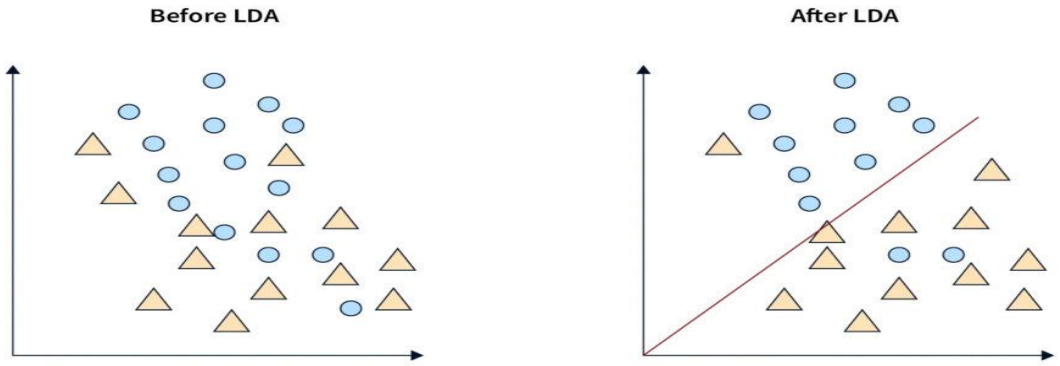
UNIT-4 LINEAR DISCRIMINANTS FOR MACHINE LEARNING

INTRODUCTION TO LINEAR DISCRIMINANTS AND LINEAR DISCRIMINANTS FOR CLASSIFICATION:

Machine learning models are often used to solve supervised learning tasks, particularly classification problems, where the goal is to assign data points to specific categories or classes. However, as datasets grow larger with more features, it becomes challenging for models to process the data effectively. This is where dimensionality reduction techniques like Linear Discriminant Analysis (LDA) come into play.

LDA not only helps to reduce the number of features but also ensures that the important class-related information is retained, making it easier for models to differentiate between classes. This article explores LDA, its working principle, and its practical applications in various fields.



What is Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a supervised learning technique used for classification tasks. It helps distinguish between different classes by projecting data points onto a lower-dimensional space, maximizing the separation between those classes.

LDA performs two key roles:

1. **Classification**: It finds a linear combination of features that best separates multiple classes.
2. **Dimensionality Reduction**: It reduces the number of input features while preserving the information necessary for classification.

For example, in a dataset where each data point belongs to one of three classes, LDA transforms the data into a space where the classes are well-separated, making it easier for models to classify them correctly.

Properties and Assumptions of LDA

To perform effectively, LDA makes several key assumptions about the data:

Key Assumptions:

1. **Gaussian Distribution**: LDA assumes that the features within each class follow a normal (Gaussian) distribution.
2. **Equal Covariance Matrices**: It assumes that the variance (or spread) of data points is the same across all classes.
3. **Linearity**: The relationship between the features and the target variable is assumed to be linear.
4. **Independence**: The features used should ideally be independent of each other.

Limitations Due to These Assumptions:

- **Sensitivity to Data Distribution**: If the features do not follow a Gaussian distribution or if class covariances differ significantly, LDA may not perform well.
- **Impact of Multicollinearity**: If the input features are highly correlated, LDA's performance may be affected.
- **Limited Use with Complex Data**: In datasets with complex, non-linear relationships, LDA may struggle to achieve high classification accuracy.

Preparing to Implement Linear Discriminant Analysis

Before applying LDA, it's essential to prepare the data carefully to ensure accurate results. Below are the key steps involved in data preparation.

Steps for Data Preparation:

1. **Data Cleaning**: Remove any missing values or incorrect data points that could affect the results.
2. **Feature Selection**: Choose features that are relevant to the classification task to avoid unnecessary complexity.
3. **Handling Multicollinearity**: If features are highly correlated, either remove or combine them to improve LDA's performance.
4. **Feature Scaling**: Although LDA is less sensitive to scaling than some other methods, applying standardization (mean=0, variance=1) can help in cases where the ranges of features vary significantly.

Carefully preparing the data ensures that the LDA model can accurately capture the relationships between the features and class labels.

How Does LDA Work?

The core idea of **Linear Discriminant Analysis (LDA)** is to find a new axis that best separates different classes by maximizing the distance between them. LDA achieves this by reducing the dimensionality of the data while retaining the class-discriminative information.

Key Concepts:

1. **Maximizing Between-Class Variance**: LDA maximizes the separation between the means of different classes.
2. **Minimizing Within-Class Variance**: It minimizes the spread (variance) within each class, ensuring that data points from the same class remain close together.
3. **Projection to Lower-Dimensional Space**: LDA projects data onto a new axis or subspace that best separates the classes. For example, in a 3-class problem, LDA can reduce the dimensionality to 2 or even 1 while preserving class-related information.

Working Mechanism:

- **Step 1**: Compute the mean vectors for each class.
- **Step 2**: Calculate the within-class and between-class scatter matrices.
- **Step 3**: Solve for the linear discriminants that maximize the ratio of between-class variance to within-class variance.
- **Step 4**: Project the data onto the new lower-dimensional space.

Mathematical Intuition Behind LDA:

LDA works by finding a new axis that best separates the classes by maximizing the **between-class variance** while minimizing the **within-class variance**. Below is a simplified breakdown of the core concepts:

1. Compute the Mean Vectors:

Calculate the mean vector for each class and the overall mean of the entire dataset.

where is the number of samples in class .

2. Calculate Scatter Matrices:

Within-Class Scatter Matrix (): Measures how far data points in the same class deviate from their class mean.

Between-Class Scatter Matrix (): Measures how far each class mean is from the overall mean.

where is the overall mean of the data.

3. Objective of LDA:

LDA tries to **maximize the ratio** of between-class scatter to within-class scatter to achieve maximum class separation. This is represented mathematically as:

Here, is the projection vector that defines the new axis onto which the data points are projected.

4. Solve for the Optimal Projection Vector:

The optimal projection vector is found by solving the **generalized eigenvalue problem** for the matrix

5. Project Data onto the New Axis:

Once the projection vector is found, the data points are projected onto this new axis using:

This transformation reduces the dimensionality of the data while maintaining the class-related information.

Python Code Implementation of LDA

Below is a simple example of how to implement **Linear Discriminant Analysis (LDA)** using Python and the scikit-learn library. This example demonstrates the key steps: training the LDA model and using it for classification.

Code Example:

```
# Step 1: Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.metrics import accuracy_score

# Step 2: Load the dataset (Iris dataset)

data = load_iris()

X = data.data  # Features
```

```
y = data.target  # Target labels (Classes)

# Step 3: Split the data into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Create and train the LDA model

lda = LinearDiscriminantAnalysis()

lda.fit(X_train, y_train)

# Step 5: Make predictions on the test set

y_pred = lda.predict(X_test)

# Step 6: Evaluate the model's performance

accuracy = accuracy_score(y_test, y_pred)

print (f'LDA Model Accuracy: {accuracy * 100:.2f} %')
```

Explanation:

1. **Loading the Dataset**: We use the popular Iris dataset, which contains data about flowers belonging to three classes.
2. **Training and Testing Split**: The data is divided into training and testing sets to evaluate the model.
3. **Training the LDA Model**: The Linear Discriminant Analysis () model from scikit-learn is trained using the training data.
4. **Making Predictions**: The model predicts class labels for the test set.
5. **Evaluating Accuracy**: The accuracy of the model is calculated and printed.

Advantages & Disadvantages of Using LDA

While **Linear Discriminant Analysis (LDA)** is a powerful tool for classification and dimensionality reduction, it has its pros and cons.

Advantages:

1. **Simplicity**: LDA is easy to implement and understand, making it suitable for beginners.
2. **Interpretability**: It provides clear insight into how features contribute to the classification task.

3. **Computational Efficiency**: LDA is computationally less intensive, making it useful for large datasets.
4. **Works Well with Linearly Separable Data**: It performs effectively when the classes are linearly separable.

Disadvantages:

1. **Sensitive to Assumptions**: LDA assumes that features follow a Gaussian distribution, which may not always hold.
2. **Struggles with Non-linear Relationships**: It may not perform well if the data contains complex, non-linear relationships.
3. **Affected by Class Imbalance**: LDA can be biased toward the majority class if the class distribution is imbalanced.
4. **Impact of Outliers**: It is sensitive to outliers, which can affect the model's performance.

Applications of Linear Discriminant Analysis (LDA):

1. Face Recognition
2. Disease Diagnosis in Healthcare
3. Customer Identification in Marketing
4. Credit Risk Assessment in Finance
5. Quality Control in Manufacturing
6. Campaign Optimization in Marketing

PERCEPTRON CLASSIFIER:

Perceptron model. Developed in the late 1950s by Frank Rosenblatt, The Perceptron model is a type of artificial neuron that functions as a **linear binary classifier**. Its purpose is to classify data points into one of two categories by learning a decision boundary from labelled training data. The model is trained using supervised learning, where the inputs and their corresponding outputs are provided. The Perceptron model calculates the weighted sum of input values, applies an activation function, and produces a binary output. If the output meets or exceeds a specific threshold, it is classified into one category; otherwise, it falls into the other category. This process makes the Perceptron particularly useful in tasks such as **binary classification** in datasets like spam detection or sentiment analysis. Although it is limited to linear separability, the Perceptron remains a fundamental algorithm for understanding neural networks.

Basic Components of a Perceptron

The Perceptron consists of several key components that work together to process input data and generate an output. The main components include:

- **Input layer:** The input layer consists of feature values or data points that the Perceptron will classify. Each input is assigned a corresponding weight.
- **Weights:** Weights determine the importance of each input in the classification process. The model adjusts these weights during training to improve accuracy.
- **Bias:** The bias term helps the Perceptron shift the decision boundary, improving flexibility in data classification.
- **Activation function:** The activation function, such as a step function or sigmoid function, determines the output based on the weighted sum of the inputs.
- **Output:** The output is the final result of the Perceptron's decision, typically a binary value (0 or 1) in the case of binary classification.

Together, these components allow the Perceptron to learn from data, adjust its parameters, and generate predictions.

How Does the Perceptron Works

The Perceptron model operates in a step-by-step process that involves computing the weighted sum of inputs, applying an activation function, and classifying the output. This process allows the model to differentiate between two classes based on the input data.

1. **Initialize Weights and Bias**: The model begins by assigning random weights to the inputs and setting a bias value.
2. **Compute Weighted Sum**: For each input, the Perceptron computes the weighted sum of the input values and adds the bias term. Mathematically, this can be represented as:

$$Z = W_1X_1 + W_2X_2 + \cdots + W_nX_n + b$$

   where W represents the weights, X represents the inputs, and b is the bias.
3. **Apply Activation Function**: The weighted sum is then passed through an activation function. If the result meets a certain threshold, the output is classified as 1; otherwise, it is classified as 0. The step function is a common activation function used for this purpose:

$$f(Z) = \begin{cases} 1, & \text{if } Z \geq 0 \\ 0, & \text{if } Z < 0 \end{cases}$$

4. **Update Weights**: During training, the model compares the predicted output to the actual output and adjusts the weights using a learning rule (typically gradient descent) to reduce the error.
5. **Repeat**: This process is repeated for multiple iterations until the model's predictions align with the actual outputs.

Types of Perceptron Models

1. **Single-Layer Perceptron Model**
   The single-layer Perceptron is the most basic form of the Perceptron algorithm. It consists of only one layer of output neurons that are directly connected to input features. This model can only handle **linearly separable data**, meaning it can draw a straight line to classify data points into two categories. The mathematical formulation for the single-layer Perceptron involves computing the weighted sum of inputs and applying an activation function to determine the output. However, it cannot solve more complex problems that involve non-linear data, such as the **XOR problem**.

2. **Multi-Layer Perceptron (MLP)**
   The **Multi-Layer Perceptron (MLP)** is an extension of the Perceptron model, introducing one or more hidden layers between the input and output layers. These hidden layers allow MLPs to handle non-linear data by using more complex activation functions like the **sigmoid** or **ReLU** (Rectified Linear Unit).
   The MLP employs **backpropagation**, a learning algorithm that adjusts weights across all layers to minimize prediction errors. MLPs are the foundation for modern deep learning models and are capable of solving more complex classification and regression tasks.
   Perceptron Function
   The mathematical function of the Perceptron can be expressed as:

$$f(x) = \text{activation}(W \cdot X + b)$$

   Where:

   W is the weight vector
   X is the input vector
   b is the bias
   activation is the function that determines the output, such as the step function or sigmoid function

The activation function plays a crucial role in determining the output. For example, the step function outputs either 0 or 1 based on the threshold, while the sigmoid function provides a smoother transition between 0 and 1, making it suitable for more complex tasks.

Characteristics of Perceptron

The Perceptron model possesses several notable characteristics:

a. Simplicity: The model is easy to understand and implement, making it an excellent starting point for learning about neural networks.
b. Efficiency: It is efficient when working with linearly separable data, as it quickly converges to a solution.

Limitations: The Perceptron can only solve problems with linear decision boundaries, making it unsuitable for more complex tasks.

Applications: Perceptrons are used in various fields such as binary classification, image recognition, and speech processing.

♣ Limitations of the Perceptron Model

Despite its foundational role in machine learning, the Perceptron model has some limitations. The most significant limitation is its inability to solve non-linearly separable problems, such as the XOR problem. The model cannot differentiate between classes when the data points cannot be separated by a straight line. Additionally, the Perceptron struggles with more complex patterns and requires additional layers (such as those in MLP) to handle intricate datasets effectively. These limitations prompted the development of more advanced neural network models that can tackle non-linear classification problems.

Perceptron Learning Algorithm:

Perceptron learning algorithm:-

(1) First multiply all input value with corresponding weight values and then add them to determine the weighted sum.

Weighted sum:-

$$\sum w_i * x_i = x_1 * w_1 + x_2 * w_2 + \cdots +$$

Add another essential term called bias 'b' to the weighted sum to improve the model performance.

$$\sum w_i * x_i + b$$

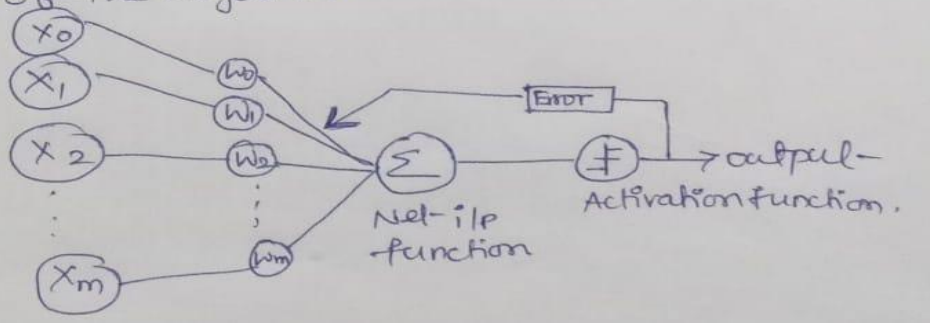
(2) Activation fnt is applied to this weighted sum, producing a binary (or) a continuous-valued output.

$$Y = f(\sum w_i * x_i + b)$$

(3) The difference between this output and the actual target value is computed to get the error term, E, generally interms of mean square error. This form the forward propagation part of the algorithm

$$E = [Y - Y_{actual}]^2$$

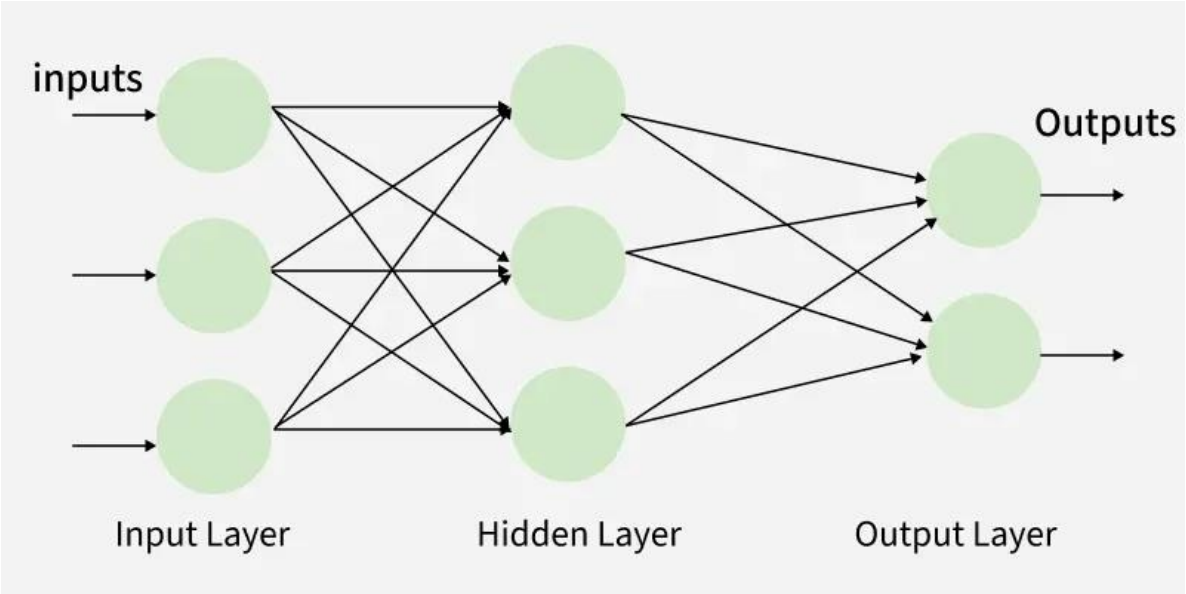
(4) We optimize this error (loss function) using an optimization algorithm. Generally, some form of gradient descent algorithm is used to find the optimal values of the hyperparameters like learning rate, weight, bias etc. This step forms the backward propagation part of the algorithm.



Multi-Layer Perceptron:

Multi-Layer Perceptron (MLP) consists of fully connected dense layers that transform input data from one dimension to another. It is called multi-layer because it contains an input layer, one or more hidden layers and an output layer. The purpose of an MLP is to model complex relationships between inputs and outputs.

Components of Multi-Layer Perceptron (MLP)



- **Input Layer**: Each neuron or node in this layer corresponds to an input feature. For instance, if you have three input features the input layer will have three neurons.
- **Hidden Layers**: MLP can have any number of hidden layers with each layer containing any number of nodes. These layers process the information received from the input layer.
- **Output Layer**: The output layer generates the final prediction or result. If there are multiple outputs, the output layer will have a corresponding number of neurons.

Every connection in the diagram is a representation of the fully connected nature of an MLP. This means that every node in one layer connects to every node in the next layer. As the data moves through the network each layer transforms it until the final output is generated in the output layer.

Working of Multi-Layer Perceptron

Let's see working of the multi-layer perceptron. The key mechanisms such as forward propagation, loss function, backpropagation and optimization.

1. Forward Propagation

In forward propagation the data flows from the input layer to the output layer, passing through any hidden layers. Each neuron in the hidden layers processes the input as follows:

1. Weighted Sum: The neuron computes the weighted sum of the inputs:

$$z = \sum_i \quad w_i x_i + b$$

Where:

- x_i is the input feature.
- w_i is the corresponding weight.
- b is the bias term.

2. Activation Function: The weighted sum z is passed through an activation function to introduce non-linearity. Common activation functions include:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
- **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$
- **Tanh (Hyperbolic Tangent)**: $\tanh(z) = \frac{2}{1+e^{-2z}} - 1$

2. Loss Function

Once the network generates an output the next step is to calculate the loss using a loss function. In supervised learning this compares the predicted output to the actual label.

For a classification problem the commonly used binary cross-entropy loss function is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Where:

- y_i is the actual label.
- $\hat{y}_i$ is the predicted label.
- N is the number of samples.

For regression problems the mean squared error (MSE) is often used:

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

3. Backpropagation

The goal of training an MLP is to minimize the loss function by adjusting the network's weights and biases. This is achieved through backpropagation:

1. **Gradient Calculation**: The gradients of the loss function with respect to each weight and bias are calculated using the chain rule of calculus.
2. **Error Propagation**: The error is propagated back through the network, layer by layer.
3. **Gradient Descent**: The network updates the weights and biases by moving in the opposite direction of the gradient to reduce the loss: $w = w - \eta \cdot \frac{\partial L}{\partial w}$

Where:

- w is the weight.
- η is the learning rate.
- $\frac{\partial L}{\partial w}$ is the gradient of the loss function with respect to the weight.

4. Optimization

MLPs rely on optimization algorithms to iteratively refine the weights and biases during training. Popular optimization methods include:

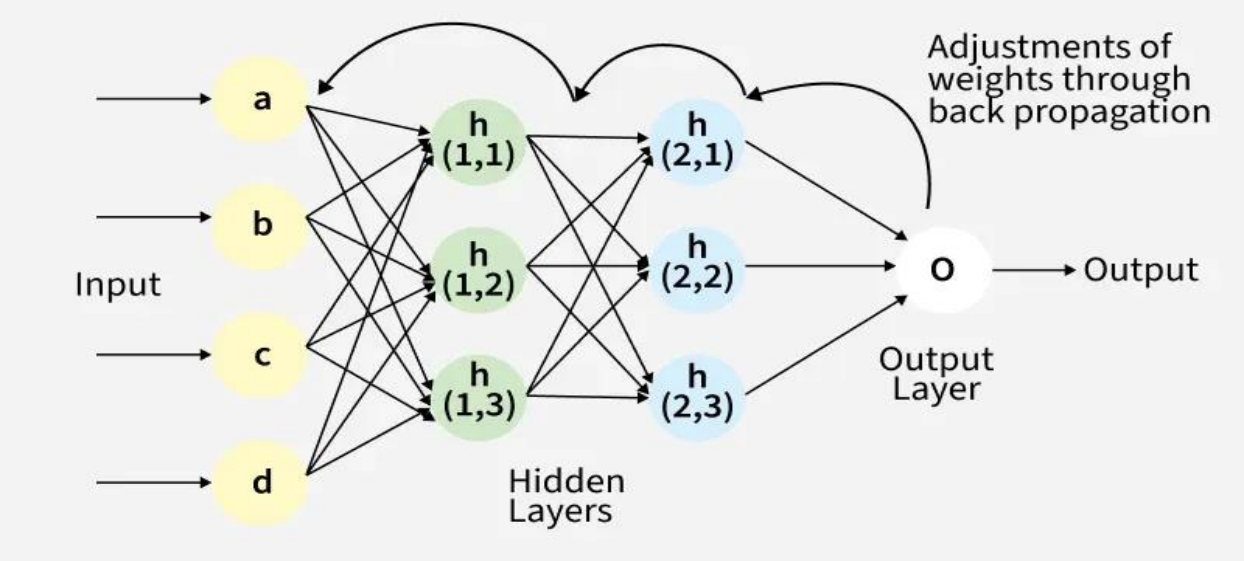
- **Stochastic Gradient Descent (SGD)**: Updates the weights based on a single sample or a small batch of data: $w = w - \eta \cdot \frac{\partial L}{\partial w}$
- **Adam Optimizer**: An extension of SGD that incorporates momentum and adaptive learning rates for more efficient training:
- $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \cdot g_t$
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2) \cdot g_t^2$

Here g_t represents the gradient at time t and β_1, β_2 are decay rates.

Backpropagation in Neural Network:

Back Propagation is also known as "Backward Propagation of Errors" is a method used to train neural network . Its goal is to reduce the difference between the model's predicted output and the actual output by adjusting the weights and biases in the network.

It works iteratively to adjust weights and bias to minimize the cost function. In each epoch the model adapts these parameters by reducing loss by following the error gradient. It often uses optimization algorithms like **gradient descent** or **stochastic gradient descent**. The algorithm computes the gradient using the chain rule from calculus allowing it to effectively navigate complex layers in the neural network to minimize the cost function.



Back Propagation plays a critical role in how neural networks improve over time. Here's why:

1. **Efficient Weight Update**: It computes the gradient of the loss function with respect to each weight using the chain rule making it possible to update weights efficiently.
2. **Scalability**: The Back Propagation algorithm scales well to networks with multiple layers and complex architectures making deep learning feasible.
3. **Automated Learning**: With Back Propagation the learning process becomes automated and the model can adjust itself to optimize its performance.

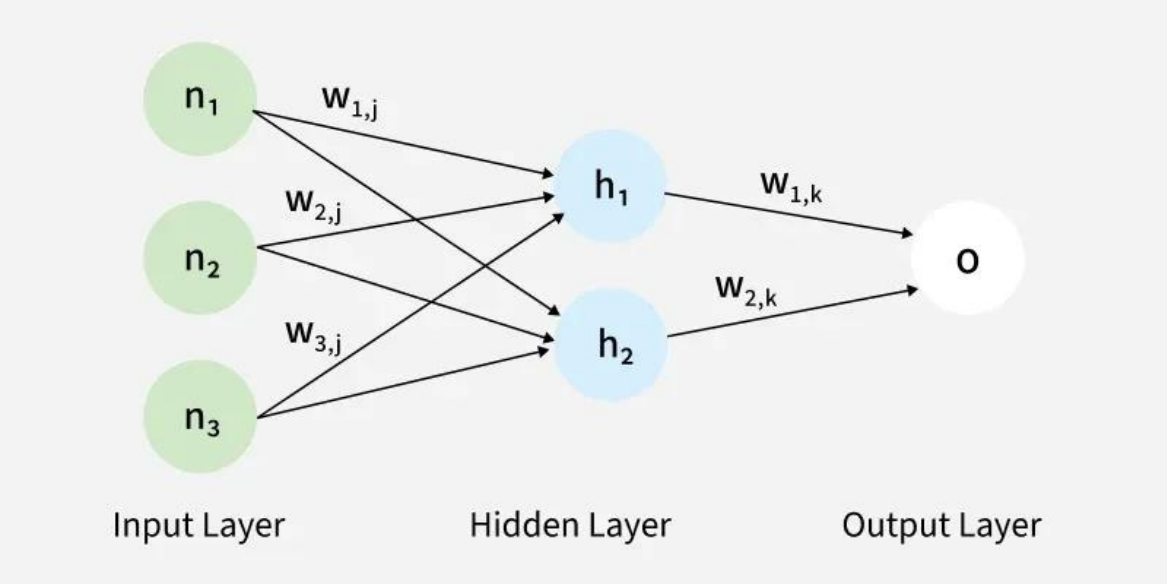
Working of Back Propagation Algorithm

The Back Propagation algorithm involves two main steps: the **Forward Pass** and the **Backward Pass**.

1. Forward Pass Work

In **forward pass** the input data is fed into the input layer. These inputs combined with their respective weights are passed to hidden layers. For example in a network with two hidden layers (h1 and h2) the output from h1 serves as the input to h2. Before applying an activation function, a bias is added to the weighted inputs.

Each hidden layer computes the weighted sum (`a`) of the inputs then applies an activation function like **ReLU (Rectified Linear Unit)** to obtain the output (`o`). The output is passed to the next layer where an activation function such as **softmax** converts the weighted outputs into probabilities for classification.



The forward pass using weights and biases

2. Backward Pass

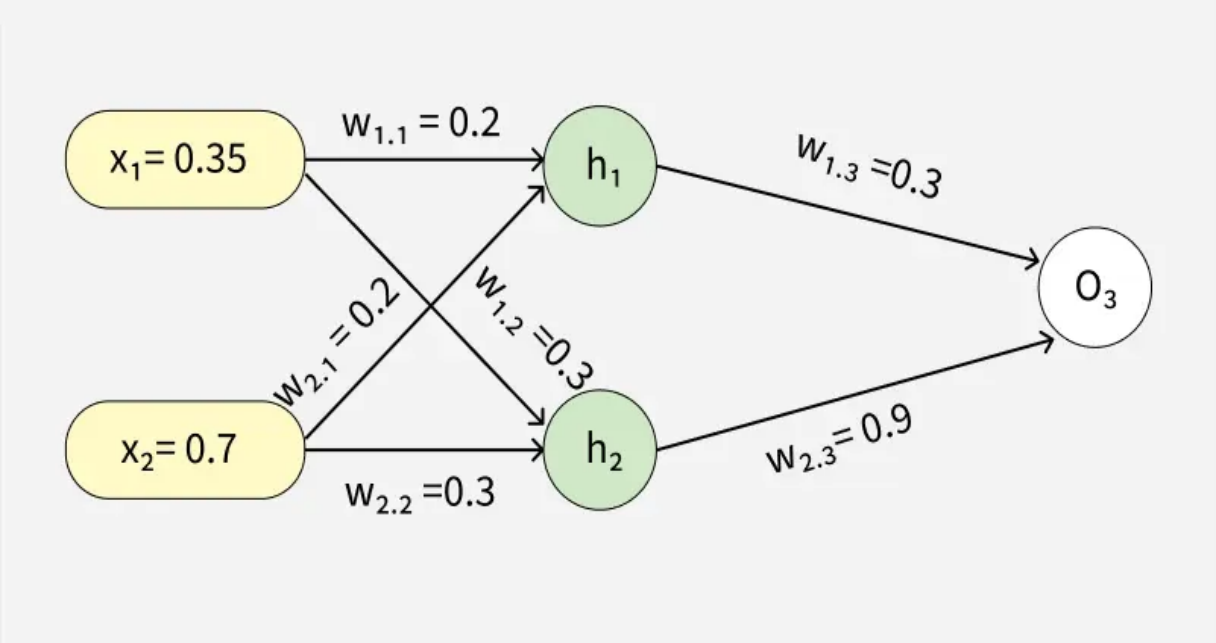
In the backward pass the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases. One common method for error calculation is the **Mean Squared Error (MSE)** given by:

$$\text{MSE} = (\text{Predicted Output} - \text{Actual Output})^2$$

Once the error is calculated the network adjusts weights using **gradients** which are computed with the chain rule. These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration. The backward pass continues layer by layer ensuring that the network learns and improves its performance. The activation function through its derivative plays a crucial role in computing these gradients during Back Propagation.

Example of Back Propagation in Machine Learning

Let's walk through an example of Back Propagation in machine learning. Assume the neurons use the sigmoid activation function for the forward and backward pass. The target output is 0.5 and the learning rate is 1.



Example (1) of backpropagation sum

Forward Propagation

1. Initial Calculation

The weighted sum at each node is calculated using:

$$aj = \sum(wi, j * xi)$$

Where,

- a_j is the weighted sum of all the inputs and weights at each node
- $w_{i,j}$ represents the weights between the i^{th}input and the j^{th} neuron
- x_i represents the value of the i^{th} input

O (output): After applying the activation function to a, we get the output of the neuron:

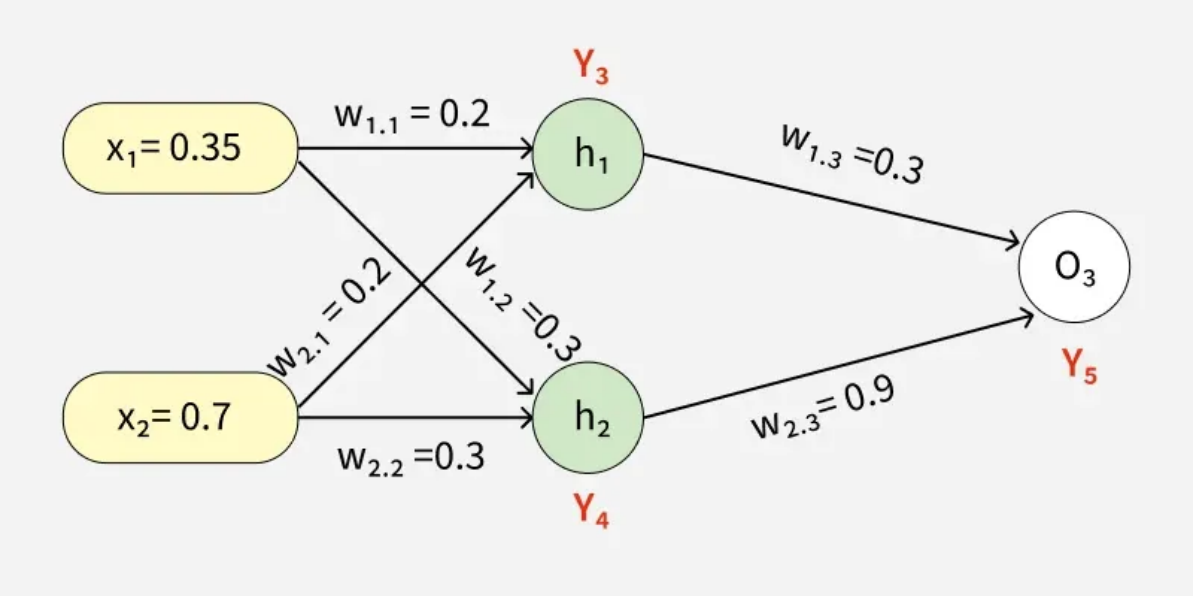
o_j= activation function(a_j)

2. Sigmoid Function

The sigmoid function returns a value between 0 and 1, introducing non-linearity into the model.

$$y_j = \frac{1}{1 + e^{-a_j}}$$

To find the outputs of y3, y4 and y5



3. Computing Outputs

At h1 node

$$\begin{aligned} a_1 &= (w_{1,1}x_1) + (w_{2,1}x_2) \\ &= (0.2 * 0.35) + (0.2 * 0.7) \\ &= 0.21 \end{aligned}$$

Once we calculated the a1 value, we can now proceed to find the y3 value:

$$y_j = F(a_j) = \frac{1}{1 + e^{-a_1}}$$

$$y_3 = F(0.21) = \frac{1}{1 + e^{-0.21}}$$

$$y_3 = 0.56$$

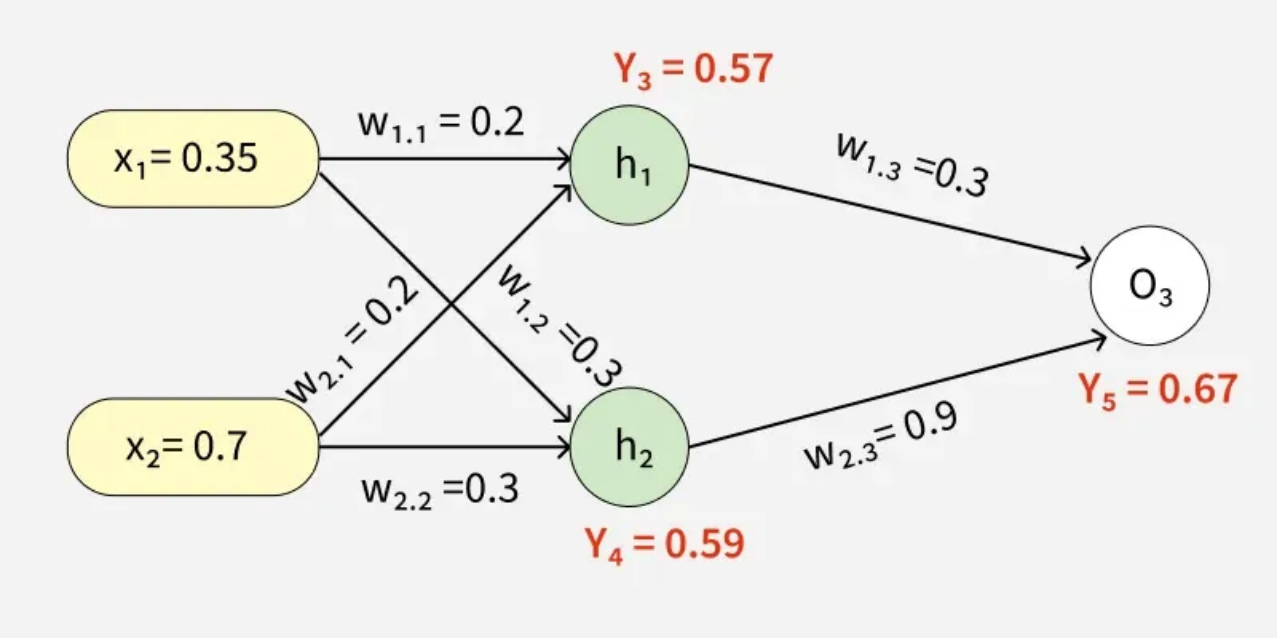
Similarly find the values of y4 at **h2** and y5 at O3

$$a_2 = (w_{1,2} * x_1) + (w_{2,2} * x_2) = (0.3 * 0.35) + (0.3 * 0.7) = 0.315$$

$$y_4 = F(0.315) = \frac{1}{1 + e^{-0.315}}$$

$$a3 = (w_{1,3} * y_3) + (w_{2,3} * y_4) = (0.3 * 0.57) + (0.9 * 0.59) = 0.702$$

$$y_5 = F(0.702) = \frac{1}{1 + e^{-0.702}} = 0.67$$



Values of y3, y4 and y5

4. Error Calculation

Our actual output is 0.5 but we obtained 0.67. To calculate the error we can use the below formula:

$$Error_j = y_{target} - y_5$$

$$=> 0.5 - 0.67 = -0.17$$

Using this error value we will be backpropagating.

Back Propagation

1. Calculating Gradients

The change in each weight is calculated as:

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

Where:

- δ_j is the error term for each unit,
- η is the learning rate.

2. Output Unit Error

For O3:

$$\delta_5 = y_5(1 - y_5)(y_{target} - y_5)$$

$$= 0.67(1 - 0.67)(-0.17) = -0.0376$$

3. Hidden Unit Error

For h1:

$$\delta_3 = y_3(1 - y_3)(w_{1,3} \times \delta_5)$$

$$= 0.56(1 - 0.56)(0.3 \times -0.0376) = -0.0027$$

For h2:

$$\delta_4 = y_4(1 - y_4)(w_{2,3} \times \delta_5)$$

$$= 0.59(1 - 0.59)(0.9 \times -0.0376) = -0.0819$$

4. Weight Updates

For the weights from hidden to output layer:

$$\Delta w_{2,3} = 1 \times (-0.0376) \times 0.59 = -0.022184$$

New weight:

$$w_{2,3}(\text{new}) = -0.022184 + 0.9 = 0.877816$$

For weights from input to hidden layer:

$$\Delta w_{1,1} = 1 \times (-0.0027) \times 0.35 = 0.000945$$

New weight:

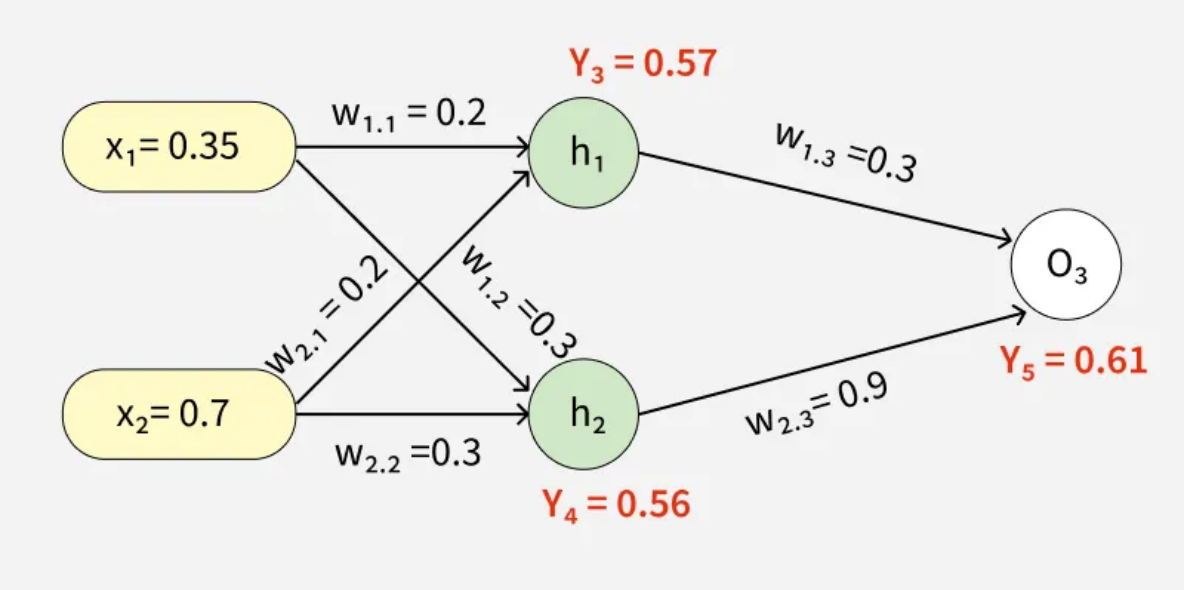
$$w_{1,1}(\text{new}) = 0.000945 + 0.2 = 0.200945$$

Similarly other weights are updated:

- $w_{1,2}(\text{new}) = 0.273225$
- $w_{1,3}(\text{new}) = 0.086615$
- $w_{2,1}(\text{new}) = 0.269445$
- $w_{2,2}(\text{new}) = 0.18534$

The updated weights are illustrated below

Through backward pass the weights are updated



After updating the weights the forward pass is repeated hence giving:

- $y_3 = 0.57$
- $y_4 = 0.56$
- $y_5 = 0.61$

Since $y_5 = 0.61$ is still not the target output the process of calculating the error and backpropagating continues until the desired output is reached.

This process demonstrates how Back Propagation iteratively updates weights by minimizing errors until the network accurately predicts the output.

$$Error = y_{target} - y_5$$

$$= 0.5 - 0.61 = -0.11$$

Logistic Regression in Machine Learning

Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class. It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between 0 and 1. In this article, we will see the basics of logistic regression and its core concepts.

Types of Logistic Regression

Logistic regression can be classified into three main types based on the nature of the dependent variable:

1. **Binomial Logistic Regression**: This type is used when the dependent variable has only two possible categories. Examples include Yes/No, Pass/Fail or 0/1. It is the most common form of logistic regression and is used for binary classification problems.
2. **Multinomial Logistic Regression**: This is used when the dependent variable has three or more possible categories that are not ordered. For example, classifying animals into categories like "cat," "dog" or "sheep." It extends the binary logistic regression to handle multiple classes.
3. **Ordinal Logistic Regression**: This type applies when the dependent variable has three or more categories with a natural order or ranking. Examples include ratings like "low," "medium" and "high." It takes the order of the categories into account when modeling.

Assumptions of Logistic Regression

Understanding the assumptions behind logistic regression is important to ensure the model is applied correctly, main assumptions are:

1. **Independent observations**: Each data point is assumed to be independent of the others means there should be no correlation or dependence between the input samples.
2. **Binary dependent variables**: It takes the assumption that the dependent variable must be binary, means it can take only two values. For more than two categories SoftMax functions are used.
3. **Linearity relationship between independent variables and log odds**: The model assumes a linear relationship between the independent variables and the log odds of the dependent variable which means the predictors affect the log odds in a linear way.
4. **No outliers**: The dataset should not contain extreme outliers as they can distort the estimation of the logistic regression coefficients.
5. **Large sample size**: It requires a sufficiently large sample size to produce reliable and stable results.

Understanding Sigmoid Function

1. The sigmoid function is a important part of logistic regression which is used to convert the raw output of the model into a probability value between 0 and 1.

2. This function takes any real number and maps it into the range 0 to 1 forming an "S" shaped curve called the sigmoid curve or logistic curve. Because probabilities must lie between 0 and 1, the sigmoid function is perfect for this purpose.

3. In logistic regression, we use a threshold value usually 0.5 to decide the class label.

- If the sigmoid output is same or above the threshold, the input is classified as Class 1.
- If it is below the threshold, the input is classified as Class 0.

This approach helps to transform continuous input values into meaningful class predictions.

How does Logistic Regression work?

Logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Suppose we have input features represented as a matrix:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e 0 or 1.

$$Y = \begin{cases} 0 & \text{if } Class1 \\ 1 & \text{if } Class2 \end{cases}$$

then, apply the multi-linear function to the input variables X.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

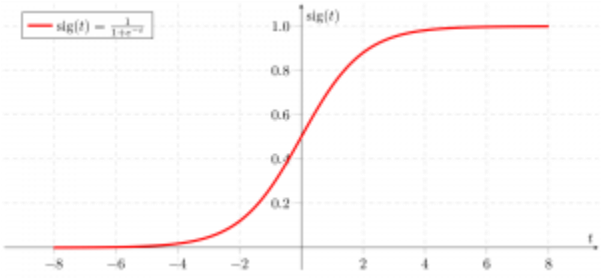
Here x_i is the ith observation of X, $w_i = [w_1, w_2, w_3, \cdots, w_m]$ is the weights or Coefficient and b is the bias term also known as intercept. Simply this can be represented as the dot product of weight and bias.

$$z = w \cdot X + b$$

At this stage, z is a continuous value from the linear regression. Logistic regression then applies the sigmoid function to z to convert it into a probability between 0 and 1 which can be used to predict the class.

Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e. predicted y.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid function

As shown above the sigmoid function converts the continuous variable data into the probability i.e between 0 and 1.

- $\sigma(z)$ tends towards 1 as $z \to \infty$
- $\sigma(z)$ tends towards 0 as $z \to -\infty$
- $\sigma(z)$ is always bounded between 0 and 1

where the probability of being a class can be measured as:

$$P(y = 1) = \sigma(z)$$

$$P(y = 0) = 1 - \sigma(z)$$

Logistic Regression Equation and Odds:

It models the odds of the dependent event occurring which is the ratio of the probability of the event to the probability of it not occurring:

$$\frac{p(x)}{1 - p(x)} = e^{z}$$

Taking the natural logarithm of the odds gives the log-odds or logit:

$$\log\left[\frac{p(x)}{1-p(x)}\right] = z$$

$$\log\left[\frac{p(x)}{1-p(x)}\right] = w \cdot X + b$$

$$\frac{p(x)}{1-p(x)} = e^{w \cdot X + b} \cdots \text{Exponentiate both sides}$$

$$p(x) = e^{w \cdot X + b} \cdot (1 - p(x))$$

$$p(x) = e^{w \cdot X + b} - e^{w \cdot X + b} \cdot p(x))$$

$$p(x) + e^{w \cdot X + b} \cdot p(x)) = e^{w \cdot X + b}$$

$$p(x)(1 + e^{w \cdot X + b}) = e^{w \cdot X + b}$$

$$p(x) = \frac{e^{w \cdot X + b}}{1 + e^{w \cdot X + b}}$$

then the final logistic regression equation will be:

$$p(X; b, w) = \frac{e^{w \cdot X + b}}{1 + e^{w \cdot X + b}} = \frac{1}{1 + e^{-w \cdot X + b}}$$

This formula represents the probability of the input belonging to Class 1.

Likelihood Function for Logistic Regression

The goal is to find weights w and bias b that maximize the likelihood of observing the data.

For each data point i

- for $y = 1$, predicted probabilities will be: p(X;b,w) =$p(x)$
- for $y = 0$ The predicted probabilities will be: 1-p(X;b,w) = $1 - p(x)$

$$L(b, w) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

Taking natural logs on both sides:

$$\log(L(b,w)) = \sum_{i=1}^{n} y_i \log p(x_i) + (1-y_i)\log(1-p(x_i))$$

$$= \sum_{i=1}^{n} y_i \log p(x_i) + \log(1-p(x_i)) - y_i \log(1-p(x_i))$$

$$= \sum_{i=1}^{n} \log(1-p(x_i)) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1-p(x_i}$$

$$= \sum_{i=1}^{n} -\log\ 1 - e^{-(w\cdot x_i+b)} + \sum_{i=1}^{n} y_i(w\cdot x_i + b)$$

$$= \sum_{i=1}^{n} -\log\ 1 + e^{w\cdot x_i+b} + \sum_{i=1}^{n} y_i(w\cdot x_i + b)$$

This is known as the log-likelihood function.

Gradient of the log-likelihood function

To find the best w and b we use gradient ascent on the log-likelihood function. The gradient with respect to each weight $w_j$is:

$$\frac{\partial J(l(b,w)}{\partial w_j} = -\sum_{i=n}^{n} \frac{1}{1+e^{w\cdot x_i+b}} e^{w\cdot x_i+b} x_{ij} + \sum_{i=1}^{n} y_i x_{ij}$$

$$= -\sum_{i=n}^{n} p(x_i; b, w) x_{ij} + \sum_{i=1}^{n} y_i x_{ij}$$

$$= \sum_{i=n}^{n} (y_i - p(x_i; b, w)) x_{ij}$$

Terminologies involved in Logistic Regression

Here are some common terms involved in logistic regression:

1. **Independent Variables:** These are the input features or predictor variables used to make predictions about the dependent variable.
2. **Dependent Variable**: This is the target variable that we aim to predict. In logistic regression, the dependent variable is categorical.
3. **Logistic Function**: This function transforms the independent variables into a probability between 0 and 1 which represents the likelihood that the dependent variable is either 0 or 1.

4. **Odds**: This is the ratio of the probability of an event happening to the probability of it not happening. It differs from probability because probability is the ratio of occurrences to total possibilities.
5. **Log-Odds (Logit)**: The natural logarithm of the odds. In logistic regression, the log-odds are modeled as a linear combination of the independent variables and the intercept.
6. **Coefficient**: These are the parameters estimated by the logistic regression model which shows how strongly the independent variables affect the dependent variable.
7. **Intercept**: The constant term in the logistic regression model which represents the log-odds when all independent variables are equal to zero.
8. **Maximum Likelihood Estimation (MLE)**: This method is used to estimate the coefficients of the logistic regression model by maximizing the likelihood of observing the given data.

Linear Regression in Machine learning:

Linear regression is a type of **supervised machine-learning algorithm** that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It assumes that there is a linear relationship between the input and output, meaning the output changes at a constant rate as the input changes. This relationship is represented by a straight line.

For example we want to predict a student's exam score based on how many hours they studied. We observe that as students study more hours, their scores go up. In the example of predicting exam scores based on hours studied. Here

- **Independent variable (input):** Hours studied because it's the factor we control or observe.
- **Dependent variable (output):** Exam score because it depends on how many hours were studied.

We use the independent variable to predict the dependent variable.

Why Linear Regression is Important?

Here’s why linear regression is important:

- **Simplicity and Interpretability:** It’s easy to understand and interpret, making it a starting point for learning about machine learning.

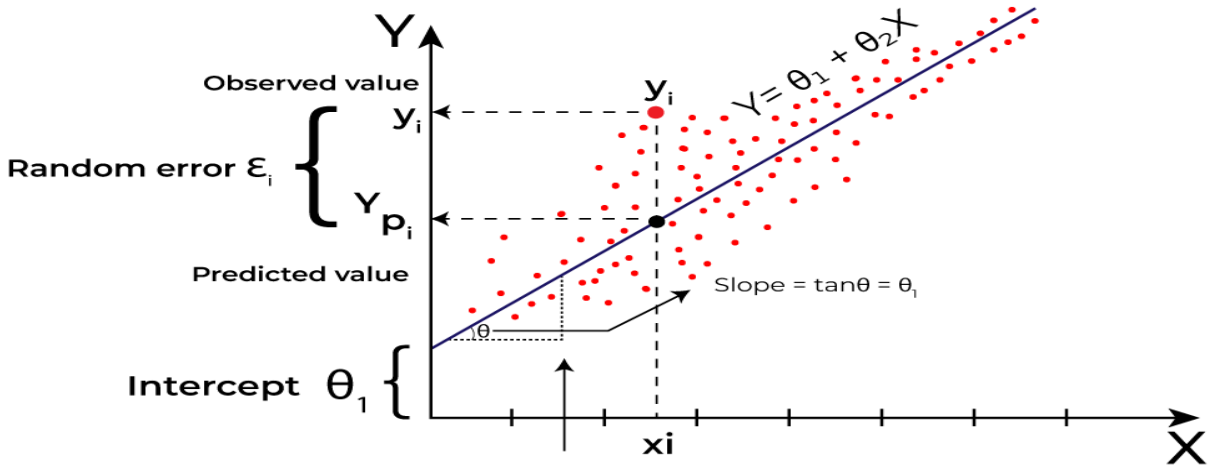
- **Predictive Ability**: Helps predict future outcomes based on past data, making it useful in various fields like finance, healthcare and marketing.
- **Basis for Other Models:** Many advanced algorithms, like logistic regression or neural networks, build on the concepts of linear regression.
- **Efficiency:** It's computationally efficient and works well for problems with a linear relationship.
- **Widely Used:** It's one of the most widely used techniques in both statistics and machine learning for regression tasks.
- **Analysis:** It provides insights into relationships between variables (e.g., how much one variable influences another).

Best Fit Line in Linear Regression

In linear regression, the best-fit line is the straight line that most accurately represents the relationship between the independent variable (input) and the dependent variable (output). It is the line that minimizes the difference between the actual data points and the predicted values from the model.

1. Goal of the Best-Fit Line

The goal of linear regression is to find a straight line that minimizes the error (the difference) between the observed data points and the predicted values. This line helps us predict the dependent variable for new, unseen data.



Linear Regression

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. There are many types of functions or modules that can be used for regression. A linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.

2. Equation of the Best-Fit Line

For simple linear regression (with one independent variable), the best-fit line is represented by the equation

$$y = mx + b$$

Where:

- **y** is the predicted value (dependent variable)
- **x** is the input (independent variable)
- **m** is the slope of the line (how much y changes when x changes)
- **b** is the intercept (the value of y when x = 0)

The best-fit line will be the one that optimizes the values of m (slope) and b (intercept) so that the predicted y values are as close as possible to the actual data points.

3. Minimizing the Error: The Least Squares Method

To find the best-fit line, we use a method called **Least Squares**. The idea behind this method is to minimize the sum of squared differences between the actual values (data points) and the predicted values from the line. These differences are called residuals.

The formula for residuals is:

$$Residual = y_i - \hat{y}_i$$

Where:

- y_i is the actual observed value
- $\hat{y}_i$ is the predicted value from the line for that x_i

The least squares method minimizes the sum of the squared residuals:

$$Sum\ of squared\ errors(SSE) = \Sigma(y_i - \hat{y}_i)^2$$

This method ensures that the line best represents the data where the sum of the squared differences between the predicted values and actual values is as small as possible.

4. Interpretation of the Best-Fit Line

- **Slope (m):** The slope of the best-fit line indicates how much the dependent variable (y) changes with each unit change in the independent variable (x). For example if the slope is 5, it means that for every 1-unit increase in x, the value of y increases by 5 units.
- **Intercept (b):** The intercept represents the predicted value of y when x = 0. It's the point where the line crosses the y-axis.

In linear regression some hypothesis are made to ensure reliability of the model's results.

Limitations

- ***Assumes Linearity:*** *The method assumes the relationship between the variables is linear. If the relationship is non-linear, linear regression might not work well.*
- ***Sensitivity to Outliers:*** *Outliers can significantly affect the slope and intercept, skewing the best-fit line.*

Hypothesis function in Linear Regression

In linear regression, the hypothesis function is the equation used to make predictions about the dependent variable based on the independent variables. It represents the relationship between the input features and the target output.

For a simple case with one independent variable, the hypothesis function is:

$$h(x) = \beta_0 + \beta_1 x$$

Where:

- $h(x) (or \hat{y})$ is the predicted value of the dependent variable (y).
- x $x$is the independent variable.
- β_0 is the intercept, representing the value of y when x is 0.
- β_1 is the slope, indicating how much y changes for each unit change in x.

For **multiple linear regression** (with more than one independent variable), the hypothesis function expands to:

$$h(x_1, x_2, \ldots, x_k) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k$$

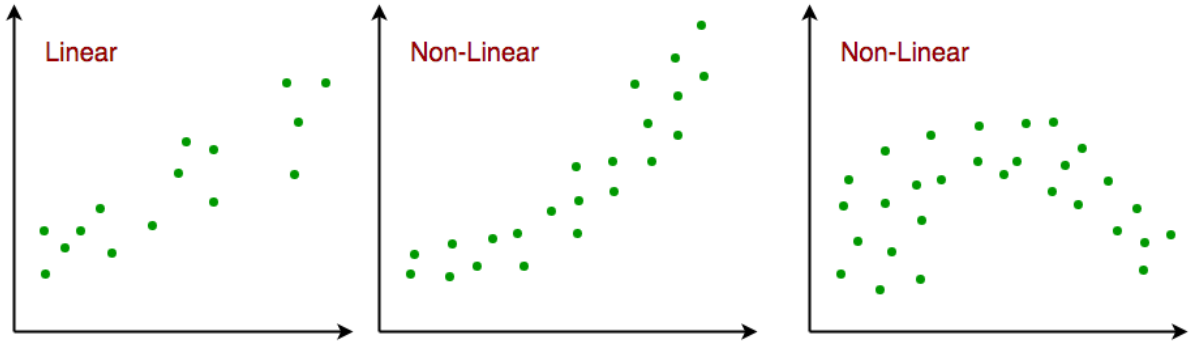
Where:

- $x_1, x_2, \ldots, x_k$ are the independent variables.
- β_0 is the intercept.

- $\beta_1, \beta_2, \ldots, \beta_k$ are the coefficients, representing the influence of each respective independent variable on the predicted output.

Assumptions of the Linear Regression

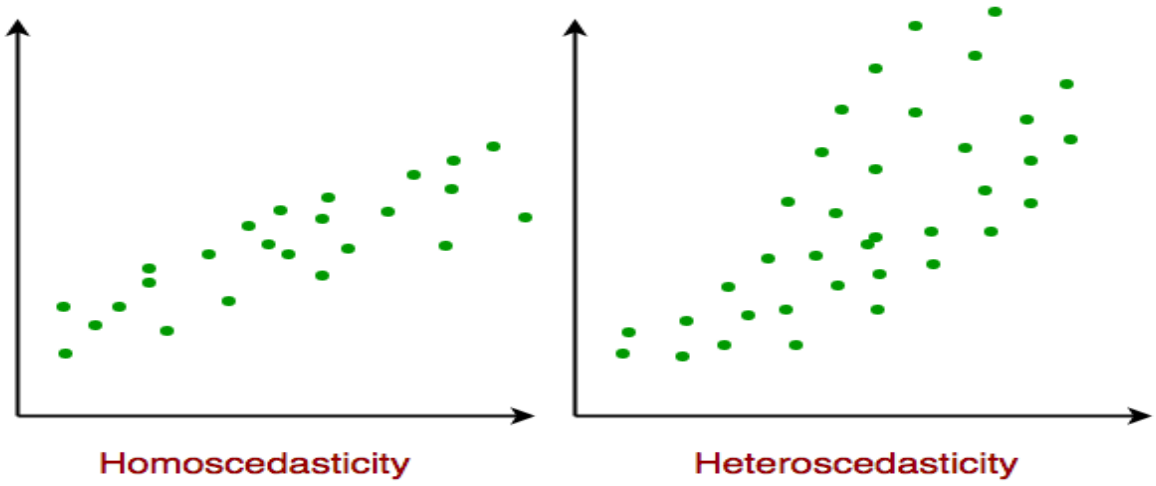
1. Linearity: The relationship between inputs (X) and the output (Y) is a straight line.



Linearity

2. Independence of Errors: The errors in predictions should not affect each other.

3. Constant Variance (Homoscedasticity): The errors should have equal spread across all values of the input. If the spread changes (like fans out or shrinks), it's called heteroscedasticity and it's a problem for the model.



Homoscedasticity

4. Normality of Errors: The errors should follow a normal (bell-shaped) distribution.

5. No Multicollinearity *(*for multiple regression): Input variables shouldn't be too closely related to each other.

6. No Autocorrelation: Errors shouldn't show repeating patterns, especially in time-based data.

7. Additivity: The total effect on Y is just the sum of effects from each X, no mixing or interaction between them.'

To understand Multicollinearity in detail refer to article: ***Multicollinearity****.*

Types of Linear Regression

When there is only one independent feature it is known as Simple Linear Regression or Univariate Linear Regression and when there are more than one feature it is known as Multiple Linear Regression or Multivariate Regression.

1. Simple Linear Regression

Simple linear regression is used when we want to predict a target value (dependent variable) using only one input feature (independent variable). It assumes a straight-line relationship between the two.

Formula:

$$\hat{y} = \theta_0 + \theta_1 x$$

Where:

- $\hat{y}$ is the predicted value
- $x$is the input (independent variable)
- θ_0 is the intercept (value of $\hat{y}$ when x=0)
- θ_1 is the slope or coefficient (how much $\hat{y}$ changes with one unit of x)

Example:

Predicting a person's salary (y) based on their years of experience (x).

2. Multiple Linear Regression

Multiple linear regression involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

where:

- $\hat{y}$ is the predicted value
- $x_1, x_2, \ldots, x_n$ are the independent variables
- $\theta_1, \theta_2, \ldots, \theta_n$ are the coefficients (weights) corresponding to each predictor.
- θ_0 is the intercept.

The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.

Use Case of Multiple Linear Regression

Multiple linear regression allows us to analyze relationship between multiple independent variables and a single dependent variable. Here are some use cases:

- **Real Estate Pricing:** In real estate MLR is used to predict property prices based on multiple factors such as location, size, number of bedrooms, etc. This helps buyers and sellers understand market trends and set competitive prices.
- **Financial Forecasting:** Financial analysts use MLR to predict stock prices or economic indicators based on multiple influencing factors such as interest rates, inflation rates and market trends. This enables better investment strategies and risk management24.
- **Agricultural Yield Prediction:** Farmers can use MLR to estimate crop yields based on several variables like rainfall, temperature, soil quality and fertilizer usage. This information helps in planning agricultural practices for optimal productivity
- **E-commerce Sales Analysis:** An e-commerce company can utilize MLR to assess how various factors such as product price, marketing promotions and seasonal trends impact sales.

Now that we have understood about linear regression, its assumption and its type now we will learn how to make a linear regression model.

Cost function for Linear Regression

As we have discussed earlier about best fit line in linear regression, its not easy to get it easily in real life cases so we need to calculate errors that affects it. These errors need to be

calculated to mitigate them. The difference between the predicted value $\hat{Y}$ and the true value Y and it is called cost function or the loss function.

In Linear Regression, the Mean Squared Error (MSE) cost function is employed, which calculates the average of the squared errors between the predicted values $\hat{y}_i$ and the actual values y_i. The purpose is to determine the optimal values for the intercept θ_1 and the coefficient of the input feature θ_2 providing the best-fit line for the given data points. The linear equation expressing this relationship is $\hat{y}_i = \theta_1 + \theta_2 x_i$.

MSE function can be calculated as:

$$\text{Cost function}(J) = \frac{1}{n}\sum_{n}^{i} \quad (\hat{y_i} - y_i)^2$$

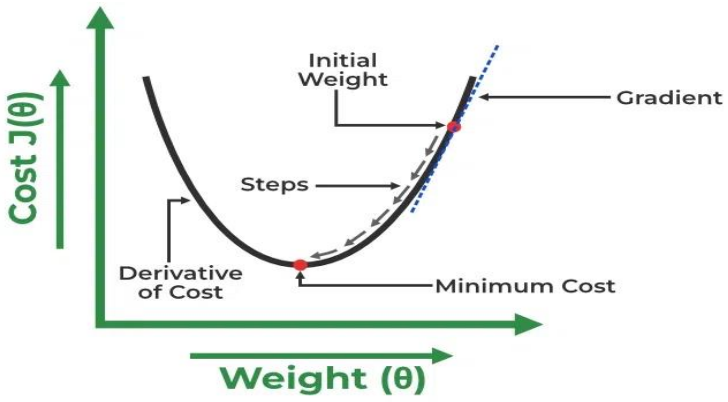
Utilizing the MSE function, the iterative process of gradient descent is applied to update the values of \θ_1&θ_2. This ensures that the MSE value converges to the global minima, signifying the most accurate fit of the linear regression line to the dataset.

This process involves continuously adjusting the parameters \(\theta_1\) and \(\theta_2\) based on the gradients calculated from the MSE. The final result is a linear regression line that minimizes the overall squared differences between the predicted and actual values, providing an optimal representation of the underlying relationship in the data.

Now we have calculated loss function we need to optimize model to mtigate this error and it is done through gradient descent.

Gradient Descent for Linear Regression

Gradient descent is an optimization technique used to train a linear regression model by minimizing the prediction error. It works by starting with random model parameters and repeatedly adjusting them to reduce the difference between predicted and actual values.



Gradient Descent

How it works:

- Start with random values for slope and intercept.
- Calculate the error between predicted and actual values.
- Find how much each parameter contributes to the error (gradient).
- Update the parameters in the direction that reduces the error.
- Repeat until the error is as small as possible.

This helps the model find the best-fit line for the data.

For more details you can refer to: Gradient Descent in Linear Regression

Evaluation Metrics for Linear Regression

A variety of evaluation measures can be used to determine the strength of any linear regression model. These assessment metrics often give an indication of how well the model is producing the observed outputs.

The most common measurements are:

1. Mean Square Error (MSE)

Mean Squared Error (MSE) is an evaluation metric that calculates the average of the squared differences between the actual and predicted values for all the data points. The difference is squared to ensure that negative and positive differences don't cancel each other out.

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

Here,

- $n$is the number of data points.
- $y_i$is the actual or observed value for thei^{th}data point.
- $\hat{y_i}$ is the predicted value for the i^{th}data point.

MSE is a way to quantify the accuracy of a model's predictions. MSE is sensitive to outliers as large errors contribute significantly to the overall score.

2. Mean Absolute Error (MAE)

Mean Absolute Error is an evaluation metric used to calculate the accuracy of a regression model. MAE measures the average absolute difference between the predicted values and actual values.

Mathematically MAE is expressed as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \quad | Y_i - \widehat{Y}_i |$$

Here,

- n is the number of observations
- Yi represents the actual values.
- $\widehat{Y}_i$ represents the predicted values

Lower MAE value indicates better model performance. It is not sensitive to the outliers as we consider absolute differences.

3. Root Mean Squared Error (RMSE)

The square root of the residuals' variance is the Root Mean Squared Error. It describes how well the observed data points match the expected values or the model's absolute fit to the data.
In mathematical notation, it can be expressed as:

$$RMSE = \sqrt{\frac{RSS}{n}} = \sqrt{\frac{\sum_{i=2}^{n} \quad (y_i^{actual} - y_i^{predicted})^2}{n}}$$

Rather than dividing the entire number of data points in the model by the number of degrees of freedom, one must divide the sum of the squared residuals to obtain an unbiased estimate. Then, this figure is referred to as the Residual Standard Error (RSE).

In mathematical notation, it can be expressed as:

$$RMSE = \sqrt{\frac{RSS}{n}} = \sqrt{\frac{\sum_{i=2}^{n} \quad (y_i^{actual} - y_i^{predicted})^2}{(n-2)}}$$

RSME is not as good of a metric as R-squared. Root Mean Squared Error can fluctuate when the units of the variables vary since its value is dependent on the variables' units (it is not a normalized measure).

4. Coefficient of Determination (R-squared)

R-Squared is a statistic that indicates how much variation the developed model can explain or capture. It is always in the range of 0 to 1. In general, the better the model matches the data, the greater the R-squared number.

In mathematical notation, it can be expressed as:

$$R^2 = 1 - (\frac{RSS}{TSS})$$

- **Residual sum of Squares**(RSS): The sum of squares of the residual for each data point in the plot or data is known as the residual sum of squares or RSS. It is a measurement of the difference between the output that was observed and what was anticipated.

$$RSS = \sum_{i=1}^{n} \quad (y_i - b_0 - b_1 x_i)^2$$

- **Total Sum of Squares (TSS):** The sum of the data points' errors from the answer variable's mean is known as the total sum of squares or TSS.

$$TSS = \sum_{i=1}^{n} \quad (y - \bar{y_i})^2.$$

R squared metric is a measure of the proportion of variance in the dependent variable that is explained the independent variables in the model.

5. Adjusted R-Squared Error

Adjusted R^2measures the proportion of variance in the dependent variable that is explained by independent variables in a regression model. Adjusted R-square accounts the number of predictors in the model and penalizes the model for including irrelevant predictors that don't contribute significantly to explain the variance in the dependent variables.

Mathematically, adjusted $R^2$is expressed as:

$$AdjustedR^2 = 1 - (\frac{(1 - R^2).(n-1)}{n-k-1})$$

Here,

- n is the number of observations
- k is the number of predictors in the model
- R2 is coeeficient of determination

Adjusted R-square helps to prevent overfitting. It penalizes the model with additional predictors that do not contribute significantly to explain the variance in the dependent variable.

While evaluation metrics help us measure the performance of a model, regularization helps in improving that performance by addressing overfitting and enhancing generalization.

Regularization Techniques for Linear Models

1. Lasso Regression (L1 Regularization)

Lasso Regression is a technique used for regularizing a linear regression model, it adds a penalty term to the linear regression objective function to prevent overfitting.

The objective function after applying lasso regression is:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{n} | \theta_j |$$

- the first term is the least squares loss, representing the squared difference between predicted and actual values.
- the second term is the L1 regularization term, it penalizes the sum of absolute values of the regression coefficient θj.

2. Ridge Regression (L2 Regularization)

Ridge regression is a linear regression technique that adds a regularization term to the standard linear objective. Again, the goal is to prevent overfitting by penalizing large coefficient in linear regression equation. It useful when the dataset has multicollinearity where predictor variables are highly correlated.

The objective function after applying ridge regression is:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

- the first term is the least squares loss, representing the squared difference between predicted and actual values.
- the second term is the L1 regularization term, it penalizes the sum of square of values of the regression coefficient θj.

3. Elastic Net Regression

Elastic Net Regression is a hybrid regularization technique that combines the power of both L1 and L2 regularization in linear regression objective.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} \quad (\hat{y_i} - y_i)^2 + \alpha\lambda \sum_{j=1}^{n} \mid \theta_j \mid + \frac{1}{2}(1-\alpha)\lambda \sum_{j=1} \quad n\theta_j^2$$

- the first term is least square loss.
- the second term is L1 regularization and third is ridge regression.
- $\lambda$is the overall regularization strength.
- αcontrols the mix between L1 and L2 regularization.