EX NO:1 Build simple neural network without using keras

Aim:

To implement a simple neural network for binary classification using Python without relying on deep learning libraries (like Keras or PyTorch). The neural network should classify whether an Iris flower belongs to the "setosa" species or not.

Procedure:

1. Import Required Libraries: Use numpy for mathematical operations, pandas for data handling, and sklearn for preprocessing and evaluation.
2. Load and Preprocess the Dataset: Load the Iris dataset into a DataFrame. Convert the target column (species) into a binary format (1 for "setosa", 0 for others). Standardize the feature values to ensure the data is normalized.
3. Split the Data: Divide the data into training and testing sets to evaluate the model.
4. Implement the Neural Network: Initialize random weights and a bias term. Define the sigmoid activation function and its derivative. Train the network using forward and backward propagation. Adjust weights and bias iteratively to minimize the error.
5. Test and Evaluate the Model: Use the trained network to predict the labels of the test set. Compute the accuracy score to evaluate performance.
6. Analyze Results: Print the accuracy score and observe the network's ability to classify the flowers.

Code:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import pandas as pd

# Load dataset
df = pd.read_csv('/content/iris.csv')

# Preprocess dataset
df['species'] = (df['species'] == 'setosa').astype(int)  # Binary classification
X = df.drop('species', axis=1).values
y = df['species'].values.reshape(-1, 1)

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define the Neural Network class
class SimpleNN:
    def __init__(self, input_size):
        np.random.seed(42)  # For reproducibility
        self.weights = np.random.rand(input_size, 1) - 0.5  # Initialize weights
        self.bias = 0

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))  # Sigmoid function

    def sigmoid_derivative(self, z):
        return z * (1 - z)  # Derivative of sigmoid

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            # Forward pass
            z = np.dot(X, self.weights) + self.bias
            output = self.sigmoid(z)

            # Backward pass
            error = y - output
            adjustments = error * self.sigmoid_derivative(output)

            # Update weights and bias
            self.weights += np.dot(X.T, adjustments) * learning_rate
            self.bias += np.sum(adjustments) * learning_rate

            # Print loss at intervals
            if epoch % 1000 == 0:
                loss = np.mean(error ** 2)
                print(f'Epoch {epoch}, Loss: {loss}')

    def predict(self, X):
        z = np.dot(X, self.weights) + self.bias
        return (self.sigmoid(z) > 0.5).astype(int)

# Initialize and train the model
nn = SimpleNN(input_size=X_train.shape[1])
nn.train(X_train, y_train, epochs=5000, learning_rate=0.1)

# Test the model
y_pred = nn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output:

```
Epoch 0, Loss: 0.252348
Epoch 1000, Loss: 0.015468
Epoch 2000, Loss: 0.008732
Epoch 3000, Loss: 0.006121
Epoch 4000, Loss: 0.004781
Accuracy: 0.9555555555555556
```

Result:

Thus to implement a simple neural network for binary classification using Python without relying on deep learning libraries was successfully executed.