Module2

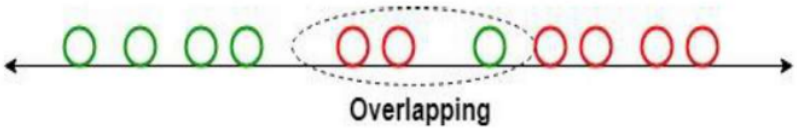
SINGLE LAYER NETWORKS

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA), also known as Normal Discriminant Analysis or Discriminant Function Analysis, is a Dimensionality reduction technique primarily utilized in supervised classification problems. It facilitates the modeling of distinctions between groups, effectively separating two or more classes.

LDA operates by projecting features from a higher-dimensional space into a lower-dimensional one. In machine learning, LDA serves as a supervised learning algorithm specifically designed for classification tasks, aiming to identify a linear combination of features that optimally segregates classes within a dataset.

For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.



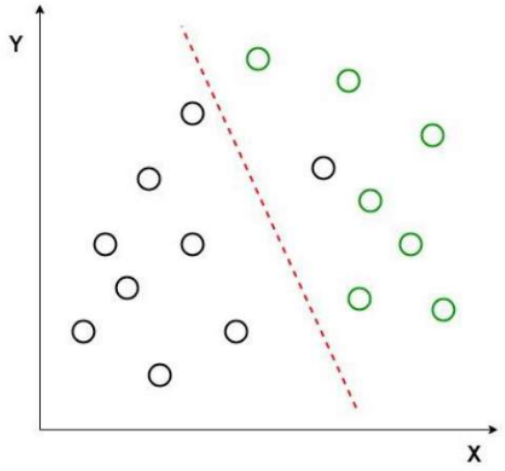
Assumptions of LDA

LDA assumes that the data has a Gaussian distribution and that the covariance matrices of the different classes are equal. It also assumes that the data is linearly separable, meaning that a linear decision boundary can accurately classify the different classes.

Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.

Linearly Separable Dataset

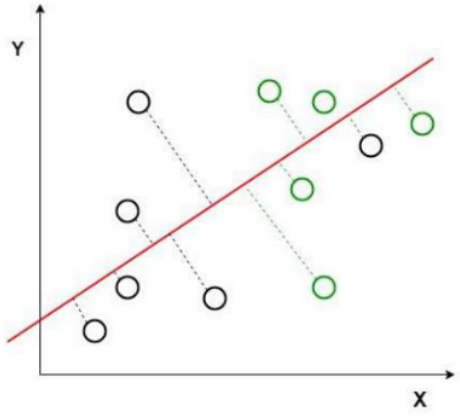
Here, Linear Discriminant Analysis uses both axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reduces the 2D graph into a 1D graph.



Two criteria are used by LDA to create a new axis

1.Maximize the distance between the means of the two classes.

2.Minimize the variation within each class.



The perpendicular distance between the line and points.

LDA working process

LDA works by projecting the data onto a lower-dimensional space that maximizes the separation between the classes. It does this by finding a set of linear discriminating that maximize the ratio of between-class variance to within-class variance. In other words, it finds the directions in the feature space that best separates the different classes of data.

Mathematical Intuition Behind LDA

Let's suppose we have two classes and d- dimensional samples such as x1, x2 … xn, where,

n1 samples coming from the class (c1) and n2 coming from the class (c2).

If xi is the data point, then its projection on the line represented by unit vector v can be written as vTxi

Let's consider u1 and u2 to be the means of samples class c1 and c2 respectively before projection and u1hat denote the mean of the samples of class after projection and it can be calculated by

$\mu 1\sim = 1n1 \sum xi \in c1 n1 vTxi = vT\mu 1$ $\mu 1 = n1 \; 1 \; \sum xi \in c1 \; n1 \; vTxi = vT\mu 1$

Extensions to LDA

1. **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance when there are multiple input variables).
2. **Flexible Discriminant Analysis (FDA):** Where non-linear combinations of inputs are used such as splines.
3. **Regularized Discriminant Analysis (RDA):** Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

Linear Separability

Linear Separability refers to the data points in binary classification problems which can be separated using linear decision boundary. if the data points can be separated using a line, linear function, or flat hyperplane are considered linearly separable.

Linear separability is an important concept in neural networks. If the separate points in n-dimensional space follows then it is said linearly separable

For two-dimensional inputs, if there exists a line (whose equation is) that separates all samples of one class from the other class, then an appropriate perception can be derived from the equation of the separating line. such classification problems are called "Linear separable" i.e, separating by a linear combination of i/p.
The logical AND gate example shown below illustrates a two-dimensional example of a linearly separable problem.

Linear Separability as Mathematics

Linear separability is introduced in the context of linear algebra and optimization theory. It speaks of the capacity of a hyperplane to divide two classes of data points in a high-dimensional space.

Let's use the example of a set of data points in a p-dimensional space, where p is the number of features or variables that each point has to characterize it.

A linear function can be used to represent the hyperplane mathematically, where are the features of the data point, are corresponding weights. so that we can separate two different categories with a straight line and can represent them on the graph then we will say it is linearly separable the condition is only that it should be in the form $y = ax + b$ form is the power of x should be 1 only then we can separate them linearly.

Linear Separability refers to the data points in binary classification problems which can be separated using linear decision boundary. if the data points can be separated using a line, linear function, or flat hyperplane are considered linearly separable.

Linear separability is an important concept in neural networks. If the separate points in n-dimensional space follows then it is said linearly separable

For two-dimensional inputs, if there exists a line (whose equation is) that separates all samples of one class from the other class, then an appropriate perception can be derived from the equation of the separating line. such classification problems are called "Linear separable" i.e, separating by a linear combination of i/p.

The logical AND gate example shown below illustrates a two-dimensional example of a linearly separable problem.

Methods for checking linear separability

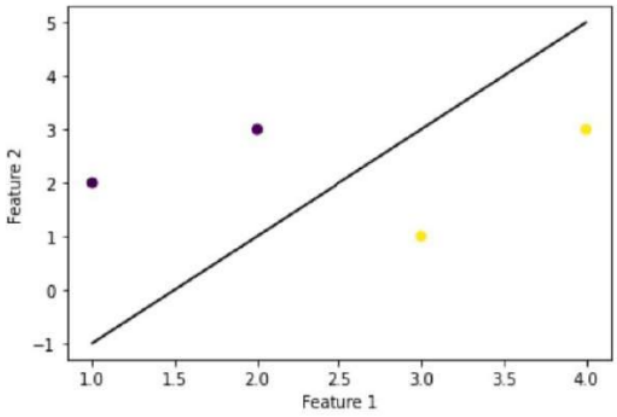
1. **Visual Inspection:** If a distinct straight line or plane divides the various groups, it can be visually examined by plotting the data points in a 2D or 3D space. The data may be linearly separable if such a boundary can be seen.
2. **Perceptron Learning Algorithm:** This binary linear classifier divides the input into two classes by learning a separating hyperplane iteratively. The data are linearly separable if the method finds a separating hyperplane and converges. If not, it is not.
3. **Support vector machines:** SVMs are a well-liked classification technique that can handle data that can be separated linearly. To optimize the margin between the two classes, they identify the separating hyperplane. The data can be linearly separated if the margin is bigger than zero.
4. **Kernel methods:** The data can be transformed into a higher-dimensional space using this family of techniques, where it might then be linearly separable. The original data is also linearly separable if the converted data is linearly separable.
5. **Quadratic programming:** Finding the separation hyperplane that reduces the classification error can be done using quadratic programming. If a solution is found, the data can be separated linearly.

A. Checking Linear separability
B. Import the necessary libraries
C. Define custom dataset
D. Build and train the linear model
E. predict from new input

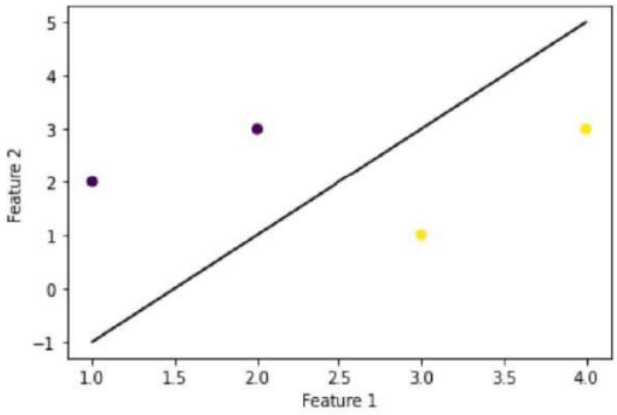
```
fromsklearn importsvm
importnumpy as np
# Making dataset
X =np.array([[1, 2], [2, 3], [3, 1], [4, 3]])
Y =np.array([0, 0, 1, 1])
# Now lets train svm model
model =svm.SVC(kernel='linear')
model.fit(X, Y)
# Lets predict for new input
n_data =np.array([[5, 2], [2, 1]])
```

pred =model.predict(n_data)

print(pred)



Output



Convert Non-separable data to separable

Import the necessary libraries

Create the non-linear dataset

plot the dataset using matplotlib

fromsklearn.datasets importmake_circles

fromsklearn.svm importSVC

importmatplotlib.pyplot as plt

importnumpy as np

first lets create non-linear dataset

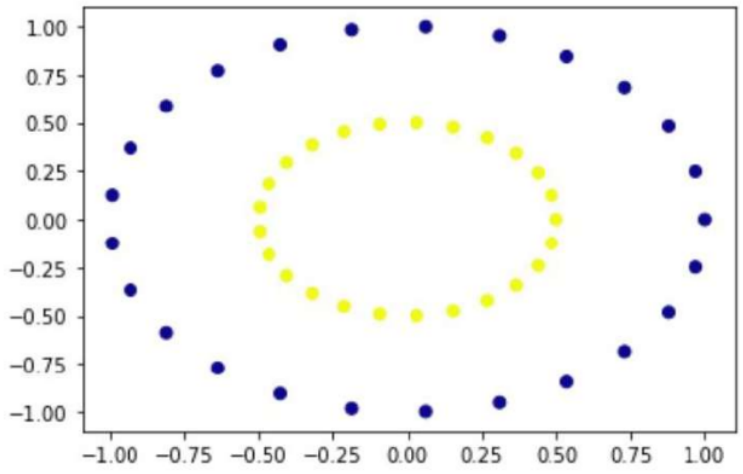
x_val, y_val =make_circles(n_samples=50, factor=0.5)

Now lets plot and see our dataset

plt.scatter(x_val[:, 0], x_val[:, 1], c=y_val, cmap='plasma')

plt.show()

Output :



Apply kernel trick to map data into higher-dimensional space

apply kernel trick to map data into higher-dimensional space.

Now fit SVM on mapped data

Plot decision boundary in mapped space

plot mapped data

apply kernel trick to map data into higher-dimensional space

x_new =np.vstack((x_val[:, 0]**2, x_val[:, 1]**2)).T

Now fit SVM on mapped data

svm =SVC(kernel='linear')

svm.fit(x_new, y_val)

plot decision boundary in mapped space

w =svm.coef_

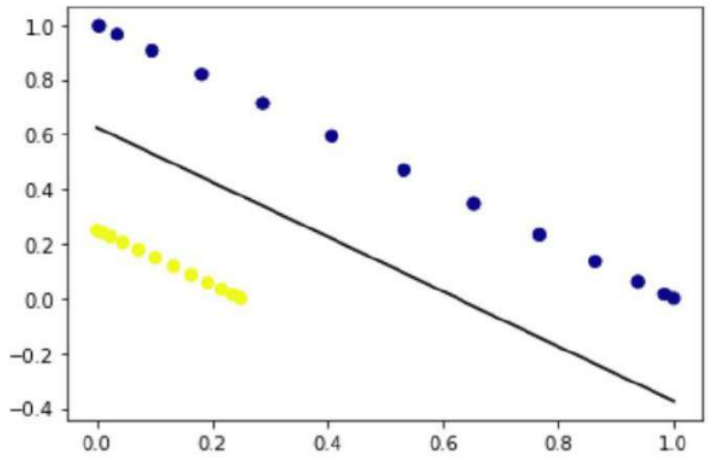
a =-w[0][0] /w[0][1]

x =np.linspace(0, 1)

y =a *x -(svm.intercept_[0]) /w[0][1]

```
plt.plot(x, y, 'k-')
# plot mapped data
plt.scatter(x_new[:, 0], x_new[:, 1], c=y_val, cmap='plasma')
plt.show()
```

Output:



Least Square Method

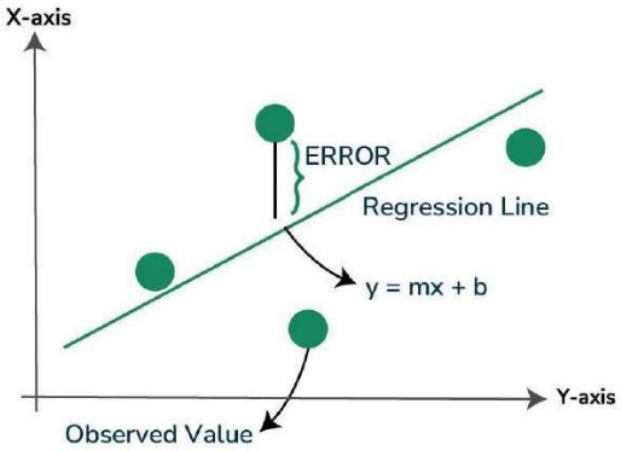
Least Square method is a fundamental mathematical technique widely used in data analysis, statistics, and regression modeling to identify the best-fitting curve or line for a given set of data points. This method ensures that the overall error is reduced, providing a highly accurate model for predicting future data trends.

In statistics, when the data can be represented on a cartesian plane by using the independent and dependent variable as the x and y coordinates, it is called scatter data. This data might not be useful in making interpretations or predicting the values of the dependent variable for the independent variable. So, we try to get an equation of a line that fits best to the given data points with the help of the Least Square Method.

In this article, we will learn the least square method, its formula, graph, and solved examples on it.

Least Square Method Definition

Least Squares method is a statistical technique used to find the equation of best-fitting curve or line to a set of data points by minimizing the sum of the squared differences between the observed values and the values predicted by the model.



Formula for Least Square Method

Least Square Method formula is used to find the best-fitting line through a set of data points. For a simple linear regression, which is a line of the form y=mx+c, where y is the dependent variable, x is the independent variable, a is the slope of the line, and b is the y-intercept, the formulas to calculate the slope (m) and intercept (c) of the line are derived from the following equations:

1.**Slope (m) Formula:** $m = n(\sum xy) - (\sum x)(\sum y) / n(\sum x2) - (\sum x)2$

2.**Intercept (c) Formula:** $c = (\sum y) - a(\sum x) / n$

Where:

n is the number of data points,

$\sum xy$ is the sum of the product of each pair of x and y values,

$\sum x$ is the sum of all x values,

$\sum y$ is the sum of all y values,

$\sum x2$ is the sum of the squares of x values.

The steps to find the line of best fit by using the least square method is discussed below:

Step 1: Denote the independent variable values as xi and the dependent ones as yi.

Step 2: Calculate the average values of xi and yi as X and Y.

Step 3: Presume the equation of the line of best fit as $y = mx + c$, where m is the slope of the line and c represents the intercept of the line on the Y-axis.

Step 4: The slope m can be calculated from the following formula:

$m = [\Sigma (X - xi) \times (Y - yi)] / \Sigma(X - xi)2$

Step 5: The intercept c is calculated from the following formula:$c = Y - mX$

Limitations of the Least Square Method

The Least Square method assumes that the data is evenly distributed and doesn't contain any outliers for deriving a line of best fit. But, this method doesn't provide accurate results for unevenly distributed data or for data containing outliers.

The Perceptron Artificial Neural Networks, the Perceptron is one of the simplest artificial neural network architectures, introduced by Frank Rosenblatt in 1957. It is primarily used for binary classification.

At that time, traditional methods like Statistical Machine Learning and Conventional Programming were commonly used for predictions. Despite being one of the simplest forms of artificial neural networks, the Perceptron model proved to be highly effective in solving specific classification problems, laying the groundwork for advancements in AI and machine learning.

Perceptron

Perceptron is a type of neural network that performs binary classification that maps input features to an output decision, usually classifying data into one of two categories, such as 0 or 1.

Perceptron consists of a single layer of input nodes that are fully connected to a layer of output nodes. It is particularly good at learning linearly separable patterns. It utilizes a variation of artificial neurons called Threshold Logic Units (TLU), which were first introduced by McCulloch and Walter Pitts in the 1940s. This foundational model has played a crucial role in the development of more advanced neural networks and machine learning algorithms.

Types of Perceptron

Single-Layer Perceptron is a type of perceptron is limited to learning linearly separable patterns. It is effective for tasks where the data can be divided into distinct categories through a straight line. While powerful in its simplicity, it struggles with more complex problems where the relationship between inputs and outputs is non-linear.

Multi-Layer Perceptron possess enhanced processing capabilities as they consist of two or more layers, adept at handling more complex patterns and relationships within the data.

Components of Perceptron

A Perceptron is composed of key components that work together to process information and make predictions.

Input Features: The perceptron takes multiple input features, each representing a characteristic of the input data.

Weights: Each input feature is assigned a weight that determines its influence on the output. These weights are adjusted during training to find the optimal values.

Summation Function: The perceptron calculates the weighted sum of its inputs, combining them with their respective weights.

Activation Function: The weighted sum is passed through the Heaviside step function, comparing it to a threshold to produce a binary output (0 or 1).

Output: The final output is determined by the activation function, often used for binary classification tasks.

Bias: The bias term helps the perceptron make adjustments independent of the input, improving its flexibility in learning.

Learning Algorithm: The perceptron adjusts its weights and bias using a learning algorithm, such as the Perceptron Learning Rule, to minimize prediction errors.

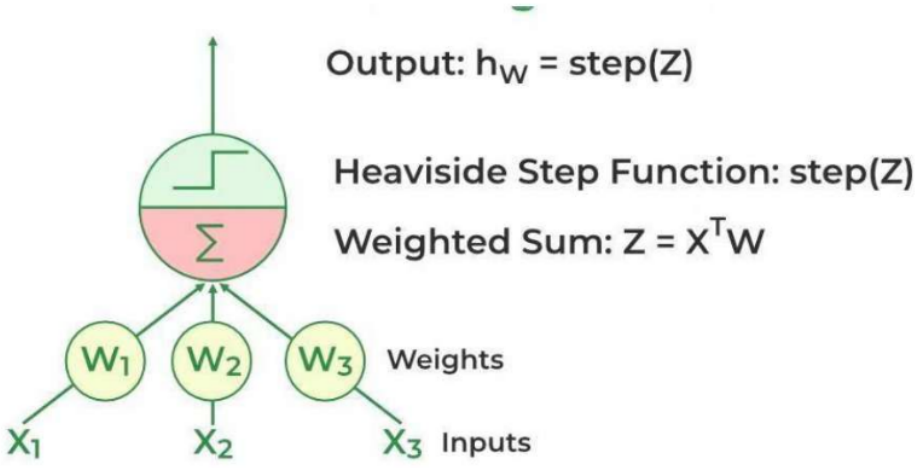
Perceptron working process

A weight is assigned to each input node of a perceptron, indicating the importance of that input in determining the output. The Perceptrons output is calculated as a weighted sum of the inputs, which is then passed through an activation function to decide whether the Perceptron will fire.

The weighted sum is computed as:

$z=w1x1+w2x2+\ldots+wnxn=XTWz=w1 \quad x1 \quad +w2 \quad x2 \quad +\ldots+wn \quad xn \quad =XTW$

$h(z)=\{01 \quad \text{if } z<\text{Threshold if } z\geq\text{Threshold}$



A perceptron consists of a single layer of Threshold Logic Units (TLU), with each TLU fully connected to all input nodes.

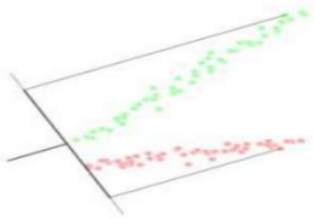
In a fully connected layer, also known as a dense layer, all neurons in one layer are connected to every neuron in the previous layer.

The output of the fully connected layer is computed as:

$fW,b(X)=h(XW+b)fW,b \quad (X)=h(XW+b)$

where XX is the input WW is the weight for each inputs neurons and bb is the bias and hh is the step function.

Fisher's linear discriminant



The terms Fisher's linear discriminant and LDA are often used interchangeably, although Fisher's original article actually describes a slightly different discriminant, which does not make some of the assumptions of LDA such as normally distributed classes or equal class co-variances.

This measure is, in some sense, a measure of the signal-to-noise ratio for the class labeling. It can be shown that the maximum separation occurs when

Be sure to note that the vector $\mathbf{w}\rightarrow$ is the normal to the discriminant hyperplane. As an example, in a two dimensional problem, the line that best divides the two groups is perpendicular to $\mathbf{w}\rightarrow$.

Otsu's method is related to Fisher's linear discriminant, and was created to binarize the histogram of pixels in a grayscale image by optimally picking the black/white threshold that minimizes intra-class variance and maximizes inter-class variance within/between gray scales assigned to black and white pixel classes.

Gradient-Based Strategy

Gradient-based strategy is commonly known as Gradient boosting which is a fundamental machine learning technique used by many gradient boosting algorithms like Light GBM to optimize and enhance the performance of predictive models. In a gradient-based strategy, multiple weak learners(commonly Decision trees) are combined to achieve a high-performance model. There are some key processes associated with a gradient-based strategy which are listed below:

Gradient Descent: In the gradient-based strategy, the optimization algorithm (usually gradient descent) is used to minimize a loss function that measures the difference between predicted values and actual target values.

Iterative Learning: The model iteratively updates its predictions for each step by calculating gradients (slopes) of the loss function with respect to the model's parameters. These gradients are calculated to know the right way to minimize the loss.
Boosting: In gradient boosting, weak learners (decision trees) are trained sequentially where each tree attempting to correct the errors made by the previous ones and the final prediction is the combination of predictions from all the trees.

Benefits of Gradient-based strategy

Utilizing a gradient-based method in our predictive model can yield various advantages, as enumerated below:

Model Accuracy: Gradient boosting, particularly Light GBM, is recognized for its superior prediction accuracy, adept at capturing intricate correlations within the data through iterative model refinement.

Robustness: The ensemble characteristic of gradient boosting renders it resilient to the issue of over fitting. Each subsequent tree addresses the errors of its predecessors, so decreasing the possibility of incorporating noise from the data.

Flexibility: Gradient boosting possesses an inherent mechanism to accommodate diverse data sources, including both numerical and categorical features, rendering it appropriate for a broad spectrum of machine learning tasks.

Interpretability: Although ensemble models may exhibit complexity, they can provide interpretability via feature importance rankings, which can be utilized alongside interpretability techniques such as SHAP values to elucidate model decisions.

Learning Rate Decay

Learning rate decay is a technique used in machine learning models, especially deep neural networks. It is sometimes referred to as learning rate scheduling or learning rate annealing. Throughout the training phase, it entails gradually lowering the learning rate. Learning rate decay is used to gradually adjust the learning rate, usually by lowering it, to facilitate the optimization algorithm's more rapid convergence to a better solution. This method tackles problems that are frequently linked to a fixed learning rate, such as oscillations and sluggish convergence.

Learning rate decay can be accomplished by a variety of techniques, such as step decay, exponential decay, and 1/t decay. Degradation strategy selection is based on the particular challenge and architecture. When training deep learning models, learning rate decay is a crucial hyper parameter that, when used properly, can result in faster training, better convergence, and increased model performance.

Learning Rate Decay working process

Learning rate decay is like driving a car towards a parking spot. At first, you drive fast to reach the spot quickly. As you get closer, you slow down to park accurately. In machine learning, the learning rate determines how much the model changes based on the mistakes it makes. If it's too high, the model might miss the best fit; too low, and it's too slow. Learning rate decay starts with a higher learning rate, letting the model learn fast. As training progresses, the rate gradually decreases, making the model adjustments more precise. This ensures the model finds a good solution efficiently. Different methods reduce the rate in various ways, either stepwise or smoothly, to optimize the training process.

Mathematical representation of Learning rate decay

A basic learning rate decay plan can be mathematically represented as follows:

Assume that the starting learning rate is \eta_{0} and that the learning rate at epoch t is {t} .

A typical decay schedule for learning rates is based on a constant decay rate alpha , where alpha epsilon (0,1) , applied at regular intervals (e.g., every n epochs):

{t} = {{0}}{1 + \alpha \c. t}

Where,

{t} is the learning rate at epoch t.

{0} is the initial learning rate at the start of training.

\alpha is the fixed decay rate, typically a small positive value, such as 0.1 or 0.01.

t is the current epoch during training.

The learning rate {t} decreases as t increases, leading to smaller step size as training progresses.

Basic decay schedules

In order to enhance the convergence of machine learning models, learning rate decay schedules are utilized to gradually lower the learning rate during training. Here are a few simple schedules for learning rate decay:

Step Decay: In step decay, after a predetermined number of training epochs, the learning rate is decreased by a specified factor (decay rate). The mathematical formula for step decay is lr = lr_{initial} * drop \;rate^{\frac{epoch}{step\;size}}

Exponential Decay: The learning rate is progressively decreased over time by exponential decay. At each epoch, a factor is used to adjust the learning rate. The mathematical formula for Exponential decay is:lr = lr_{initial}\ * \e^{-decay\rate * \}

Inverse Time Decay: A factor inversely proportional to the number of epochs is used to reduce the learning rate through inverse decay.

The mathematical formula for Inverse Time decay is:lr = lr_{initial} \ * \\frac{1}{1 + decay * epoch}

Polynomial Decay: When a polynomial function, usually a power of the epoch number, is followed, polynomial decay lowers the learning rate.The mathematical formula for Polynomial decay is:lr = lr_{initial} \ * \ \left (1 - \frac{epoch}{max\epoch}\right)^{power}.

Steps Needed to implement Learning Rate Decay

Set Initial Learning Rate: Start by establishing a base learning rate. It shouldn't be too high to cause drastic updates, nor too low to stall the learning process.

Choose a Decay Method: Common methods include exponential decay, step decay, or inverse time decay. The choice depends on your specific machine learning problem.

Implement the Decay: Apply the chosen decay method after a set number of epochs, or based on the performance of the model.

Monitor and Adjust: Keep an eye on the model's performance. If it's not improving, you might need to adjust the decay rate or the method.

Momentum-based Gradient Optimizer

Gradient Descent is an optimization method employed in Machine Learning frameworks to train various models. The training process involves an objective function, also known as the error function, which quantifies the error of a Machine Learning model on a specific dataset. During training, the parameters of this method are initialized to arbitrary values. As the algorithm progresses, the parameters are adjusted to approach the ideal value of the function more closely.

Nonetheless, Adaptive Optimization Algorithms are becoming increasingly popular owing to their rapid convergence capability. All these techniques, unlike traditional Gradient Descent, utilize statistics from prior iterations to enhance the convergence process.

The Momentum-based Gradient Optimizer is a method employed in optimization techniques, including Gradient Descent, to enhance convergence speed and circumvent local minima. In the Momentum-based Gradient Optimizer, a portion of the preceding update is incorporated into the current update, generating a momentum effect that facilitates the algorithm's accelerated convergence towards the minimum.

The momentum term can be regarded as a moving average of the gradients. A bigger momentum term results in a smoother moving average and enhances resistance to gradient fluctuations. The momentum parameter is generally assigned a value ranging from 0 to 1, with elevated values yielding a more stable optimization procedure.

The update rule for the Momentum-based Gradient Optimizer can be expressed as follows:

makefilev = beta * v - learning_rate * gradientparameters = parameters + v

Where v is the velocity vector, beta is the momentum term, learning_rate is the step size, gradient is the gradient of the cost function with respect to the parameters, and parameters are the parameters of the model.

Momentum-based Optimization

An Adaptive Optimization Algorithm uses exponentially weighted averages of gradients over previous iterations to stabilize the convergence, resulting in quicker optimization. For example, in most real-world applications of Deep Neural Networks, the training is carried out on noisy data. It is, therefore, necessary to reduce the effect of noise when the data are fed in batches during Optimization. This problem can be tackled using Exponentially Weighted Averages (or Exponentially Weighted Moving Averages).

Implementing Exponentially Weighted Averages:

In order to approximate the trends in a noisy dataset of size N: θ0,θ1,θ2,…,θN θ0 ,θ1 ,θ2 ,…,θN , we maintain a set of parameters v0,v1,v2,v3,…,vN v0 ,v1 ,v2 ,v3 ,…,vN . As we iterate through all the values in the dataset, we calculate the parameters below:

On iteration t: Get next θtθt t:vθ=βvθ+(1−β)θtt:vθ =βvθ +(1−β)θt

This algorithm averages the value of vθ over its values from previous 11–β 1–β1 iterations. This averaging ensures that only the trend is retained and the noise is averaged out. This method is used as a strategy in momentum-based gradient descent to make it robust against noise in data samples, resulting in faster training.

Learning Rate in Neural Network

Learning rate is a hyper parameter that controls how much to change the model in response to the estimated error each time the model weights are updated. It determines the size of the steps taken towards a minimum of the loss function during optimization.

In mathematical terms, when using a method like Stochastic Gradient Descent (SGD), the learning rate (often denoted as α or η) is multiplied by the gradient of the loss function to update the weights:

$w=w–\alpha\cdot \nabla L(w)w=w–\alpha\cdot \nabla L(w)$

Where:

w represents the weights,

α is the learning rate,

$\nabla L(w)\nabla L(w)$ is the gradient of the loss function concerning the weights.

Impact of Learning Rate on Model

The learning rate influences the training process of a machine learning model by controlling how much the weights are updated during training. A well-calibrated learning rate balances convergence speed and solution quality.

If set too low, the model converges slowly, requiring many epochs and leading to inefficient resource use. Conversely, a high learning rate can cause the model to overshoot optimal weights, resulting in instability and divergence of the loss function. An optimal learning rate should be low enough for accurate convergence while high enough for reasonable training time. Smaller rates require more epochs, potentially yielding better final weights, whereas larger rates can cause fluctuations around the optimal solution.

Imagine learning to play a video game where timing your jumps over obstacles is crucial. Jumping too early or late leads to failure, but small adjustments

can help you find the right timing to succeed. In machine learning, a low learning rate results in longer training times and higher costs, while a high learning rate can cause overshooting or failure to converge. Thus, finding the optimal learning rate is essential for efficient and effective training.

Techniques for Adjusting the Learning Rate in Neural Networks

1. Fixed Learning Rate

A fixed learning rate is a common optimization approach where a constant learning rate is selected and maintained throughout the training process. Initially, parameters are assigned random values, and a cost function is generated based on these initial values. The algorithm then iteratively improves the parameter estimations to minimize the cost function. While simple to implement, a fixed learning rate may not adapt well to the complexities of various training scenarios.

2. Learning Rate Schedules

Learning rate schedules adjust the learning rate based on predefined rules or functions, enhancing convergence and performance. Some common methods include:

Step Decay: The learning rate decreases by a specific factor at designated epochs or after a fixed number of iterations.

Exponential Decay: The learning rate is reduced exponentially over time, allowing for a rapid decrease in the initial phases of training.

Polynomial Decay: The learning rate decreases polynomially over time, providing a smoother reduction.

3. Adaptive Learning Rate

Adaptive learning rates dynamically adjust the learning rate based on the model's performance and the gradient of the cost function. This approach can lead to optimal results by adapting the learning rate depending on the steepness of the cost function curve.

AdaGrad: This method adjusts the learning rate for each parameter individually based on historical gradient information, reducing the learning rate for frequently updated parameters.

RMSprop: A variation of AdaGrad, RMSprop addresses overly aggressive learning rate decay by maintaining a moving average of squared gradients to adapt the learning rate effectively.

Adam: Combining concepts from both AdaGrad and RMSprop, Adam incorporates adaptive learning rates and momentum to accelerate convergence.

4. Scheduled Drop Learning Rate

In this technique, the learning rate is decreased by a specified proportion at set intervals, contrasting with decay techniques where the learning rate continuously diminishes. This allows for more controlled adjustments during training.

5. Cycling Learning Rate

Cycling learning rate techniques involve cyclically varying the learning rate within a predefined range throughout the training process. The learning rate fluctuates in a triangular shape between minimum and maximum values, maintaining a constant frequency. One popular strategy is the triangular learning rate policy, where the learning rate is linearly increased and then decreased within a cycle. This method aims to explore various learning rates during training, helping the model escape poor local minima and speeding up convergence.

6. Decaying Learning Rate

In this approach, the learning rate decreases as the number of epochs or iterations increases. This gradual reduction helps stabilize the training process as the model converges to a minimum.

Cliffs and Higher-Order Instability

Cliffs and Exploding Gradients Neural networks with multiple layers frequently exhibit pronounced steep sections akin to cliffs. This result arises from the simultaneous growth of specific substantial weights. The gradient information step may significantly alter the parameters on a steep cliff construction.It typically involves leaping from the cliff structure as a whole.

The cliff poses a risk regardless of whether we approach it from above or below. However, its most significant values can be circumvented through the application of the gradient clipping heuristic. The fundamental concept is to

remember that the gradient does not necessitate the optimal step size. The gradient clipping heuristic is employed to reduce the step size to a sufficiently small magnitude. It is less feasible to exit the zone where the gradient indicates the direction of greatest descent.

Cliff structures frequently occur in the cost functions of recurrent neural networks, as these models entail the multiplication of numerous factors, with one factor corresponding to each time step. Prolonged temporal sequences consequently require a substantial amount of multiplication.

Identification and catching of Exploding Gradients

The proof of identity of these gradient problems is difficult to understand before the training process is even started. We have to continually monitor the logs and record unexpected jumps in the cost function when the network is a deep recurrent one.

This would tell us whether these jumps are recurrent. And if the norm of the gradient is growing exponentially. The best way to do this is by checking logs in a visualization dashboard.

Fixation of Exploding Gradients

There are various methods to address the exploding gradients. Below is the list of some best-practice methods that we can use.

Gradient Clipping

Gradient Clipping is the process that helps maintain numerical stability by preventing the gradients from growing too large. When training a neural network, the loss gradients are computed through backpropagation. However, if these gradients become too large, the updates to the model weights can also become excessively large, leading to numerical instability. This can result in the model producing NaN (Not a Number) values or overflow errors, which can be problematic. This problem is often referred to as 'gradient exploding', it could be solved by clipping the gradient to the value that we want it to be. Let's thoroughly discuss gradient clipping.

Gradient Clipping working

Let's discuss the step-by-step description of gradient clipping:

Calculate Gradients: The model's learning process resembles a student undertaking an examination. Backpropagation resembles an educator evaluating an examination and providing comments to the learner. It computes the gradients of the model's parameters in relation to the loss function, facilitating the model's learning and enhancement of its performance. Consider backpropagation as an instructive mentor directing the model towards achievement.

Calculate Gradient Norm: To assess the magnitude of the gradients, various norms can be employed, such the L2 norm (often referred to as the Euclidean norm) or the L1 norm. These norms assist in quantifying the magnitude of the gradients and comprehending the rate of parameter changes. The L2 norm computes the square root of the sum of the squares of the individual gradients, whereas the L1 norm computes the sum of the absolute values of the gradients. By assessing the norm of the gradients, we may oversee the training process and modify the learning rate as necessary to guarantee the model converges well.

Gradient Clipping: When the calculated gradient norm over the specified clipping threshold, the gradients are reduced in magnitude to maintain the norm within this limit. The scaling factor is calculated by dividing the clip threshold by the gradient norm.

clip \factor = \frac{clip_threshold}{gradient_norm}

The clipped gradients become, clip \,factor * gradients.

Update Model Parameters: The clipped gradients are used to update the model parameters. By using the clipped gradients to update the model parameters, we can prevent the weights from being updated by excessively large amounts, which can lead to numerical instability and slow down the training process. This helps to ensure that the model is learning effectively and converging towards a good solution.

The clip_threshold discussed here is a type of hyper parameter whose value could be determined by experimenting on the dataset present in front of us.

Types of Gradient Clipping Techniques

There are two different gradient clipping techniques that are used, gradient clipping by value and gradient clipping by norm, let's discuss them thoroughly.

Clipping by Value:

'Clipping by value' is the most straightforward and effective gradient clipping technique, in this method the gradients are individually clipped so that they lie in the predefined range that is mentioned. This technique is done element wise, so each component of the gradient vector is clipped individually. In this gradient clipping technique, the minimum and maximum thresholds are defined, and the range is set accordingly so that the gradient's value lies in between the minimum and maximum value.

After the computation of the gradients through back-propagation, inspection of the gradient component is done, if the gradient component is greater than the maximum threshold it's value is set to maximum threshold and if the gradient component is lower than the minimum threshold value mentioned then the value of the gradient component is set to minimum threshold value and if the value of the gradient component lies in between the range of minimum and maximum threshold value then the gradient component is set as it is and not changed.

Clipping by Norm:

In the 'clipping by norm' technique of gradient clipping the gradients are clipped if their norm (or their size) is greater than the specified threshold value. In contrast to the 'clipping by value' here in this case the values of the gradients greater than or less than the threshold values are not set to the threshold values. It makes sure that the norm of the updated gradients remains small and manageable, and the learning process is more stable. There are different types of 'clipping by norm' techniques let's explore them one by one.

L2 Norm Clipping:In this form of norm clipping technique the gradient value is clipped down if it's L2 norm (Euclidean norm) exceeds the predefined threshold value. The L2 norm or the Euclidean norm is calculated as the square root of the squared values of its components. Considering gradient vector as g = [\nabla\theta_1, \nabla\theta_2, ..., \nabla\theta_n] where g_i is the gradient with respect to the i^{th} parameter of the model and n is the total number of model parameters.

Therefore, the L2 norm is represented as:||\nabla\theta||_2 = \sqrt{\Sigma_{i=1}^n {\nabla{\theta{_i^2}}}}Now, if the L2 norm exceeds the threshold value the upgraded gradient after clipping of the components becomes:\nabla\theta = \frac{threshold} {||\nabla\theta||_2}.\nabla\theta.

L1 Norm Clipping:L1 norm technique of gradient is similar to the L2 norm gradient clipping technique, in this technique if the L1 norm exceeds the threshold that we defined in alignment with our specific requirements. L1 norm of a gradient is the sum of the absolute values of all its components. Therefore, the L1 norm is represented as:||\nabla\theta||_1 = \Sigma_{i=1}^n|\nabla\theta_i|If the L1 norm exceeds the threshold value, the upgraded gradient after clipping becomes:\nabla\theta = \frac{threshold} {||\nabla\theta||_1}.\nabla\theta.
Gradient clipping by norm provides a more global control over the gradients, and it is often used to address the exploding gradient problem.

Necessity of Gradient Clipping

Gradient Clipping is a crucial step in the training of neural networks since it helps in addressing the issue of exploding gradients. The exploding gradients problem arises when the gradients in the backpropagation process becomes excessively large in value, causing instability in training of model. Now we will see some of the most important points which explains the necessity of gradient clipping in neural network model training.

Stability of Training: During the training of neural network, the optimization algorithm adjusts the value of model parameters with the help of obtained gradients. If the gradients are obtained to be too large the weights of the model updates by a large value causing the model to oscillate and diverge instead of converging to an optimal solution. Whereas gradient clipping limits the size of the gradient and eliminating this issue of instability in the model.

Improving Generalization: Large gradients might cause the model to over fit to the training data, which in turn might capture more noise and makes the model bad at generalization. Gradient clipping removes this hindrance and makes the model generalize better on new data preventing extreme updates.

Convergence to Optimal Solution: Exploding gradient prevents the model to converge to optimal solution and instead produces more unstable modelling of data. By clipping the gradient values, we can suspend the possibility of instability, the model gets better at navigating the parameter space enabling consistent progress toward optimal solution.

Compatibility with Activation Function: Some of the activation functions such as 'Sigmoid' and 'tanh' functions are sensitive to large input. Gradient clipping ensures the gradient passed through the activation function is within a reasonable range which also helps in removing undesirable behavior like saturation.

Mitigating Vanishing Gradient Problem: Sometimes the gradient of the loss function with respect to the weights become extremely small which causes the weight to stop updating or even halt the process of training the model. Norm-based gradient clipping helps in preventing the vanishing gradient problem by maintaining the range of value for the gradient that is effective for training the model.

Gradient Clipping in Keras

Keras helps gradient clipping on every optimization algorithm. It supports the similar order applied to all layers in the model. Gradient clipping may be used with an optimization algorithm, for example, stochastic gradient descent, with an extra argument when configuring the optimization algorithm.

We can use two types of gradient clipping.

1. Gradient norm scaling
2. Gradient value clipping

Second Order Derivatives

The Second Order Derivative is defined as the derivative of the first derivative of the given function. The first-order derivative at a given point gives us the information about the slope of the tangent at that point or the instantaneous rate of change of a function at that point.

Second-Order Derivative gives us the idea of the shape of the graph of a given function. The second derivative of a function f(x) is usually denoted as f''(x). It is also denoted by D2y or y2 or y'' if y = f(x).

Let y = f(x)

Then, dy/dx = f'(x)

If f'(x) is differentiable, we may differentiate (1) again w.r.t x. Then, the left-hand side becomes d/dx(dy/dx) which is called the second order derivative of y w.r.t x.

Second Order Derivatives Overview

The second order derivative of a functionf(x)f(x)f(x)f(x)f(x)f(x) is the derivative of its first derivative f′(x)f′(x)f′(x)f′(x)f′(x)f′(x).It measures how the rate of change of f′(x)f′(x)f′(x) itself changes.

Notation Denoted as

f′′(x)f''(x)f′′(x),d2ydx2d2ydx2dx2d2y ,ord2fdx2d2fdx2dx2d2f .f′′(x)f''(x)f′′(x),d2y dx2dx2d2y dx2d2y ,ord2fdx2dx2d2f dx2d2f.

Significance

Concavity: [Tex]f′′(x)>0f''(x) > 0f′′(x)>0[/Tex] indicates the graph of f(x)f(x)f(x)f(x)f(x)f(x) is concave up. [Tex]f′′(x)<0f''(x) < 0f′′(x)<0[/Tex] indicates concave down

Inflection Points: Points where f′′(x)=0f''(x) = 0f′′(x)=0 may indicate a change in concavity, known as inflection points.

Basic Rules

ConstantRule:f(x)=c⇒f′′(x)=0f(x)=c⇒f''(x)=0f(x)=c⇒f′′(x)=0–

PowerRule:f(x)=xn⇒f′′(x)=n(n−1)xn−2f(x)=xn⇒f''(x)=n(n−1)xn−2f(x)=xn⇒f′′(x)=n(n−1)xn−2–

ExponentialRule:f(x)=ex⇒f′′(x)=exf(x)=ex⇒f''(x)=exf(x)=ex⇒f′′(x)=ex–

LogarithmicRule:f(x)=ln⁡(x)⇒f′′(x)=−1x2f(x)=ln⁡(x)⇒f''(x)=−1x2f(x)=ln(x)⇒f′′(x)=−x21

Second Order Derivatives Examples

Example 1: Find d2y/dx2, if y = x3

Given that, y = x3

Then, first derivative will bedy/dx = d/dx (x3) = 3x2

Again, we will differentiate further to find its second derivative,

Therefore, d2y/dx2 = d/dx (dy/dx)

= d/dx (3x2)

= 6x

Example 2: y = logx, Find d2y/dx2?

Solution:

Given that, y = logx

Then first derivative will be,

dy/dx = d/dx (logx)

= (1 / x)

Again, we will further differentiate to find its second derivative,

d2y/dx2 = d/dx (dy/dx)

= d/dx (1 / x) (from first derivative)

= -1 / x2

Polyak Averaging Technique

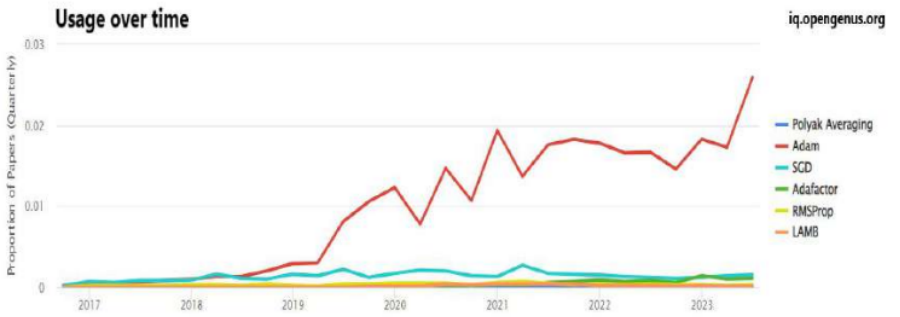
Polyak averaging addresses this issue by maintaining a running average of the model parameters throughout the training process. Instead of using the parameters from the final iteration as the solution, Polyak averaging computes a weighted average of the model parameters obtained during all iterations. This averaging smoothens out the parameter updates and helps in reducing the impact of noise and oscillations, leading to a more stable and accurate final model.

Polyak Averaging provides numerous advantages to optimization algorithms. One crucial benefit is its ability to mitigate over fitting. Over fitting occurs when an algorithm becomes excessively intricate, fitting the noise in the data rather than the actual pattern. By computing the average of recent parameters, Polyak Averaging smooths the algorithm's trajectory, minimizing the risk of over fitting.

Objective Function and Optimization Algorithm

Suppose you have an optimization problem, such as training a machine learning model. In this context, you have a loss function that you want to minimize, typically representing he difference between your model's predictions and the actual target values.

You employ an optimization algorithm, like stochastic gradient descent (SGD), Adam, or RMSprop, to update the model's parameters iteratively.



Noisy or Fluctuating Objective Function

In practical situations, the objective function may lack smoothness and display variations due to causes like as noisy data, unpredictability from batch sampling, or intrinsic randomness in the problem.

Optimization algorithms may occasionally become ensnared in local minima or display oscillatory behavior when addressing non-smooth and variable functions.

Parameter Averaging:

Instead of considering only the final set of parameters obtained after a fixed number of optimization iterations, Polyak averaging introduces the concept of maintaining two sets of parameters:

Current Parameters: These are the parameters that the optimization algorithm actively updates during each iteration.

Averaged Parameters: These are the parameters obtained by averaging the current parameters over multiple iterations.

Advantages of Polyak Averaging:

1. **Improved Generalization:** Averaging the parameters helps in finding a solution that generalizes better to unseen data. It reduces the risk of over fitting by producing a smoother and more stable model.
2. **Reduced Sensitivity to Hyper parameters:** Polyak averaging can make the model less sensitive to the learning rate and other hyper parameters, as the averaging process mitigates the impact of abrupt parameter changes.
3. **Enhanced Robustness:** By reducing the impact of noisy updates, Polyak averaging makes the training process more robust, especially in scenarios where the data is noisy or the optimization landscape is complex.

Applications of Polyak Averaging

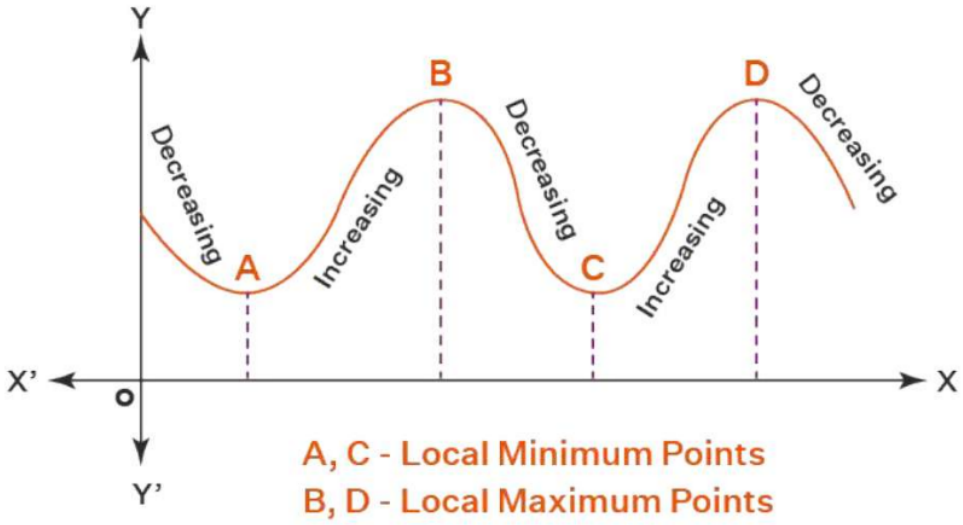
Polyak averaging is particularly useful in deep learning, where finding an optimal set of parameters is challenging due to the high dimensionality of the model. By providing a stable and accurate solution, Polyak averaging contributes significantly to the training of deep neural networks.

Local Minima

A Local Minima point is a point on any function where the function attains its minimum value within a certain interval. A point ($x = a$) of a function $f(a)$ is called a Local minimum if the value of $f(a)$ is lesser than or equal to all the values of $f(x)$. Mathematically, $f(a) \leq f(a - h)$ and $f(a) \leq f(a + h)$ where $h > 0$, then a is called the Local minimum point.

Definition of Local Maxima and Local Minima

Local Maxima and Minima are the initial values of any function to get an idea about its boundaries such as the highest and lowest output values. Local Minima and Local Maxima are also called Local Extrema.



Local Maxima

A Local Maxima point is a place on a function where the function reaches its greatest value within a specific interval. A point (x = a) of a function f(a) is termed a Local maximum if the value of f(a) is larger than or equal to all values of f(x).

Terminology Pertaining to Local Maxima and Local Minima

Key vocabulary pertaining to Local Maxima and Minima is elucidated below:

Optimal Value

If any function yields the maximum output value for the input value of x. The value of x is referred to as the greatest value. If it is delineated within a particular range. That point is referred to as a Local Maximum.

Utmost Maximum

If a function yields the maximum output value for the input value of x across the full domain of the function. The value of x is referred to as the Absolute Maximum.

Minimal Value

If any function yields the minimal output value for the input value of x. The value of x is referred to as the minimum value. If it is delineated within a particular range. That point is referred to as a Local Minimum.

Absolute Minimum

If a function yields the minimum output value for the input value of x across the whole domain of the function. The value of x is referred to as the Absolute Minimum.

Inversion Point

If the value of x within the specified function does not yield the maximum or minimum output, it is referred to as the Point of Inversion.

Characteristics of Local Maxima and Minima

Comprehending the characteristics of local maxima and minima facilitates their recognition:

If a function f(x) is continuous within its domain, it must possess at least one extremum, either maximum or minimum, between any two sites where the function values are identical.

Local maxima and minima alternate; between two minima, there exists a maximum, and conversely.

If f(x) tends to infinity as x approaches the interval's endpoints and possesses a singular critical point within the interval, that critical point constitutes an extremum.

Illustrated Instances of Local Maxima and Local Minima

Example 1: Examine the Local Maxima and Local Minima of the function $f(x) = 2x^3 – 3x^2 – 12x + 5$ utilizing the first derivative technique.

Resolution:

The provided function is $f(x) = 2x^3 - 3x^2 - 12x + 5$.

The first derivative of the function is $f'(x) = 6x^2 – 6x – 12$, which will be utilized to determine the crucial locations.

To determine the critical point, set f'(x) equal to zero.

$6x^2 - 6x - 12 = 0$, $6(x^2 - x - 2) = 0$, $6(x + 1)(x - 2) = 0$

Therefore, the critical points are $x = -1$ and $x = 2$.

Examine the first derivative at the critical point $x = -1$.

The coordinates are $\{-2, 0\}$.

$f'(-2) = 6(4 + 2 - 2) = 6(4) = 24$ and $f'(0) = 6(0 + 0 - 2) = 6(-2) = -12$

The derivative is positive to the left of $x = -1$ and negative to the right. Therefore, it indicates that $x = -1$ is the local maximum.

Let us now examine the first derivative at the critical point $x = 2$. The coordinates are $\{1,3\}$. $f'(1) = 6(1 - 1 - 2) = 6(-2) = -12$ and $f'(3) = 6(9 - 3 - 2) = 6(4) = 24$

The derivative is negative to the left of $x = 2$ and positive to the right. Therefore, it signifies that $x = 2$ is the local minimum.