,

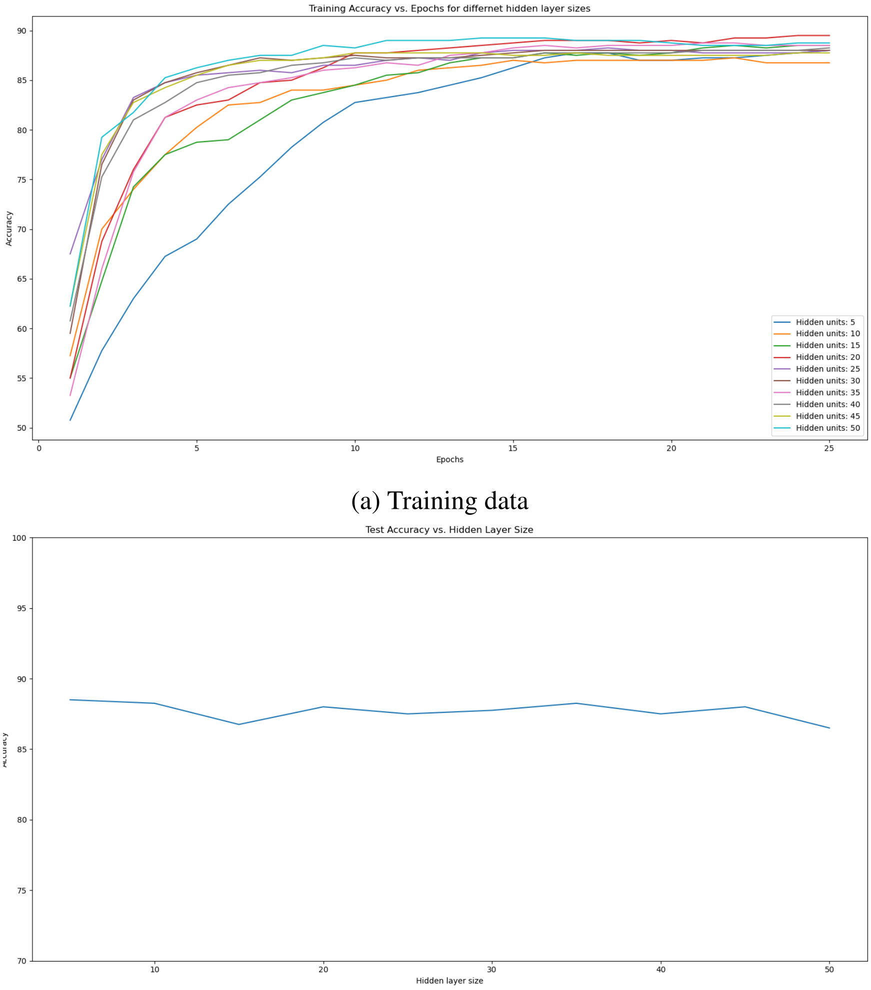
Learning Based Control HW1: Neural Network

Submitted By: Ashutosh Gupta
OSU ID: 934533517

1 Training performance of the network on training set 1

1.1 How does the number of hidden units impact the results?

To observe the effects of hidden units on the result, we plot a graph comparing the training accuracy over the epochs for different hidden unit sizes. We plot another graph comparing test accuracy to different hidden unit sizes. The model was trained for 25 epochs with stochastic gradient descent and a learning rate of 0.01. Weights were initialized using He normal initialization, and biases were zero-initialized. The sigmoid activation function was used for both layers.



(a) Training data

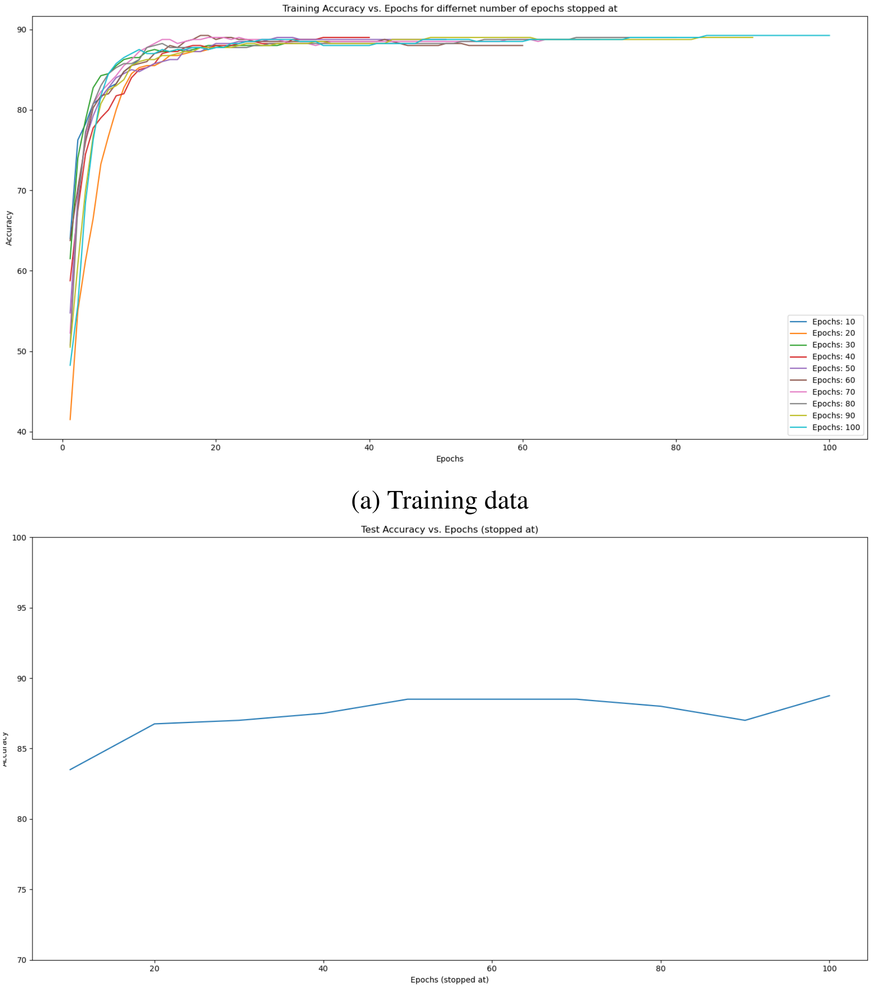
(b) Test data

Figure 1: Accuracy for different hidden layer size

Analyzing the graphs, we can say that with a higher number of hidden units, the model takes less training time to achieve higher accuracy, i.e. we get faster training. However, the test performance of the network starts deteriorating. So the model stops generalizing, and it seems a case of over-fitting on the training set, causing bad performance on the testing data. So, there is a trade-off between over-fitting and faster training when controlling the hidden unit size.

1.2 How does the training time impact the results?

To observe the effects of training time on the result, we plot a graph comparing the training accuracy over the epochs for different stop times. We plot another graph comparing test accuracy to different stop times. The model was trained with 10 hidden units with stochastic gradient descent and a learning rate of 0.01. Weights were initialized using He normal initialization, and biases were zero-initialized. The sigmoid activation function was used for both layers.



(a) Training data

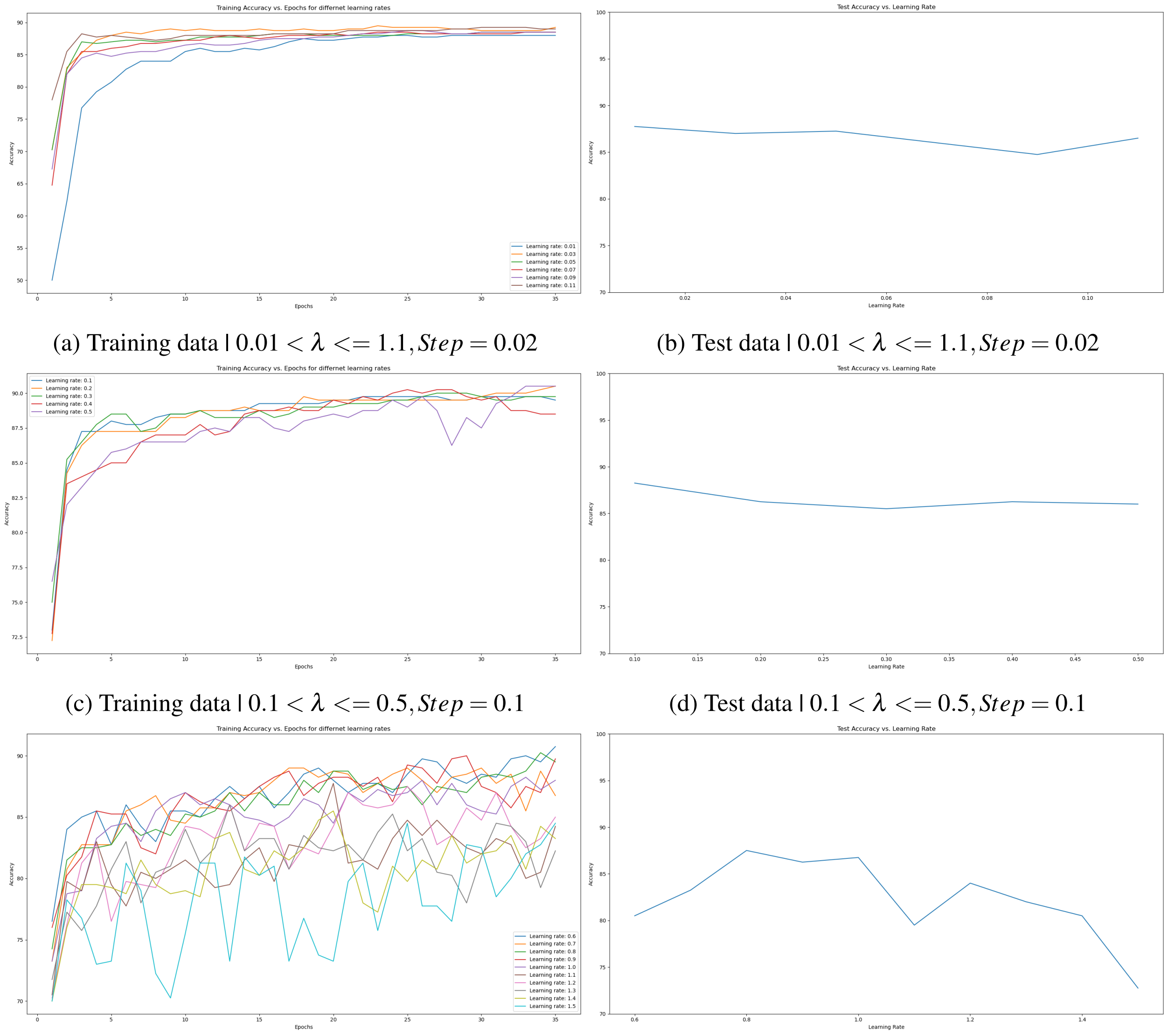
(b) Test data

Figure 2: Accuracy for different training times

The above figure shows that the training accuracy increases significantly in the first 10 epochs but mostly converges around the epoch-30. Training the model for longer post this, results in not very significant improvement over the accuracy. As evident in Fig:2a, the accuracy virtually doesn't change post epoch-60. So, we can conclude that longer training times of more than 40 epochs do not affect the model much.

1.3 How does the learning rate impact the results?

To observe the effects of learning rate on the result, we plot a set of graphs comparing the training accuracy over the epochs for different learning rates. We plot another set of graphs comparing test accuracy to different learning rates. The model was trained with 10 hidden units with stochastic gradient descent for 35 epochs. Weights were initialized using He normal initialization, and biases were zero-initialized. The sigmoid activation function was used for both layers.



(a) Training data | $0.01 < \lambda <= 1.1, Step = 0.02$

(b) Test data | $0.01 < \lambda <= 1.1, Step = 0.02$

(c) Training data | $0.1 < \lambda <= 0.5, Step = 0.1$

(d) Test data | $0.1 < \lambda <= 0.5, Step = 0.1$

(e) Training data | $0.6 < \lambda <= 1.5, Step = 0.1$

(f) Test data | $0.6 < \lambda <= 1.5, Step = 0.1$

Figure 3: Accuracy for different learning rates

In Fig:3a and Fig:3b the learning rate range is $[0.01, 1.1]$ with 0.02 step size. In Fig:3c and Fig:3d the learning rate range is $[0.1, 0.5]$ with 0.1 step size and lastly, in Fig:3e and Fig:3f the learning rate range is $[0.6, 1.5]$ with 0.1 step size.

In Fig:3a and Fig:3b, we see that increasing the learning rate from 0.01 to 1.1 results in faster training with some compromise to the test accuracy. In Fig:3c and Fig:3d, we see some instability in learning accuracy as we keep increasing the learning rate further up to 0.5. We also now see more significant decrease in test accuracy, overall affecting the model performance. Further in Fig:3e and Fig"3f as we increase the learning rate the model becomes unstable. As evident the accuracy is oscillating and the model performs even worse on the test data. So, we can conclude that a learning rate in the range of $[0.09, 0.2]$ achieves the best performance.

1.4 What other critical parameters impacted the results?

1.4.1 Activation Function

Changing the activation function for the hidden layer from "sigmoid" to "relu". The model is trained with 10 hidden units and learning rate 0.1 for 35 epochs. Weights were initialized using He normal initialization, and biases were zero-initialized. We plot the training accuracy over the epochs for both the activation functions.

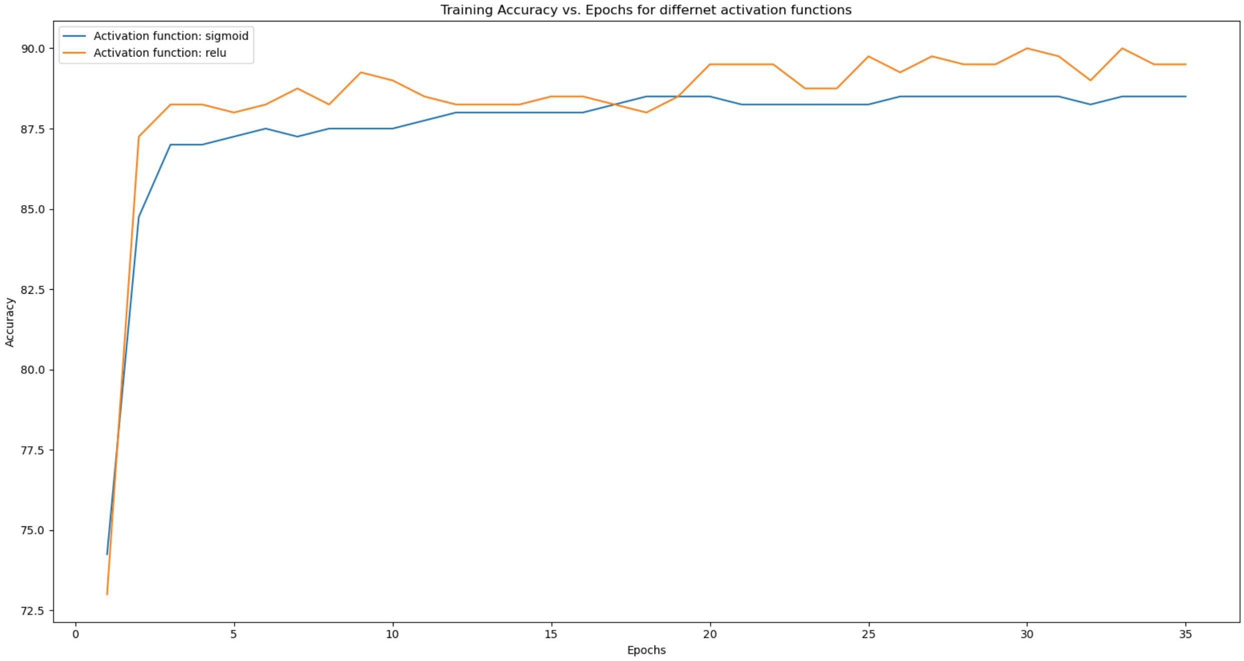


Figure 4: Different activation function for hidden layer

We can see that using the relu activation function results in faster training than sigmoid. However, it also results in some instability as we train longer. Using sigmoid function results in a more stable learning but overall lesser accuracy.

1.4.2 Weight Initialization

Changing the weight initialization from "He Normal" to "Normal". The model is trained with 10 hidden units and learning rate 0.1 for 35 epochs. The sigmoid activation function was used, and biases were zero-initialized. We plot the training accuracy over the epochs for both the weight initialization methods.

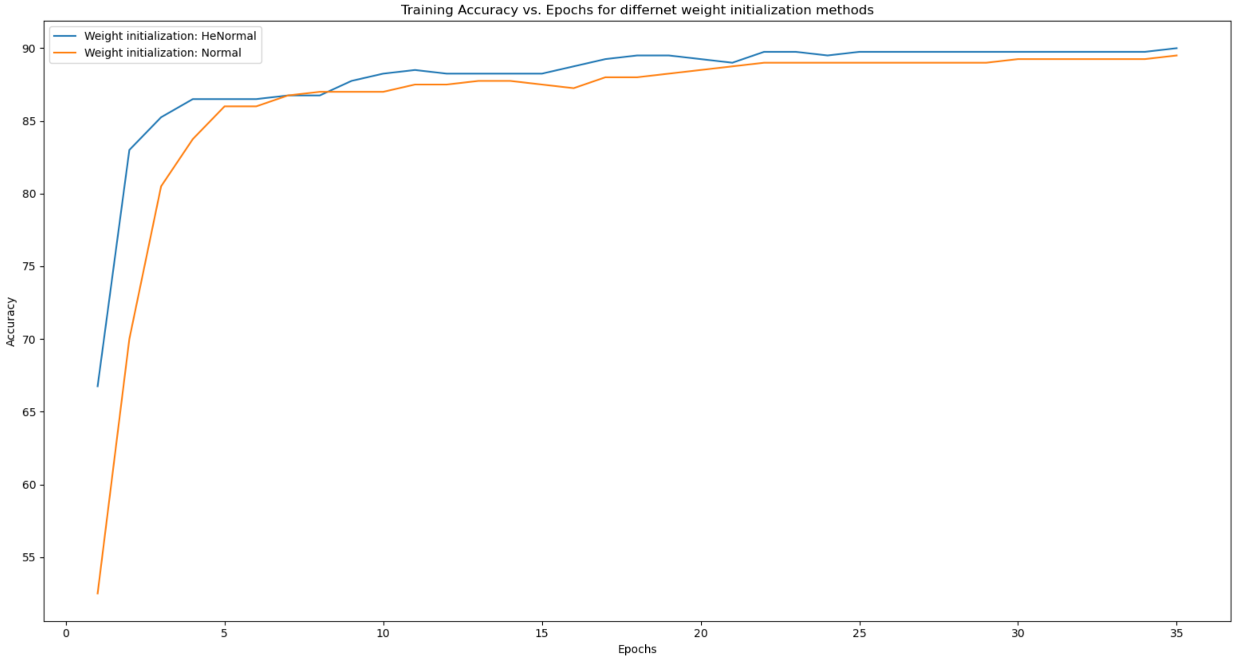


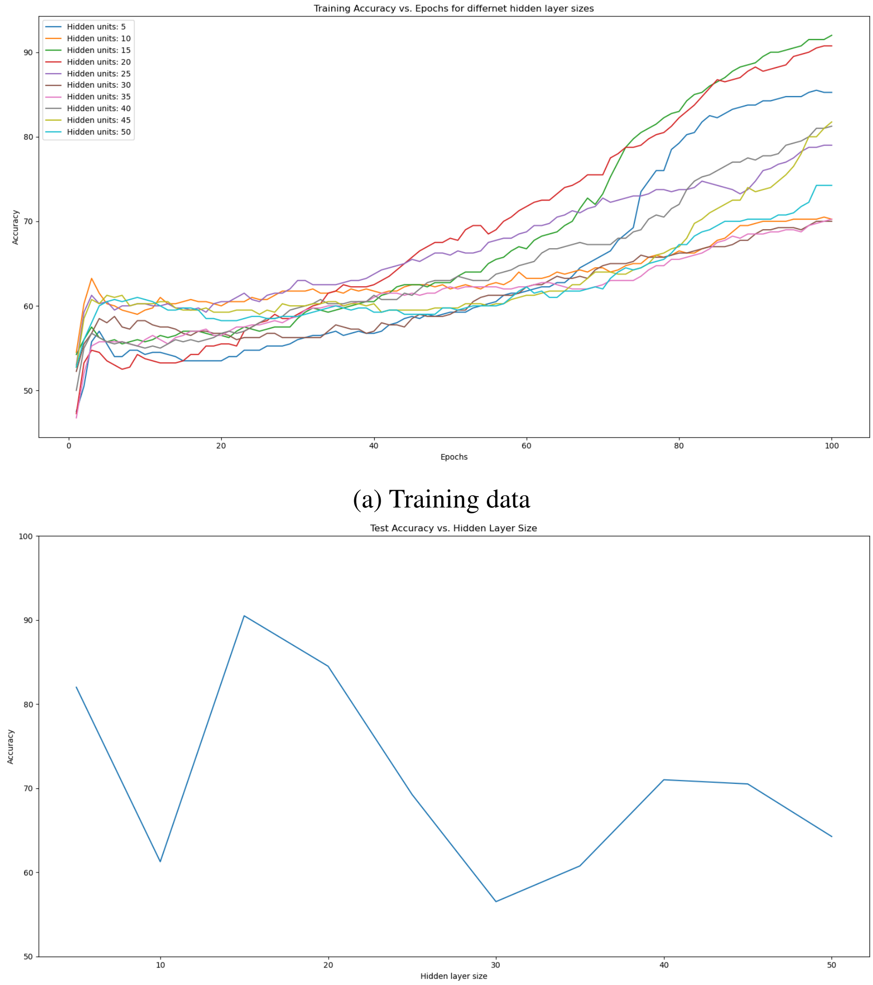
Figure 5: Different weight initialization methods

We can see that using the He Normal weight initialization method results in faster training than just Normal distribution. It also results in overall better accuracy.

2 Training performance of the network on training set 2

2.1 How does the number of hidden units impact the results?

To observe the effects of hidden units on the result, we plot a graph comparing the training accuracy over the epochs for different hidden unit sizes. We plot another graph comparing test accuracy to different hidden unit sizes. The model was trained for 100 epochs with stochastic gradient descent and a learning rate of 0.1. Weights were initialized using He normal initialization, and biases were zero-initialized. The sigmoid activation function was used for both layers.



(a) Training data

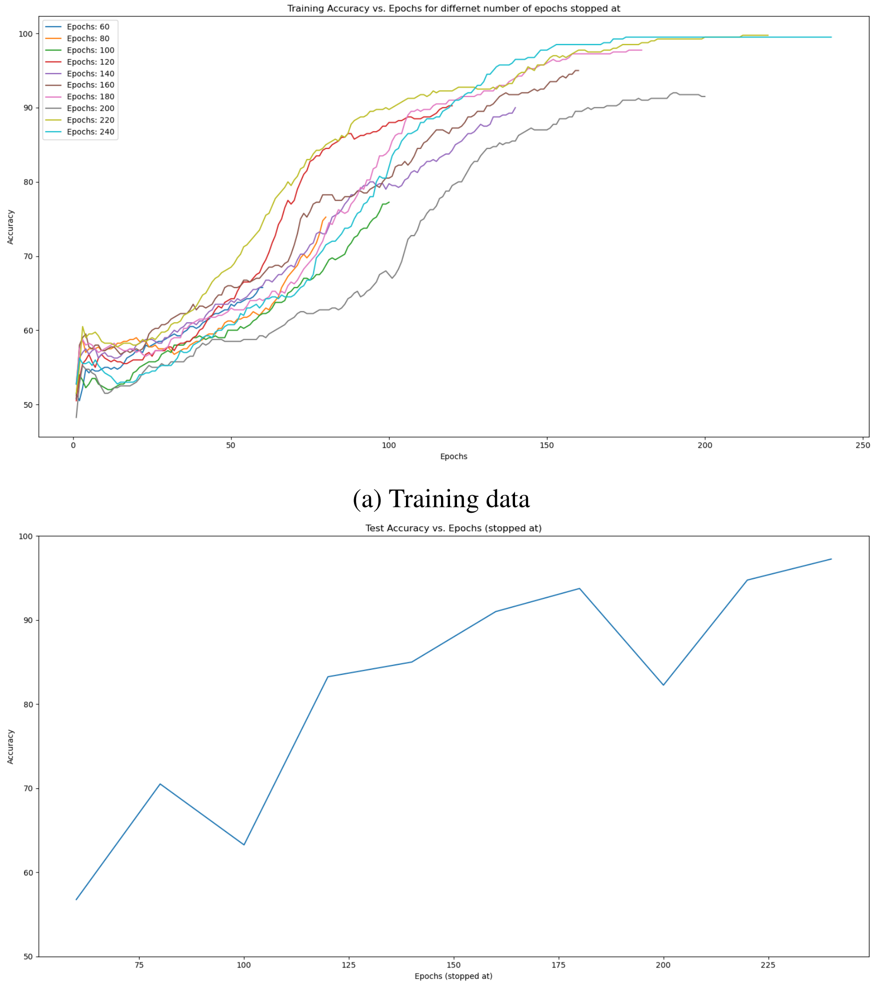
(b) Test data

Figure 6: Accuracy for different hidden layer size

Analyzing the graphs, we can say that a higher number of hidden units initially perform better but their performance decreases significantly as we train longer post epoch-50. After epoch-50 smaller hidden unit sizes (especially 15 and 20) start to perform much better ending with higher training accuracy and comparable test accuracy. The larger models also perform worse on test accuracy, indicating an over-fitting case. As compared to dataset-1 this dataset had to be trained for significantly longer time to achieve a good accuracy.

2.2 How does the training time impact the results?

To observe the effects of training time on the result, we plot a graph comparing the training accuracy over the epochs for different stop times. We plot another graph comparing test accuracy to different stop times. The model was trained with 15 hidden units with stochastic gradient descent and a learning rate of 0.1. Weights were initialized using He normal initialization, and biases were zero-initialized. The sigmoid activation function was used for both layers.



(a) Training data

(b) Test data

Figure 7: Accuracy for different training times

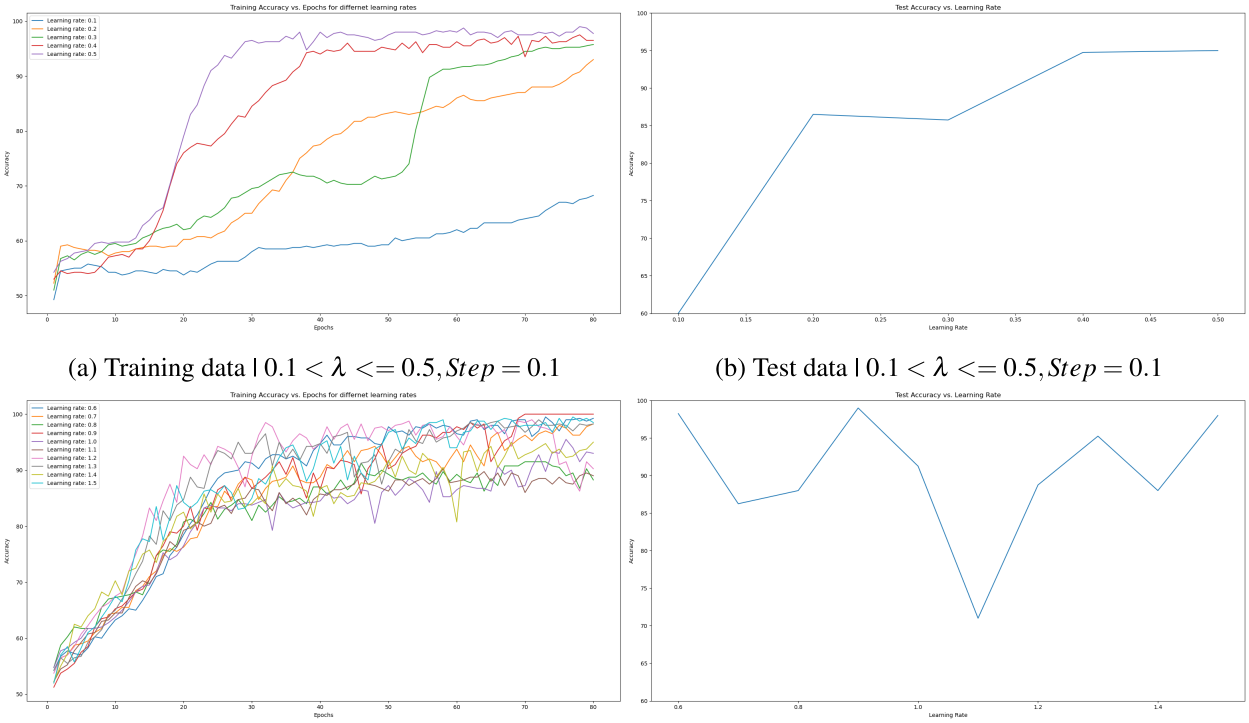
The above figure shows that the training accuracy increases significantly until 180 epochs and mostlyu converges with no significant improvement post 180 epochs. As evident in Fig:7a, the accuracy virtually doesn't change post epoch-190. The model takes significantly takes a longer time to train with the current parameters and even the test accuracy keeps increasing. Training for longer might result in overfitting.

2.3 How does the learning rate impact the results?

To observe the effects of learning rate on the result, we plot a set of graphs comparing the training accuracy over the epochs for different learning rates. We plot another set of graphs comparing test accuracy to different learning rates. The model was trained with 15 hidden units with stochastic gradient descent for 80 epochs. Weights were initialized using He normal initialization, and biases were zero-initialized. The sigmoid activation function was used for both layers.

In Fig:8a and Fig:8b the learning rate range is $[0.1, 0.5]$ with 0.1 step size. In Fig:8c and Fig:8d the learning rate range is $[0.6, 1.5]$ with 0.1 step size.

In Fig:8a and Fig:8b, we see that increasing the learning rate from 0.1 to 0.5 results in faster training with an increase to the test accuracy. In Fig:8c and Fig:8d, we see a lot instability in learning accuracy as we keep increasing the learning rate further up to 1.5. As we train these models longer with higher learning rate there is more oscillations seen in the accuracy. We also now see more significant jumps in test accuracy, overall



(a) Training data | $0.1 < \lambda <= 0.5, Step = 0.1$

(b) Test data | $0.1 < \lambda <= 0.5, Step = 0.1$

(c) Training data | $0.6 < \lambda <= 1.5, Step = 0.1$

(d) Test data | $0.6 < \lambda <= 1.5, Step = 0.1$

Figure 8: Accuracy for different learning rates

affecting the model performance. So, we can conclude that a learning rate in the range of $[0.4, 0.6]$ achieves the best performance.

2.4 What other critical parameters impacted the results?

2.4.1 Activation Function

Changing the activation function for the hidden layer from "sigmoid" to "relu". The model is trained with 15 hidden units and learning rate 0.5 for 80 epochs. Weights were initialized using He normal initialization, and biases were zero-initialized. We plot the training accuracy over the epochs for both the activation functions.

We can see in Fig:9 that using the relu activation function model performs better initially until 15 epochs but then the sigmoid function model starts to perform significantly well. Overall the sigmoid model achieves a higher accuracy as we train longer.

2.4.2 Weight Initialization

Changing the weight initialization from "He Normal" to "Normal". The model is trained with 15 hidden units and learning rate 0.5 for 80 epochs. The sigmoid activation function was used, and biases were zero-initialized. We plot the training accuracy over the epochs for both the weight initialization methods.

We can see in Fig:10 that using the He Normal weight initialization method achieves faster training around 30-40 epochs than just Normal distribution. But, as we train longer it falls below, although the accuracy difference in both models isn't that significant.

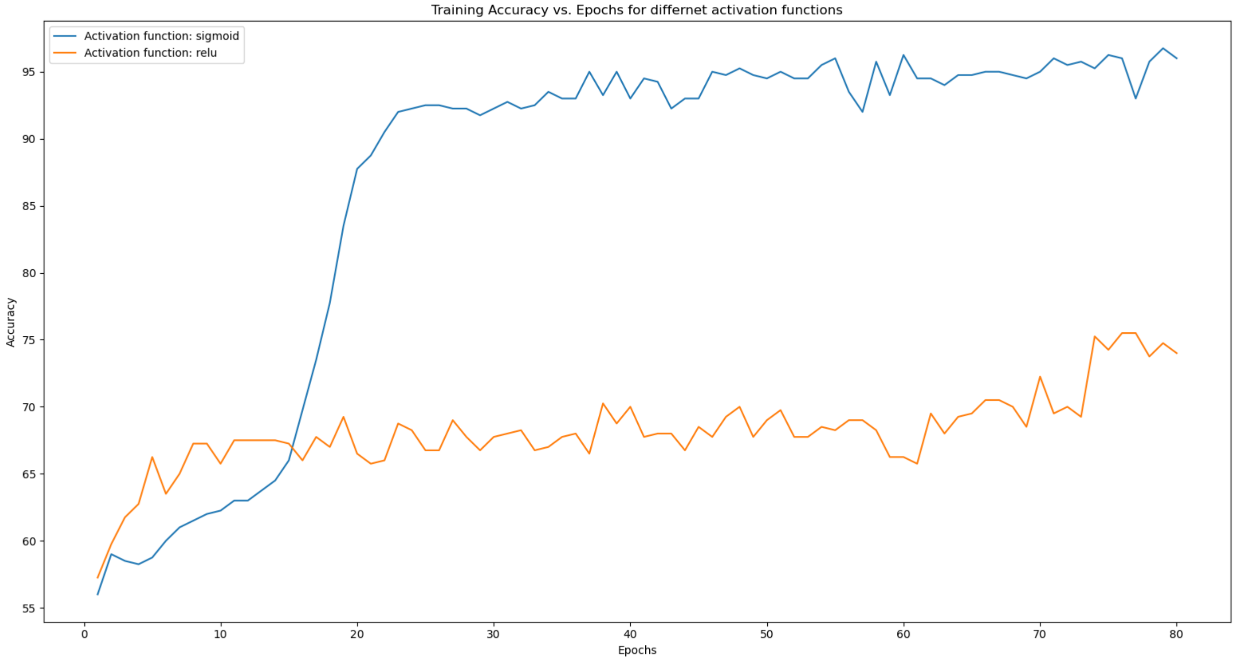


Figure 9: Different activation function for hidden layer

3 Difference in performance

For the second dataset, it takes significantly longer to train and achieve a comparable performance to the first dataset. We see more hidden units, larger learning rate and more epochs needed. It also takes more fine tuning of these parameters to achieve a good accuracy for the second dataset. This could be due to more patterned data observed in the second dataset that results in difference in performance.

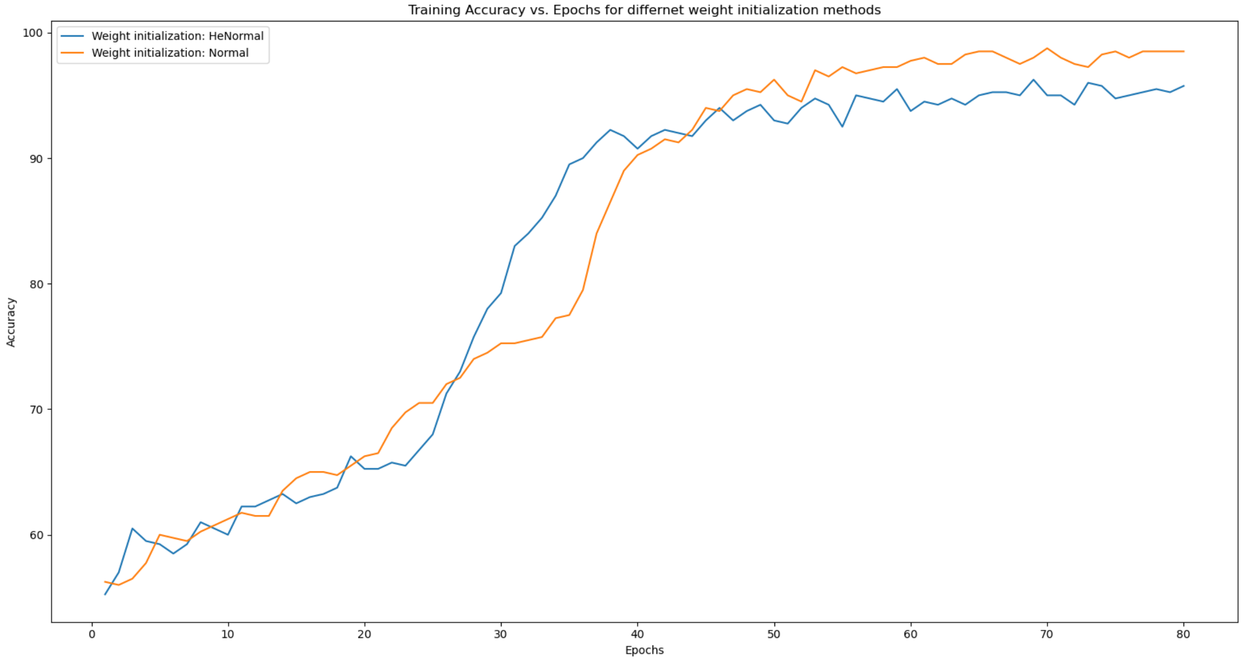


Figure 10: Different weight initialization methods

```
import os
import csv
import numpy as np
import matplotlib.pyplot as plt

class NeuralNetwork():
    """Class to define a simple feed forward neural network with 1 hidden layer"""

    def __init__(self, input_size:int, output_size:int, hidden_size:int, learning_rate:float,
epochs:int,
                 activation_function:str=’sigmoid’, initialization:str=’HeNormal’ , save_flag:
bool=True):
        """
        Initialize the parameters of the network

        Parameters:
            input_size: Number of features in the input
            output_size: Number of features in the output
            save_flag: Flag to save the model parameters and plots
        Hyperparameters:
            hidden_size: Number of neurons in the hidden layer
            learning_rate: Learning rate of the network
            epochs: Number of epochs to train the network
            activation_function: Activation function to use in the hidden layer, ’sigmoid’ or
’relu’. Default is sigmoid
            initialization: Initialization method for the weights, ’HeNormal’ or ’Normal’. Def
ault is HeNormal
        """

        # Weights are initialized using initialization method specified
        if initialization == ’HeNormal’:
            self.W1 = np.random.randn(input_size, hidden_size) / np.sqrt(input_size) # (i,h)
            self.W2 = np.random.randn(hidden_size, output_size) / np.sqrt(hidden_size) # (h,o)
        else:
            self.W1 = np.random.randn(input_size, hidden_size) # (i,h)
            self.W2 = np.random.randn(hidden_size, output_size) # (h,o)

        # Biases are initialized to 0
        self.b1 = np.zeros((1, hidden_size)) # (1,h)
        self.b2 = np.zeros((1, output_size)) # (1,o)

        # Hyperparameters
        self.hidden_size = hidden_size
        self.learning_rate = learning_rate
        self.epochs = epochs

        if activation_function == ’sigmoid’ or activation_function == ’relu’:
            self.activation_function = activation_function
        else:
            print("Invalid activation function. Using sigmoid.")
            self.activation_function = ’sigmoid’

        self.save_flag = save_flag

        # Performance metrics
        self.training_accuracy = []

    def sigmoid(self, z:float):
        """Sigmoid activation function"""

        a = 1 / (1 + np.exp(-z))
        return a

    def sigmoid_prime(self, z:float):
        """Derivative of the sigmoid activation function"""
```

```python
        a = self.sigmoid(z)
        a = a * (1 - a)
        return a

    def relu(self, z:float):
        """ReLU activation function"""

        a = z * (z > 0)
        return a

    def relu_prime(self, z:float):
        """Derivative of the ReLU activation function"""

        a = 1 * (z > 0)
        return a

    def forward(self, X:np.ndarray):
        """
        Forward pass of the network

        Parameters:
            X: Input to the network - 1 data point - (1,i)
        """

        # Input to the hidden layer
        self.z1 = np.dot(X, self.W1) + self.b1 # (1,h)

        # Activation function used for hidden layer
        if self.activation_function == ’relu’:
            self.a1 = self.relu(self.z1) # (1,h)
        else:
            self.a1 = self.sigmoid(self.z1) # (1,h)

        # Hidden to the output layer
        self.z2 = np.dot(self.a1, self.W2) + self.b2 # (1,o)
        # Sigmoid activation function used for output as classification is binary
        self.a2 = self.sigmoid(self.z2) # (1,o)

        return

    def backward(self, X:np.ndarray, Y:np.ndarray):
        """
        Backward pass of the network

        Parameters:
            X: Input to the network - 1 data point - (1,i)
            Y: True output for the input X - (1,o)
        """

        # Compute gradient components from output to hidden layer
        self.dEda2 = 2 * (self.a2 - Y) # (1,o)
        self.da2dz2 = self.sigmoid_prime(self.z2) # (1,o)
        self.dEdz2 = self.dEda2 * self.da2dz2 # (1,o)
        self.dz2dW2 = self.a1 # (1,h)

        self.dEdW2 = np.dot(self.dz2dW2.T, self.dEdz2) # (h,o)
        self.dEdb2 = self.dEdz2 # (1,o)
        self.dEda1 = np.dot(self.dEdz2, self.W2.T) # (1,h)

        # Compute gradient components from hidden to input layer
        if self.activation_function == ’relu’:
            self.da1dz1 = self.relu_prime(self.z1) # (1,h)
        else:
            self.da1dz1 = self.sigmoid_prime(self.z1) # (1,h)
        self.dEdz1 = self.dEda1 * self.da1dz1 # (1,h)
        self.dz1dW1 = X # (1,i)

        self.dEdW1 = np.dot(self.dz1dW1.T, self.dEdz1) # (i,h)
        self.dEdb1 = self.dEdz1 # (1,h)

        # Update weights and biases
        self.W1 -= self.learning_rate * self.dEdW1
        self.b1 -= self.learning_rate * self.dEdb1

        self.W2 -= self.learning_rate * self.dEdW2
        self.b2 -= self.learning_rate * self.dEdb2

        return

    def train(self, X:np.ndarray, Y:np.ndarray):
        """
        Train the network

        Parameters:
            X: Input to the network - All data points - (n,i)
            Y: True output for all the input X - (n,o)

        Returns:
            training_accuracy: List of accuracies for each epoch in percentage
        """

        for epoch in range(self.epochs):
            correct = 0
            for i in range(len(X)):
                # Convert the input and output to 2D arrays
                x_in = X[i].reshape(1, len(X[i]))
                y_out = Y[i].reshape(1, len(Y[i]))

                self.forward(x_in)
                self.backward(x_in, y_out)

                # Compute the number of correct predictions
                predicted = np.zeros_like(self.a2)
                predicted[np.where(self.a2 == np.max(self.a2))] = 1
                if np.array_equal(predicted, y_out):
                    correct += 1

            # Accuracy for the epoch in percentage
            self.accuracy = correct / len(X) * 100.0
            self.training_accuracy.append(self.accuracy)

        return self.training_accuracy

    def test(self, X:np.ndarray, Y:np.ndarray):
        """
        Test the network

        Parameters:
            X: Input to the network - All data points - (n,i)
            Y: True output for all the input X - (n,o)

        Returns:
            test_accuracy: Accuracy for the test data in percentage
        """

        correct = 0
        for i in range(len(X)):
            # Convert the input and output to 2D arrays
            x_in = X[i].reshape(1, len(X[i]))
            y_out = Y[i].reshape(1, len(Y[i]))

            self.forward(x_in)

            # Compute the number of correct predictions
            predicted = np.zeros_like(self.a2)
            predicted[np.where(self.a2 == np.max(self.a2))] = 1
            if np.array_equal(predicted, y_out):
                correct += 1

        # Accuracy for the test data in percentage
        self.test_accuracy = correct / len(X) * 100.0

        return self.test_accuracy

    def generate_csv(self, filename:str):
        """Save the model parameters in a csv file"""

        if not self.save_flag:
            return

        with open(filename, ’w’) as f:
            writer = csv.writer(f)

            # Write the hyperparameters
            writer.writerow([’Learning rate’, [self.learning_rate]])
            writer.writerow([’Epochs’, [self.epochs]])
            writer.writerow([’Hidden units’, [self.hidden_size]])

            # Write the weights and biases
            writer.writerow([’W1’, self.W1])
            writer.writerow([’b1’, self.b1])
            writer.writerow([’W2’, self.W2])
            writer.writerow([’b2’, self.b2])

            # Write the performance metrics
            writer.writerow([’Training accuracy’, self.training_accuracy])
            writer.writerow([’Test accuracy’, [self.test_accuracy]])

        print(f"Model parameters saved in {filename}")

        return

    def generate_plots(self, filename:str):
        """Generate plot for the training accuracy"""

        plt.figure()
        plt.plot(self.training_accuracy)
        plt.xlabel(’Epochs’)
        plt.ylabel(’Training accuracy’)

        plt.title(f"Accuracy with {self.hidden_size} hidden units, {self.learning_rate} learni
ng rate and {self.epochs} epochs")
        plt.show()

        if not self.save_flag:
            return

        plt.savefig(filename)
        print(f"Plot saved in {filename}")

        return
```

```
def main(hidden_size:int=25, learning_rate:float=0.02, epochs:int=20,
         activation_function:str='sigmoid', initialization:str='HeNormal',
         save_flag:bool=False, shuffle_flag:bool=False, show_plot:bool=True, file_number:int=1
):
    """
    Main function to train and test the network

    Parameters:
        hidden_size: Number of neurons in the hidden layer
        learning_rate: Learning rate of the network
        epochs: Number of epochs to train the network
        activation_function: Activation function to use in the hidden layer, 'sigmoid' or 'rel
u'. Default is sigmoid
        initialization: Initialization method for the weights, 'HeNormal' or 'Normal'. Default
 is HeNormal
        save_flag: Flag to save the model parameters and plots
        shuffle_flag: Flag to shuffle the training data
        show_plot: Flag to show the training accuracy plot
        file_number: Number of the data file to use for training and testing

    Returns:
        training_accuracy: List of accuracies for each epoch in percentage
        test_accuracy: Accuracy for the test data in percentage
    """

    # Check if the data files exist. Otherwise, use the default files
    train_file = os.path.join(os.getcwd(), f"data/train{file_number}.csv")
    test_file = os.path.join(os.getcwd(), f"data/test{file_number}.csv")

    if not (os.path.exists(train_file) and os.path.exists(test_file)):
        print("Invalid file number. Defaulting to 1.")
        file_number = 1
        train_file = os.path.join(os.getcwd(), f"data/train1.csv")
        test_file = os.path.join(os.getcwd(), f"data/test1.csv")

    # Read the training data
    with open(train_file, 'r') as f:
        reader = csv.reader(f)
        train_data = np.array(list(reader), dtype=np.float64)

    # Read the test data
    with open(test_file, 'r') as f:
        reader = csv.reader(f)
        test_data = np.array(list(reader), dtype=np.float64)

    # Shuffle the training data
    if shuffle_flag:
        rng = np.random.default_rng()
        rng.shuffle(train_data, axis=0)

    # Split the data into input and output
    X_train = train_data[:, :5]
    Y_train = train_data[:, 5:]
    X_test = test_data[:, :5]
    Y_test = test_data[:, 5:]

    # Input and output sizes
    input_size = X_train.shape[1]
    output_size = Y_train.shape[1]

    # Filename to save the model parameters and plots
    csv_file = os.path.join(os.getcwd(), f"results/d{file_number}_h{hidden_size}_l{learning_ra
te}_e{epochs}.csv")
    plot_file = os.path.join(os.getcwd(), f"results/d{file_number}_h{hidden_size}_l{learning_r
ate}_e{epochs}.png")
```

```
    # Create the network
    nn = NeuralNetwork(input_size, output_size, hidden_size, learning_rate, epochs,
                       activation_function, initialization, save_flag)

    # Train the network
    training_accuracy = nn.train(X_train, Y_train)

    # Test the network
    test_accuracy = nn.test(X_test, Y_test)

    # Save the model parameters
    nn.generate_csv(csv_file)

    # Generate the training accuracy plot
    if show_plot:
        nn.generate_plots(plot_file)

    return training_accuracy, test_accuracy


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

```python
import nn
import numpy as np
import matplotlib.pyplot as plt
%matplotlib qt

# Rarely changed parameters
shuffle = True # Shuffle the training data for stichastic gradient descent
activation_function = 'sigmoid' # Activation function for hidden layer ('sigmoid', 'relu')
initialization = 'HeNormal' # Weight initialization method ('HeNormal', 'Normal')
save = False # Save the trained model parameters
show_each_plot = False # Show each plot during training
file_num = 2 # The dataset to use

## Experiments with hidden layer size

# Hyperparameters
learning_rate = 0.1
epochs = 100
hidden_layer_size = np.arange(5, 51, 5)

# Accuracy arrays
train_acc = np.zeros((len(hidden_layer_size), epochs))
test_acc = np.zeros((len(hidden_layer_size), 1))

# Call neural network for each hidden layer size
for i in range(len(hidden_layer_size)):
    train, test = nn.main(hidden_layer_size[i], learning_rate, epochs,
                          activation_function, initialization,
                          save, shuffle, show_each_plot, file_num)
    train_acc[i] = train
    test_acc[i] = test
    print(f"Hidden layer size: {hidden_layer_size[i]} | Final Train accuracy : {train[-1]} | Test accuracy: {test}")

# Plot training accuracy vs. epochs
plt.figure("Training Accuracy")
for i in range(len(hidden_layer_size)):
    plt.plot(np.arange(1, epochs+1), train_acc[i], label=f"Hidden units: {hidden_layer_size[i]}")

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs. Epochs for differnet hidden layer sizes")
plt.legend()

# Plot test accuracy vs. hidden layer size
```

```python
plt.figure("Test Accuracy")
plt.plot(hidden_layer_size, test_acc)
plt.xlabel("Hidden layer size")
plt.ylabel("Accuracy")
plt.ylim(50,100)
plt.title("Test Accuracy vs. Hidden Layer Size")

plt.show()
```

```
Hidden layer size: 5 | Final Train accuracy : 85.25 | Test accuracy:
82.0
Hidden layer size: 10 | Final Train accuracy : 70.25 | Test accuracy:
61.25000000000001
Hidden layer size: 15 | Final Train accuracy : 92.0 | Test accuracy:
90.5
Hidden layer size: 20 | Final Train accuracy : 90.75 | Test accuracy:
84.5
Hidden layer size: 25 | Final Train accuracy : 79.0 | Test accuracy:
69.25
Hidden layer size: 30 | Final Train accuracy : 70.0 | Test accuracy:
56.49999999999999
Hidden layer size: 35 | Final Train accuracy : 70.25 | Test accuracy:
60.75000000000001
Hidden layer size: 40 | Final Train accuracy : 81.25 | Test accuracy:
71.0
Hidden layer size: 45 | Final Train accuracy : 81.75 | Test accuracy:
70.5
Hidden layer size: 50 | Final Train accuracy : 74.25 | Test accuracy:
64.25
```

```python
## Experiments with epochs

# Hyperparameters
learning_rate = 0.1
hidden_layer_size = 15
epochs = np.arange(60, 250, 20)

# Accuracy arrays
train_acc = np.zeros((len(epochs), epochs[-1]))
test_acc = np.zeros((len(epochs), 1))

# Call neural network for each number of epochs
for i in range(len(epochs)):
    train, test = nn.main(hidden_layer_size, learning_rate, epochs[i],
                          activation_function, initialization,
                          save, shuffle, show_each_plot, file_num)
    train_acc[i,:len(train)] = train
    test_acc[i] = test
    # Pad with NaNs to not have zero values in plot
    train_acc[i,len(train):] = np.nan
    print(f"Epochs: {epochs[i]} | Final Train accuracy : {train[-1]} |
Test accuracy: {test}")

# Plot training accuracy vs. epochs
plt.figure("Training Accuracy")
for i in range(len(epochs)):
    plt.plot(np.arange(1, epochs[-1]+1), train_acc[i], label=f"Epochs:
{epochs[i]}")

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs. Epochs for differnet number of epochs
stopped at")
plt.legend()

# Plot test accuracy vs. hidden layer size
plt.figure("Test Accuracy")
plt.plot(epochs, test_acc)
plt.xlabel("Epochs (stopped at)")
plt.ylabel("Accuracy")
plt.ylim(50,100)
plt.title("Test Accuracy vs. Epochs (stopped at)")

plt.show()
```

```
Epochs: 60 | Final Train accuracy : 65.75 | Test accuracy: 56.75
Epochs: 80 | Final Train accuracy : 75.25 | Test accuracy: 70.5
Epochs: 100 | Final Train accuracy : 77.25 | Test accuracy:
63.24999999999999
Epochs: 120 | Final Train accuracy : 90.25 | Test accuracy: 83.25
Epochs: 140 | Final Train accuracy : 90.0 | Test accuracy: 85.0
Epochs: 160 | Final Train accuracy : 95.0 | Test accuracy: 91.0
Epochs: 180 | Final Train accuracy : 97.75 | Test accuracy: 93.75
Epochs: 200 | Final Train accuracy : 91.5 | Test accuracy: 82.25
Epochs: 220 | Final Train accuracy : 99.75 | Test accuracy: 94.75
Epochs: 240 | Final Train accuracy : 99.5 | Test accuracy: 97.25
```

```
## Experiments with learning rate

# Hyperparameters
learning_rate = np.arange(0.6, 1.51, 0.1).round(2)
epochs = 80
hidden_layer_size = 15

# Accuracy arrays
train_acc = np.zeros((len(learning_rate), epochs))
test_acc = np.zeros((len(learning_rate), 1))

# Call neural network for each learning rate
for i in range(len(learning_rate)):
    train, test = nn.main(hidden_layer_size, learning_rate[i], epochs,
                          activation_function, initialization,
                          save, shuffle, show_each_plot, file_num)
    train_acc[i] = train
    test_acc[i] = test
    print(f"Learning rate: {learning_rate[i]} | Final Train accuracy :
{train[-1]} | Test accuracy: {test}")

# Plot training accuracy vs. epochs
plt.figure("Training Accuracy")
for i in range(len(learning_rate)):
    plt.plot(np.arange(1, epochs+1), train_acc[i], label=f"Learning
rate: {learning_rate[i]}")

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs. Epochs for differnet learning rates")
plt.legend()

# Plot test accuracy vs. hidden layer size
plt.figure("Test Accuracy")
plt.plot(learning_rate, test_acc)
plt.xlabel("Learning Rate")
plt.ylabel("Accuracy")
plt.ylim(60,100)
plt.title("Test Accuracy vs. Learning Rate")

plt.show()

Learning rate: 0.6 | Final Train accuracy : 99.25 | Test accuracy:
98.25
Learning rate: 0.7 | Final Train accuracy : 98.25 | Test accuracy:
86.25
Learning rate: 0.8 | Final Train accuracy : 88.25 | Test accuracy:
88.0
Learning rate: 0.9 | Final Train accuracy : 100.0 | Test accuracy:
99.0
Learning rate: 1.0 | Final Train accuracy : 93.0 | Test accuracy:
91.25
Learning rate: 1.1 | Final Train accuracy : 89.0 | Test accuracy: 71.0
Learning rate: 1.2 | Final Train accuracy : 90.25 | Test accuracy:
88.75
Learning rate: 1.3 | Final Train accuracy : 98.25 | Test accuracy:
95.25
Learning rate: 1.4 | Final Train accuracy : 95.0 | Test accuracy: 88.0
Learning rate: 1.5 | Final Train accuracy : 98.5 | Test accuracy: 98.0

## Experiments with other parameters

# Hyperparameters
```

```
learning_rate = 0.5
epochs = 80
hidden_layer_size = 15

## Experiments with activation function for hidden layer
activation_function = ['sigmoid', 'relu']
initialization = 'HeNormal'

# Accuracy arrays
train_acc = np.zeros((len(activation_function), epochs))

# Call neural network for each activation function
for i in range(len(activation_function)):
    train, test = nn.main(hidden_layer_size, learning_rate, epochs,
                          activation_function[i], initialization,
                          save, shuffle, show_each_plot, file_num)
    train_acc[i] = train
    print(f"Activation function: {activation_function[i]} | Final
Train accuracy : {train[-1]} | Test accuracy: {test}")

# Plot training accuracy vs. epochs
plt.figure("Activation Function")
for i in range(len(activation_function)):
    plt.plot(np.arange(1, epochs+1), train_acc[i], label=f"Activation
function: {activation_function[i]}")

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs. Epochs for differnet activation
functions")
plt.legend()

## Experiments with weight initialization
initialization = ['HeNormal', 'Normal']
activation_function = 'sigmoid'

# Accuracy arrays
train_acc = np.zeros((len(initialization), epochs))

# Call neural network for each weight initialization method
for i in range(len(initialization)):
    train, test = nn.main(hidden_layer_size, learning_rate, epochs,
                          activation_function, initialization[i],
                          save, shuffle, show_each_plot, file_num)
    train_acc[i] = train
    print(f"Weight initialization: {initialization[i]} | Final Train
accuracy : {train[-1]} | Test accuracy: {test}")

# Plot training accuracy vs. epochs
plt.figure("Weight Initialization")
```

```
for i in range(len(initialization)):
    plt.plot(np.arange(1, epochs+1), train_acc[i], label=f"Weight
initialization: {initialization[i]}")

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs. Epochs for differnet weight
initialization methods")
plt.legend()

plt.show()
```

```
Activation function: sigmoid | Final Train accuracy : 96.0 | Test
accuracy: 92.5
Activation function: relu | Final Train accuracy : 74.0 | Test
accuracy: 71.5
Weight initialization: HeNormal | Final Train accuracy : 95.75 | Test
accuracy: 92.5
Weight initialization: Normal | Final Train accuracy : 98.5 | Test
accuracy: 95.75
```