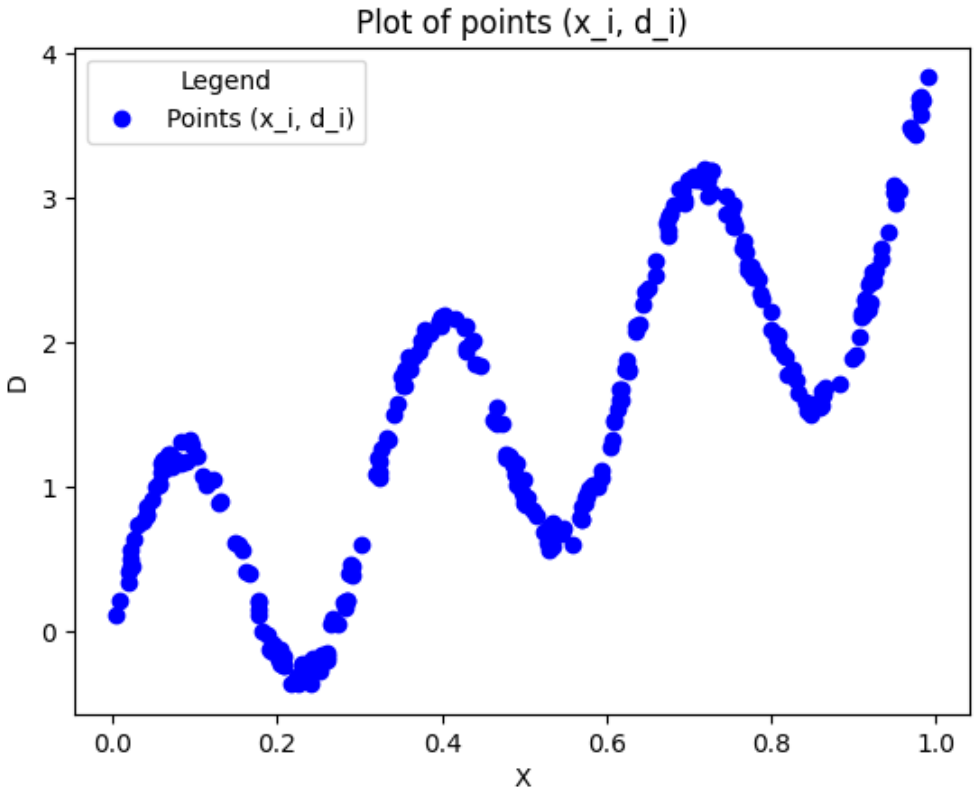
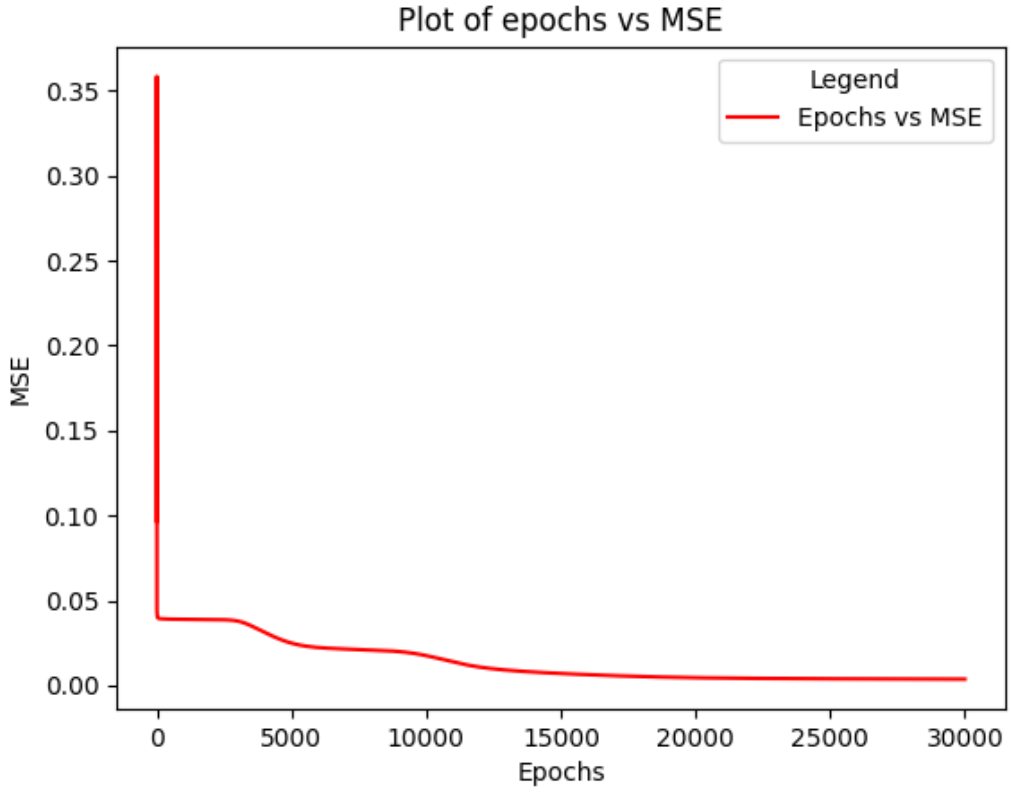
Name: Sai Anish Garapati
UIN: 650208577
Assignment 5:

Q1)

3)

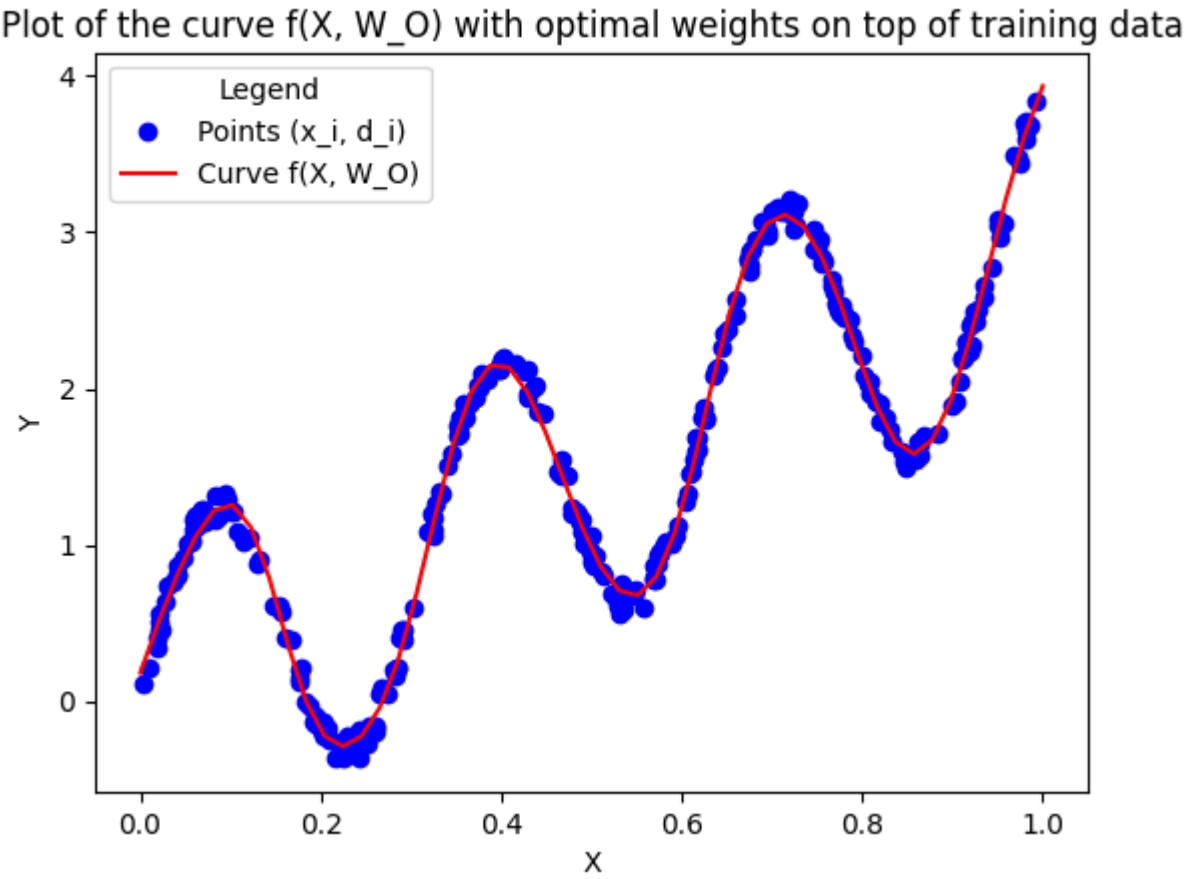


4)



5) Choice of eta

- eta_1 = 0.006 (for training weights in the first layer)
- eta_2 = 0.0001 (for training weights in the output layer)



-> Gradient descent is terminated after 30000 iterations with final MSE = 0.003726364084759235

6) Pseudo code:

function gradient_descent():

1. Initialize Weights randomly uniformly between -1 and 1
2. X = Append column of 1's to X
3. Epochs = 0
4. Training until convergence or till 30000 epochs:
   a. Calculate layer_1_fields = $W_{layer\_1}.X^T$
   b. Calculate layer_1_output = append row of 1's to $\tanh(layer\_1\_fields)$ which is the new input to output layer
   c. Calculate Y = $(W.layer\_1\_output)^T$
   d. for i = 0 to n:
      i. $W_{layer_1} = W_{layer_1} - eta\_1 * (-(d-y) * (X.(W_{output\_layer}.*(1-tanh^2(layer\_1\_fields)))^T))$
      ii. $W_{output\_layer} = W_{output\_layer} - eta\_2 * (-layer\_1\_output * (d-y))$
   e. epochs += 1

Python code:

```
# Name: Sai Anish Garapati
# UIN: 650208577
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(55)

# Mean Square Error
def mean_square_error(n, D, Y):
    return 1/n * np.linalg.norm(D - Y, 2)

# Backpropagation to find gradient vector
def backpropagation(N, x, d, W, layer_1_fields, layer_1_output, y):
    output_layer_g = -layer_1_output*(d - y)
    layer_1_g = -(d - y)*np.dot(x, np.multiply(W[2*N + 1:].reshape(N, 1), (1 -
(np.tanh(layer_1_fields))**2)).T)
    return np.append(layer_1_g.T.reshape(2*N, 1), output_layer_g).reshape(3*N + 1, 1)

# gradient descent algorithm
def gradient_descent(n, N, X, D):
    W = np.random.uniform(-1, 1, size=(3*N + 1, 1))
    X = np.append(np.ones(shape=(n, 1)), X, axis=1)
    epochs = 0
    # eta_1 for training weights in first layer
    eta_1 = 0.006
    # eta_2 for training weights in output layer
    eta_2 = 0.0001
    mse = []
    while (True):
        layer_1_fields = np.dot(W[:2*N].reshape(N, 2), X.T)
        layer_1_output = np.append(np.ones(shape=(1, n)), np.tanh(layer_1_fields), axis=0)
        Y = np.dot(W[2*N:].reshape(1, N+1), layer_1_output).T
        mse.append(mean_square_error(n, D, Y))
        print('eta_1:{} eta_2:{} epoch: {} cost: {}'.format(eta_1, eta_2, epochs, mse[-1]))
        if (epochs == 30000):
            break
        if (len(mse) > 1 and mse[-1] >= mse[-2]):
            eta_1 -= 0.00001

        if (len(mse) > 1 and abs(mse[-1] - mse[-2]) <= 1e-9):
            break

        for i in range(0, n):
            g = backpropagation(N, X[[i], :].reshape(2, 1), D[i], W, layer_1_fields[:, [i]],
layer_1_output[:, [i]], Y[i])
            # Updating weights in first layer
            W[:2*N] = W[:2*N] - eta_1*g[:2*N]
            # Updating weights in output layer
            W[2*N:] = W[2*N:] - eta_2*g[2*N:]
        epochs += 1

    return W, epochs, mse
```

```
if __name__ == '__main__':
    n = 300
    N = 24
    X = np.random.uniform(0, 1, size=(n, 1))
    V = np.random.uniform(-0.1, 0.1, size=(n, 1))

    D = np.sin(20*X) + 3*X + V

    # Plot of training data
    plt.title('Plot of points (x_i, d_i)')
    plt.xlabel('X')
    plt.ylabel('D')
    plt.plot(X, D, 'bo', label='Points (x_i, d_i)')
    plt.legend(title='Legend')
    plt.show()

    W_optimal, epochs, mse = gradient_descent(n, N, X, D)

    # Plotting learned curve on the range (0, 1)
    X_1 = np.linspace(0, 1)
    l = len(X_1)
    X_1 = np.asarray(X_1).reshape(l, 1)
    X_1 = np.append(np.ones(shape=(l, 1)), X_1, axis=1)
    layer_1_fields = np.dot(W_optimal[:2*N].reshape(N, 2), X_1.T)
    layer_1_output = np.append(np.ones(shape=(1, l)), np.tanh(layer_1_fields), axis=0)
    Y = np.dot(W_optimal[2*N:].reshape(1, N+1), layer_1_output).T

    # Plot of epochs vs MSE
    plt.title('Plot of epochs vs MSE')
    plt.xlabel('Epochs')
    plt.ylabel('MSE')
    plt.plot([i for i in range(0, epochs + 1)], mse, 'r', label='Epochs vs MSE')
    plt.legend(title='Legend')
    plt.show()

    # Plot of curve f(X, W_O) on top of training data
    plt.title('Plot of the curve f(X, W_O) with optimal weights on top of training data')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.plot(X, D, 'bo', label='Points (x_i, d_i)')
    plt.plot([x[1] for x in X_1], Y, 'r', label='Curve f(X, W_O)')
    plt.legend(title='Legend')
    plt.show()
```