Date 04/03/2024

16.Aim: Write the python program to implement Feed forward neural Network

Program: import numpy as np

```
# Define the sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the derivative of the sigmoid function
def sigmoid_derivative(x):
    return x * (1 - x)

# Function to get user input for dataset
def get_user_input():
    print("Enter the number of samples:")
    num_samples = int(input())
    print("Enter the number of features:")
    num_features = int(input())

    X = []
    y = []

    print("Enter the values for the dataset (one sample per line):")
    for i in range(num_samples):
        sample = list(map(float, input().split()))
        X.append(sample[:-1])  # Features
        y.append([sample[-1]])    # Target labels (assuming single output neuron)

    return np.array(X), np.array(y)
```

```
# Main function
def main():
    # Get user input for dataset
    X, y = get_user_input()

    # Define network parameters
    input_size = X.shape[1]
    hidden_size = int(input("Enter the number of neurons in the hidden layer: "))
    output_size = y.shape[1]
    learning_rate = float(input("Enter the learning rate: "))
    epochs = int(input("Enter the number of epochs: "))

    # Initialize weights randomly
    np.random.seed(1)
    weights_input_hidden = np.random.uniform(size=(input_size, hidden_size))
    weights_hidden_output = np.random.uniform(size=(hidden_size, output_size))

    # Train the neural network
    for epoch in range(epochs):
        # Forward propagation
        hidden_layer_input = np.dot(X, weights_input_hidden)
        hidden_layer_output = sigmoid(hidden_layer_input)

        output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
        output_layer_output = sigmoid(output_layer_input)

        # Backpropagation
        error = y - output_layer_output
```

```
        d_output = error * sigmoid_derivative(output_layer_output)

        error_hidden_layer = d_output.dot(weights_hidden_output.T)
        d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

        # Update weights
        weights_hidden_output += hidden_layer_output.T.dot(d_output) * learning_rate
        weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate

        # Print the mean squared error at each epoch
        if epoch % 1000 == 0:
            mse = np.mean(np.square(error))
            print(f"Epoch {epoch}, Mean Squared Error: {mse}")

    print("Training complete.")

if __name__ == "__main__":
    main()
```

Output:

```
Enter the number of samples:
2
Enter the number of features:
2
Enter the values for the dataset (one sample per line):
12 2212
65 48
Enter the number of neurons in the hidden layer: 2
Enter the learning rate: 2
Enter the number of epochs: 8
Epoch 0, Mean Squared Error: 2446324.768146071
Training complete.
```

Result: The given program has been executed successfully