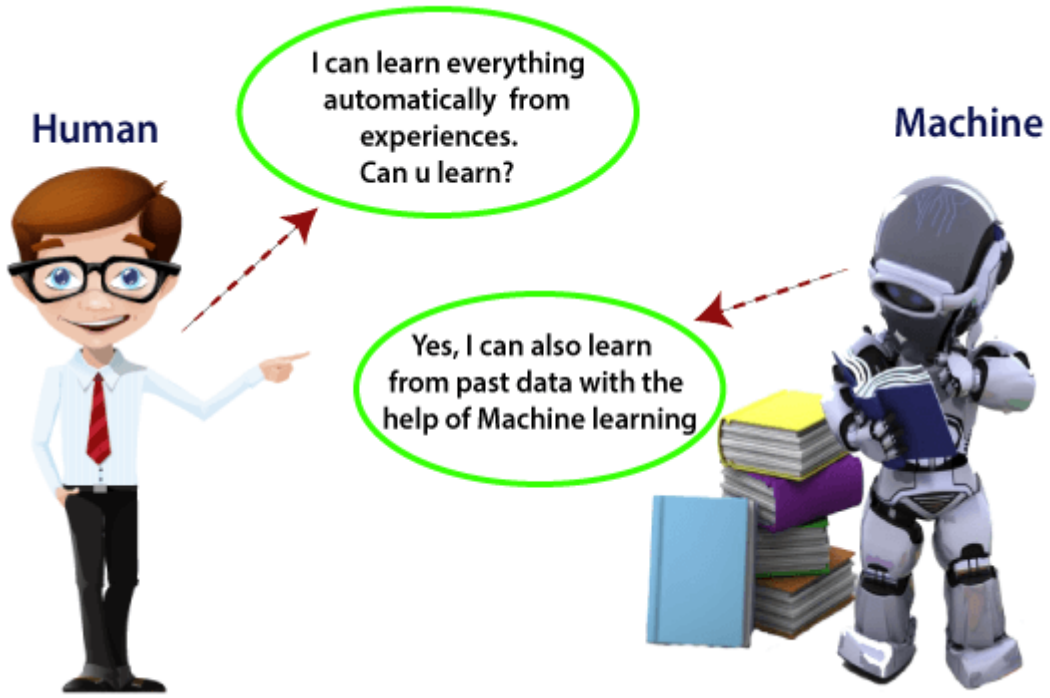
A rapidly developing field of technology, machine learning allows computers to automatically learn from previous data. For building mathematical models and making predictions based on historical data or information, machine learning employs a variety of algorithms. It is currently being used for a variety of tasks, including speech recognition, email filtering, auto-tagging on Facebook, a recommender system, and image recognition.

You will learn about the many different methods of machine learning, including reinforcement learning, supervised learning, and unsupervised learning, in this machine learning tutorial. Regression and classification models, clustering techniques, hidden Markov models, and various sequential models will all be covered.

What is Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.





Need for Machine Learning

The demand for machine learning is steadily rising. Because it is able to perform tasks that are too complex for a person to directly implement, machine learning is required. Humans are constrained by our inability to manually access vast amounts of data; as a result, we require computer systems, which is where machine learning comes in to simplify our lives.

By providing them with a large amount of data and allowing them to automatically explore the data, build models, and predict the required output, we can train machine learning algorithms. The cost function can be used to determine the amount of data and the machine learning algorithm's performance. We can save both time and money by using machine learning.

The significance of AI can be handily perceived by its utilization's cases, Presently, AI is utilized in self-driving vehicles, digital misrepresentation identification, face acknowledgment, and companion idea by Facebook, and so on. Different top organizations, for example, Netflix and Amazon have constructed AI models that are utilizing an immense measure of information to examine the client interest and suggest item likewise.

Following are some key points which show the importance of Machine Learning:

- Rapid increment in the production of data
- Solving complex problems, which are difficult for a human
- Decision making in various sector including finance

- Finding hidden patterns and extracting useful information from data.

Classification of Machine Learning

At a broad level, machine learning can be classified into three types:

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**

1) Supervised Learning

In supervised learning, sample labeled data are provided to the machine learning system for training, and the system then predicts the output based on the training data.

The system uses labeled data to build a model that understands the datasets and learns about each one. After the training and processing are done, we test the model with sample data to see if it can accurately predict the output.

The mapping of the input data to the output data is the objective of supervised learning. The managed learning depends on oversight, and it is equivalent to when an understudy learns things in the management of the educator. Spam filtering is an example of supervised learning.

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

The system uses labeled data to build a model that understands the datasets and learns about each one. After the training and processing are done, we test the model with sample data to see if it can accurately predict the output.

The mapping of the input data to the output data is the objective of supervised learning. The managed learning depends on oversight, and it is equivalent to when an understudy learns things in the management of the educator. Spam filtering is an example of supervised learning.

Supervised learning can be grouped further in two categories of algorithms:

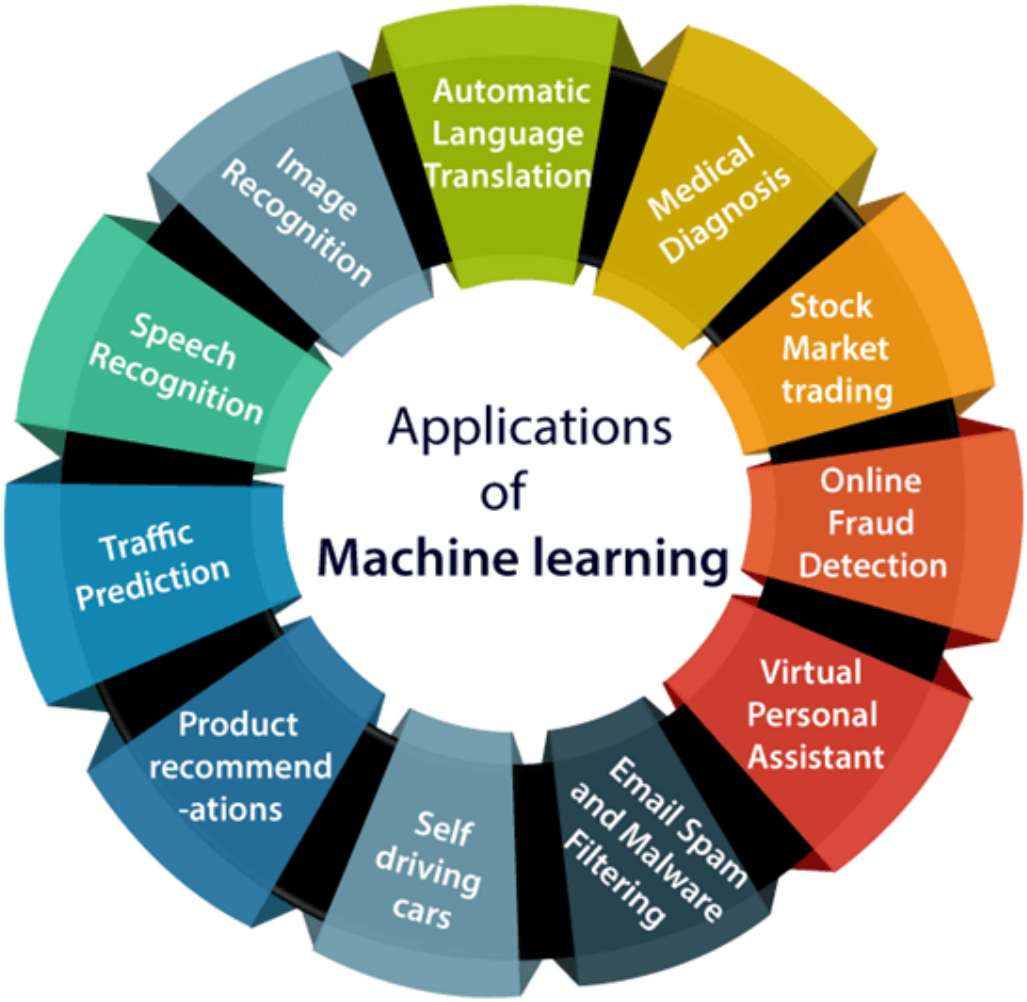
- **Classification**
- **Regression**

3) Reinforcement Learning

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

Applications of Machine learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

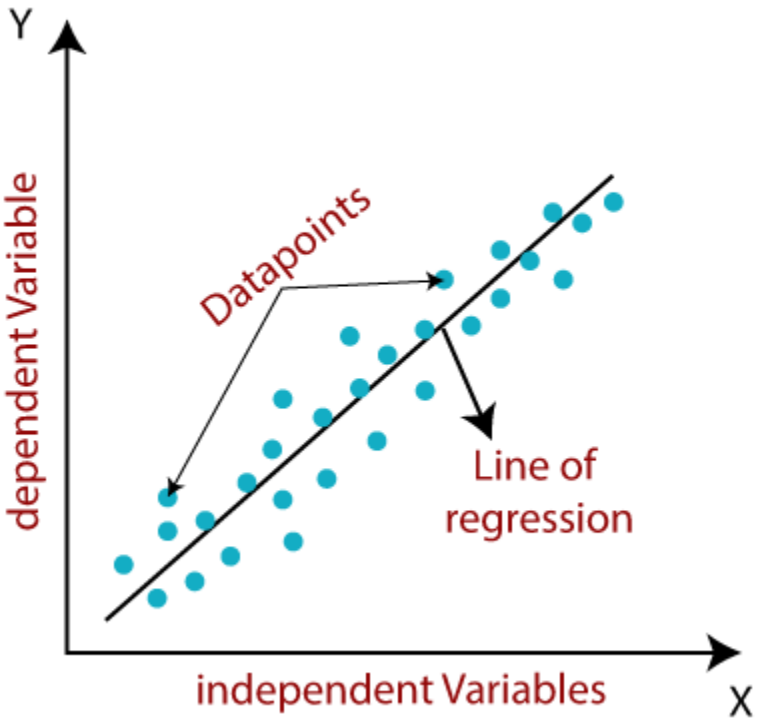


Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



$$y= a_0+a_1x+ \varepsilon$$

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

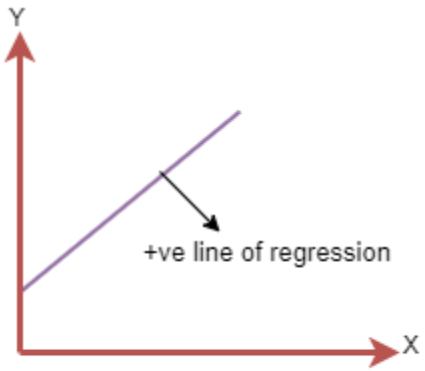
- **SimpleLinearRegression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- **MultipleLinearregression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

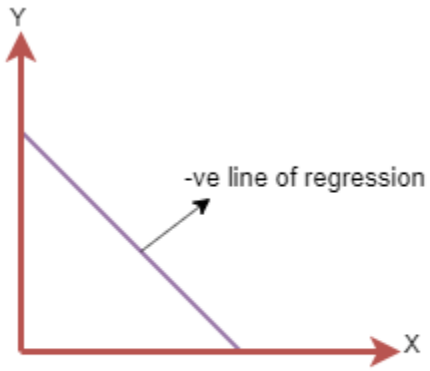
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **PositiveLinearRelationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

- **NegativeLinearRelationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.

- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:** Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:** Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.
- **Homoscedasticity Assumption:** Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.

- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Simple Linear Regression in Machine Learning

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the ***dependent variable must be a continuous/real value***. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

Simple Linear Regression Model:

The Simple Linear Regression model can be represented using the below equation:

$$y= a_0+a_1x+ \varepsilon$$

Where,

a0= It is the intercept of the Regression line (can be obtained putting x=0)
a1= It is the slope of the regression line, which tells whether the line is increasing or decreasing.
ε = The error term. (For a good model it will be negligible)

Implementation of Simple Linear Regression Algorithm using Python

Problem Statement example for Simple Linear Regression:

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

- **We want to find out if there is any correlation between these two variables**
- **We will find the best fit line for the dataset.**
- **How the dependent variable is changing by changing the independent variable.**

In this section, we will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

- First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

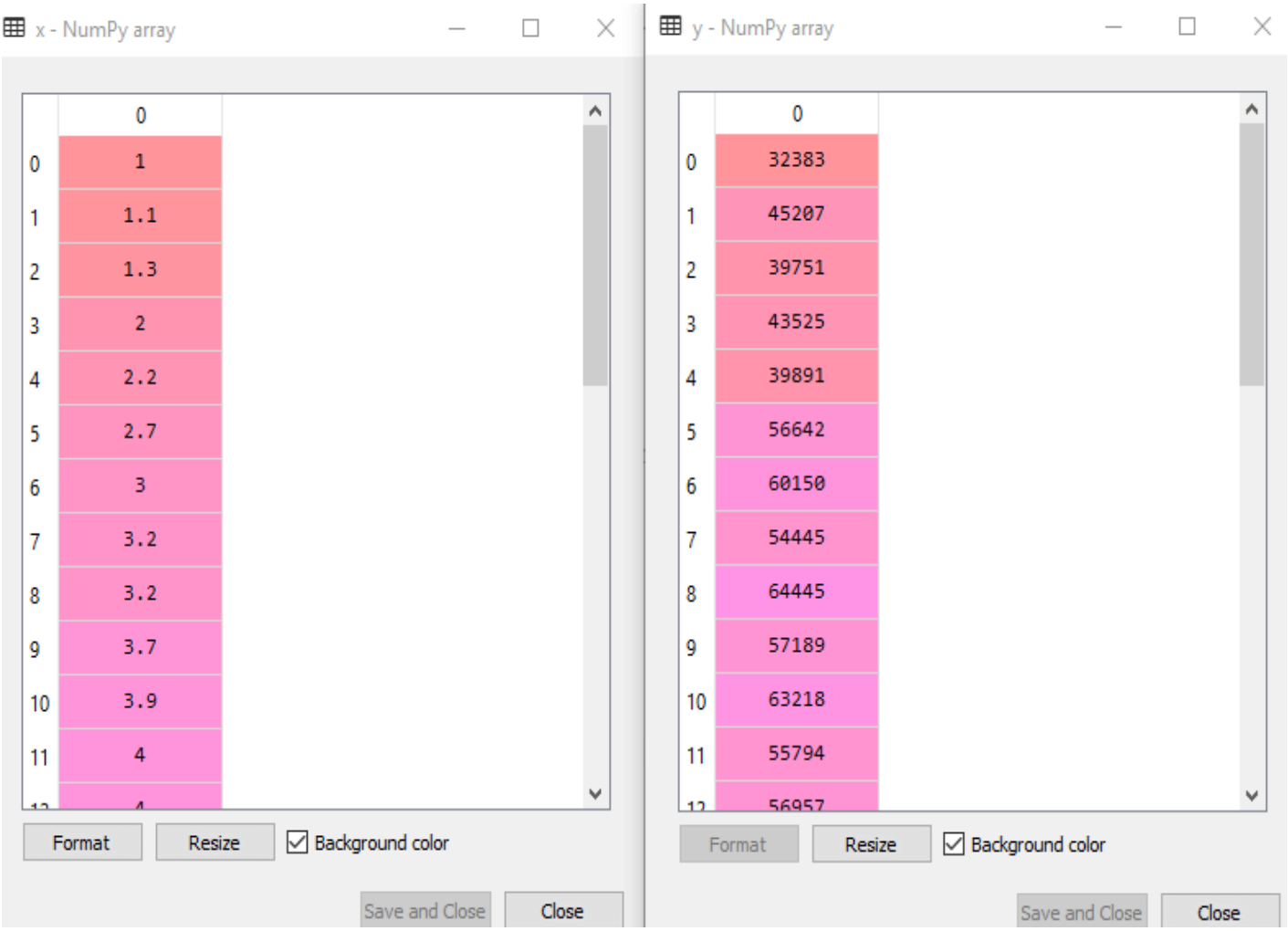
4. Next, we will load the dataset into our code:

- After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

```
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values
```

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as:



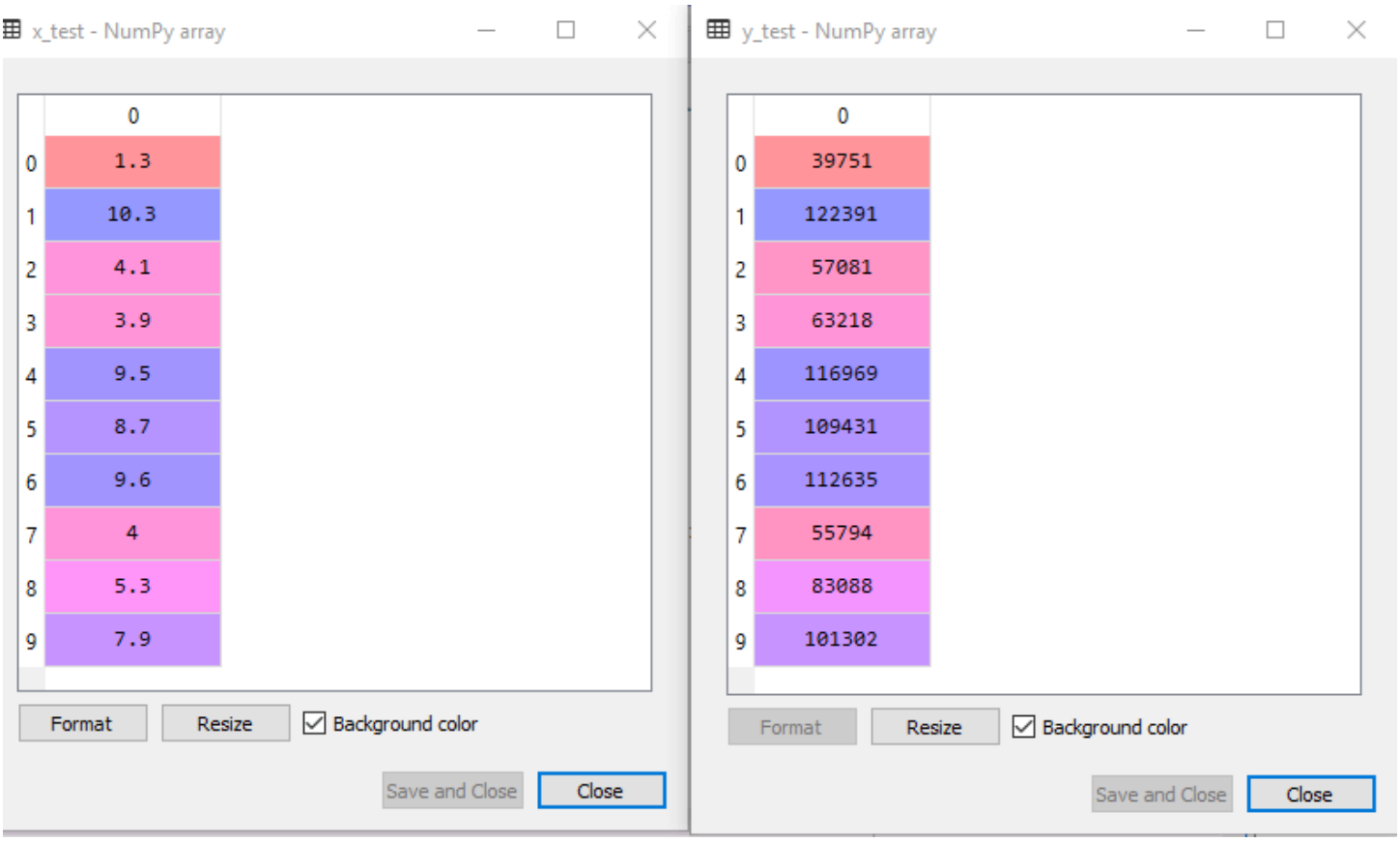
In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

- Next, we will split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

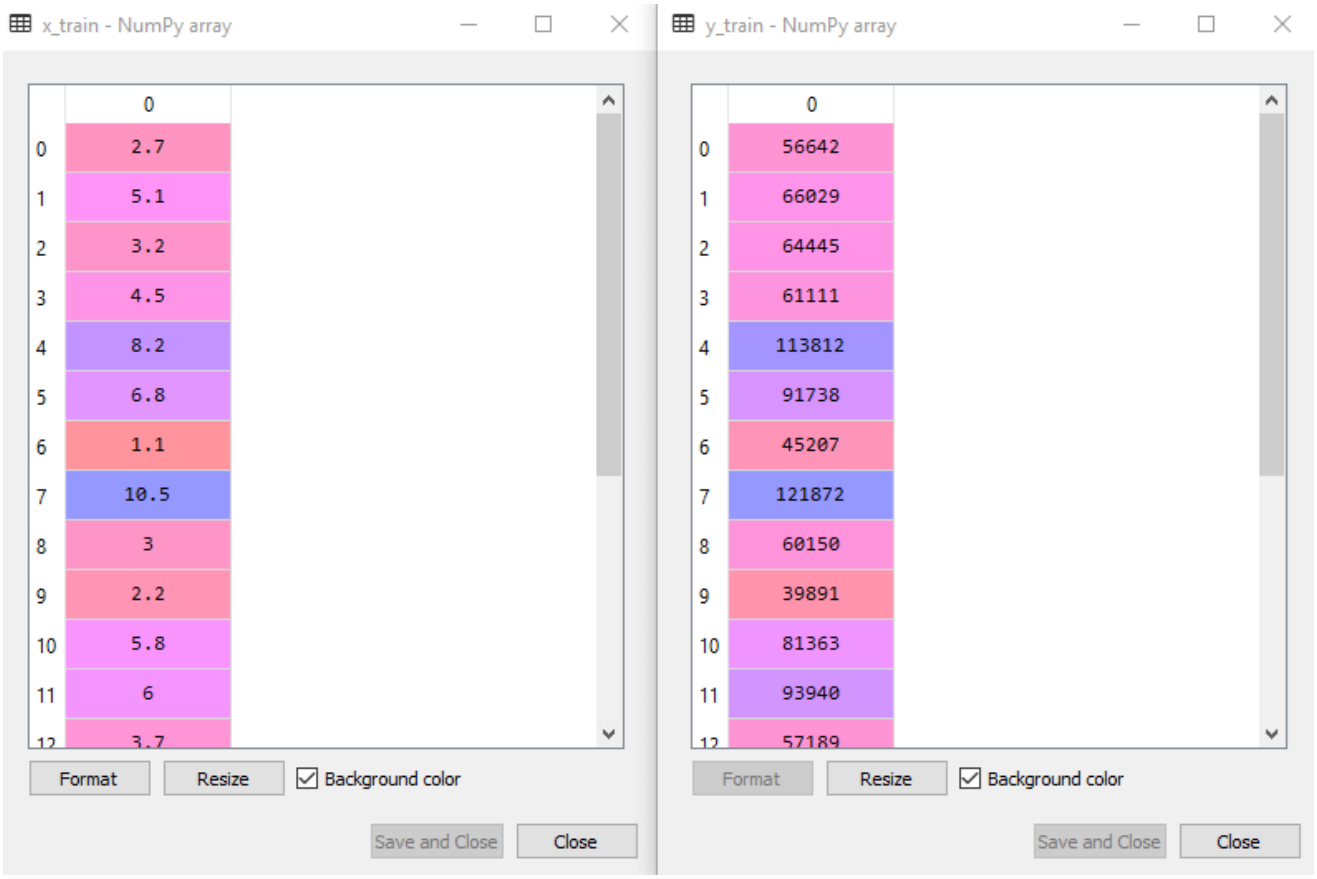
```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_state=0)
```

By executing the above code, we will get x-test, x-train and y-test, y-train dataset. Consider the below images:

Test-dataset:



Training Dataset:



- For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

Step-2: Fitting the Simple Linear Regression to the Training Set:

Now the second step is to fit our model to the training dataset. To do so, we will import the **LinearRegression** class of the **linear_model** library from the **scikit learn**. After importing the class, we are going to create an object of the class named as a **regressor**. The code for this is given below:

```
1. #Fitting the Simple Linear Regression model to the training dataset
2. from sklearn.linear_model import LinearRegression
3. regressor= LinearRegression()
4. regressor.fit(x_train, y_train)
```

In the above code, we have used a **fit()** method to fit our Simple Linear Regression object to the training set. In the fit() function, we have passed the x_train and y_train, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

Output:

```
Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Step: 3. Prediction of test set result:

dependent (salary) and an independent variable (Experience). So, now, our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector **y_pred**, and **x_pred**, which will contain predictions of test dataset, and prediction of training set respectively.

```
1. #Prediction of Test and Training set result
2. y_pred= regressor.predict(x_test)
3. x_pred= regressor.predict(x_train)
```

On executing the above lines of code, two variables named y_pred and x_pred will generate in the variable explorer options that contain salary predictions for the training set and test set.

Output:

You can check the variable by clicking on the variable explorer option in the IDE, and also compare the result by comparing values from y_pred and y_test. By comparing these values, we can check how good our model is performing.

Step: 4. visualizing the Training set results:

Now in this step, we will visualize the training set result. To do so, we will use the scatter() function of the pyplot library, which we have already imported in the pre-processing step. The **scatter () function** will create a scatter plot of observations.

In the x-axis, we will plot the Years of Experience of employees and on the y-axis, salary of employees. In the function, we will pass the real values of training set, which means a year of experience x_train, training set of Salaries y_train, and color of the observations. Here we are taking a green color for the observation, but it can be any color as per the choice.

Now, we need to plot the regression line, so for this, we will use the **plot() function** of the pyplot library. In this function, we will pass the years of experience for training set, predicted salary for training set x_pred, and color of the line.

Next, we will give the title for the plot. So here, we will use the **title()** function of the **pyplot** library and pass the name ("Salary vs Experience (Training Dataset)".

After that, we will assign labels for x-axis and y-axis using **xlabel() and ylabel() function**.

Finally, we will represent all above things in a graph using show(). The code is given below:

```
1. mtp.scatter(x_train, y_train, color="green")
2. mtp.plot(x_train, x_pred, color="red")
3. mtp.title("Salary vs Experience (Training Dataset)")
4. mtp.xlabel("Years of Experience")
5. mtp.ylabel("Salary(In Rupees)")
6. mtp.show()
```

Output:

By executing the above lines of code, we will get the below graph plot as an output.

In the above plot, we can see the real values observations in green dots and predicted values are covered by the red regression line. The regression line shows a correlation between the dependent and independent variable.

The good fit of the line can be observed by calculating the difference between actual values and predicted values. But as we can see in the above **plot, most of the observations are close to the regression line, hence our model is good for the training set**.

Step: 5. visualizing the Test set results:

In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use x_test, and y_test instead of x_train and y_train.

Here we are also changing the color of observations and regression line to differentiate between the two plots, but it is optional.

```
#visualizing the Test set results
mtp.scatter(x_test, y_test, color="blue")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Test Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

Output:

By executing the above line of code, we will get the output as:

In the above plot, there are observations given by the blue color, and prediction is given by the red regression line. As we can see, most of the observations are close to the regression line, hence we can say our Simple Linear Regression is a good model and able to make good predictions

Multiple Linear Regression

In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.

Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Assumptions for Multiple Linear Regression:

- A **linear relationship** should exist between the Target and predictor variables.
- The regression residuals must be **normally distributed**.
- MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

Implementation of Multiple Linear Regression model using Python:

To implement MLR using Python, we have below problem:

Problem Description:

We have a dataset of **50 start-up companies**. This dataset contains five main information: **R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year**. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

1. **Data Pre-processing Steps**
2. **Fitting the MLR model to the training set**
3. **Predicting the result of the test set**

Step-1: Data Pre-processing Step:

The very first step is data pre-processing, which we have already discussed in this tutorial. This process contains the below steps:

- **Importing libraries:** Firstly we will import the library which will help in building the model. Below is the code for it:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

ML Polynomial Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

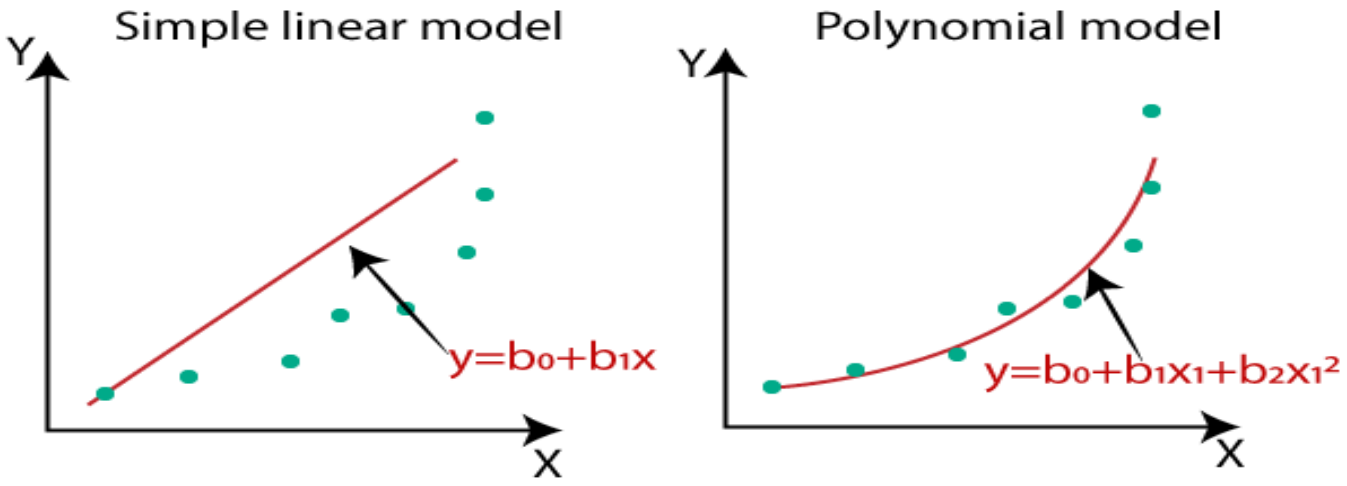
$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+..... b_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence, "*In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model.*"**

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression*

Equation of the Polynomial Regression Model:

Simple Linear Regression equation: $y = b_0+b_1x$ .....(a)

Multiple Linear Regression equation: $y= b_0+b_1x+ b_2x_2+ b_3x_3+....+ b_nx_n$ .....(b)

Polynomial Regression equation: $y= b_0+b_1x + b_2x^2+ b_3x^3+....+ b_nx^n$ .....(c)

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

Classification Algorithm in Machine Learning

As we know, the Supervised Machine Learning algorithm can be broadly classified into Regression and Classification Algorithms. In Regression algorithms, we have predicted the output for continuous values, but to predict the categorical values, we need Classification algorithms.

What is the Classification Algorithm?

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog,** etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.

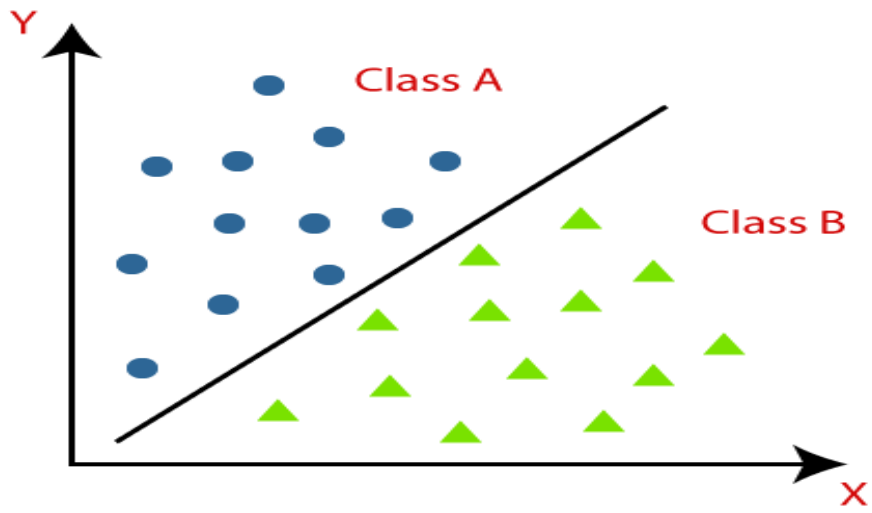
In classification algorithm, a discrete output function(y) is mapped to input variable(x).

1. y=f(x), where y = categorical output

The best example of an ML classification algorithm is **Email Spam Detector**.

The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

Classification algorithms can be better understood using the below diagram. In the below diagram, there are two classes, class A and Class B. These classes have features that are similar to each other and dissimilar to other classes.



The algorithm which implements the classification on a dataset is known as a classifier. There are two types of Classifications:

- **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.
  Examples: YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.
- **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.
  Example: Classifications of types of crops, Classification of types of music.

Learners in Classification Problems:

In the classification problems, there are two types of learners:

1. **Lazy Learners:** Lazy Learner firstly stores the training dataset and wait until it receives the test dataset. In Lazy learner case, classification is done on the basis of the most related data stored in the training dataset. It takes less time in training but more time for predictions.
   Example: K-NN algorithm, Case-based reasoning
2. **Eager Learners:**Eager Learners develop a classification model based on a training dataset before receiving a test dataset. Opposite to Lazy learners, Eager Learner takes more time in learning, and less time in prediction. **Example:** Decision Trees, Naïve Bayes, ANN.

Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the Mainly two category:

- **Linear Models**
  - Logistic Regression
  - Support Vector Machines
- **Non-linear Models**
  - K-Nearest Neighbours
  - Kernel SVM
  - Naïve Bayes
  - Decision Tree Classification
  - Random Forest Classification

Evaluating a Classification model:

Once our model is completed, it is necessary to evaluate its performance; either it is a Classification or Regression model. So for evaluating a Classification model, we have the following ways:

1. Log Loss or Cross-Entropy Loss:

- It is used for evaluating the performance of a classifier, whose output is a probability value between the 0 and 1.
- For a good binary Classification model, the value of log loss should be near to 0.
- The value of log loss increases if the predicted value deviates from the actual value.
- The lower log loss represents the higher accuracy of the model.
- For Binary classification, cross-entropy can be calculated as:

1. ?(ylog(p)+(1?y)log(1?p))

Where y= Actual output, p= predicted output.

2. Confusion Matrix:

- The confusion matrix provides us a matrix/table as output and describes the performance of the model.
- It is also known as the error matrix.

3. AUC-ROC curve:

- ROC curve stands for **Receiver Operating Characteristics Curve** and AUC stands for **Area Under the Curve**.
- It is a graph that shows the performance of the classification model at different thresholds.
- To visualize the performance of the multi-class classification model, we use the AUC-ROC Curve.
- The ROC curve is plotted with TPR and FPR, where TPR (True Positive Rate) on Y-axis and FPR(False Positive Rate) on X-axis.

Use cases of Classification Algorithms

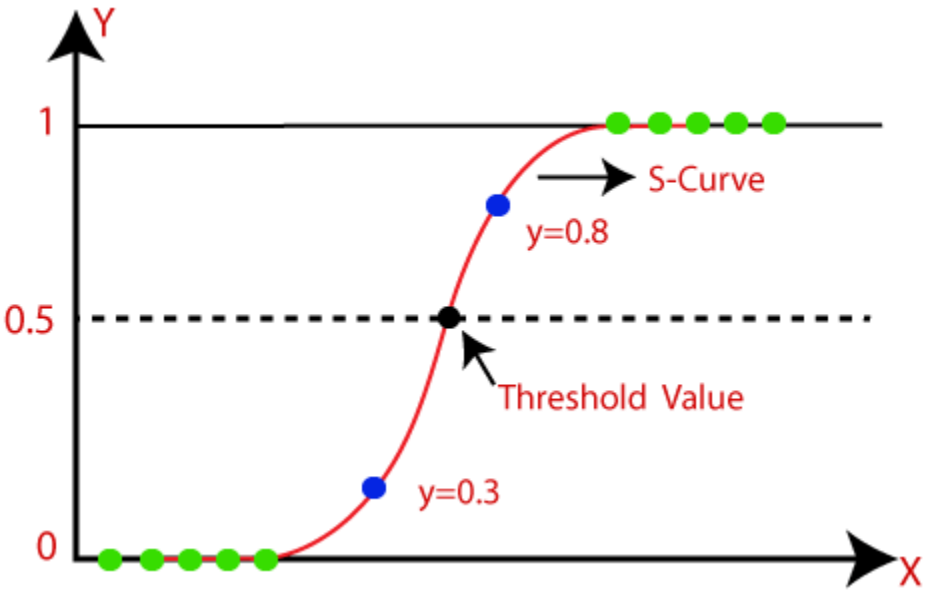
Classification algorithms can be used in different places. Below are some popular use cases of Classification Algorithms:

- Email Spam Detection
- Speech Recognition
- Identifications of Cancer tumor cells.
- Drugs Classification
- Biometric Identification, etc.

Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Note: ***Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.***

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

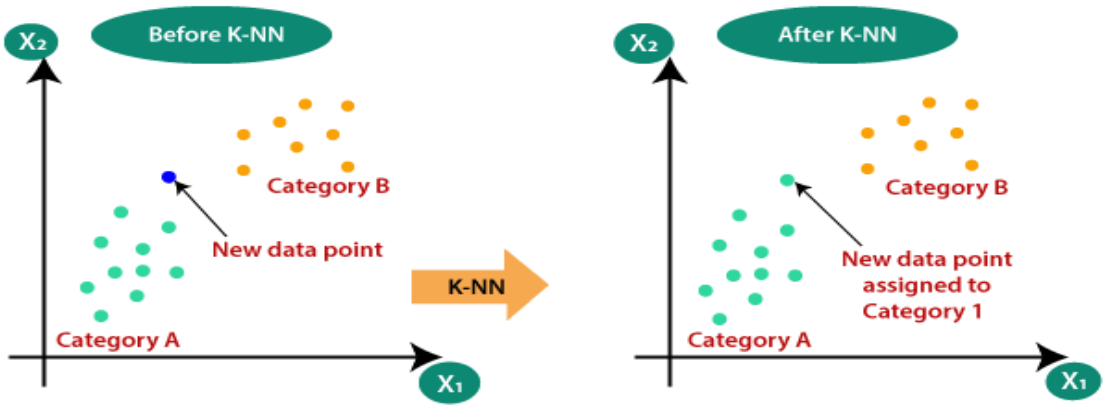
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

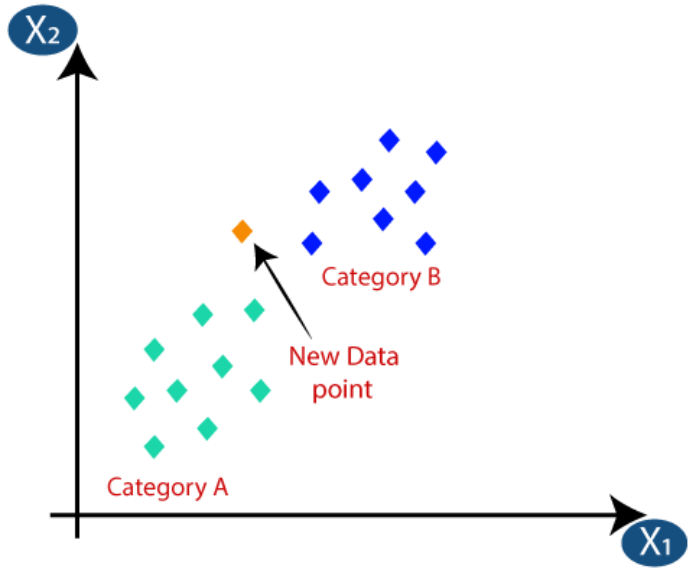
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



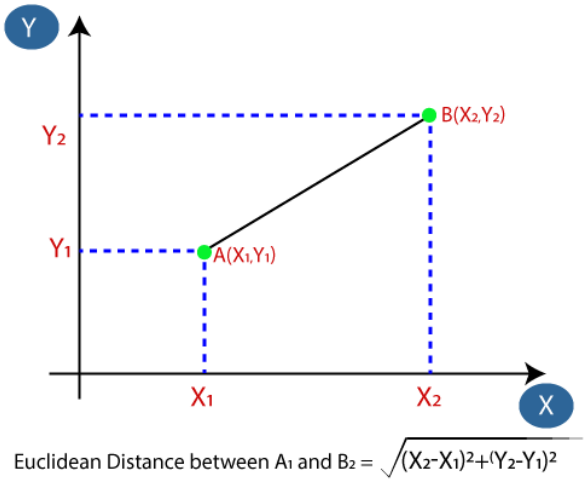
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

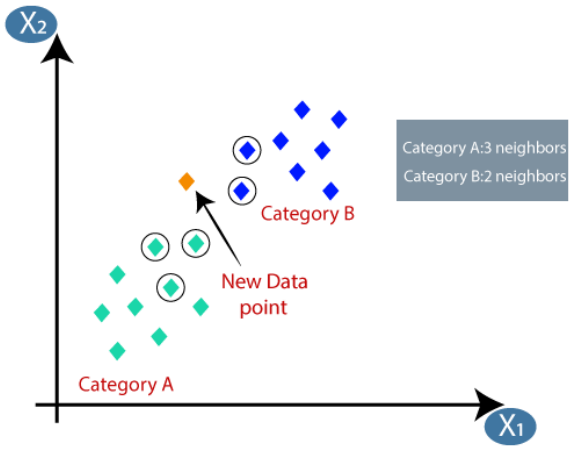
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.
- Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

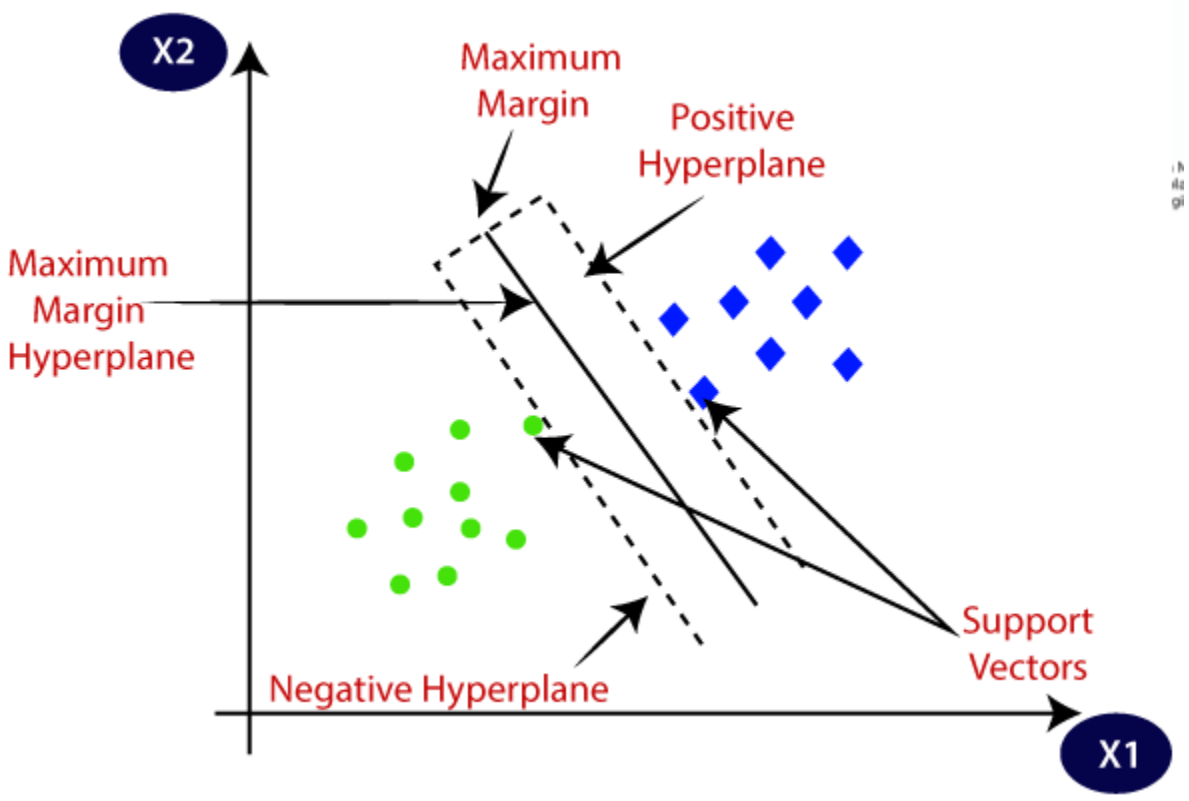
- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Support Vector Machine Algorithm

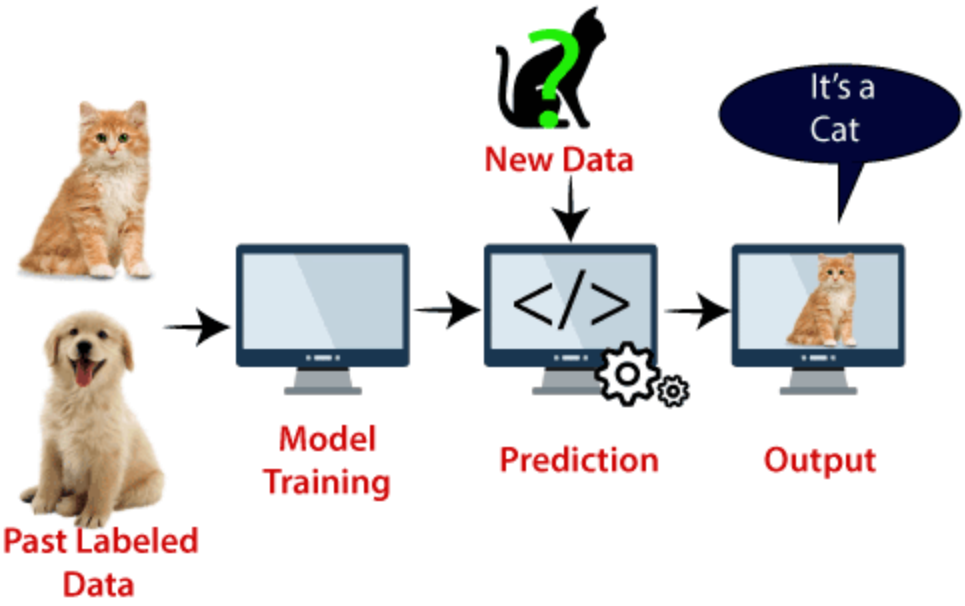
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



Types of SVM

SVM can be of two types:

- **inear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

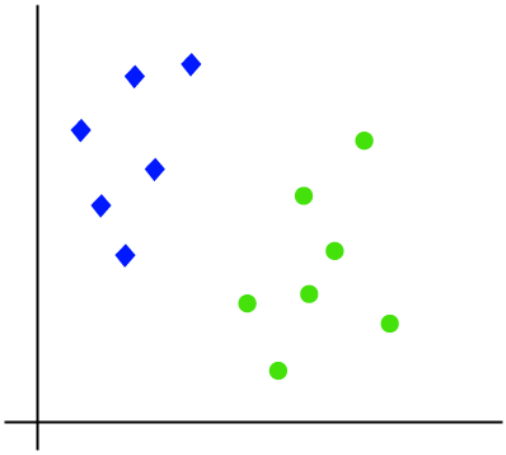
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

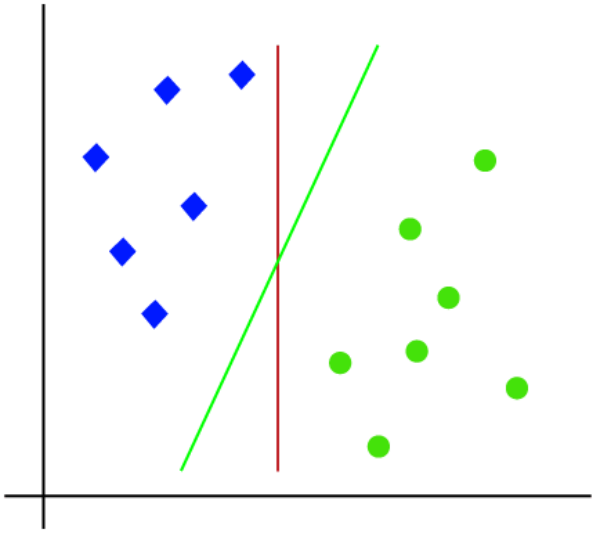
How does SVM works?

Linear SVM:

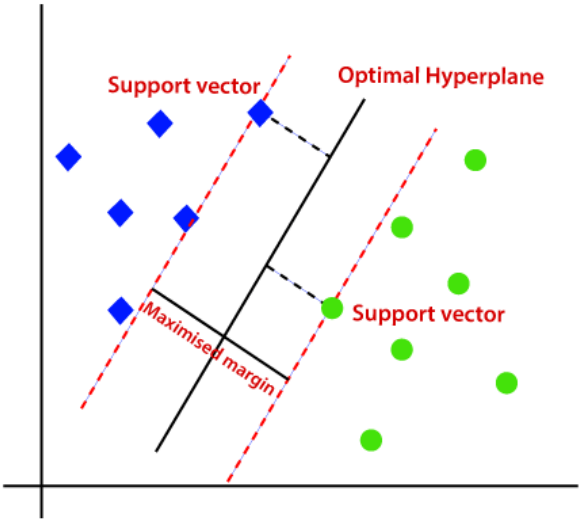
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

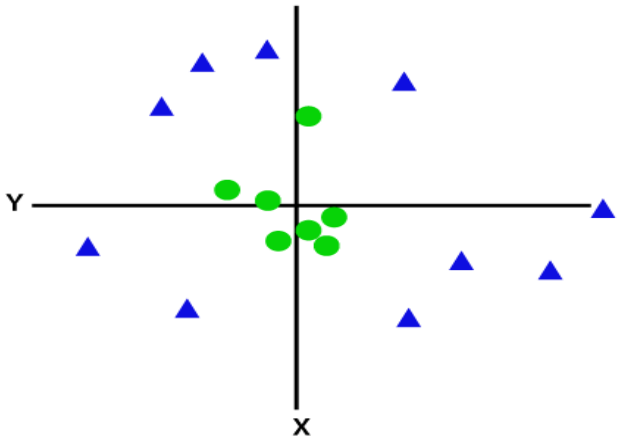


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM:

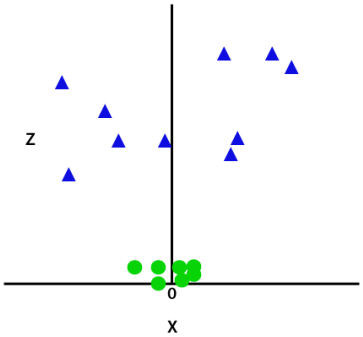
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



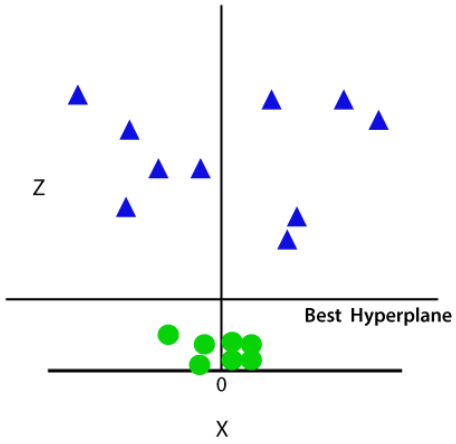
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z=x^2 +y^2$$

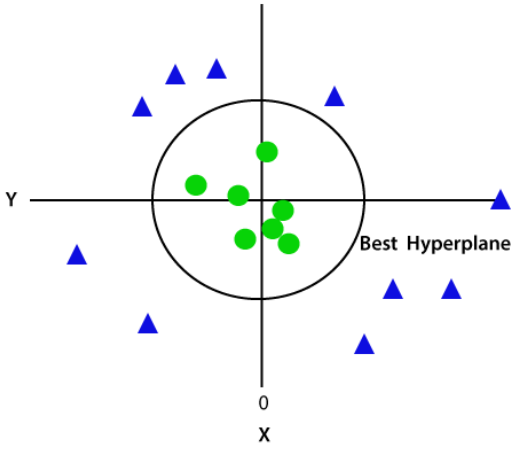
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

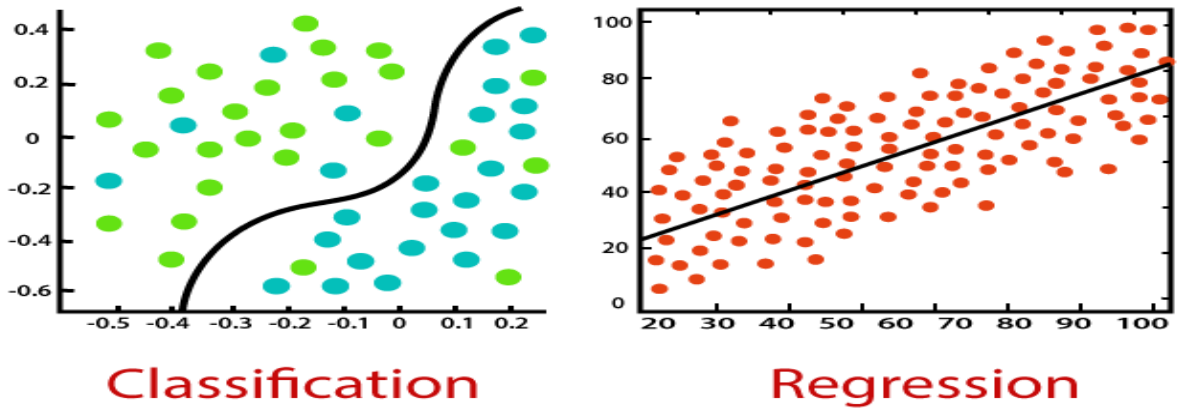
Solution: To solve this, first consider the below dataset:

Regression vs. Classification in Machine Learning

Regression and Classification algorithms are Supervised Learning algorithms. Both the algorithms are used for prediction in Machine learning and work with the labeled datasets. But the difference between both is how they are used for different machine learning problems.

The main difference between Regression and Classification algorithms that Regression algorithms are used to **predict the continuous** values such as price, salary, age, etc. and Classification algorithms are used to **predict/Classify the discrete values** such as Male or Female, True or False, Spam or Not Spam, etc.

Consider the below diagram:



Classification:

Classification is a process of finding a function which helps in dividing the dataset into classes based on different parameters. In Classification, a computer program is trained on the training dataset and based on that training, it categorizes the data into different classes.

The task of the classification algorithm is to find the mapping function to map the input(x) to the discrete output(y).

Example: The best example to understand the Classification problem is Email Spam Detection. The model is trained on the basis of millions of emails on different parameters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.

Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the following types:

- Logistic Regression
- K-Nearest Neighbours
- Support Vector Machines
- Kernel SVM
- Naïve Bayes
- Decision Tree Classification
- Random Forest Classification

Regression:

Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of **Market Trends**, prediction of House prices, etc.

The task of the Regression algorithm is to find the mapping function to map the input variable(x) to the continuous output variable(y).

Example: Suppose we want to do weather forecasting, so for this, we will use the Regression algorithm. In weather prediction, the model is trained on the past data, and once the training is completed, it can easily predict the weather for future days.

Types of Regression Algorithm:

- Simple Linear Regression
- Multiple Linear Regression

- Polynomial Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

Difference between Regression and Classification

Regression Algorithm	Classification Algorithm
In Regression, the output variable must be of continuous nature or real value.	In Classification, the output variable must be a discrete value.
The task of the regression algorithm is to map the input value (x) with the continuous output variable(y).	The task of the classification algorithm is to map the input value(x) with the discrete output variable(y).
Regression Algorithms are used with continuous data.	Classification Algorithms are used with discrete data.
In Regression, we try to find the best fit line, which can predict the output more accurately.	In Classification, we try to find the decision boundary, which can divide the dataset into different classes.
Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, etc.	Classification Algorithms can be used to solve classification problems such as Identification of spam emails, Speech Recognition, Identification of cancer cells, etc.
The regression Algorithm can be further divided into Linear and Non-linear Regression.	The Classification algorithms can be divided into Binary Classifier and Multi-class Classifier.

What is Deep Learning?

Deep Learning is a part of Machine Learning that uses artificial neural networks to learn from lots of data without needing explicit programming. These networks are inspired by the human brain and can be used for things like recognizing images, understanding speech, and processing language. There are different types of deep learning networks, like feedforward neural networks, convolutional neural networks, and recurrent neural networks. Deep Learning needs lots of labeled data and powerful computers to work well, but it can achieve very good results in many applications.

Activation Functions in Pytorch

What is an activation function and why to use them?

Activation functions are the building blocks of Pytorch. Before coming to types of activation function, let us first understand the working of neurons in the human brain. In the Artificial Neural Networks, we have an input layer which is the input by the user in some format, a hidden layer that performs the hidden calculations and identifies features and output is the result. So the whole structure is like a network with neurons connected to one another. So we have artificial neurons which are activated by these activation functions. The activation function is a function that performs calculations to provide an output that may act as input for the next neurons. An ideal activation function should handle non-linear relationships by using the linear concepts and it should be differentiable so as to reduce the errors and adjust the weights accordingly. All activation functions are present in the torch.nn library.

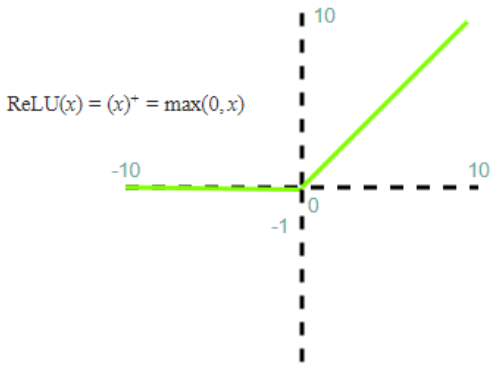
Types of Pytorch Activation Function

Let us look at the different Pytorch Activation functions:

- ReLU Activation Function
- Leaky ReLU Activation Function
- Sigmoid Activation Function
- Tanh Activation Function
- Softmax Activation Function

ReLU Activation Function:

ReLU stands for Rectified Linear Activation function. It is a non-linear function and, graphically ReLU has the following transformative behavior:

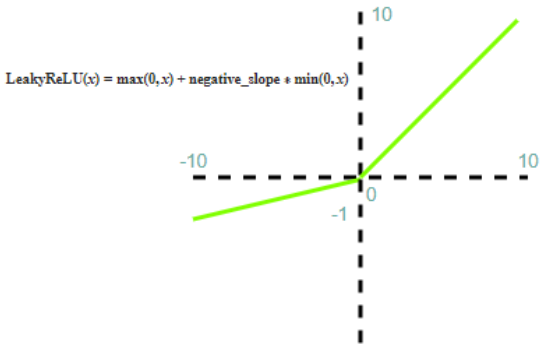


ReLU is a popular activation function since it is differentiable and nonlinear. If the inputs are negative its derivative becomes zero which causes the ‘dying’ of neurons and

learning doesn't take place. Let us illustrate the use of ReLU with the help of the Python program.

Leaky ReLU Activation Function:

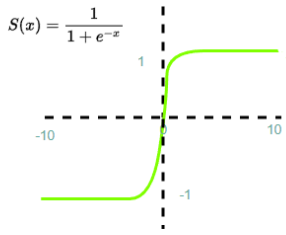
Leaky ReLU Activation Function or LReLU is another type of activation function which is similar to ReLU but solves the problem of 'dying' neurons and, graphically Leaky ReLU has the following transformative behavior:



This function is very useful as when the input is negative the differentiation of the function is not zero. Hence the learning of neurons doesn't stop. Let us illustrate the use of LReLU with the help of the Python program.

Sigmoid Activation Function:

Sigmoid Function is a non-linear and differentiable activation function. It is an S-shaped curve that does not pass through the origin. It produces an output that lies between 0 and 1. The output values are often treated as a probability. It is often used for binary classification. It is slow in computation and, graphically Sigmoid has the following transformative behavior:

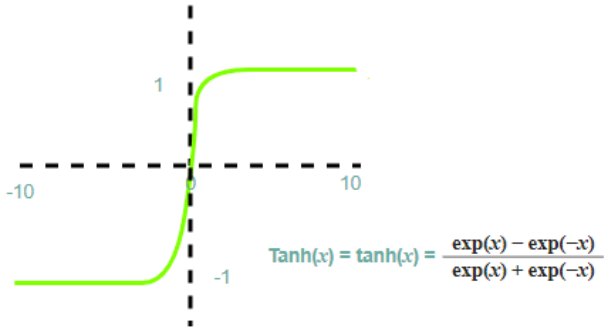


Sigmoid activation function has a problem of "Vanishing Gradient". Vanishing Gradient is a significant problem as a large number of inputs are fed to the neural network and the

number of hidden layers increases, the gradient or derivative becomes close to zero thus leading to inaccuracy in the neural network.

Tanh Activation Function:

Tanh function is a non-linear and differentiable function similar to the sigmoid function but output values range from -1 to +1. It is an S-shaped curve that passes through the origin and, graphically Tanh has the following transformative behavior:



Softmax Activation Function:

The softmax function is different from other activation functions as it is placed at the last to normalize the output. We can use other activation functions in combination with Softmax to produce the output in probabilistic form. It is used in multiclass classification and generates an output of probabilities whose sum is 1. The range of output lies between 0 and 1. Softmax has the following transformative behavior:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Artificial Neural Network

Gradient Descent Optimization in Tensorflow

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function. In other words, gradient descent is an iterative algorithm that helps to find the optimal solution to a given problem.

In this blog, we will discuss gradient descent optimization in TensorFlow, a popular deep-learning framework. TensorFlow provides several optimizers that implement

different variations of gradient descent, such as stochastic gradient descent and mini-batch gradient descent.
Before diving into the details of gradient descent in TensorFlow, let's first understand the basics of gradient descent and how it works.

What is Gradient Descent?

Gradient descent is an iterative optimization algorithm that is used to minimize a function by iteratively moving in the direction of the steepest descent as defined by the negative of the gradient. In other words, the gradient descent algorithm takes small steps in the direction opposite to the gradient of the function at the current point, with the goal of reaching a global minimum.
The gradient of a function tells us the direction in which the function is increasing or decreasing the most. For example, if the gradient of a function is positive at a certain point, it means that the function is increasing at that point, and if the gradient is negative, it means that the function is decreasing at that point.
The gradient descent algorithm starts with an initial guess for the parameters of the function and then iteratively improves these guesses by taking small steps in the direction opposite to the gradient of the function at the current point. This process continues until the algorithm reaches a local or global minimum, where the gradient is zero (i.e., the function is not increasing or decreasing).

How does Gradient Descent work?

The gradient descent algorithm is an iterative algorithm that updates the parameters of a function by taking steps in the opposite direction of the gradient of the function. The gradient of a function tells us the direction in which the function is increasing or decreasing the most. The gradient descent algorithm uses the gradient to update the parameters in the direction that reduces the value of the cost function.
The gradient descent algorithm works in the following way:

1. Initialize the parameters of the function with some random values.
2. Calculate the gradient of the cost function with respect to the parameters.
3. Update the parameters by taking a small step in the opposite direction of the gradient.
4. Repeat steps 2 and 3 until the algorithm reaches a local or global minimum, where the gradient is zero.
5. Here is a simple example to illustrate the gradient descent algorithm in action. Let's say we have a function $f(x) = x^2$, and we want to find the value of x that minimizes the function. We can use the gradient descent algorithm to find this value.

First, we initialize the value of x with some random value, say x = 3. Next, we calculate the gradient of the function with respect to x, which is 2x. In this case, the gradient is 6 (2 * 3). Since the gradient is positive, it means that the function is increasing at x = 3, and we need to take a step in the opposite direction to reduce the value of the function.
We update the value of x by subtracting a small step size (called the learning rate) from the current value of x. For example, if the learning rate is 0.1, we can update the value of x as follows:

```
x = x - 0.1 * gradient
  = 3 - 0.1 * 6
  = 2.4
```

We repeat this process until the algorithm reaches a local or global minimum. To implement gradient descent in TensorFlow, we first need to define the cost function that we want to minimize. In this example, we will use a simple linear regression model to illustrate how gradient descent works.
Linear regression is a popular machine learning algorithm that is used to model the relationship between a dependent variable (y) and one or more independent variables (x). In a linear regression model, we try to find the best-fit line that describes the relationship between the dependent and independent variables. To create a linear regression model in TensorFlow, we first need to define the placeholders for the input and output data. A placeholder is a TensorFlow variable that we can use to feed data into our model.
Here is the code to define the placeholders for the input and output data:
Python

```python
# Import tensorflow 2 as tensorflow 1
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

# Define the placeholders for
# the input and output data
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Define the placeholders for
# the input and output data
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
```

Next, we need to define the variables that represent the parameters of our linear regression model. In this example, we will use a single variable (w) to represent the slope of the best-fit line. We initialize the value of w with a random value, say 0.5.
Here is the code to define the variable for the model parameters:
Python

```python
# Define the model parameters
w = tf.Variable(0.5, name="weights")
```

Once we have defined the placeholders and the model parameters, we can define the linear regression model by using the TensorFlow tf.add() and tf.multiply() functions. The tf.add() function is used to add the bias term to the model, and the tf.multiply() function is used to multiply the input data (x) and the model parameters (w).
Here is the code to define the linear regression model:
Python

```python
# Define the linear regression model
model = tf.add(tf.multiply(x, w), 0.5)
```

Once we have defined the linear regression model, we need to define the cost function that we want to minimize. In this example, we will use the mean squared error (MSE) as the cost function. The MSE is a popular metric that is used to evaluate the performance of a linear regression model. It measures the average squared difference between the predicted values and the actual values.

To define the cost function, we first need to calculate the difference between the predicted values and the actual values using the TensorFlow tf.square() function. The tf.square() function squares each element in the input tensor and returns the squared values.
Here is the code to define the cost function using the MSE:
Python

```
# Define the cost function (MSE)
cost = tf.reduce_mean(tf.square(model - y))
```

Once we have defined the cost function, we can use the TensorFlow tf.train.GradientDescentOptimizer() function to create an optimizer that uses the gradient descent algorithm to minimize the cost function. The tf.train.GradientDescentOptimizer() function takes the learning rate as an input parameter. The learning rate is a hyperparameter that determines the size of the steps that the algorithm takes to reach the minimum of the cost function.
Here is the code to create the gradient descent optimizer:
Python

```
# Create the gradient descent optimizer
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
```

Once we have defined the optimizer, we can use the minimize() method of the optimizer to minimize the cost function. The minimize() method takes the cost function as an input parameter and returns an operation that, when executed, performs one step of gradient descent on the cost function.
Here is the code to minimize the cost function using the gradient descent optimizer:
Python

```
# Minimize the cost function
train = optimizer.minimize(cost)
```

Once we have defined the gradient descent optimizer and the train operation, we can use the TensorFlow Session class to train our model. The Session class provides a way to execute TensorFlow operations. To train the model, we need to initialize the variables that we have defined earlier (i.e., the model parameters and the optimizer) and then run the train operation in a loop for a specified number of iterations.
Here is the code to train the linear regression model using the gradient descent optimizer:
Python

```
# Define the toy dataset
x_train = [1, 2, 3, 4]
y_train = [2, 4, 6, 8]

# Create a TensorFlow session
with tf.Session() as sess:
    # Initialize the variables
    sess.run(tf.global_variables_initializer())

    # Training loop
    for i in range(1000):
        sess.run(train,
                 feed_dict={x: x_train,
```

```
                                y: y_train})

    # Evaluate the model
    w_val = sess.run(w)

# Mean Squared Error (MSE) between
# the predicted and true output values
print(w_val)
```

In the above code, we have defined a Session object and used the global_variables_initializer() method to initialize the variables. Next, we have run the train operation in a loop for 1000 iterations. In each iteration, we have fed the input and output data to the train operation using the feed_dict parameter. Finally, we evaluated the trained model by running the w variable to get the value of the model parameters. This will train a linear regression model on the toy dataset using gradient descent. The model will learn the weights w that minimizes the mean squared error between the predicted and true output values.

Visualizing the convergence of Gradient Descent using Linear Regression

Linear regression is a method for modeling the linear relationship between a dependent variable (also known as the response or output variable) and one or more independent variables (also known as the predictor or input variables). The goal of linear regression is to find the values of the model parameters (coefficients) that minimize the difference between the predicted values and the true values of the dependent variable.

The linear regression model can be expressed as follows:

$$y\hat{}=w1x1+w2x2+...+wnxn+by\hat{}=w_1x_1+w_2x_2+...+w_nx_n+b$$

where:

- $y\hat{}$ $y\hat{}$ is the predicted value of the dependent variable
- x1,x2,…,xn $x_1,x_2,...,x_n$ are the independent variables w1,w2,…,wn $w_1,w_2,...,w_n$ are the coefficients (model parameters) associated with the independent variables.
- b is the intercept (a constant term).

To train the linear regression model, you need a dataset with input features (independent variables) and labels (dependent variables). You can then use an optimization algorithm, such as gradient descent, to find the values of the model parameters that minimize the loss function.

The loss function measures the difference between the predicted values and the true values of the dependent variable. There are various loss functions that can be used for linear regression, such as mean squared error (MSE) and mean absolute error (MAE).

The MSE loss function is defined as follows:

$$MSE=1N\sum i=1N(yi\hat{}-yi)2MSE=N1\sum_{i=1N}(y_i\hat{}-y_i)2$$

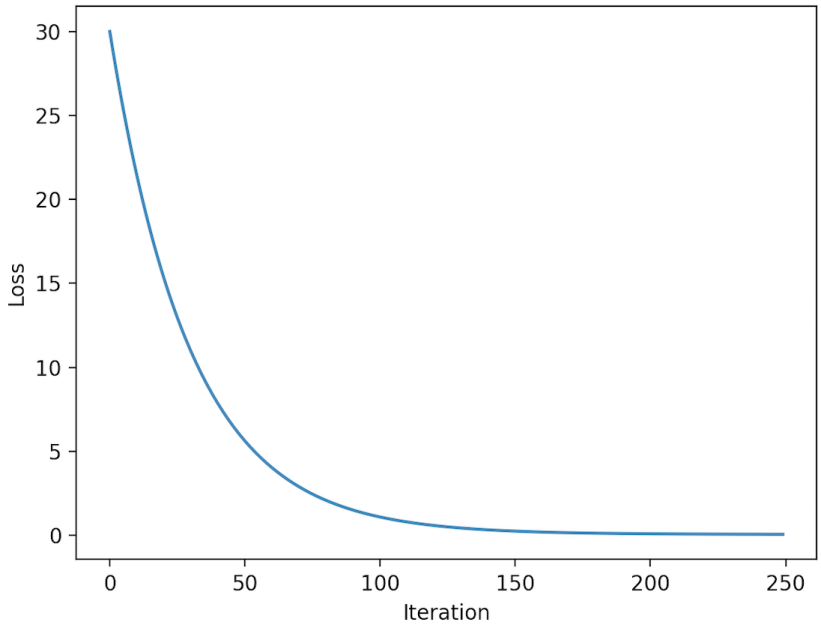
where:

- $yi\hat{}$ $y_i\hat{}$ is the predicted value for the i i th sample
- yi y_i is the true value for the i i th sample
- N is the total number of samples
- The MSE loss function measures the average squared difference between the predicted values and the true values. A lower MSE value indicates that the model is performing better.

Python

```python
import tensorflow as tf
import matplotlib.pyplot as plt

# Set up the data and model
X = tf.constant([[1.], [2.], [3.], [4.]])
y = tf.constant([[2.], [4.], [6.], [8.]])

w = tf.Variable(0.)
b = tf.Variable(0.)

# Define the model and loss function
def model(x):
    return w * x + b

def loss(predicted_y, true_y):
    return tf.reduce_mean(tf.square(predicted_y - true_y))

# Set the learning rate
learning_rate = 0.001

# Training loop
losses = []
for i in range(250):
    with tf.GradientTape() as tape:
        predicted_y = model(X)
        current_loss = loss(predicted_y, y)
    gradients = tape.gradient(current_loss, [w, b])
    w.assign_sub(learning_rate * gradients[0])
    b.assign_sub(learning_rate * gradients[1])

    losses.append(current_loss.numpy())

# Plot the loss
plt.plot(losses)
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()
```

Output:



Loss vs Iteration

The loss function calculates the mean squared error (MSE) loss between the predicted values and the true labels. The model function defines the linear regression model, which is a linear function of the form $w*x+b$ $w*x+b$.
The training loop performs 250 iterations of gradient descent. At each iteration, the with tf.GradientTape() as tape: block activates the gradient tape, which records the operations for computing the gradients of the loss with respect to the model parameters. Inside the block, the predicted values are calculated using the model function and the current values of the model parameters. The loss is then calculated using the loss function and the predicted values and true labels.
After the loss has been calculated, the gradients of the loss with respect to the model parameters are computed using the gradient method of the gradient tape. The model parameters are then updated by subtracting the learning rate multiplied by the gradients from the current values of the parameters. This process is repeated until the training loop is completed.
Finally, the model parameters will contain the optimized values that minimize the loss function, and the model will be trained to predict the dependent variable given the independent variables.
A list called losses store the loss at each iteration. After the training loop is completed, the losses list contains the loss values at each iteration.
The plt.plot function plots the losses list as a function of the iteration number, which is simply the index of the loss in the list. The plt.xlabel and plt.ylabel functions add labels to the x-axis and y-axis of the plot, respectively. Finally, the plt.show function displays the plot.
The resulting plot shows how the loss changes over the course of the training process. As the model is trained, the loss should decrease, indicating that the model is learning and the model parameters are being optimized to minimize the loss. Eventually, the loss should converge to a minimum value, indicating that the model has reached a good solution. The rate at which the loss decreases and the final value of the loss will depend

on various factors, such as the learning rate, the initial values of the model parameters, and the complexity of the model.

Visualizing the Gradient Descent

Gradient descent is an optimization algorithm that is used to find the values of the model parameters that minimize the loss function. The algorithm works by starting with initial values for the parameters and then iteratively updating the values to minimize the loss. The equation for the gradient descent algorithm for linear regression can be written as follows:

$$\text{wi}=\text{wi}-\alpha\partial\partial\text{wiMSE}(\text{w1},\text{w2},\ldots,\text{wn})\quad w_i=w_i-\alpha\partial w_i\partial\text{MSE}(w_1,w_2,...,w_n)$$

$$\text{b}=\text{b}-\alpha\partial\partial\text{bMSE}(\text{w1},\text{w2},\ldots,\text{wn})\quad b=b-\alpha\partial b\partial\text{MSE}(w_1,w_2,...,w_n)$$

where:

- w_i is the ith model parameter.
- α is the learning rate (a hyperparameter that determines the step size of the update).
- $\partial\partial\text{wiMSE}(\text{w1},\text{w2},\ldots,\text{wn})$ $\partial w_i\partial\text{MSE}(w_1,w_2,...,w_n)$ is the partial derivative of the MSE loss function with respect to the ith model parameter.

This equation updates the value of each parameter in the direction that reduces the loss. The learning rate determines the size of the update, with a smaller learning rate resulting in smaller steps and a larger learning rate resulting in larger steps.

The process of performing gradient descent can be visualized as taking small steps downhill on a loss surface, with the goal of reaching the global minimum of the loss function. The global minimum is the point on the loss surface where the loss is the lowest.

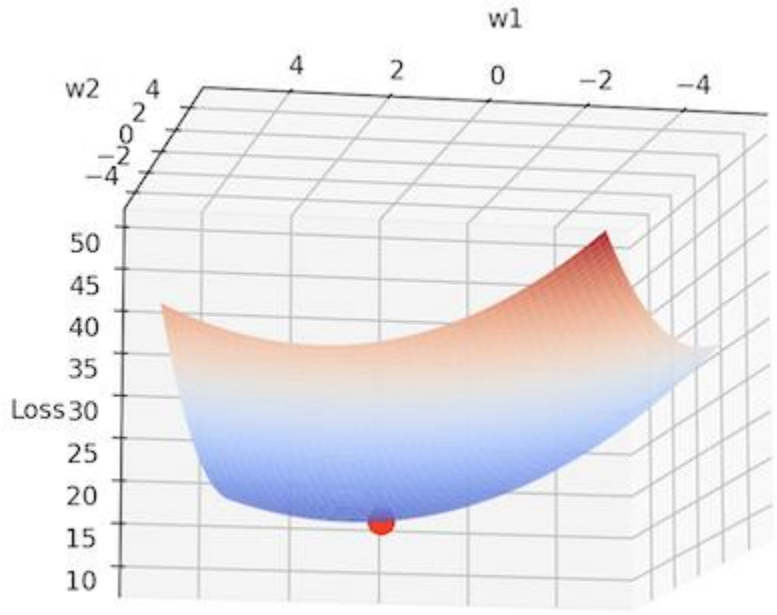
Here is an example of how to plot the loss surface and the trajectory of the gradient descent algorithm:

Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data
np.random.seed(42)
X = 2 * np.random.rand(100, 1) - 1
y = 4 + 3 * X + np.random.randn(100, 1)

# Initialize model parameters
w = np.random.randn(2, 1)
b = np.random.randn(1)[0]

# Set the learning rate
alpha = 0.1

# Set the number of iterations
num_iterations = 20

# Create a mesh to plot the loss surface
w1, w2 = np.meshgrid(np.linspace(-5, 5, 100),
                     np.linspace(-5, 5, 100))
```

```
# Compute the loss for each point on the grid
loss = np.zeros_like(w1)
for i in range(w1.shape[0]):
    for j in range(w1.shape[1]):
        loss[i, j] = np.mean((y - w1[i, j] \
                              * X - w2[i, j] * X**2)**2)

# Perform gradient descent
for i in range(num_iterations):
    # Compute the gradient of the loss
    # with respect to the model parameters
    grad_w1 = -2 * np.mean(X * (y - w[0] \
                                * X - w[1] * X**2))
    grad_w2 = -2 * np.mean(X**2 * (y - w[0] \
                                   * X - w[1] * X**2))

    # Update the model parameters
    w[0] -= alpha * grad_w1
    w[1] -= alpha * grad_w2

# Plot the loss surface
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(projection='3d')
ax.plot_surface(w1, w2, loss, cmap='coolwarm')
ax.set_xlabel('w1')
ax.set_ylabel('w2')
ax.set_zlabel('Loss')

# Plot the trajectory of the gradient descent algorithm
ax.plot(w[0], w[1], np.mean((y - w[0]\
                             * X - w[1] * X**2)**2),
        'o', c='red', markersize=10)
plt.show()
```

Output:



Gradient Descent finding global minima

This code generates synthetic data for a quadratic regression problem, initializes the model parameters, and performs gradient descent to find the values of the model parameters that minimize the mean squared error loss. The code also plots the loss surface and the trajectory of the gradient descent algorithm on the loss surface.
The resulting plot shows how the gradient descent algorithm takes small steps downhill on the loss surface and eventually reaches the global minimum of the loss function. The global minimum is the point on the loss surface where the loss is the lowest.
It is important to choose an appropriate learning rate for the gradient descent algorithm. If the learning rate is too small, the algorithm will take a long time to converge to the global minimum. On the other hand, if the learning rate is too large, the algorithm may overshoot the global minimum and may not converge to a good solution.
Another important consideration is the initialization of the model parameters. If the initialization is too far from the global minimum, the gradient descent algorithm may take a long time to converge. It is often helpful to initialize the model parameters to small random values.
It is also important to choose an appropriate stopping criterion for the gradient descent algorithm. One common stopping criterion is to stop the algorithm when the loss function stops improving or when the improvement is below a certain threshold. Another option is to stop the algorithm after a fixed number of iterations.
Overall, gradient descent is a powerful optimization algorithm that can be used to find the values of the model parameters that minimize the loss function for a wide range of machine learning problems.

Conclusion

In this blog, we have discussed gradient descent optimization in TensorFlow and how to implement it to train a linear regression model. We have seen that TensorFlow provides several optimizers that implement different variations of gradient descent, such as stochastic gradient descent and mini-batch gradient descent.
Gradient descent is a powerful optimization algorithm that is widely used in machine learning and deep learning to find the optimal solution to a given problem. It is an iterative algorithm that updates the parameters of a function by taking steps in the opposite direction of the gradient of the function. TensorFlow makes it easy to implement gradient descent by providing built-in optimizers and functions for computing gradients.

Convolution Neural Network

A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.
When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use **Recurrent Neural Networks** more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.
In a regular Neural Network there are three types of layers:

1. **Input Layers:** It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
2. **Hidden Layer:** The input from the Input layer is then fed into the hidden layer. There can be many hidden layers depending on our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of the output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

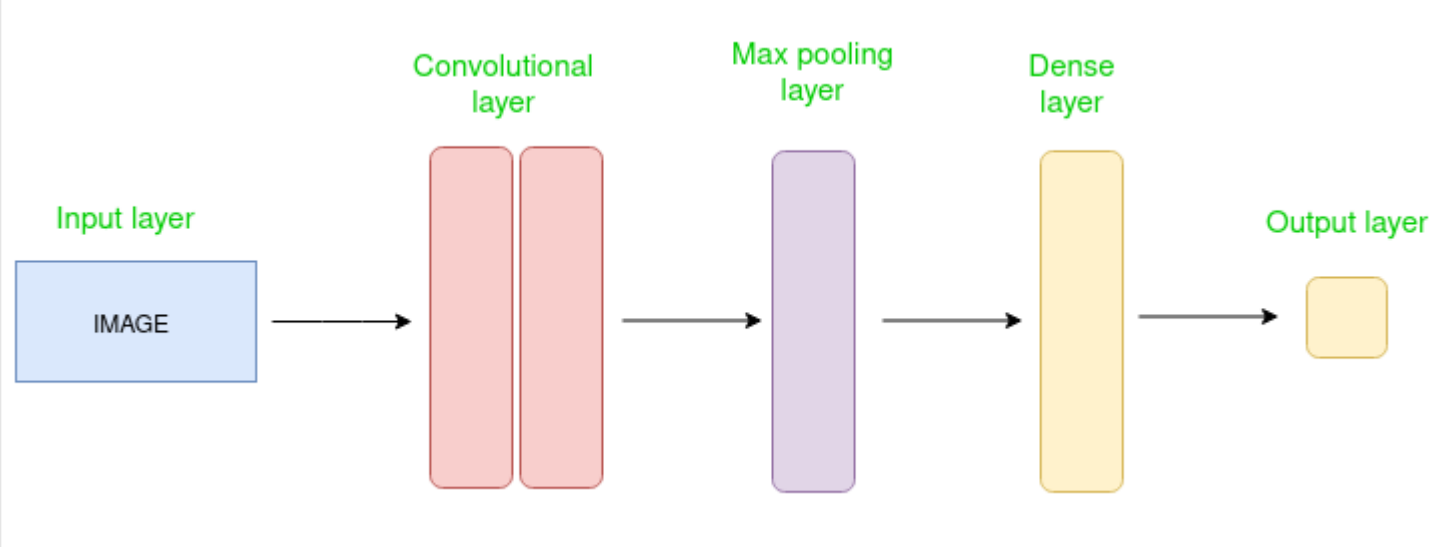
The data is fed into the model and output from each layer is obtained from the above step is called **feedforward**, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the model by calculating the derivatives. This step is called **Backpropagation** which basically is used to minimize the loss.

Convolution Neural Network

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

CNN architecture

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

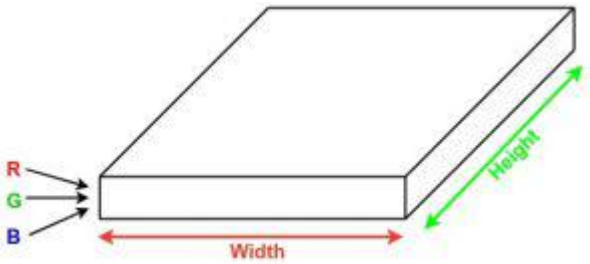


Simple CNN architecture

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

How Convolutional Layers works

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Now let's talk about a bit of mathematics that is involved in the whole convolution process.

- Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where 'a' can be anything like 3, 5, or 7 but smaller as compared to the image dimension.
- During the forward pass, we slide each filter across the whole input volume step by step where each step is called **stride** (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume.

- As we slide our filters we'll get a 2-D output for each filter and we'll stack them together as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

Layers used to build ConvNets

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers: datasets

Let's take an example by running a covnets on of image of dimension 32 x 32 x 3.

- **Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.
- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: max(0, x), **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.
- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

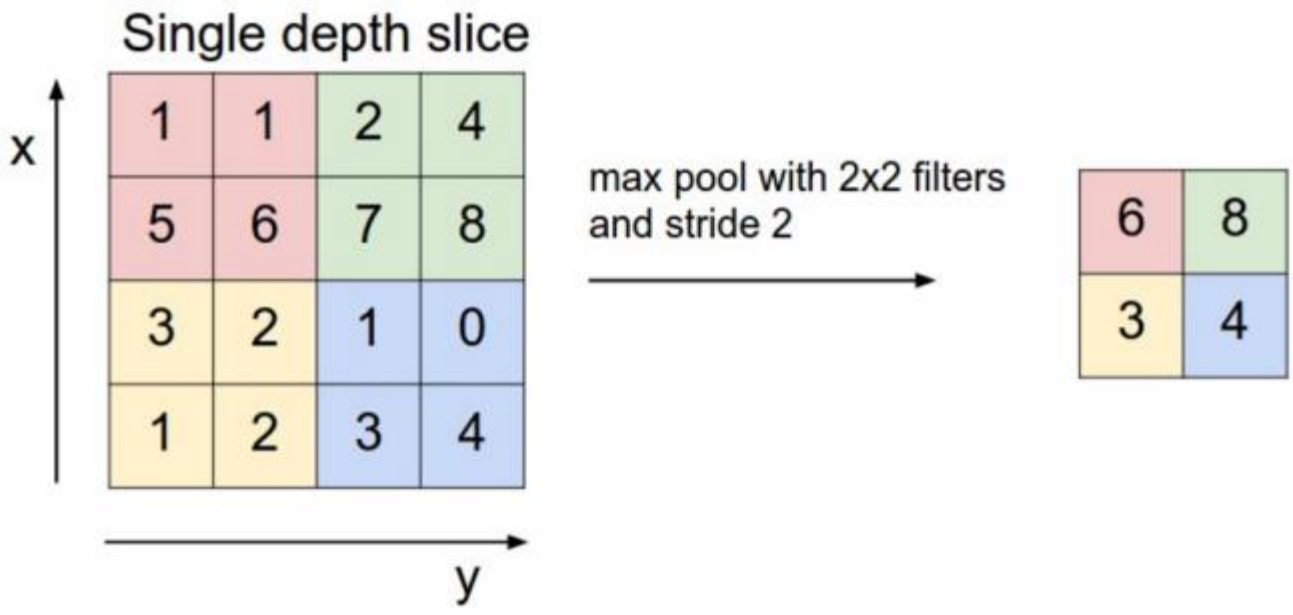


Image source: cs231n.stanford.edu

- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

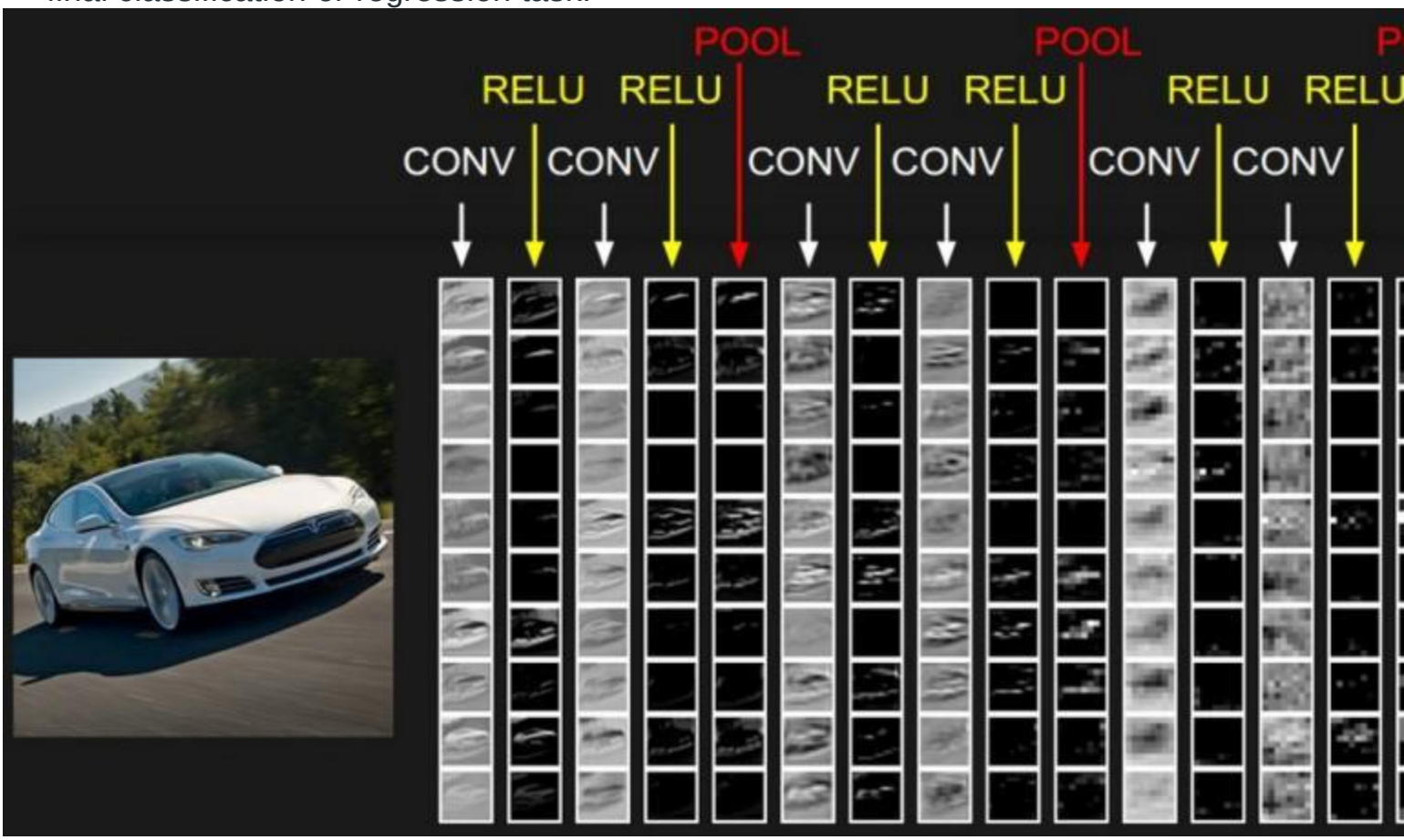


Image source: cs231n.stanford.edu

- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Example:

Let’s consider an image and apply the convolution layer, activation layer, and pooling layer operation to extract the inside feature.

Input image:

Input image

Step:

- import the necessary libraries
- set the parameter
- define the kernel
- Load the image and plot it.
- Reformat the image
- Apply convolution layer operation and plot the output image.
- Apply activation layer operation and plot the output image.
- Apply pooling layer operation and plot the output image.

- Python3

```
# import the necessary libraries

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from itertools import product



# set the param

plt.rc('figure', autolayout=True)

plt.rc('image', cmap='magma')



# define the kernel

kernel = tf.constant([[-1, -1, -1],

                      [-1,  8, -1],

                      [-1, -1, -1],

                     ])



# load the image

image = tf.io.read_file('Ganesh.jpg')

image = tf.io.decode_jpeg(image, channels=1)

image = tf.image.resize(image, size=[300, 300])
```

```
# plot the image

img = tf.squeeze(image).numpy()

plt.figure(figsize=(5, 5))

plt.imshow(img, cmap='gray')

plt.axis('off')

plt.title('Original Gray Scale image')

plt.show();
```

```
# Reformat

image = tf.image.convert_image_dtype(image, dtype=tf.float32)

image = tf.expand_dims(image, axis=0)

kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])

kernel = tf.cast(kernel, dtype=tf.float32)
```

```
# convolution layer

conv_fn = tf.nn.conv2d
```

```
image_filter = conv_fn(

    input=image,
```

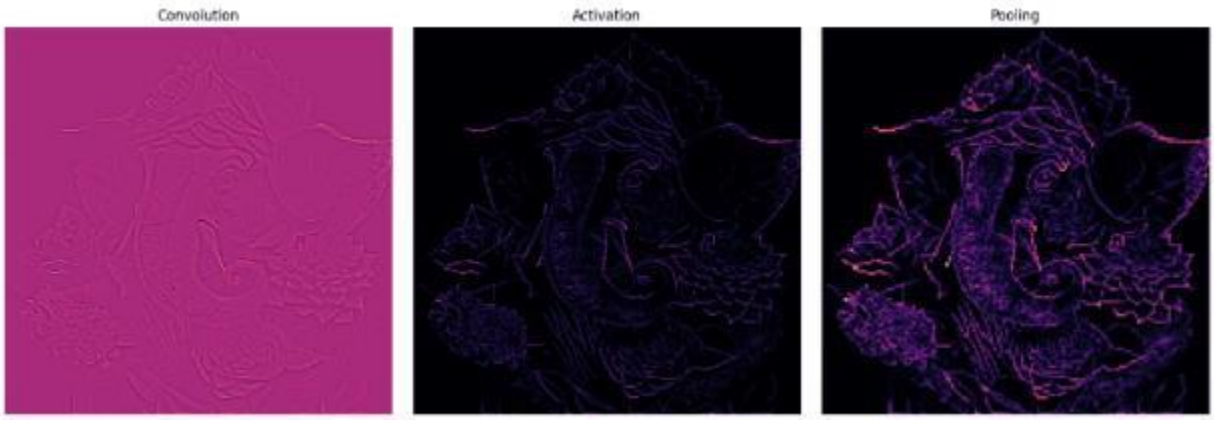
```
    filters=kernel,
    strides=1, # or (1, 1)
    padding='SAME',
)


plt.figure(figsize=(15, 5))


# Plot the convolved image
plt.subplot(1, 3, 1)


plt.imshow(
    tf.squeeze(image_filter)
)
plt.axis('off')
plt.title('Convolution')


# activation layer
relu_fn = tf.nn.relu
# Image detection
image_detect = relu_fn(image_filter)
```

```
plt.subplot(1, 3, 2)

plt.imshow(

    # Reformat for plotting

    tf.squeeze(image_detect)

)



plt.axis('off')

plt.title('Activation')



# Pooling layer

pool = tf.nn.pool

image_condense = pool(input=image_detect,

                              window_shape=(2, 2),

                              pooling_type='MAX',

                              strides=(2, 2),

                              padding='SAME',

                             )



plt.subplot(1, 3, 3)

plt.imshow(tf.squeeze(image_condense))

plt.axis('off')

plt.title('Pooling')
```

```
plt.show()
```

Output:

Original Grayscale image



Output

Advantages of Convolutional Neural Networks (CNNs):

1. Good at detecting patterns and features in images, videos, and audio signals.
2. Robust to translation, rotation, and scaling invariance.
3. End-to-end training, no need for manual feature extraction.
4. Can handle large amounts of data and achieve high accuracy.

Disadvantages of Convolutional Neural Networks (CNNs):

1. Computationally expensive to train and require a lot of memory.
2. Can be prone to overfitting if not enough data or proper regularization is used.
3. Requires large amounts of labeled data.
4. Interpretability is limited, it’s hard to understand what the network has learned.

Recurrent Neural Network

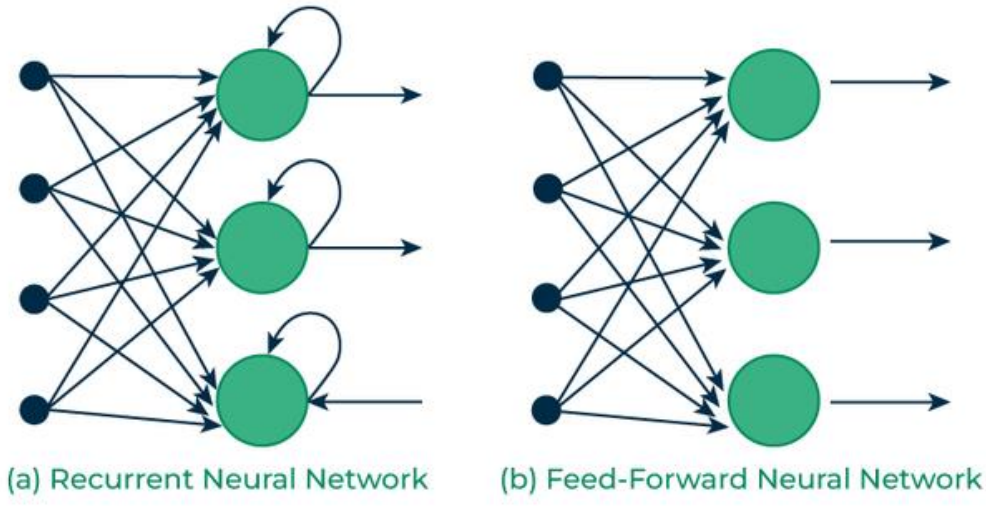
Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as *Memory State* since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

Recurrent Neural Network

How RNN differs from Feedforward Neural Network?

Artificial neural networks that do not have looping nodes are called feed forward neural networks. Because all information is only passed forward, this kind of neural network is also referred to as a multi-layer neural network.

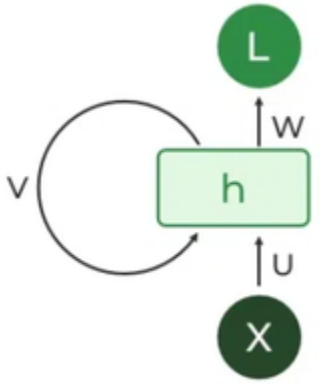
Information moves from the input layer to the output layer – if any hidden layers are present – unidirectionally in a feedforward neural network. These networks are appropriate for image classification tasks, for example, where input and output are independent. Nevertheless, their inability to retain previous inputs automatically renders them less useful for sequential data analysis.



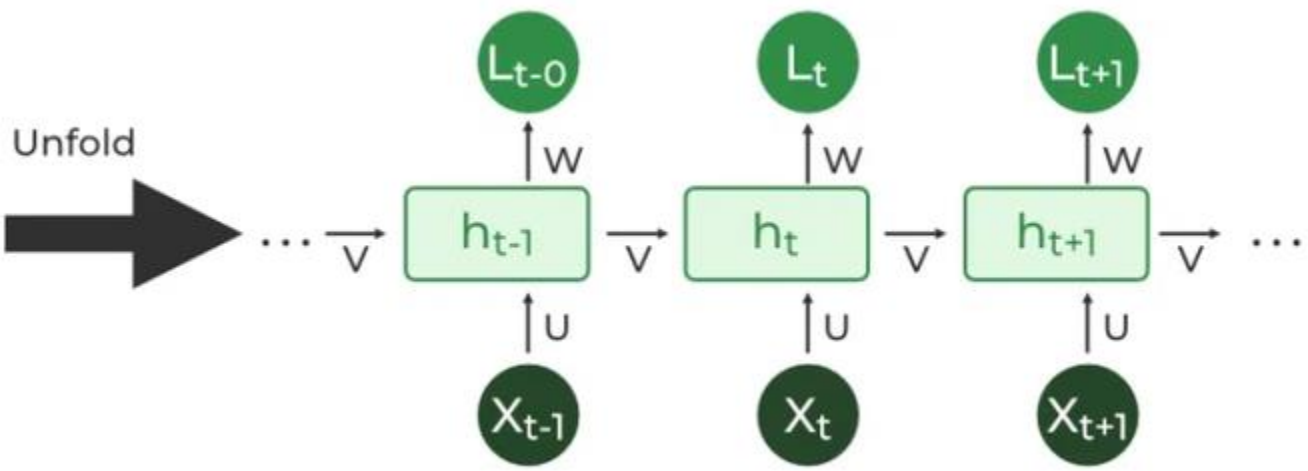
Recurrent Vs Feedforward networks

Recurrent Neuron and RNN Unfolding

The fundamental processing unit in a Recurrent Neural Network (RNN) is a Recurrent Unit, which is not explicitly called a “Recurrent Neuron.” This unit has the unique ability to maintain a hidden state, allowing the network to capture sequential dependencies by remembering previous inputs while processing. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) versions improve the RNN’s ability to handle long-term dependencies.



Recurrent Neuron



Types Of RNN

There are four types of RNNs based on the number of inputs and outputs in the network.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

One to One

This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.

One To Many

In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.

One to Many RNN

Many to One

In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.

Many to One RNN

Many to Many

In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.

Many to Many RNN

Recurrent Neural Network Architecture

RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state H_i for every input X_i . **By using the following formulas:**

$h = \sigma(UX + Wh_{-1} + B)$

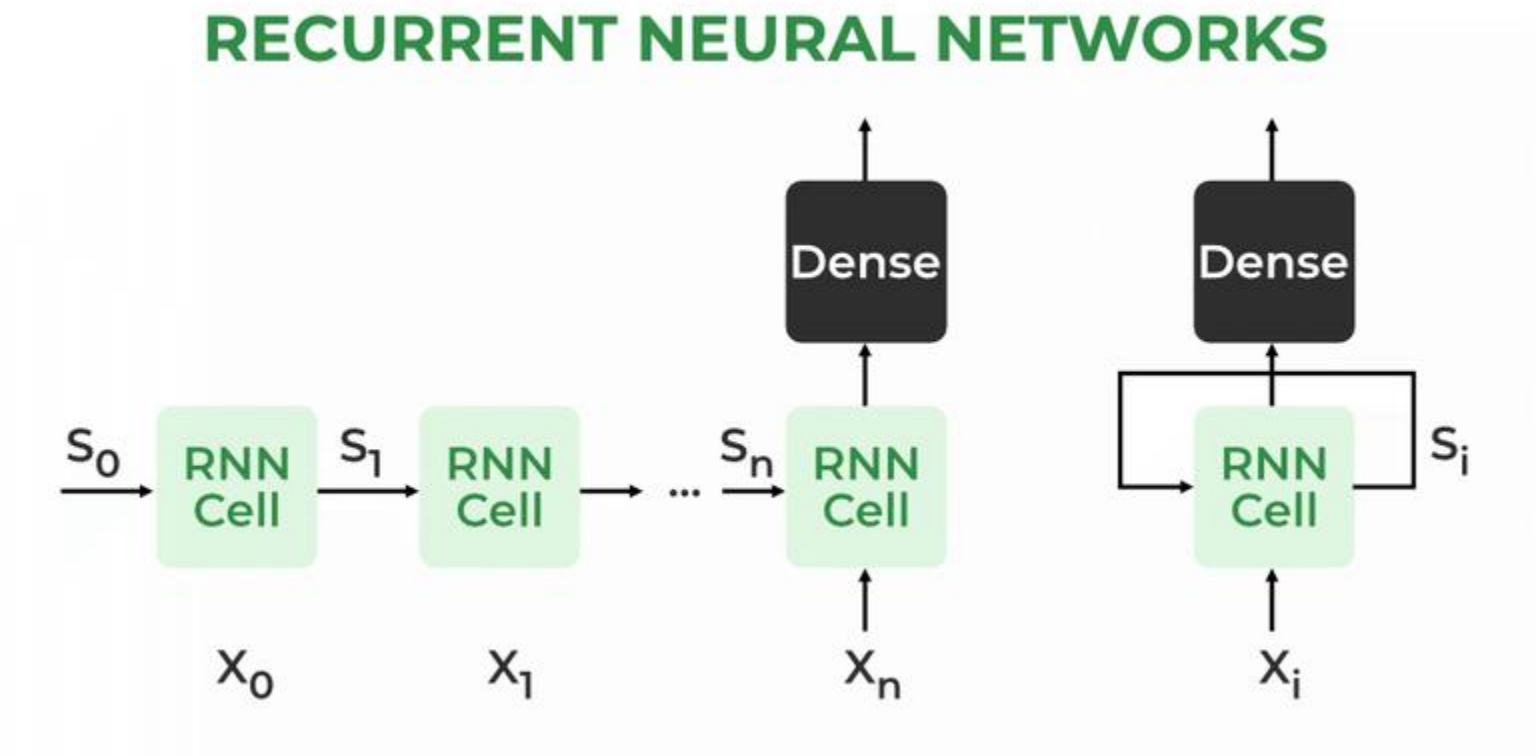
$Y = O(Vh + C)$

Hence

$Y = f(X, h, W, U, V, B, C)$

Here S is the State matrix which has element si as the state of the network at timestep i

The parameters in the network are W, U, V, c, b which are shared across timestep



Recurrent Neural Architecture

How does RNN work?

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

The formula for calculating the current state:

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

where,

- h_t -> current state

- h_{t-1} -> previous state
- x_t -> input state

Formula for applying Activation function(tanh)

$ht=\tanh(Whhht-1+Wxhxt)h_t=tanh(W_{hh}h_{t-1}+W_{xh}x_t)$

where,

- w_{hh} -> weight at recurrent neuron
- w_{xh} -> weight at input neuron

The formula for calculating output:

$yt=Whyhty_t=W_{hy}h_t$

- Y_t -> output
- W_{hy} -> weight at output layer

These parameters are updated using Backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

Backpropagation Through Time (BPTT)

In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h1 then h2 then h3 so on. Hence we will apply backpropagation throughout all these hidden time states sequentially.

- L(θ)(loss function) depends on h3
- h3 in turn depends on h2 and W
- h2 in turn depends on h1 and W
- h1 in turn depends on h0 and W
- where h0 is a constant starting state.

$\partial L(\theta)\partial W=\sum t=1T\partial L(\theta)\partial W\partial W\partial \mathbf{L}(\theta)=\sum t=1T\partial W\partial \mathbf{L}(\theta)$

For simplicity of this equation, we will apply backpropagation on only one row

$\partial L(\theta)\partial W=\partial L(\theta)\partial h3\partial h3\partial W\partial W\partial L(\theta)=\partial h3\partial L(\theta)\partial W\partial h3$

We already know how to compute this one as it is the same as any simple deep neural network backpropagation.

$\partial L(\theta)\partial h3\partial h3\partial L(\theta)$

.However, we will see how to apply backpropagation to this term $\partial h3\partial W\partial W\partial h3$

```
As we know h3 = σ(Wh2 + b)
```

And In such an ordered network, we can't compute $\partial h3\partial W\partial W\partial h3$ by simply treating h3 as a constant because as it also depends on W. the total derivative $\partial h3\partial W\partial W\partial h3$ has two parts:

1. **Explicit:** $\partial h3+\partial W\partial W\partial h3+$ treating all other inputs as constant
2. **Implicit:** Summing over all indirect paths from h_3 to W

Let us see how to do this

$\partial h3\partial W=\partial h3+\partial W+\partial h3\partial h2\partial h2\partial W=\partial h3+\partial W+\partial h3\partial h2[\partial h2+\partial W+\partial h2\partial h1\partial h1\partial W]=\partial h3+\partial W+\partial h3\partial h2\partial h2+\partial W+\partial h3\partial h2\partial h2\partial h1[\partial h1+\partial W]\partial W\partial h3=\partial W\partial h3+\partial h2\partial h3\partial W\partial h2=\partial W\partial h3+\partial h2\partial h3[\partial W\partial h2+\partial h1\partial h2\partial W\partial h1]=\partial W\partial h3+\partial h2\partial h3\partial W\partial h2+\partial h2\partial h3\partial h1\partial h2[\partial W\partial h1+]$

For simplicity, we will short-circuit some of the paths

$\partial h3\partial W=\partial h3+\partial W+\partial h3\partial h2\partial h2+\partial W+\partial h3\partial h1\partial h1+\partial W\partial W\partial h3=\partial W\partial h3+\partial h2\partial h3\partial W\partial h2+\partial h1\partial h3\partial W\partial h1+$

Finally, we have

$\partial L(\theta)\partial W=\partial L(\theta)\partial h3\cdot\partial h3\partial W\partial W\partial L(\theta)=\partial h3\partial L(\theta)\cdot\partial W\partial h3$

Where

$\partial h3\partial W=\sum k=13\partial h3\partial hk\cdot\partial hk\partial W\partial W\partial h3=\sum k=13\partial hk\partial h3\cdot\partial W\partial hk$

Hence,

$\partial L(\theta)\partial W=\partial L(\theta)\partial h3\sum k=13\partial h3\partial hk\cdot\partial hk\partial W\partial W\partial L(\theta)=\partial h3\partial L(\theta)\sum k=13\partial hk\partial h3\cdot\partial W\partial hk$

This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

Issues of Standard RNNs

1. **Vanishing Gradient:** Text generation, machine translation, and stock market prediction are just a few examples of the time-dependent and sequential data problems that can be modelled with recurrent neural networks. You will discover, though, that the gradient problem makes training RNN difficult.
2. **Exploding Gradient:** An Exploding Gradient occurs when a neural network is being trained and the slope tends to grow exponentially rather than decay. Large error gradients that build up during training lead to very large updates to the neural network model weights, which is the source of this issue.

Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.

5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained using Backpropagation through time.

Advantages and Disadvantages of Recurrent Neural Network

Advantages

1. An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

Applications of Recurrent Neural Network

1. Language Modelling and Generating Text
2. Speech Recognition
3. Machine Translation
4. Image Recognition, Face detection
5. Time series Forecasting

Variation Of Recurrent Neural Network (RNN)

To overcome the problems like vanishing gradient and exploding gradient descent several new advanced versions of RNNs are formed some of these are as;

1. Bidirectional Neural Network (BiNN)
2. Long Short-Term Memory (LSTM)

Bidirectional Neural Network (BiNN)

A BiNN is a variation of a Recurrent Neural Network in which the input information flows in both direction and then the output of both direction are combined to produce the input. BiNN is useful in situations when the context of the input is more important such as Nlp tasks and Time-series analysis problems.

Long Short-Term Memory (LSTM)

Long Short-Term Memory works on the read-write-and-forget principle where given the input information network reads and writes the most useful information from the data and it forgets about the information which is not important in predicting the output. For doing this three new gates are introduced in the RNN. In this way, only the selected information is passed through the network.

Difference between RNN and Simple Neural Network

RNN is considered to be the better version of deep neural when the data is sequential. There are significant differences between the RNN and deep neural networks they are listed as:

Recurrent Neural Network	Deep Neural Network
Weights are same across all the layers number of a Recurrent Neural Network	Weights are different for each layer of the network
Recurrent Neural Networks are used when the data is sequential and the number of inputs is not predefined.	A Simple Deep Neural network does not have any special method for sequential data also here the number of inputs is fixed
The Numbers of parameter in the RNN are higher than in simple DNN	The Numbers of Parameter are lower than RNN
Exploding and vanishing gradients is the the major drawback of RNN	These problems also occur in DNN but these are not the major problem with DNN