11. Demonstrate Multi-layer Perceptron algorithm for a classification problem.

**Description:**

A **Multi-Layer Perceptron (MLP)** is a type of **artificial neural network (ANN)** that consists of multiple layers of neurons, enabling it to learn complex patterns in data. It is primarily used for **classification and regression problems**.

**Architecture of MLP:**

MLP consists of three main types of layers:

1. **Input Layer** – Receives input features.
2. **Hidden Layers** – One or more layers where neurons perform weighted computations and apply activation functions.
3. **Output Layer** – Provides the final prediction (class labels for classification problems).

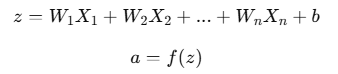
Each neuron in a layer is **fully connected** to the neurons in the next layer.

**MLP Works for Classification:**

1. **Forward Propagation:**
   * Each neuron computes a weighted sum of its inputs and applies an **activation function** (e.g., ReLU, Sigmoid).
   * The process continues through multiple layers until the output layer produces predictions.
2. **Activation Functions:**
   * **ReLU (Rectified Linear Unit):** Used in hidden layers to introduce non-linearity.
   * **Sigmoid:** Used in the output layer for binary classification.
   * **Softmax:** Used in the output layer for multi-class classification.
3. **Loss Function:**
   * **Binary Cross-Entropy Loss:** Used for binary classification.
   * **Categorical Cross-Entropy Loss:** Used for multi-class classification.
4. **Backpropagation & Optimization:**
   * The **error** (difference between predicted and actual values) is calculated.
   * Using the **gradient descent** algorithm and an optimizer like **Adam** or **SGD**, weights are updated to minimize the loss.

**Mathematical Representation:**

For each neuron:



Where:

* X = Input features
* W = Weights
* b = Bias
* f(z) = Activation function (e.g., ReLU, Sigmoid)

**Steps Involved:**

1. Load and preprocess the dataset
2. Train an **MLP classifier** using MLPClassifier from sklearn.neural\_network
3. Make predictions and evaluate the model
4. Visualize results

**Dataset Loading:** We use the **Iris dataset**, which contains **three classes (Setosa, Versicolor, Virginica)**.

**Data Preprocessing:**

* Data is **split into training (80%) and testing (20%) sets**.
* **Standardization** is applied to improve neural network performance.

**MLP Classifier Model:**

* **Hidden Layers:** (10,10) → Two hidden layers with 10 neurons each.
* **Activation Function:** 'relu' → Rectified Linear Unit (ReLU) for non-linearity.
* **Optimizer:** 'adam' → Adaptive learning rate optimization.
* **Epochs (max\_iter)** = 500.

**Model Evaluation:**

* Accuracy, Confusion Matrix, and Classification Report are displayed.
* A **confusion matrix** is plotted for visualization.

**Python Code for MLP Classification**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Load the dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # Target labels (0, 1, 2)

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features (MLP performs better with normalized data)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Define and train the MLP classifier

mlp = MLPClassifier(hidden\_layer\_sizes=(10, 10), activation='relu', solver='adam', max\_iter=500, random\_state=42)

mlp.fit(X\_train, y\_train)

# Make predictions

y\_pred = mlp.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Print results

print(f"Accuracy: {accuracy:.2f}")

print("Confusion Matrix:\n", conf\_matrix)

print("Classification Report:\n", class\_report)

# Plot Confusion Matrix

plt.imshow(conf\_matrix, cmap="Blues", interpolation="nearest")

plt.colorbar()

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix - MLP Classifier")

plt.show()