GE461 Spring2024/25 Supervised Learning Project

Preprocessing

In this project a small dataset is provided. Test and train data are loaded from train1.txt and test1.txt files. Training set input and outputs are obtained by slicing the training data and test inputs and outputs are obtained by slicing the test data. Afterwards, batch normalizaiton applied to test and train set seperately to prevent information leakeage from training set. It is important to mention that test and train set are carefully handled throught the assignment in order to prevent information leakage.

Part a)

To start with, a linear regressor is implemented. Beforehand the dataset is observed to be follow some kind of polynomial function so it is expected to have poor performance with the linear regressor. Figure 1 demonstrates the result of the line generated by the linear regressor vs the true training data outputs. As a response Universal Approximation Theorem can be referred. Universal Approximation Theorem states that any nonlinear relation can be approximated with 1 hidden layer ANN, hence for this application a single hidden layer ANN will be sufficient to obtain a polynomial-wise relation. The number of hidden units can be determined experimentally which will be demonstrated in the following sections.

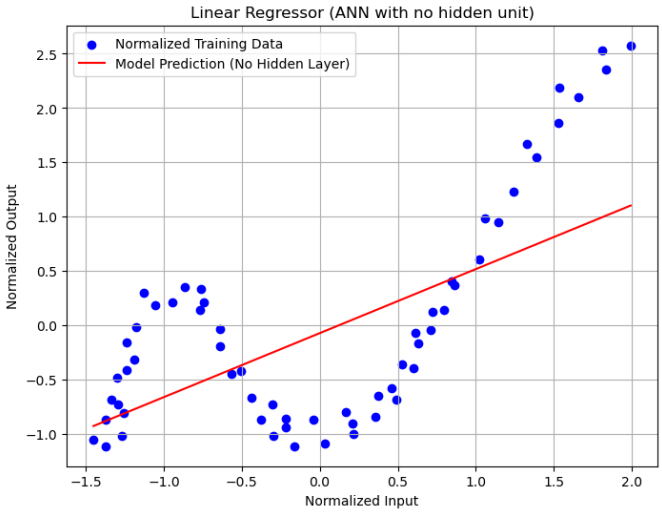
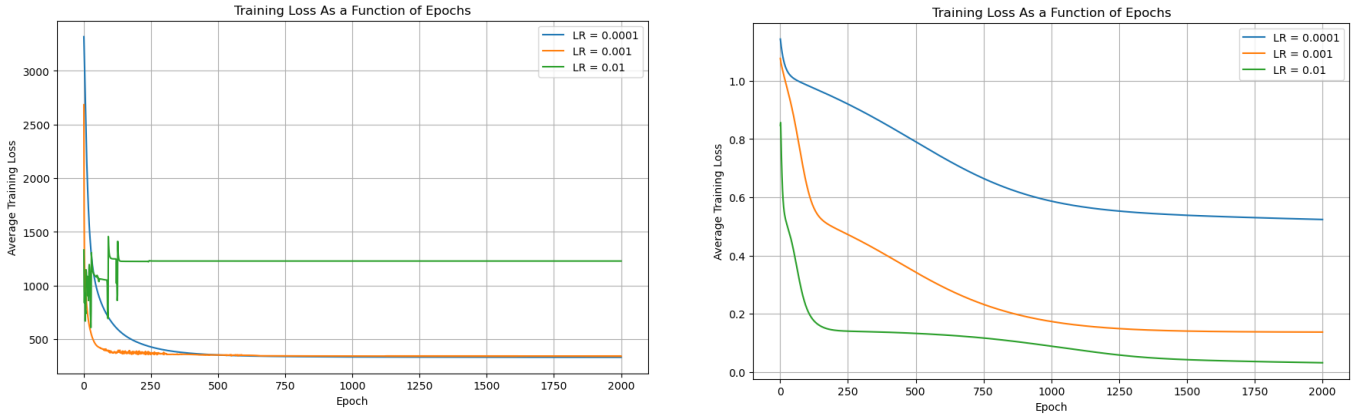


Figure 1 Linear regressor output vs training true labels

The learning rate is important due to obtain fast convergance and to solve the ‘stuck in local minima’ problem. In Figure 2 (b) training loss vs epoch plot for different learning rates are provided. As it can be observed, as learning rate becomes smaller the convergance process extends with more careful steps, additionally the 0.0001 learning rate may even be an indicator of stuck at local minima problem. In addition, larger values of learning rates cause ‘jump’ between equal error contours due to larger step size which makes the ANN learn nothing at all. Large learning rates can also be problematic in terms of exploding gradient problem. Hence as a moderate value with fast convergance 0.01 learning rate can be selected. The learning rate can be determined by inspection without any validation methods due to the explained reasons above. Moreover, normalization plays an important role in the ANN. Figure 2 (a) demonstrates the training loss without normalization, the figure is an explicit indicator of exploding graient problem due to large input values and ANN is not able to learn the weights, by normalization this problem is solved which can be observable in Figure 2 (a).



(a) (b)

Figure 2 Training Loss vs Epochs with different learning rates with (a) and without (b) Batch Normalization

The model should also be a generalizable model in other words overfitting underfitting should be considered. The no hidden unit model underfits to data hence a more compex model with a single hidden layer is proposed. However as Figure 2 (b) is observed training loss approaches to zero which mans the model memorizes the training hence, making it overfit to the training set. As a result a number of epoch should be determined such that the model is generalizable. Observe Figure 3, the test loss stops decreasing and increases with a small rate of change (small rate of change is related to the similarity with the training set). According to Figure 3 the desired number of epochs can be determined as 2000. In other words training should be stopped at 2000 epochs to achieve best computational efficiency and a generalizable model.

To mention about the initialization of weights and biases, random uniform initialization between -0.5 and 0.5 is used. The upper and lower bounds of the uniform sampling is again crucial for exposing the model to explpding gradients problem, with the [-50,50] random sampling a similar plot compared to Figure 2 (a) is observed. Also, Xavier initialization is tested howeever no significant changes are observed due to both [-0.5,0.5] random sampling and Xavier initialization are stable.

To obtain an insight about the number of hidden units, LOOCV is performed by fixing the learning rate to 0.01 and number of epochs to 2000. The hidden units are 2, 4, 8, 16, 32. The corresponding LOOCV loss calculations are as 0.0980, 0.0421, 0.0381, 0.0426, 0.0411 respectively which is also an indicator of generalizable model concept with moderate complexity which will be eleborated aslo on part c.

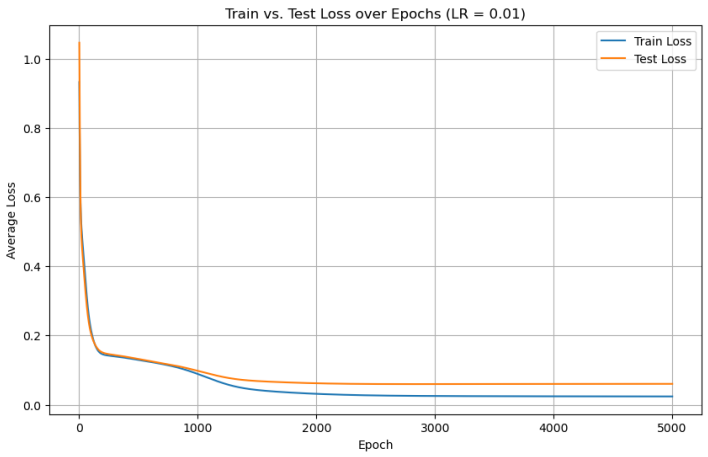


Figure 3 Test loss and training loss demonstration with fixed number of epochs and learning rate

Part b)

As the result of discussions and findings of the previous part, information of the model can be found as following:

ANN used: 8 hidden units

Learning rate: 0.01

Range of initial weights: [-0.5,0.5] with uniform random sampling

Number of epochs: When to stop: 2000 epochs

Is normalization used: Batch normalization used

Training loss (averaged over training instances): 0.039755069722146326

Test loss (averaged over test instances): 0.06840610377310763

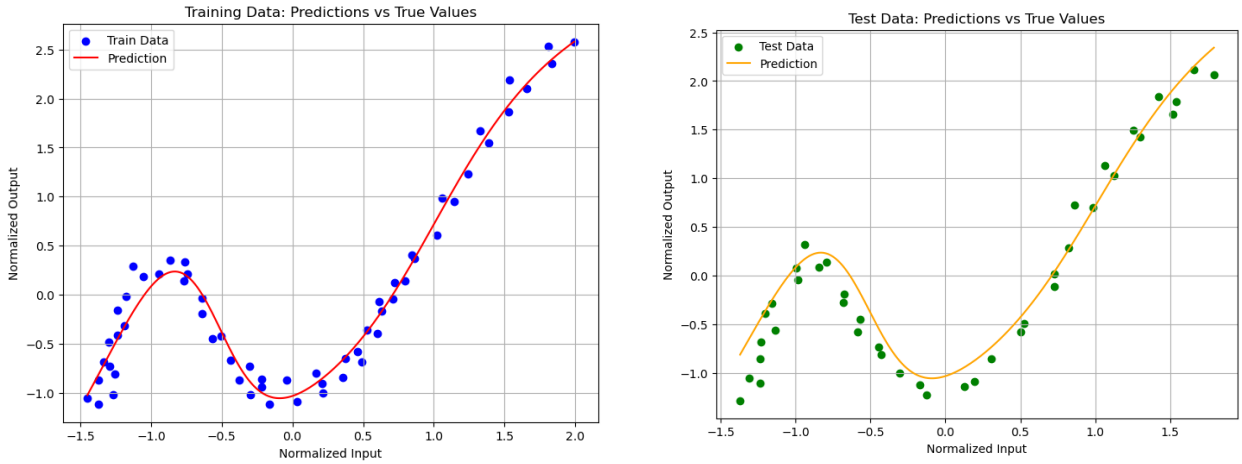


Figure 4 Demonstration of prediction curves of test and training data in seperate plots

Our goal is to obtain a generalizable model. Figure 4 indicates that the model can be considered as generalizable due to the fact that the test curve follows the patterns of test true label in some way. Hyperparameters in this case are determined as a result of the discussion in part a hence Figure 3 demonstrates the training and test loss for the particular model in part b. The ongoing discussion about the generalizable model is continued in part c.

Part c)

In this part to make the influence of the number of hidden units more observable the number of epochs for all ANN's are decreased to 1300 epochs to better observe the changing shapes of the curves as complexity (number of hidden units) increases. In figure 5 5 different hidden units are tested and the linear regressor output can already be observed from Figure 5. Also in the question it is desired to plot the hidden units for each ANN which provies a better understanding of how the seperation works. After observing the outputs it is clear to see that as the complexity increases it better suits to training output samples. As the number of hidden units increase the ability of the model to describe a nonlinear relationship increases. In other words the model memorizes the training data better as it gets more complex. However if the test curves were also plotted the result would be different. Due to memorizing higher complexity ANN's would perform poorly compared to moderate complexity (i.e. 8 hidden unit case). This outcome can also be verified from Table 1. The assignment was benefitial and educating in terms of evaluating models, grasping supervised learning and scratch implementation.

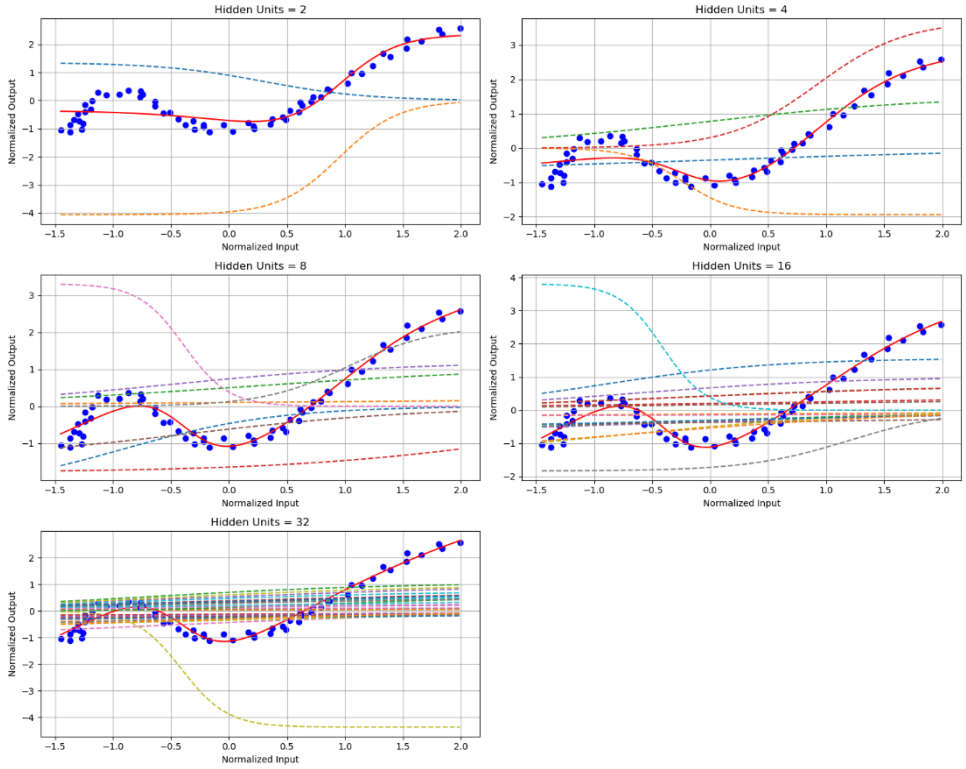


Figure 5 Prediction curves over training set demonstration with different number of hidden units (dotted lines are hidden units)

Hidden Units	Averaged Training Loss	Std Deviation of Training Loss	Averaged Test Loss	Std Deviation of Test Loss
2	0.050469	0.062150	0.076668	0.103540
4	0.020693	0.033954	0.057243	0.081622
8	0.020255	0.034823	0.057017	0.082852
16	0.022511	0.036773	0.061947	0.085923
32	0.022791	0.037680	0.061994	0.084002

Table 1 Demonstration of desired information about losses

APPENDIX

Python Codes

```
import numpy as np
import matplotlib.pyplot as plt
train_data = np.loadtxt('train1.txt', dtype=np.float32)
test_data = np.loadtxt('test1.txt', dtype=np.float32)
train_in = train_data[:,0]
train_out = train_data[:,1]
print(train_in.shape)
plt.figure(figsize=(8, 6))
plt.scatter(train_in, train_out, color='blue', label='Training Data')
plt.show()
def normalize_data(data):
    mean = np.mean(data)
    std = np.std(data)
    return (data - mean) / std, mean, std

def compute_error(y, y_pred):
    return (y - y_pred) ** 2


X_train_norm, X_train_mean, X_train_std = normalize_data(train_in)
y_train_norm, y_train_mean, y_train_std = normalize_data(train_out)

w = np.random.uniform(-0.5, 0.5)
b = np.random.uniform(-0.5, 0.5)

rate = 0.01
epochs = 100

train_loss_array = []
for epoch in range(epochs):
    loss_in_epoch = 0
    for i in range(len(X_train_norm)) :

        x = X_train_norm[i]
        true_y = y_train_norm[i]
        y_hat = w*x +b
        err = true_y - y_hat
        train_loss = compute_error(true_y, y_hat)

        loss_in_epoch += train_loss
        w_gradient = -2*x*err
        b_gradient = -2* err

        w = w - rate * w_gradient
        b = b - rate * b_gradient
    epoch_loss = loss_in_epoch / len(X_train_norm)
```

```
    train_loss_array.append(epoch_loss)
    print(f"Epoch {epoch+1}/{epochs}, Train Loss: {epoch_loss:.4f}")

plt.figure(figsize=(8, 6))
plt.plot(range(1, epochs + 1), train_loss_array, marker='o', label='Training
Loss')
plt.xlabel("Epoch")
plt.ylabel("Average Training Loss")
plt.title("Training Loss Over Epochs")
plt.legend()
plt.grid(True)
plt.show()
plt.figure(figsize=(8, 6))
plt.scatter(X_train_norm, y_train_norm, color='blue', label='Normalized
Training Data')

x_line = np.linspace(np.min(X_train_norm), np.max(X_train_norm), 100)
y_line = w * x_line + b

plt.plot(x_line, y_line, color='red', label='Model Prediction (No Hidden
Layer)')
plt.xlabel('Normalized Input')
plt.ylabel('Normalized Output')
plt.title('Linear Regressor (ANN with no hidden unit)')
plt.legend()
plt.grid(True)
plt.show()
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def train_model(learning_rate, epochs=1000, hid_units=16):
    W_in = np.random.uniform(-0.5, 0.5, size=(hid_units,))
    b_in = np.random.uniform(-0.5, 0.5, size=(hid_units,))
    W_hid = np.random.uniform(-0.5, 0.5, size=(hid_units,))
    b_hid = np.random.uniform(-0.5, 0.5)

    #limit_in = np.sqrt(6 / (1 + hid_units))
    #W_in = np.random.uniform(-limit_in, limit_in, size=(hid_units,))
    #b_in = np.zeros(hid_units)

    #limit_hid = np.sqrt(6 / (hid_units + 1))
    #W_hid = np.random.uniform(-limit_hid, limit_hid, size=(hid_units,))
    #b_hid = 0.0
    #W_in = np.ones((hid_units,))
    #b_in = np.ones((hid_units,))

    #W_hid = np.ones((hid_units,))
    #b_hid = 1.0
    train_loss_array = []
```

```
    for epoch in range(epochs):
        loss_in_epoch = 0.0
        for i in range(len(X_train_norm)):
            x = X_train_norm[i]
            true_y = y_train_norm[i]
            #x = train_in[i]
            #true_y = train_out[i]
            z = W_in * x + b_in
            h = sigmoid(z)
            y_hat = np.dot(W_hid, h) + b_hid
            error = true_y - y_hat
            new_loss = compute_error(true_y, y_hat)
            loss_in_epoch += new_loss
            W_hid_grad = -2 * error * h
            b_hid_grad = -2 * error
            W_in_grad = -2 * error * W_hid * h * (1 - h) * x
            b_in_grad = -2 * error * W_hid * h * (1 - h)
            W_hid = W_hid - learning_rate * W_hid_grad
            b_hid = b_hid - learning_rate * b_hid_grad
            W_in = W_in - learning_rate * W_in_grad
            b_in = b_in - learning_rate * b_in_grad
        epoch_loss = loss_in_epoch / len(X_train_norm)
        train_loss_array.append(epoch_loss)
        #print(f"Epoch {epoch+1}/{epochs}, Average Train Loss:
{epoch_loss:.4f}")
    return train_loss_array, W_in, b_in, W_hid, b_hid

learning_rates = [0.0001, 0.001, 0.01]
epochs = 2000
results = {}
for lr in learning_rates:
    loss_curve,W_in, b_in, W_hid, b_hid = train_model(lr, epochs=epochs,
hid_units=16)
    results[lr] = loss_curve
    print(f"Completed training for learning rate: {lr}")

plt.figure(figsize=(10, 6))
for lr, loss_array in results.items():
    plt.plot(range(1, epochs + 1), loss_array, label=f"LR = {lr}")
plt.xlabel("Epoch")
plt.ylabel("Average Training Loss")
plt.title("Training Loss As a Function of Epochs")
plt.legend()
plt.grid(True)
plt.show()
import itertools

def loocv_model(X, y, learning_rate, epochs, hid_units):
    val_losses = []
```

```
    for i in range(len(X)):
        X_train_loocv = np.delete(X, i)
        y_train_loocv = np.delete(y, i)
        x_val = X[i]
        y_val = y[i]
        W_in = np.random.uniform(-0.5, 0.5, size=(hid_units,))
        b_in = np.random.uniform(-0.5, 0.5, size=(hid_units,))
        W_hid = np.random.uniform(-0.5, 0.5, size=(hid_units,))
        b_hid = np.random.uniform(-0.5, 0.5)
        for epoch in range(epochs):
            for j in range(len(X_train_loocv)):
                x = X_train_loocv[j]
                true_y = y_train_loocv[j]
                z = W_in * x + b_in
                h = sigmoid(z)
                y_hat = np.dot(W_hid, h) + b_hid
                error = true_y - y_hat
                W_hid_grad = -2 * error * h
                b_hid_grad = -2 * error
                W_in_grad = -2 * error * W_hid * h * (1 - h) * x
                b_in_grad = -2 * error * W_hid * h * (1 - h)
                W_hid = W_hid - learning_rate * W_hid_grad
                b_hid = b_hid - learning_rate * b_hid_grad
                W_in = W_in - learning_rate * W_in_grad
                b_in = b_in - learning_rate * b_in_grad
        z_val = W_in * x_val + b_in
        h_val = sigmoid(z_val)
        y_val_hat = np.dot(W_hid, h_val) + b_hid
        val_loss = compute_error(y_val, y_val_hat)
        val_losses.append(val_loss)
    return np.mean(val_losses)

learning_rates = [0.01]
hid_units_list = [2, 4, 8, 16,32]
epoch_list = [2000]

results = []

for lr, hu, ep in itertools.product(learning_rates, hid_units_list,
epoch_list):
    avg_val_loss = loocv_model(X_train_norm, y_train_norm, lr, ep, hu)
    results.append((lr, hu, ep, avg_val_loss))
    print(f"LR: {lr}, Hidden Units: {hu}, Epochs: {ep}, LOOCV Loss:
{avg_val_loss:.4f}")
results.sort(key=lambda x: x[3])
best_config = results[0]
print("\nBest configuration:")
print(f"Learning rate: {best_config[0]}, Hidden units: {best_config[1]},
Epochs: {best_config[2]}, Validation Loss: {best_config[3]:.4f}")
```

```
plt.figure(figsize=(8, 6))

plt.scatter(X_train_norm, y_train_norm, color='blue', label='True Labels')


x_line = np.linspace(np.min(X_train_norm), np.max(X_train_norm), 100)
#plt.scatter(train_in, train_out, color='blue', label='True Labels')


#x_line = np.linspace(np.min(train_in), np.max(train_out), 100)
pred_line = []

for x in x_line:

    z = W_in * x + b_in
    h = sigmoid(z)
    y_hat = np.dot(W_hid, h) + b_hid
    pred_line.append(y_hat)

plt.plot(x_line, pred_line, color='red', label='Predictions')
plt.xlabel("Normalized Input")
plt.ylabel("Normalized Output")
plt.title("Predictions vs True Labels (Training Data)")
plt.legend()
plt.grid(True)
plt.show()

def train_model_testloss(learning_rate, epochs=2000, hid_units=8):
    W_in = np.random.uniform(-0.5, 0.5, size=(hid_units,))
    b_in = np.random.uniform(-0.5, 0.5, size=(hid_units,))
    W_hid = np.random.uniform(-0.5, 0.5, size=(hid_units,))
    b_hid = np.random.uniform(-0.5, 0.5)
    train_loss_array = []
    test_loss_array = []
    for epoch in range(epochs):
        loss_in_epoch = 0.0
        for i in range(len(X_train_norm)):
            x = X_train_norm[i]
            true_y = y_train_norm[i]
            z = W_in * x + b_in
            h = sigmoid(z)
            y_hat = np.dot(W_hid, h) + b_hid
            error = true_y - y_hat
            new_loss = compute_error(true_y, y_hat)
            loss_in_epoch += new_loss
            W_hid_grad = -2 * error * h
            b_hid_grad = -2 * error
            W_in_grad = -2 * error * W_hid * h * (1 - h) * x
            b_in_grad = -2 * error * W_hid * h * (1 - h)
```

```
            W_hid = W_hid - learning_rate * W_hid_grad
            b_hid = b_hid - learning_rate * b_hid_grad
            W_in = W_in - learning_rate * W_in_grad
            b_in = b_in - learning_rate * b_in_grad
        epoch_train_loss = loss_in_epoch / len(X_train_norm)
        train_loss_array.append(epoch_train_loss)

        test_loss_in_epoch = 0.0
        for i in range(len(X_test_norm)):
            x = X_test_norm[i]
            true_y = y_test_norm[i]
            z = W_in * x + b_in
            h = sigmoid(z)
            y_hat = np.dot(W_hid, h) + b_hid
            test_loss_in_epoch += compute_error(true_y, y_hat)
        epoch_test_loss = test_loss_in_epoch / len(X_test_norm)
        test_loss_array.append(epoch_test_loss)
    print("Train loss:" ,train_loss_array[-1])
    print("Test loss:" ,test_loss_array[-1])
    return train_loss_array, test_loss_array, W_in, b_in, W_hid, b_hid

test_in = test_data[:,0]
test_out = test_data[:,1]
X_test_norm, X_test_mean, X_test_std = normalize_data(test_in)
y_test_norm, y_test_mean, y_test_std = normalize_data(test_out)
learning_rate = 0.01
epochs = 2000
train_loss_array, test_loss_array, W_in, b_in, W_hid, b_hid =
train_model_testloss(learning_rate, epochs=epochs, hid_units=8)

plt.figure(figsize=(10, 6))
plt.plot(range(1, epochs+1), train_loss_array, label="Train Loss")
plt.plot(range(1, epochs+1), test_loss_array, label="Test Loss")
plt.xlabel("Epoch")
plt.ylabel("Average Loss")
plt.title("Train vs. Test Loss over Epochs (LR = 0.01)")
plt.legend()
plt.grid(True)
plt.show()
plt.figure(figsize=(8,6))
plt.scatter(X_train_norm, y_train_norm, color='blue', label='Train Data')
x_line = np.linspace(np.min(X_train_norm), np.max(X_train_norm), 200)
pred_line = [np.dot(W_hid, sigmoid(W_in * x + b_in)) + b_hid for x in x_line]
plt.plot(x_line, pred_line, color='red', label='Prediction')
plt.xlabel("Normalized Input")
plt.ylabel("Normalized Output")
plt.title("Training Data: Predictions vs True Values")
plt.legend()
plt.grid(True)
```

```
plt.show()

plt.figure(figsize=(8,6))
plt.scatter(X_test_norm, y_test_norm, color='green', label='Test Data')
x_line_test = np.linspace(np.min(X_test_norm), np.max(X_test_norm), 200)
pred_line_test = [np.dot(W_hid, sigmoid(W_in * x + b_in)) + b_hid for x in
x_line_test]
plt.plot(x_line_test, pred_line_test, color='orange', label='Prediction')
plt.xlabel("Normalized Input")
plt.ylabel("Normalized Output")
plt.title("Test Data: Predictions vs True Values")
plt.legend()
plt.grid(True)
plt.show()
import pandas as pd

hidden_units_list = [2, 4, 8, 16, 32]
learning_rate = 0.01
epochs = 5000

custom_table = {
    "Hidden Units": [],
    "Train Loss ": [],
    "Train Loss Std": [],
    "Test Loss ": [],
    "Test Loss Std": []
}

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 12))
axes = axes.flatten()

for idx, hu in enumerate(hidden_units_list):
    train_loss_array, test_loss_array, W_in, b_in, W_hid, b_hid =
train_model_testloss(learning_rate, epochs=epochs, hid_units=hu)

    train_instance_losses = []
    for i in range(len(X_train_norm)):
        x = X_train_norm[i]
        true_y = y_train_norm[i]
        z = W_in * x + b_in
        h = sigmoid(z)
        y_hat = np.dot(W_hid, h) + b_hid
        train_instance_losses.append(compute_error(true_y, y_hat))

    test_instance_losses = []
    for i in range(len(X_test_norm)):
        x = X_test_norm[i]
        true_y = y_test_norm[i]
        z = W_in * x + b_in
```

```
        h = sigmoid(z)
        y_hat = np.dot(W_hid, h) + b_hid
        test_instance_losses.append(compute_error(true_y, y_hat))

    train_loss_mean = np.mean(train_instance_losses)
    train_loss_std = np.std(train_instance_losses)
    test_loss_mean = np.mean(test_instance_losses)
    test_loss_std = np.std(test_instance_losses)

    custom_table["Hidden Units"].append(hu)
    custom_table["Train Loss "].append(train_loss_mean)
    custom_table["Train Loss Std"].append(train_loss_std)
    custom_table["Test Loss "].append(test_loss_mean)
    custom_table["Test Loss Std"].append(test_loss_std)

    ax = axes[idx]
    ax.scatter(X_train_norm, y_train_norm, color='blue')
    x_line = np.linspace(np.min(X_train_norm), np.max(X_train_norm), 200)
    overall_pred = [np.dot(W_hid, sigmoid(W_in * x + b_in)) + b_hid for x in
x_line]
    ax.plot(x_line, overall_pred, color='red')
    for h in range(hu):
        hidden_curve = [W_hid[h] * sigmoid(W_in[h] * x + b_in[h]) for x in
x_line]
        ax.plot(x_line, hidden_curve, '--')
    ax.set_title(f"Hidden Units = {hu}")
    ax.set_xlabel("Normalized Input")
    ax.set_ylabel("Normalized Output")
    ax.grid(True)

for i in range(len(hidden_units_list), len(axes)):
    fig.delaxes(axes[i])

plt.tight_layout()
plt.show()

df = pd.DataFrame(custom_table)
#print(df)
```