Name: Priansh Madan

Roll No: 58

Batch: E4

Lab 5

Aim: Write a program to implement backpropagation algorithm on iris dataset.

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [ ]:
```
n_epochs=100

# Load iris dataset
iris = datasets.load_iris()

data=pd.DataFrame(iris.data)
data['class']=iris.target

data.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
print(data.head())
```

```
   sepal_len  sepal_wid  petal_len  petal_wid  class
0        5.1        3.5        1.4        0.2      0
1        4.9        3.0        1.4        0.2      0
2        4.7        3.2        1.3        0.2      0
3        4.6        3.1        1.5        0.2      0
4        5.0        3.6        1.4        0.2      0
```

In [ ]:
```
#defining derivative functions
def sigmoid(x):
   return 1/(1+np.exp(-x.astype(float)))
def sigmoid_derivative(x):
   return x*(1-x)
```

In [ ]:
```
# Load Iris dataset
iris = datasets.load_iris()

# Standardize input features
scaler = StandardScaler()
X = scaler.fit_transform(iris.data)

# One-hot encode target variable
y = np.zeros((len(iris.target), 3))
y[np.arange(len(iris.target)), iris.target] = 1

# 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_st
```

In [ ]:
```
# Define neural network architecture
input_size = 4
hidden_size = 3
```

```
output_size = 3

# Initialize weights and biases
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))

# Define activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

In [ ]:

```
# Define learning rate and number of iterations
learning_rate = 0.001
num_iterations = 1001

# Train neural network using backpropagation
for i in range(num_iterations):
    # Forward propagation
    z1 = np.dot(X_train, W1) + b1
    a1 = sigmoid(z1)
    z2 = np.dot(a1, W2) + b2
    a2 = sigmoid(z2)

    # Calculate loss and accuracy
    loss = -np.mean(y_train * np.log(a2) + (1 - y_train) * np.log(1 - a2))
    predictions = np.argmax(a2, axis=1)
    true_labels = np.argmax(y_train, axis=1)
    accuracy = np.mean(predictions == true_labels)

    # Backward propagation
    dL_da2 = (a2 - y_train) / (a2 * (1 - a2))
    da2_dz2 = sigmoid_derivative(z2)
    dL_dz2 = dL_da2 * da2_dz2
    dz2_dW2 = a1
    dL_dW2 = np.dot(dz2_dW2.T, dL_dz2)
    dz2_db2 = 1
    dL_db2 = np.sum(dL_dz2 * dz2_db2, axis=0)
    dL_da1 = np.dot(dL_dz2, W2.T)
    da1_dz1 = sigmoid_derivative(z1)
    dL_dz1 = dL_da1 * da1_dz1
    dz1_dW1 = X_train
    dL_dW1 = np.dot(dz1_dW1.T, dL_dz1)
    dz1_db1 = 1
    dL_db1 = np.sum(dL_dz1 * dz1_db1, axis=0)

# Update weights and biases
    W2 -= learning_rate * dL_dW2
    b2 -= learning_rate * dL_db2
    W1 -= learning_rate * dL_dW1
    b1 -= learning_rate * dL_db1

    # Print loss and accuracy every 100 iterations
    if i % 100 == 0:
      print(f"Iteration {i}: Loss = {loss:.4f}, Accuracy = {100*accuracy:.4f}%")
```

```
Iteration 0: Loss = 0.7423, Accuracy = 32.5000%
Iteration 100: Loss = 0.4544, Accuracy = 76.6667%
Iteration 200: Loss = 0.3748, Accuracy = 80.8333%
Iteration 300: Loss = 0.3392, Accuracy = 82.5000%
Iteration 400: Loss = 0.3170, Accuracy = 82.5000%
Iteration 500: Loss = 0.2985, Accuracy = 84.1667%
Iteration 600: Loss = 0.2799, Accuracy = 85.0000%
Iteration 700: Loss = 0.2591, Accuracy = 88.3333%
Iteration 800: Loss = 0.2346, Accuracy = 90.8333%
Iteration 900: Loss = 0.2074, Accuracy = 92.5000%
Iteration 1000: Loss = 0.1819, Accuracy = 95.0000%
```

In [ ]:

```
# Test neural network on testing set
z1 = np.dot(X_test, W1) + b1
a1 = sigmoid(z1)
z2 = np.dot(a1, W2) + b2
a2 = sigmoid(z2)

# Calculate testing accuracy
predictions = np.argmax(a2, axis=1)
true_labels = np.argmax(y_test, axis=1)
accuracy = np.mean(predictions == true_labels)

print(f"Testing Accuracy = {100*accuracy:.4f}%")
```

```
Testing Accuracy = 96.6667%
```