IE 440 - NONLINEAR MODELS IN OPERATIONS RESEARCH

Homework # 5

24.12.2023

Regression & MLP

TEAM OPTIMIZERS

Fatmanur Yaman - 2019402204

Hüseyin Emre Bacak - 2021402279

Question 1. Least Square Method

a. **Linear Regression Model**

The objective is minimizing the mean squared error function. The regression model is constructed by using x values in the data. The corresponding y values were used in the minimizing the objective function process.

b. **Polynomial Regression Model**

The objective is minimizing the mean squared error function. The regression model is constructed by using x and x^2 values in the data. The corresponding y values were used in the minimizing the objective function process.

- **Parameters Used**

The data frame given to the model, the degree of the regression, the model that is used, the epoch number, the learning rate, and the loss function are the parameters that have been used in Question 1.

- **Results**

Linear Regression Model Results:

```
In [64]: linear_reg_model_results = learning_model(df, 2, linear_regression, 20000, 0.5, mean_sqaured_error)
epoch: 1
Epoch 1/20000 - Train Loss: 0.4482 - Val Loss: 0.1258
New best model saved.Epoch 1With loss 0.12583807730116764
epoch: 2
Epoch 2/20000 - Train Loss: 0.1078 - Val Loss: 0.1232
New best model saved.Epoch 2With loss 0.12319007844763315
epoch: 3
Epoch 3/20000 - Train Loss: 0.1040 - Val Loss: 0.1235
```

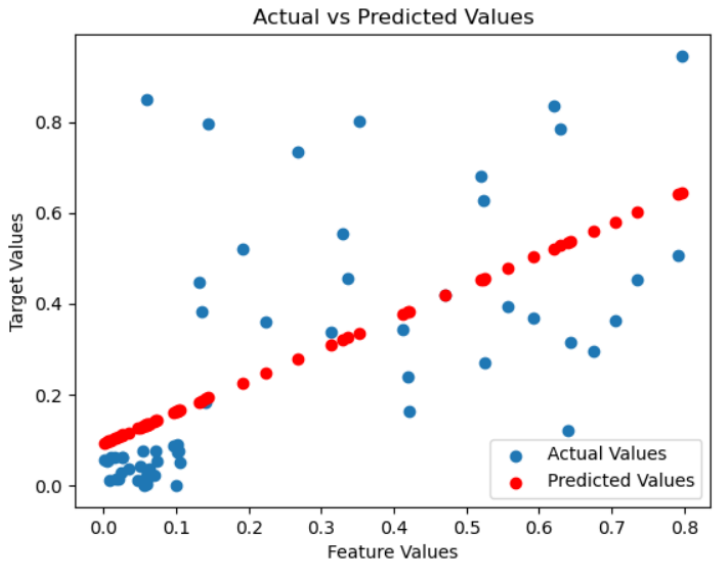
There are 200 epochs that iterate the dataset and the learning rate is 0,5. The MSE is used as an error function. The results are in the below:

```
In [65]: linear_reg_model_results
Out[65]: {'Test MSE': 0.10735390810582557,
          'Test Variance': 0.00462803670415804,
          'Training SSE': 6.519777153031311,
          'Weights': array([0.11354108, 0.59706048]),
          'Test Predictions': array([0.17614558, 0.55449468, 0.53486729, 0.1445206 , 0.1435579 ,
                 0.49789844, 0.30975106, 0.38699935, 0.42571516, 0.1282072 ,
                 0.19399106, 0.11939938, 0.17361914, 0.1509515 , 0.22850899,
                 0.12507402, 0.58918957, 0.1545058 , 0.44563971, 0.12854019,
                 0.35934569, 0.42307477, 0.13649604, 0.14887161]),
```

The Test MSE: 0.11, The Test Variance: 0.005, The Training SSE: 6.52

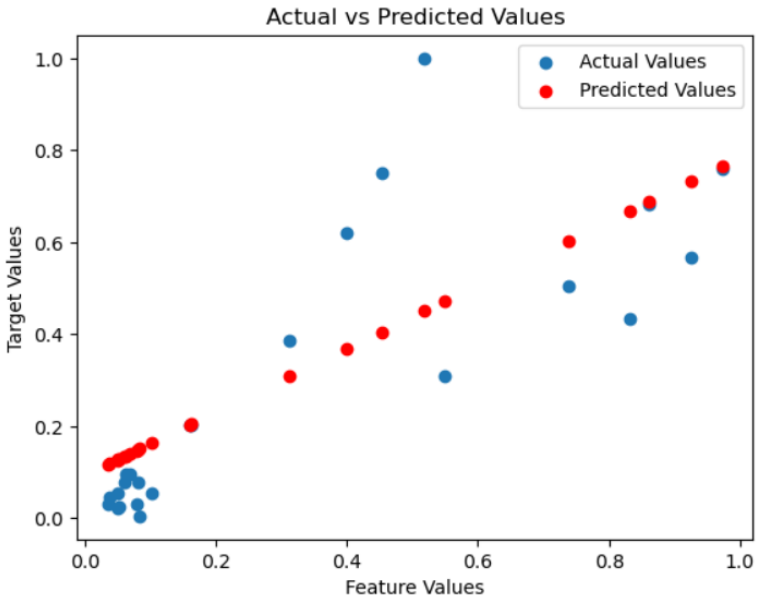
Linear Regression: Training Data & Regression Function

```
In [45]: plot_results(linear_reg_model_results['Train X'],
                      linear_reg_model_results['Train Y'],
                      linear_reg_model_results['Train Predictions'])
```



Linear Regression: Test Data & Regression Function

```
In [44]: plot_results(linear_reg_model_results['Test X'],
                      linear_reg_model_results['Test Y'],
                      linear_reg_model_results['Test Predictions'])
```



Polynomial Regression Results:

```
In [67]: polynomial_reg_model_results = learning_model(df, 3, polynomial_regression, 10000, 0.5, mean_sqaured_error)

         epoch: 1
         Epoch 1/10000 - Train Loss: 1.0021 - Val Loss: 0.0670
         New best model saved.Epoch 1With loss 0.06703549462494307
         epoch: 2
         Epoch 2/10000 - Train Loss: 0.1419 - Val Loss: 0.0933
```

There are 100 epochs that iterate the dataset and the learning rate is 0,5. The MSE is used as an error function. The results are in the below:

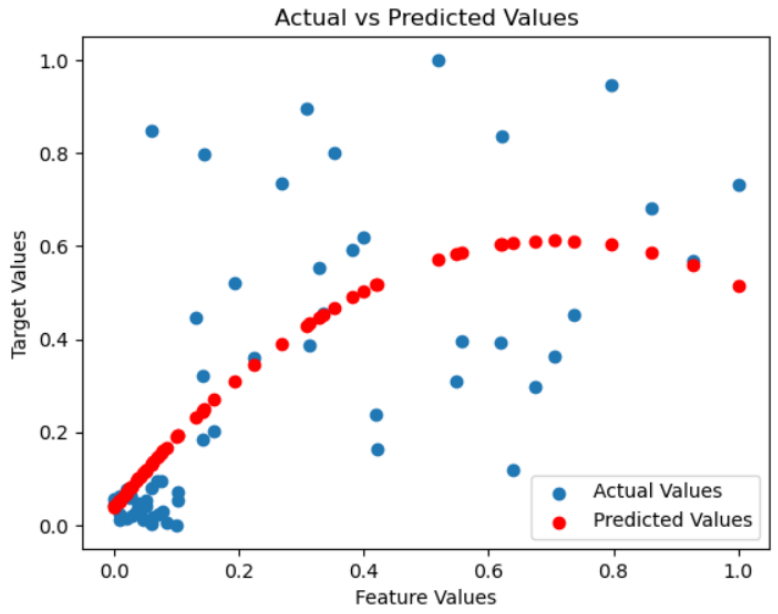
```
In [68]: polynomial_reg_model_results

Out[68]: {'Test MSE': 0.11031876244860889,
          'Test Variance': 0.003733228714103682,
          'Training SSE': 8.513092560649522,
          'Weights': array([ 0.0859916 , -0.11545697,  0.86592295]),
          'Test Predictions': array([0.7931043 , 0.082182  , 0.19026941, 0.08218143, 0.13451684,
                 0.08528663, 0.08223546, 0.17811373, 0.34756428, 0.08367801,
                 0.6869981 , 0.14492153, 0.0822117 , 0.08314143, 0.0821686 ,
                 0.08694788, 0.08215544, 0.08318486, 0.08305041, 0.0821594 ,
                 0.08236767, 0.28336672, 0.08498647, 0.08231548]),
```

The Test MSE: 0.11, The Test Variance: 0.004, The Training SSE: 8.51

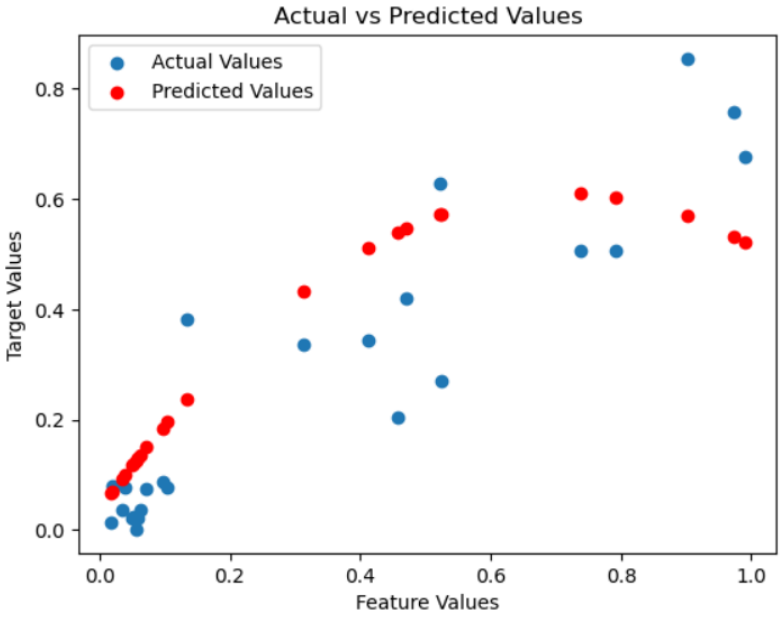
Polynomial Regression: Training Data & Regression Line

```
In [54]: plot_results(polynomial_reg_model_results['Train X'],
                     polynomial_reg_model_results['Train Y'],
                     polynomial_reg_model_results['Train Predictions'])
```



Polynomial Regression: Test Data & Regression Line

```
In [53]: plot_results(polynomial_reg_model_results['Test X'],
                      polynomial_reg_model_results['Test Y'],
                      polynomial_reg_model_results['Test Predictions'])
```



- **The Source Code:**

  - linear_regression: Performs simple linear regression operation.
  - polynomial_regression: Performs polynomial regression by using x^2 values.
  - mean_squared_error: Computes the mean squared error by taking the mean of the squares of the differences between predicted and actual values.
  - training: Runs the linear regression model, takes the derivative of the error function to update the weights.
  - validation: Makes one forward pass by using the learned weights in training and computes the loss.
  - learning_model: Takes the functions above to predict the new values, gives values like mse, variance, weights, and predictions.
  - data_loader: By taking the degree of the regression model, prepares the dataset for the training.

[5]:
```
def linear_regression(W, X):
    y_predicted = np.dot(X, W)
    return y_predicted

def polynomial_regression(W, X):
    w0, w1, w2 = W
    y_predicted = w2 * X[:, 2] + w1 * X[:, 1] + w0
    return y_predicted

def mean_sqaured_error(Y, y_predicted):
    loss = np.mean((Y - y_predicted)**2)
    return loss
```

```
def training(X, Y, W, model, learning_rate, criterion):

    #Zero The Gradients:
    gradient = np.zeros(W.shape)

    #Run The Model - Forward Pass:
    y_predicted = model(W,X)

    #Compute The Loss with Mean Squared Error:
    loss = criterion(Y, y_predicted)

    #Backpropagation:
    gradient = -2 * np.dot(X.T, (Y.ravel() - y_predicted))/len(Y)

    #Update The Weights:
    W = W - learning_rate * gradient

    return dict([('weights', W),
                 ('loss', loss),
                 ('predictions', y_predicted)])
```

```
def validation(X, Y, W, model, criterion):

    #Run The Model - Forward Pass:
    y_predicted = model(W,X)

    #Compute The Loss with Mean Squared Error:
    loss = criterion(Y, y_predicted)

    return dict([('weights', W),
                 ('loss', loss),
                 ('predictions', y_predicted)])

def data_loader(df, degree):
    column_names = ["x", "y"]
    df = pd.read_csv(r"C:\Users\fatma\OneDrive\Masaüstü\regression_data.dat", sep=' ', names=column_names)

    scaler = MinMaxScaler()
    X = scaler.fit_transform(df["x"].values.reshape(-1,1))
    Y = scaler.fit_transform(df["y"].values.reshape(-1,1))

    df["x"] = X
    df["y"] = Y
    df['0'] = 1
```

```
    for i in range(1,degree):
        df[str(i)] = df['x'] ** i
    X = np.concatenate([df[str(j)].values.reshape(-1, 1) for j in range(0,int(degree))], axis=1)
    Y = df["y"].values.reshape(-1,1)

    # Assuming you have your features X and labels Y
    X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, test_size=0.4)

    # Split the temporary set into validation and test sets
    X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp, test_size=0.6)

    return [X_train, Y_train, X_val, Y_val, X_test, Y_test]
```

[60]:
```
def learning_model(df, n_features, model, nb_epoch, learning_rate, criterion):
    #The Random Initialization for The Weights:
    W = np.random.randn(n_features)

    trigger = 0
    best_val_loss = float('inf')

    data = data_loader(df,n_features)

    for epoch in range(nb_epoch):
        print("epoch:", epoch + 1)

        #Training
        train_results = training(data[0], data[1], W, model, learning_rate, criterion)

        # Validation
        val_results = validation(data[2], data[3], train_results['weights'], model, criterion)
```

```
        num_epochs = str(nb_epoch)
        print(f"Epoch {epoch + 1}/{nb_epoch} - "
          f"Train Loss: {train_results['loss']:.4f} - "
          f"Val Loss: {val_results['loss']:.4f}")

        # Check if the current model is the best so far
        if val_results['loss'] < best_val_loss + 0.001:
            best_val_loss = val_results['loss']
            # Save the model weights
            # Update the best weights
            best_weights = val_results['weights']
            np.save('best_weights.npy', best_weights)
            W = best_weights
            print("New best model saved." + f"Epoch {epoch + 1}" +f"With loss {best_val_loss}")

    training_results = validation(data[0], data[1], best_weights, model, criterion)
    test_results = validation(data[4], data[5], best_weights, model, criterion)
    if n_features == 3:
        test_variance = np.var([validation(X.reshape(1,3), Y, best_weights, model, criterion)['loss'] for X, Y in zip(data[4], da
    else:
        test_variance = np.var([validation(X, Y, best_weights, model, criterion)['loss'] for X, Y in zip(data[4], data[5])])

    print("Final Test Loss:", test_results['loss'])
```

```
    return dict([('Test MSE', test_results['loss']),
                 ('Test Variance', test_variance),
                ('Training SSE', 60*training_results['loss']),
                ('Weights', test_results['weights']),
                ('Test Predictions', test_results['predictions']),
                ('Test Y',  data[5]),
                ('Test X', data[4]),
                ('Train X', data[0]),
                ('Train Y' , data[1]),
                ('Train Predictions',train_results['predictions'])])
```

The code is designed for regression tasks, specifically linear and polynomial regression. It utilizes gradient descent for training and includes functions for data loading, model training, and validation. The learning process is monitored, and the best model is saved based on validation loss. The final test loss, variance, and other relevant information are reported at the end of the training process.

Question 2. Nonlinear Regression

- **Parameters Used**

The data frame given to the model, the data loader, the dataframe, the input size, the hidden size, the epoch number, the learning rate, the loss function, and the learning rate multiplier are the parameters that have been used in Question 1. The hidden size optimization has been done.

- **Results of the Code**

MLP With 1 Input Unit Model Results:

```
In [73]: mlp_model_results = training_mlp(data_loader_1, df, 1, 3, 5000, 0.5, mean_sqaured_error, 0.9)

         Epoch 0/5000 - Loss: 0.0037868632675621934
         New best model saved.Epoch 1With loss 0.0037868632675621934
         Epoch 100/5000 - Loss: 0.0017577716135764759
         Epoch 200/5000 - Loss: 0.0019821771927129943
         Epoch 300/5000 - Loss: 0.0023266099972085764
         Epoch 400/5000 - Loss: 0.002485535526055076
         Epoch 500/5000 - Loss: 0.002584423383535972
         Epoch 600/5000 - Loss: 0.0026626099016677186
```

There are 5000 epochs that iterate the dataset and the initial learning rate is 0,5. The MSE is used as an error function. The results are in the below:

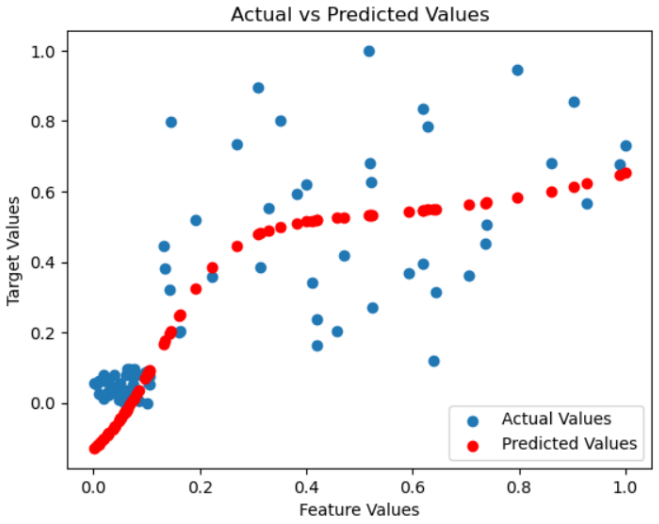
```
In [74]: mlp_model_results

Out[74]: {'Test MSE': 0.05864729063623083,
          'Test Variance': 0.026595870677969657,
          'Training SSE': 3.200119851974068,
          'Weights': array([ 0.98615377,  0.94907831,  0.60077753, -0.25148248, -0.29301353,
                 -0.27523531,  0.43465059,  0.67218129,  0.56717331, -0.69322984]),
          'Test Predictions': [array([[0.59132119]]),
           array([[0.53728938]]),
           array([[0.02135268]]),
```

The Test MSE: 0.059 , The Test Variance: 0.027, The Training SSE: 3.20

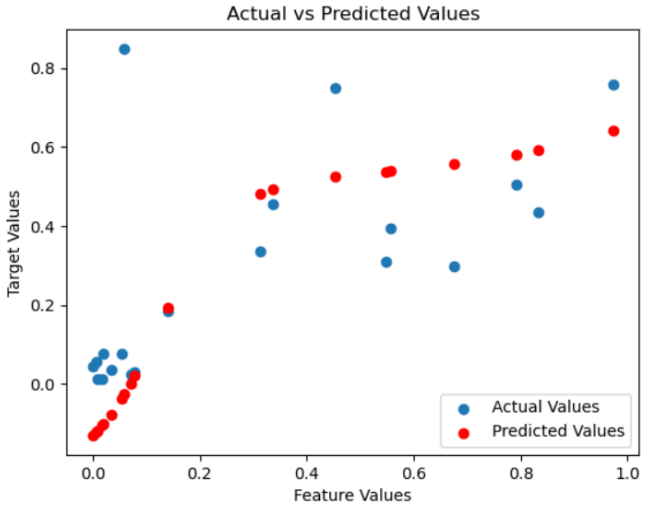
MLP With 1 Input Unit Model: Training Data & Regression Line

```
In [76]: plot_results(mlp_model_results['Train X'],
                      mlp_model_results['Train Y'],
                      mlp_model_results['Train Predictions'])
```



MLP With 1 Input Unit Model: Test Data & Regression Line

```
In [82]: plot_results(mlp_model_results['Test X'],
                      mlp_model_results['Test Y'],
                      mlp_model_results['Test Predictions'])
```



MLP With 2 Input Units Model Results:

```
In [77]: mlp_model_results_2 = training_mlp(data_loader_2, df, 2, 3, 5000, 0.5, mean_sqaured_error, 0.9)

         Epoch 0/5000 - Loss: 0.5450815600021557
         New best model saved.Epoch 1With loss 0.5450815600021557
         Epoch 100/5000 - Loss: 0.18713630232713924
         Epoch 200/5000 - Loss: 0.18053772442084415
```

There are 5000 epochs that iterate the dataset and the initial learning rate is 0,5. The data_loader_2 is used as the data requires another column as x^2. The MSE is used as an error function. The results are in the below:

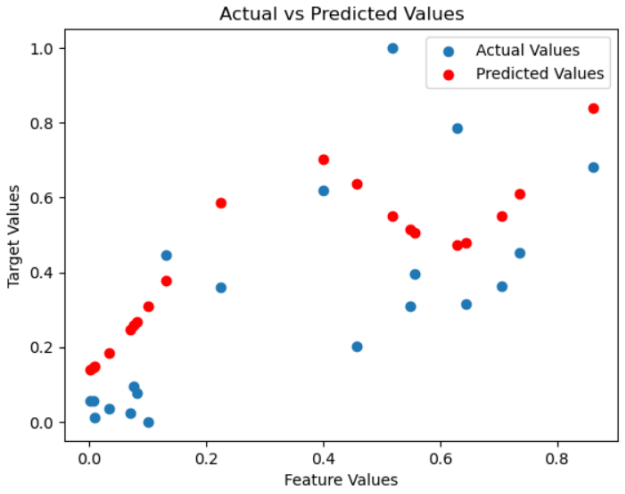
```
In [78]: mlp_model_results_2

Out[78]: {'Test MSE': 0.04886576393303674,
          'Test Variance': 0.0029858758962448303,
          'Training SSE': 3.7902930656361473,
          'Weights': array([ 0.27515433,  0.77416172,  0.47139188,  0.57313076,  0.123971  ,
                  0.6792633 , -0.22815325, -0.21574784, -0.19100724,  0.54263679,
                  0.44236537,  0.29514763, -0.44235161]),
          'Test Predictions': [array([[0.47751004]]),
           array([[0.63574465]]),
           array([[0.51320778]]),
```

The Test MSE: 0.049 , The Test Variance: 0.003, The Training SSE: 3.79

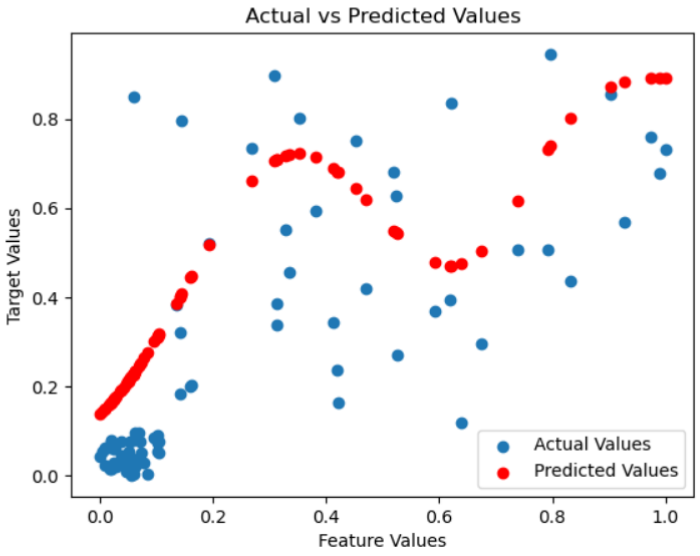
MLP With 2 Input Units Model: Test Data & Regression Line

```
In [83]: plot_results(mlp_model_results_2['Test X'][:,0],
                      mlp_model_results_2['Test Y'],
                      mlp_model_results_2['Test Predictions'])
```



MLP With 2 Input Units Model: Training Data & Regression Line

In [79]:
```
plot_results(mlp_model_results_2['Train X'][:,0],
             mlp_model_results_2['Train Y'],
             mlp_model_results_2['Train Predictions'])
```



The Source Code:

In [18]:
```
def data_loader_1(df):
    column_names = ["x", "y"]
    df = pd.read_csv(r"C:\Users\fatma\OneDrive\Masaüstü\regression_data.dat", sep=' ', names=column_names)

    scaler = MinMaxScaler()
    X = scaler.fit_transform(df["x"].values.reshape(-1,1))
    Y = scaler.fit_transform(df["y"].values.reshape(-1,1))

    # Assuming you have your features X and labels Y
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

    return [X_train, Y_train, X_test, Y_test]

def data_loader_2(df):
    column_names = ["x", "y"]
    df = pd.read_csv(r"C:\Users\fatma\OneDrive\Masaüstü\regression_data.dat", sep=' ', names=column_names)

    df["x2"] = df['x'] ** 2
    y = df["y"]
    x = df.drop("y", axis=1)

    scaler = MinMaxScaler()
    X = scaler.fit_transform(x)
    Y = scaler.fit_transform(y.values.reshape(-1,1))

    # Assuming you have your features X and labels Y
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

    return [X_train, Y_train, X_test, Y_test]
```

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def parameter_init(input_size, J_initial):
    #Initiallize The Parameters
    W1 = np.random.rand(J_initial, input_size)
    b1 = np.zeros((J_initial, 1))
    W2 = np.random.rand(1, J_initial)
    b2 = np.zeros((1, 1))
    return  W1, b1, W2, b2

def forwardprop(W1, X, W2, Y, b1, b2, criterion):
    #Forward Propagation
    Z1 = np.dot(W1, X.reshape(-1, 1)) + b1
    A1 = sigmoid(Z1)

    Z2 = np.dot(W2, A1) + b2
    A2 = Z2  # Linear activation for regression

    #Compute The Loss with Mean Squared Error:
    loss = criterion(Y, A2)
    return Z1, A1, Z2, A2, loss
```

```
def backprop(Z1, A1, Z2, A2, X, Y, W1, b1, W2, b2, learning_rate):
    #Backpropagation:
    # Compute gradients for the output layer
    dZ2 = A2 - Y.reshape(-1, 1)
    dW2 = np.dot(dZ2, A1.T)
    db2 = np.sum(dZ2, axis=1, keepdims=True)

    # Compute gradients for the hidden layer
    dZ1 = np.dot(W2.T, dZ2) * (A1 * (1 - A1))
    dW1 = np.dot(dZ1, X.reshape(-1, 1).T)
    db1 = np.sum(dZ1, axis=1, keepdims=True)

    #Update The Weights:
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    return W1, b1, W2, b2
```

In [72]:
```
from statistics import mean
def training_mlp(data_loader, df, input_size, hidden_size, num_epochs, learning_rate, criterion, lr_multip):
    #Load the data.
    data = data_loader(df)
    hidden_size_counter = 1
    X_train = data[0]
    Y_train = data[1]
    X_test = data[2]
    Y_test = data[3]

    best_train_loss = float('inf')
    test_loss_previous = float('inf')
    test_loss = 100
    #initial_learning_rate = learning_rate
    while test_loss < test_loss_previous:
        W1, b1, W2, b2 = parameter_init(input_size, hidden_size)
        learning_rate = initial_learning_rate
        for epoch in range(num_epochs):
            total_loss = 0
            for X,Y in zip(X_train,Y_train):
                # Forward propagation & Loss
                Z1, A1, Z2, A2, loss = forwardprop(W1, X, W2, Y, b1, b2, criterion)
                total_loss += loss
                # Backward propagation & Updating The Weights
                W1, b1, W2, b2 = backprop(Z1, A1, Z2, A2, X, Y, W1, b1, W2, b2, learning_rate)
```

```
        #Update the learning rate.
        learning_rate = learning_rate * lr_multip
        if epoch % 100 == 0:
            print(f"Epoch {epoch}/{num_epochs} - Loss: {loss}")

        avg_loss = (total_loss)
        if avg_loss < best_train_loss + 0.001:
                    best_train_loss = loss
                    # Save the model weights
                    # Update the best weights
                    best_weights = np.concatenate([W1.flatten(), b1.flatten(), W2.flatten(), b2.flatten()])
                    np.save('best_weights.npy', best_weights)
                    print("New best model saved." + f"Epoch {epoch + 1}" +f"With loss {best_train_loss}")

    train_predictions = [forwardprop(W1, X, W2, Y, b1, b2, criterion)[2] for X, Y in zip(X_train, Y_train)]

    test_loss_previous = test_loss

    test_predictions = [forwardprop(W1, X, W2, Y, b1, b2, criterion)[2] for X, Y in zip(X_test, Y_test)]
    test_loss = mean([forwardprop(W1, X, W2, Y, b1, b2, criterion)[4] for X, Y in zip(X_test, Y_test)])
    test_variance = np.var([forwardprop(W1, X, W2, Y, b1, b2, criterion)[4] for X, Y in zip(X_test, Y_test)])
```

```
    print("Final Test Error:", test_loss)
    print("Final Test Variance:", test_variance)
    print("Hidden Layer Size:", hidden_size)

    hidden_size += 1
    hidden_size_counter += 1

return dict([('Test MSE', test_loss),
             ('Test Variance', test_variance),
             ('Training SSE', avg_loss),
             ('Weights', best_weights),
            ('Test Predictions', test_predictions),
            ('Test Y',  data[3]),
            ('Test X', data[2]),
            ('Train X', data[0]),
            ('Train Y' , data[1]),
            ('Train Predictions',train_predictions),
            ('Hidden Layer Size', hidden_size)])
```

- sigmoid: Performs the sigmoid function with given x values..
- forwardprop: Makes the forward pass of the data from the input units to the output unit.
- backprop: Computes the gradients for each layer and updates the corresponding weights.
- training_mlp: Takes the data to predict the new values, gives values like mse, variance, weights, and predictions.
- data_loader_1: Prepares the data for the usage of 1 Input Unit MLP.
- data_loader_2: Prepares the data for the usage of 2 Input UnitS MLP.

The code trains an MLP regression model with a varying hidden layer size to find the optimal architecture. It uses a custom training loop that adjusts the learning rate and hidden layer size during training. The training stops when the test loss starts to increase, indicating potential overfitting. The code reports the final test mean squared error, variance, and other relevant information. The model architecture and weights are saved for the best-performing model.

Results:

	Training SSE	Test MSE	Test Variance
0	6.519777	0.107354	0.004628
1	8.513093	0.110319	0.003733
2	3.200120	0.058647	0.026596
3	3.790293	0.048866	0.002986

The Training SSE, Test MSE, and Test Variance values regarding the Linear Reg, Multiple Reg, 1 Input Unit MLP, and 2 Input Units MPL are given in the table above. Since there was too much variance on the dataset, the simple linear regression model and the polynomial model do not fit well to the dataset. The largest Test MSE's are in those models. The test MSE is the lowest at the 2 Input Units MPL. In addition to this, the test variance is the lowest which indicates that the error is less spread around zero. Also MLP models perform better on the training datasets as it can be seen in Training SSE's.

Both linear and polynomial regression models have similar Test MSE values, indicating comparable performance in terms of prediction accuracy. The MLP models outperform both linear and polynomial regression models in terms of Test MSE. Lower Test MSE values suggest better predictive performance. Among the MLP models, the one with 2 input units performs slightly better (lower Test MSE) compared to the one with 1 input unit. Training SSE is relatively lower for the MLP models compared to traditional regression models, indicating better fit to the training data.

To sum up, The MLP models, especially the one with 2 input units, demonstrate better predictive performance as evidenced by lower Test MSE values compared to linear and polynomial regression models. The choice of model depends on the specific requirements, but in terms of predictive accuracy, the MLP models appear to be more effective in this particular dataset. It's essential to consider trade-offs between model complexity, interpretability, and predictive performance when choosing a model for a specific task.