Nathan Butler
ROB 537 | HW1 Report
10/10/2023

Homework 1: Neural Networks

Nathan Butler | butlnath@oregonstate.edu

Training Set 1 Discussion

Unless otherwise noted, all training for part one was conducted using 25 hidden units, a step size of 0.001, and 100 epochs.

1. Hidden Units

It can be observed in Figure 1 that greater numbers of hidden units yield increased accuracy and reduced loss during training. This is likely due to the ability of a larger network to capture more information about relationships between input data and the output classification. Greater numbers of internal units allow for the tuning of many more parameters.

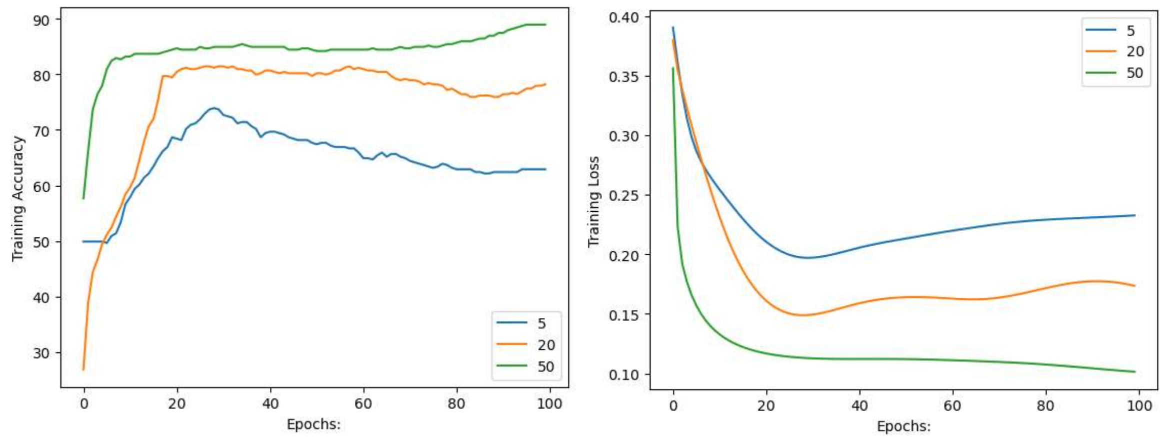


Figure 1. Accuracy and loss over training epochs for 5, 20, and 50 hidden layer units

These observations are further reinforced by the testing accuracy results stored in Table 1, which shows an accuracy of over 85% for a network with 50 hidden units and an accuracy of only 60.65% for a network with 5 hidden units.

Table 1. Testing accuracy for 5, 20, and 50 hidden layer units

# Hidden Units	Testing Accuracy
5	60.65%
20	76.44%
50	85.21%

2. Training Time

Contrary to the results obtained with increased hidden layers, an increase in training time is shown in Figure 2 to not necessarily yield better performance. In fact, the opposite effect was observed for training set 1. After an initial peak in accuracy around 60 epochs, a downward trend appears in each training cycle. These results may be impacted by the other network parameters, such as the limited number of hidden units.

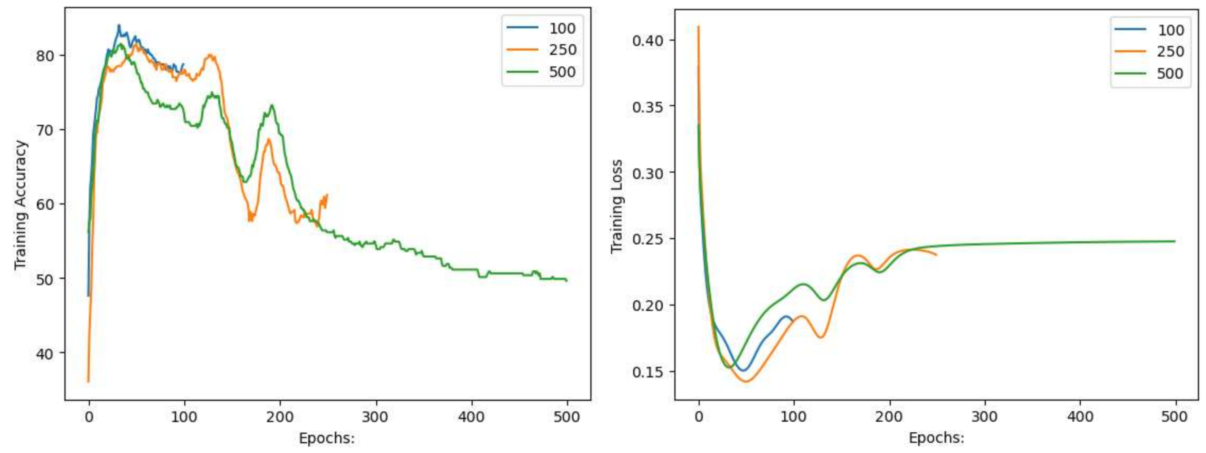


Figure 2. Accuracy and loss over training epochs for 100, 250, and 500 epochs

These observations are further reinforced by the testing accuracy results stored in Table 2, which shows an accuracy of over 75% for a network trained over 100 epochs and an accuracy of only 51.88% for a network trained over 500 epochs.

Table 2. Testing accuracy for 100, 250, and 500 epochs

# Epochs	Testing Accuracy
100	75.94%
250	58.65%
500	51.88%

3. Learning Rate

Learning rate results are somewhat skewed as an overflow error was often encountered in the sigmoid activation function at rates greater than 0.001. As a result, special attention will be given to the earlier training period of each chart displayed in Figure 3. These results indicate that the middle value of 0.01 may have been on the best trajectory towards optimal results before the error occurred. This may be due to the size of the network showing preference to the 0.01 step size at earlier time intervals. However, overall the smaller step size of 0.001 showed a steady increase in accuracy and decrease in loss over the course of training. While this smaller rate resulted in a longer training process, it also yielded the most steady training performance.

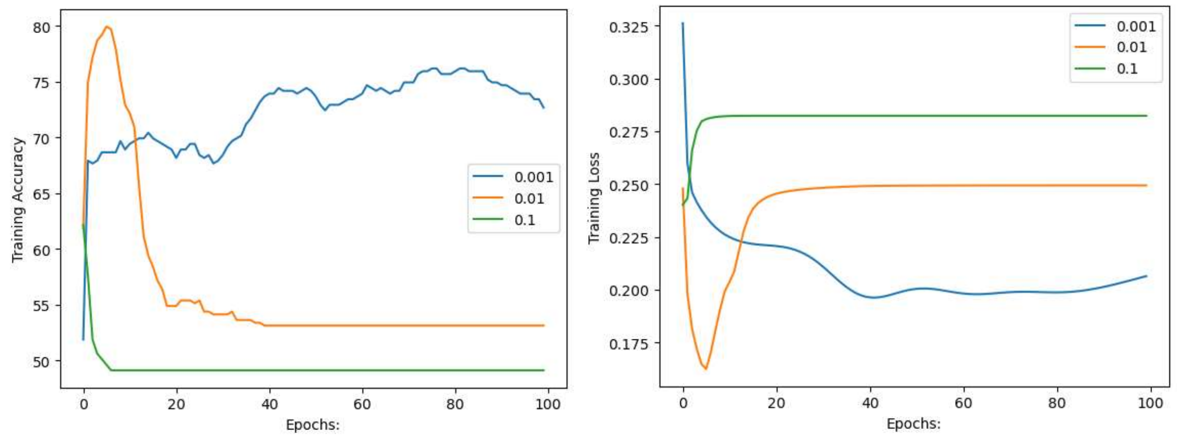


Figure 3. Accuracy and loss over training epochs for step sizes of 0.001, 0.01, and 0.1

Unfortunately, the failed learning from the overflow error for the 0.01 and 0.1 step sizes limits the accuracy results. Still, a 70% accuracy for 0.001 shows that the model was not overfit during training.

Table 3. Testing accuracy for 0.001, 0.01, and 0.1 step sizes

Step Size	Testing Accuracy
0.001	70.43%
0.01	50.13%
0.1	50.13%

4. Other Parameters

In training this network, weights and biases were initialized randomly. The initial accuracies and losses shown in the figures above illustrate the different effects that this initialization can have on the following training steps. While random sampling may be beneficial in certain settings, other approaches to initialization (such as sampling from a distribution) may result in more predictable training performances.

Additionally, the sigmoid activation function was used for the hidden and output layers of this network. There are many other types of activation functions, so future work may entail testing with ReLU, softmax, or other functions to determine which seems to have the best impact on training.

Training Set 2 Discussion

Unless otherwise noted, all training for part one was conducted using a 25 hidden units, a step size of 0.001, and 100 epochs.

1. Hidden Units

Figure 4a shows training accuracy and loss over training set 2 according to varying hidden unit sizes. Unlike set 1, there is not as noticeable of a difference in training performance according to hidden layer size. Dataset 2 is reportedly more complex, so perhaps increased training time may have shown greater separation between unit sizes and better performance overall.

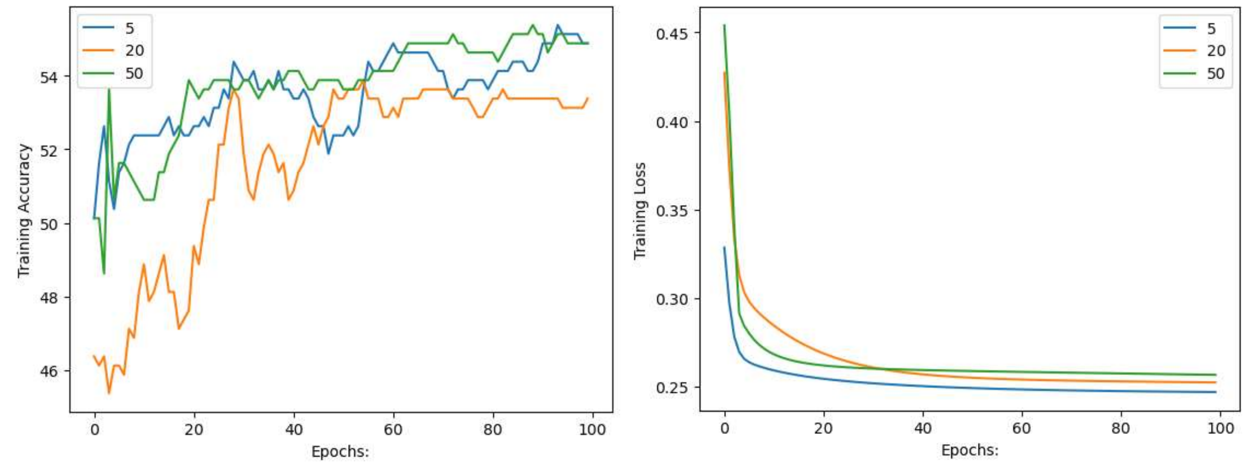


Figure 4a. Accuracy and loss over training epochs for 5, 20, and 50 hidden layer units

Because of these results, an additional test was conducted with larger hidden layers over 1000 training epochs. This appeared to have captured the complexity of training set 2 more effectively, and the results align better with those expected. Figure 4b and Table 4b show training and testing data.

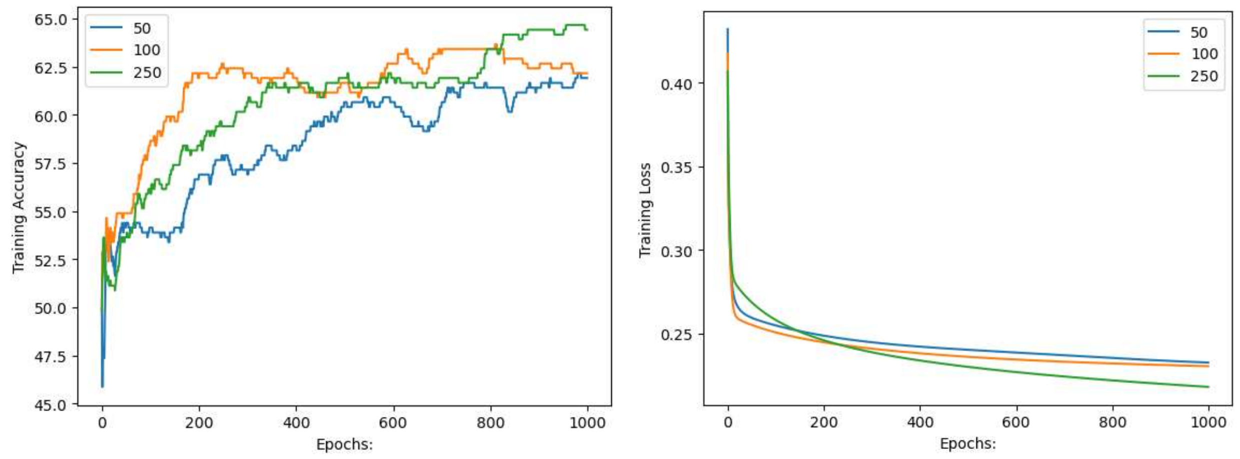


Figure 4b. Accuracy and loss over training epochs for 50, 100, and 250 hidden layer units over 1000 epochs

However, these training modifications may have resulted in some overfitting as the testing data performed poorly compared to training.

Table 4b. Testing accuracy for 50, 100, and 250 hidden layer units and 1000 epochs

# Hidden Units	Testing Accuracy
5	46.36%
20	47.87%
50	52.63%

2. Training Time

Figure 5 shows that additional training time allowed the relatively basic networks to achieve better training performance, but like hidden units the simple network appears to not be capable of capturing the relationships present in the more complex data. Table 5 supports this conclusion with accuracy results around 50%. In order to effectively compare results between training sets 1 and 2, the remaining training on set 2 was conducted using the same simple network design implemented for set 1.

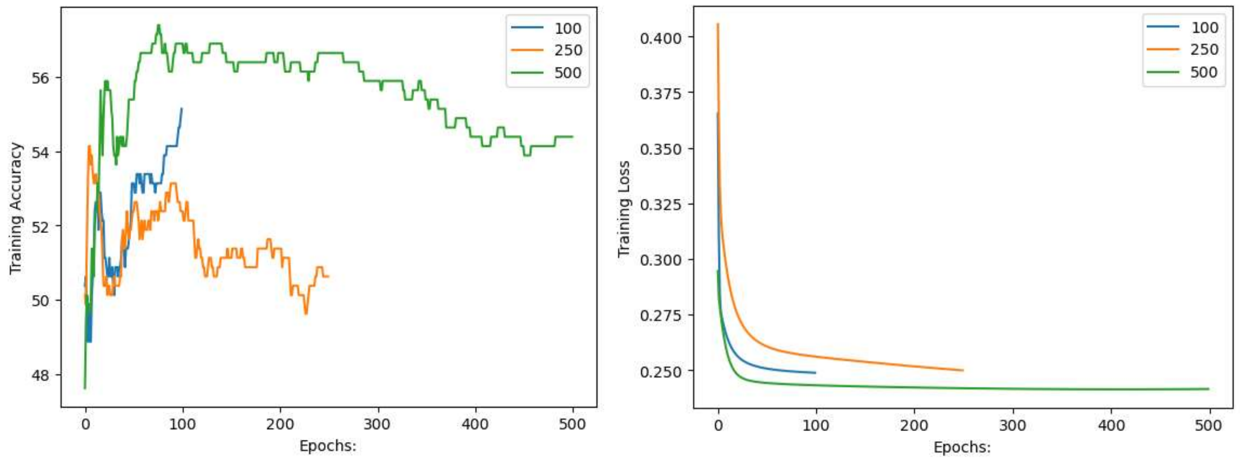


Figure 5. Accuracy and loss over training epochs for 100, 250, and 500 epochs

Table 5. Testing accuracy for 100, 250, and 500 epochs

# Epochs	Testing Accuracy
100	52.88%
250	54.88%
500	52.63%

3. Learning Rate

Learning rate results indicate that 0.01 may be a more optimal training rate for this network to capture relationships in the complex data in set 2. It might be the case the a rate of 0.1 resulted in changes that were too extreme while 0.001 was unable to produce results that could move out of local minima. Figure 6 shows the improved performance of the 0.01 rate, with similar testing performance reported in Table 6. It may be beneficial to train model at 0.01 for longer epochs, however the overflow error provides a constraint over more complex training configurations.

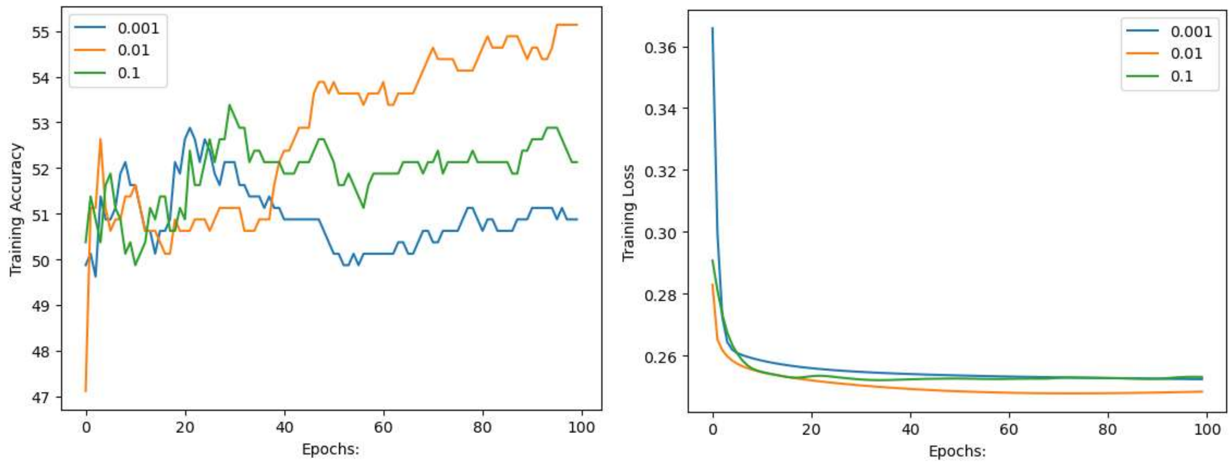


Figure 6. Accuracy and loss over training epochs for step sizes of 0.001, 0.01, and 0.1

Table 6. Testing accuracy for 0.001, 0.01, and 0.1 step sizes

Step Size	Testing Accuracy
0.001	52.13%
0.01	54.88%
0.1	48.87%

4. Other Parameters

In the case of this training, use of the sigmoid activation function provided a major bottleneck in computational abilities due to the overflow error observed in more complex training sizes. Other less computationally expensive activation functions like ReLU may be useful for expanding training capabilities.

Conclusions

Overall, I believe that the model parameters that yielded successful training results over training set 1 is too simple to capture the relationships present in the more complex set 2. 25 hidden layers cannot learn the more nuanced relationships, 100 epochs is too short of a training time, and the step size 0.001 may be too small for the training stage to break free of local minima. Going forward, I would be interest in observing the network's performance in training set 2 with over 100 hidden layers, at least 1000 epochs, and a step size around 0.01.

ROB537: Homework 1

Nathan Butler | butlnath@oregonstate.edu

Implement a one hidden-layer feed forward neural network to classify products into “pass” or “fail” categories. The neural network classifier will assume the role of quality control for a manufacturing plant. We use a simplified dataset for this assignment.

Each file has 400 data points, with one data point on each line where the data points have five inputs (x1, x2, x3, x4,x5) and two outputs (y1, y2):

x1, x2, x3, x4, x5, y1, y2

In this case, (x1, x2, x3, x4, x5) are features of products, such as specifications for dimensions, weight, and functionality. These features have been quantified by the values x1 through x5. The values y1 and y2 denote the classification of the product (pass or fail), where (y1 = 0, y2 = 1) indicates the product has passed, and (y1 = 1, y2 = 0) indicates the product has failed.

- train1.csv contains 400 training patterns (200 pass and 200 fail) from a simple power plant
- train2.csv contains 400 training patterns (200 pass and 200 fail) from a more complex power plant
- test1.csv and test2.csv are data sets to verify the accuracy of your models for the two power plants

Use the gradient descent algorithm to train a five input, two output (one for each class) neural network using file train1.csv. Write a report addressing the following questions (you should run experiments to support each of your answers):

1. Describe the training performance of the network:

How does the number of hidden units impact the results? How does the training time impact the results? How does the learning rate impact the results? What other critical parameters impacted the results? Note, this is a classification problem, meaning that each data pattern (x1, x2, x3, x4, x5) belongs to one of two classes (y1 or y2). Consequently, use correct classification percentage (instead of MSE) to report your results. You will still use MSE to train the neural networks; you will simply report the classification percentage (or classification error) to assess the performance of the neural networks.

1. Use train2.csv to train another neural network. Answer questions 1.1-1.4 from above for the test set. What conclusions can you draw from your results? What do you think is causing the difference in performance?

In [253…
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# import random
```

Data Loader

In [254…
```python
def load_data(path):
    data = pd.read_csv(path)
    x = data.iloc[:,[0,1,2,3,4]] # first 5 columns of csv as input
    x = np.array(x)
    y = data.iloc[:,[5,6]]       # last 2 as output
    y = np.array(y)

    return x, y

x,y = load_data("train1.csv") # function call

print(x[0], "\n", y[0])
```

```
[-1.4390433  -0.06364541  0.68397101  0.89095435  1.69263034]
 [1. 0.]
```

Define Model

Setup for Tools & Helpers

In [255…

```
# Reference resource: https://www.geeksforgeeks.org/implementation-of-neural-network-fro


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

# derivative for backprop
def sigmoid_prime(x):
    # print("sig'(x): ", np.multiply(sigmoid(x), (1-sigmoid(x))).shape)
    return np.multiply(sigmoid(x), (1-sigmoid(x)))

# feed forward network
def feedForward(x,w1,w2,b1,b2):

    # hidden layer
    z1 = x.dot(w1) + b1 # input from layer 1
    a1 = sigmoid(z1) # output from hidden layer

    # output layer
    z2 = a1.dot(w2) + b2 # input from hidden layer
    a2 = sigmoid(z2) # output from output layer

    return a2

# mse loss
def loss_mse(out, Y):
    s = np.square(out-Y)
    out = np.sum(s)/len(Y)

    return out
```

Training Definition

In [317…

```
# random initialization of weights
def generate_weights(x,y):
    l=[]
    for i in range(x*y):
        l.append(np.random.randn())

    return np.array(l).reshape(x,y)

# random initialization of biases
def generate_biases(x):
    l=[]
```

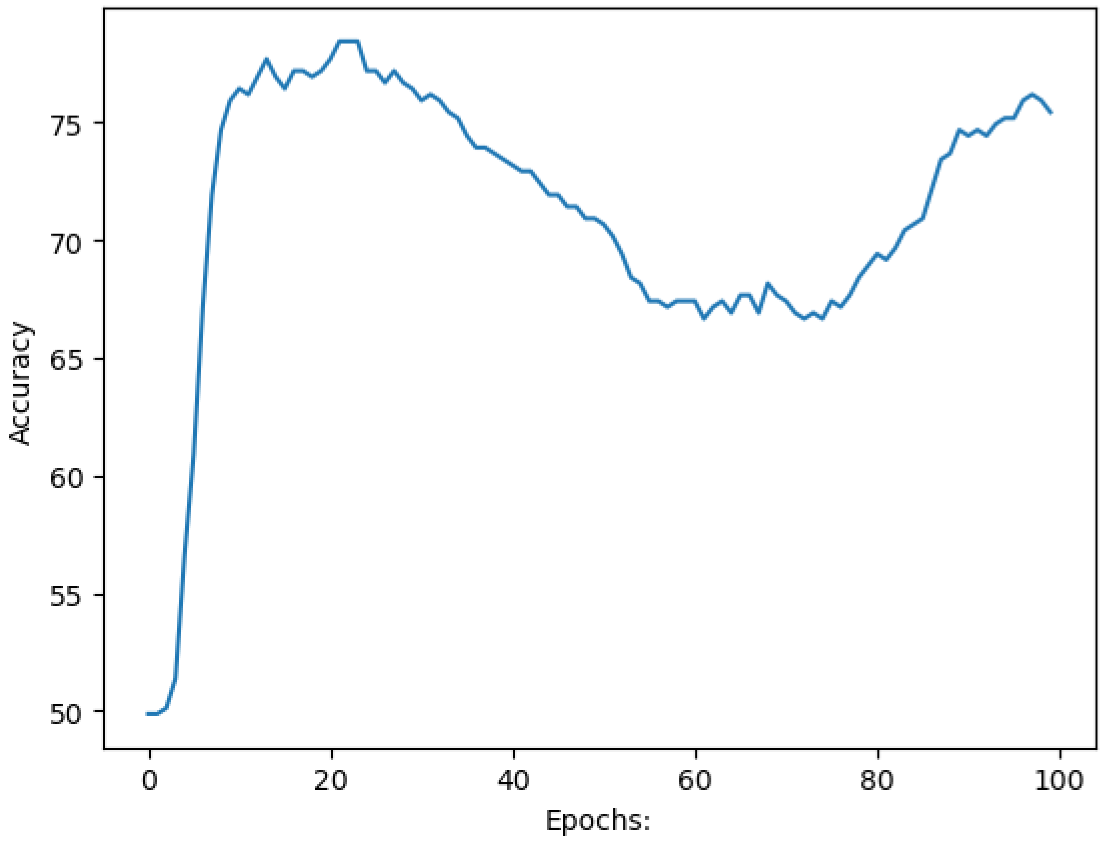
```python
    for i in range(x):
        l.append(np.random.randn())

    return np.array(l).reshape(x)

# backpropegation
def backpropegation(x,y,w1,w2,b1,b2,alpha):

    # hidden layer
    z1 = x.dot(w1) + b1 # input from layer 1
    a1 = sigmoid(z1) # output from hidden layer

    # output layer
    z2 = a1.dot(w2) + b2 # input from hidden layer
    a2 = sigmoid(z2) # output from output layer

    # errors
    output_grad = a2-y
    dE2 = output_grad  # last layer error
    dE1 = np.multiply((w2.dot((dE2.transpose()))).transpose(), sigmoid_prime(a1)) # hidd

    # determine gradients (dE/dw, dE/db)
    w2_adj = a1.reshape(-1,1).dot(dE2.reshape(-1,1).transpose())
    w1_adj = x.reshape(-1,1).dot(dE1.reshape(-1,1).transpose())
    b2_adj = dE2
    b1_adj = dE1

    # update parameters with step size
    w2 = w2 - (alpha * w2_adj)
    b2 = b2 - (alpha * b2_adj)
    w1 = w1 - (alpha * w1_adj)
    b1 = b1 - (alpha * b1_adj)


    return (w1,w2,b1,b2)

# training
def train(x, Y, w1, w2, b1, b2, alpha = 0.01, epoch = 10):
    acc = []
    loss1 = []
    total = len(x)

    for j in range(epoch): #for each epoch
        l = []
        correct = 0
        for i in range(len(x)): # for each datapoint
            out = feedForward(x[i], w1, w2, b1, b2)

            #pass/fail categories
            out_pass = [0,1]
            out_fail = [1,0]

            # classification based on output
            if out[0] < out[1]:
                classif = out_pass
            else:
                classif = out_fail

            #enumerate correct outputs
            if classif[0] == Y[i][0]:
                correct += 1

            l.append(loss_mse(out, Y[i]))

            w1,w2,b1,b2 = backpropegation(x[i],y[i],w1,w2,b1,b2,alpha)
            temp_acc = (correct/total)*100

        if (j % 25 == 0): print("epochs: ", j, "======= acc: ", temp_acc)

        acc.append(temp_acc)
        loss1.append(sum(l)/len(x))

    return (acc, loss1, w1, w2, b1, b2)
```

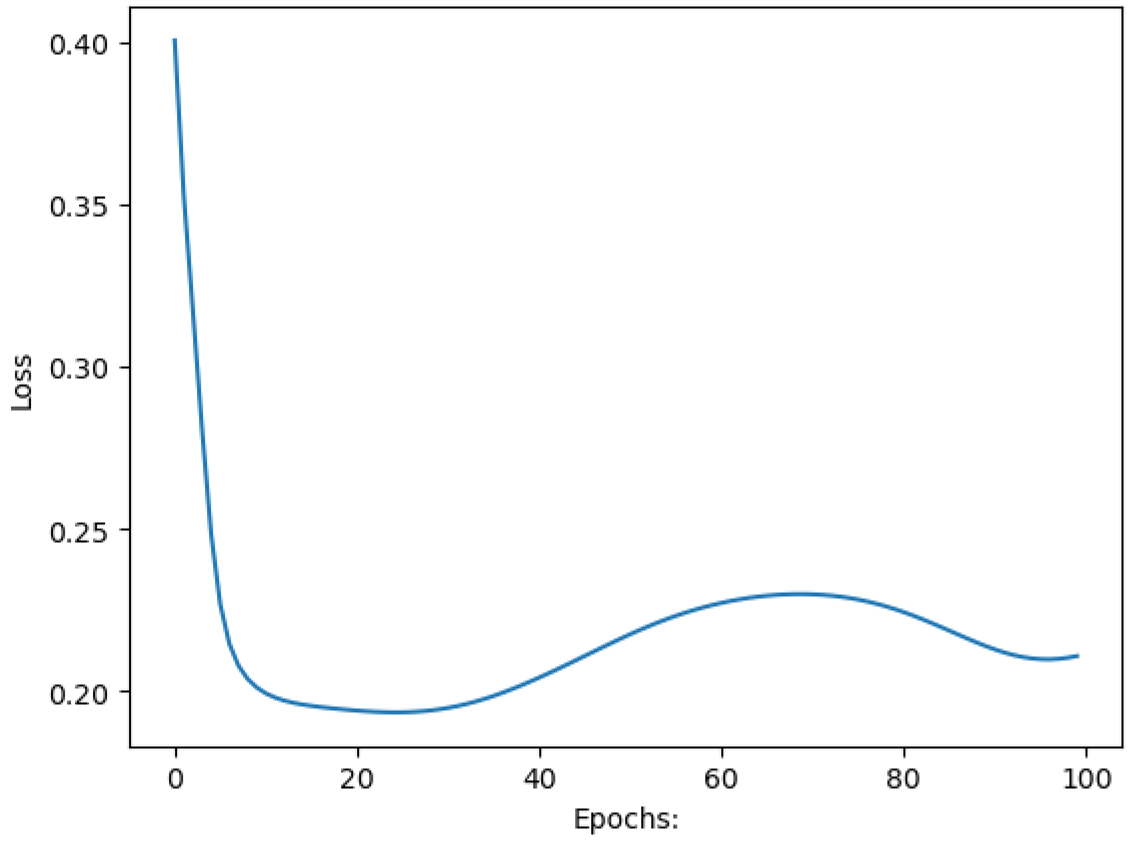
Training

In [318…

```
# initialize network
w1 = generate_weights(5, 10)
b1 = generate_biases(10)
w2 = generate_weights(10, 2)
b2 = generate_biases(2)

STEP_SIZE = 0.001
EPOCH = 100

# do training
acc, loss1, w1, w2, b1, b2 = train(x, y, w1, w2, b1, b2, STEP_SIZE, EPOCH)

# plotting accuracy
plt.plot(acc)
plt.ylabel('Accuracy')
plt.xlabel("Epochs:")
plt.show()

# plotting Loss
plt.plot(loss1)
plt.ylabel('Loss')
plt.xlabel("Epochs:")
plt.show()
```

```
epochs:  0 ======= acc:  49.87468671679198
epochs:  25 ======= acc:  77.19298245614034
epochs:  50 ======= acc:  70.67669172932331
epochs:  75 ======= acc:  67.41854636591479
```





Validation

In [336…

```
x,Y = load_data("test1.csv") # load test data


def test(x,Y,w1,w2,b1,b2):
    out = feedForward(x,w1,w2,b1,b2)
    total = len(x)
    correct = 0
    for i in range(len(x)): # for each datapoint
        out = feedForward(x[i], w1, w2, b1, b2)

        #pass/fail categories
        out_pass = [0,1]
        out_fail = [1,0]

        # classification based on output
        if out[0] < out[1]:
            classif = out_pass
        else:
            classif = out_fail

        #enumerate correct outputs
        if classif[0] == Y[i][0]:
            correct += 1

    acc = (correct/total)*100

    return acc
```

Automated Testing

In [348…

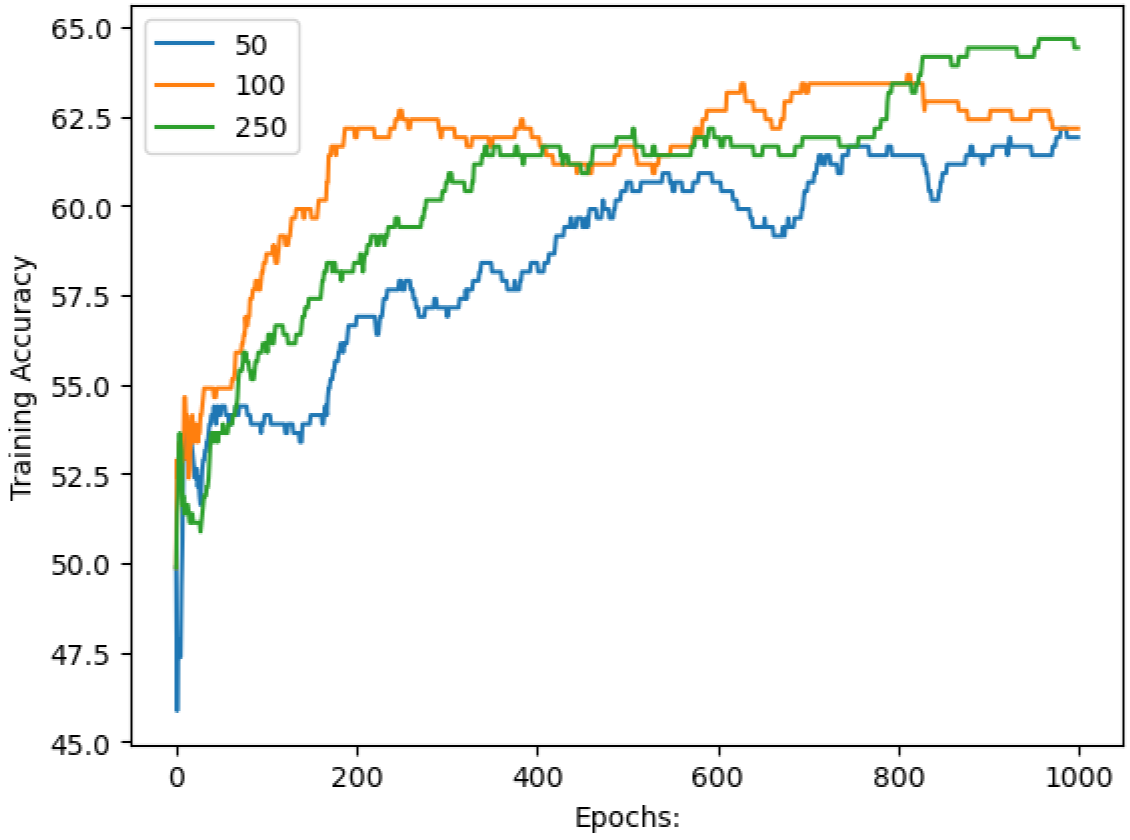
```
def train_and_test(filepath_train, filepath_test, HIDDEN_UNITS, ALPHA, EPOCHS):
    print("Training on ", filepath_train, " and testing with ", filepath_test)
    print("Hidden Units: ", HIDDEN_UNITS, " | STEP SIZE: ", ALPHA, " | EPOCHS: ", EPOCHS

    x,Y = load_data(filepath_train) # load test data

    # initialize network
    w1 = generate_weights(5, HIDDEN_UNITS)
    b1 = generate_biases(HIDDEN_UNITS)
    w2 = generate_weights(HIDDEN_UNITS, 2)
    b2 = generate_biases(2)

    STEP_SIZE = ALPHA
    EPOCH = EPOCHS

    # do training
    acc, loss1, w1, w2, b1, b2 = train(x, y, w1, w2, b1, b2, STEP_SIZE, EPOCH)

    # do testing
    x,Y = load_data(filepath_test) # load test data
    acc_test = test(x,Y,w1,w2,b1,b2)

    return acc, loss1, acc_test

# TESTING SETUP
accs = []
losses = []
acc_tests = []

acc, loss1, acc_test = train_and_test("train2.csv", "test2.csv", 50, 0.001, 1000)
accs.append(acc)
losses.append(loss1)
acc_tests.append([acc_test, 50])
acc, loss1, acc_test = train_and_test("train2.csv", "test2.csv", 100, 0.001, 1000)
accs.append(acc)
losses.append(loss1)
acc_tests.append([acc_test, 100])
acc, loss1, acc_test = train_and_test("train2.csv", "test2.csv", 250, 0.001, 1000)
accs.append(acc)
losses.append(loss1)
acc_tests.append([acc_test, 250])
```

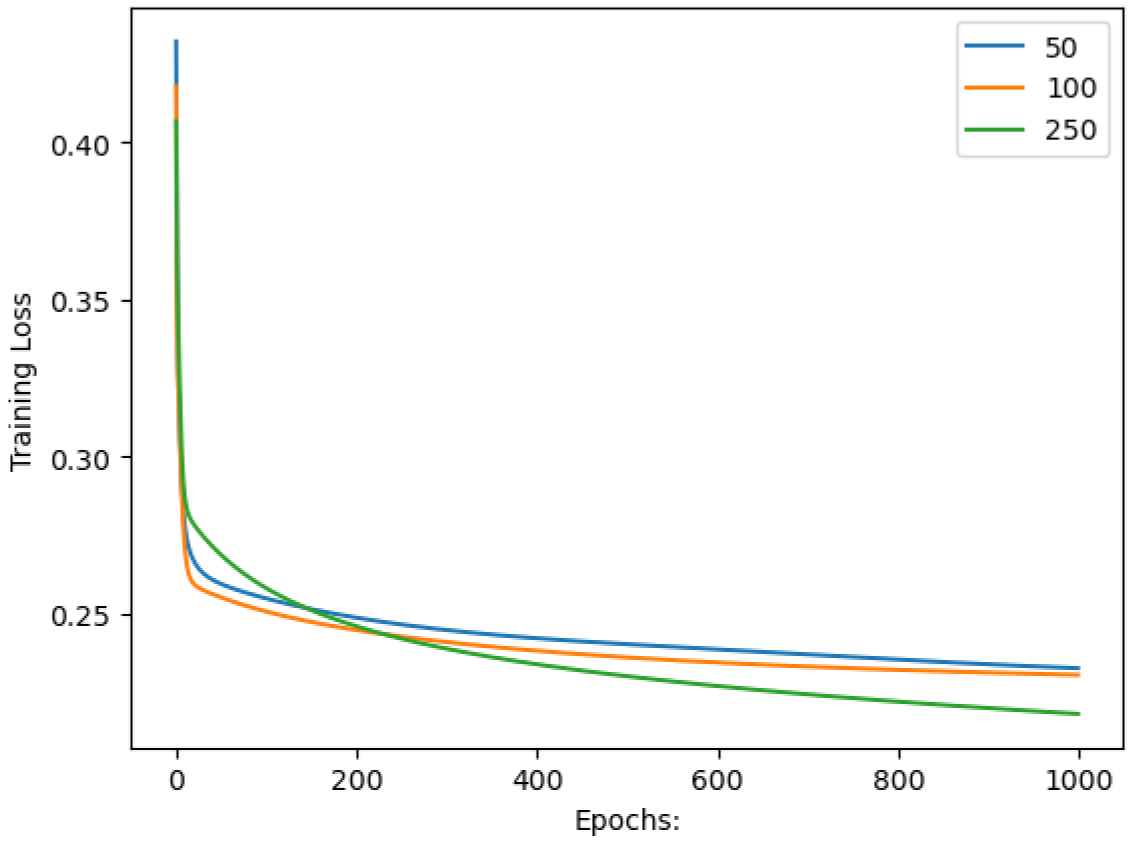
```
# plotting accuracy
for i in accs:
    plt.plot(i)
plt.ylabel('Training Accuracy')
plt.xlabel("Epochs:")
plt.legend([50, 100, 250])
plt.show()

# plotting Loss
for i in losses:
    plt.plot(i)
plt.ylabel('Training Loss')
plt.xlabel("Epochs:")
plt.legend([50, 100, 250])
plt.show()

# testing results
print("Testing Accuracies: ", acc_tests)
```

```
Training on  train2.csv  and testing with  test2.csv
Hidden Units:  50  | STEP SIZE:  0.001  | EPOCHS:  1000
epochs:  0 ======= acc:  49.87468671679198
epochs:  25 ======= acc:  52.38095238095239
epochs:  50 ======= acc:  54.385964912280706
epochs:  75 ======= acc:  54.385964912280706
epochs:  100 ======= acc:  54.13533834586466
epochs:  125 ======= acc:  53.88471177944862
epochs:  150 ======= acc:  54.13533834586466
epochs:  175 ======= acc:  55.388471177944865
epochs:  200 ======= acc:  56.89223057644111
epochs:  225 ======= acc:  56.64160401002506
epochs:  250 ======= acc:  57.64411027568922
epochs:  275 ======= acc:  56.89223057644111
epochs:  300 ======= acc:  56.89223057644111
epochs:  325 ======= acc:  57.64411027568922
epochs:  350 ======= acc:  58.1453634085213
epochs:  375 ======= acc:  57.89473684210527
epochs:  400 ======= acc:  58.39598997493734
epochs:  425 ======= acc:  59.14786967418546
epochs:  450 ======= acc:  59.3984962406015
epochs:  475 ======= acc:  59.899749373433586
epochs:  500 ======= acc:  60.6516290726817
epochs:  525 ======= acc:  60.6516290726817
epochs:  550 ======= acc:  60.40100250626567
epochs:  575 ======= acc:  60.6516290726817
epochs:  600 ======= acc:  60.6516290726817
epochs:  625 ======= acc:  59.899749373433586
epochs:  650 ======= acc:  59.3984962406015
epochs:  675 ======= acc:  59.3984962406015
epochs:  700 ======= acc:  60.6516290726817
epochs:  725 ======= acc:  61.152882205513784
epochs:  750 ======= acc:  61.65413533834586
epochs:  775 ======= acc:  61.40350877192983
epochs:  800 ======= acc:  61.40350877192983
epochs:  825 ======= acc:  61.40350877192983
epochs:  850 ======= acc:  60.902255639097746
epochs:  875 ======= acc:  61.40350877192983
epochs:  900 ======= acc:  61.40350877192983
epochs:  925 ======= acc:  61.65413533834586
epochs:  950 ======= acc:  61.40350877192983
epochs:  975 ======= acc:  61.904761904761905
Training on  train2.csv  and testing with  test2.csv
Hidden Units:  100  | STEP SIZE:  0.001  | EPOCHS:  1000
epochs:  0 ======= acc:  49.87468671679198
epochs:  25 ======= acc:  53.63408521303258
epochs:  50 ======= acc:  54.88721804511278
```

```
epochs:  75 ======= acc:  56.390977443609025
epochs:  100 ======= acc:  58.64661654135338
epochs:  125 ======= acc:  59.14786967418546
epochs:  150 ======= acc:  59.64912280701754
epochs:  175 ======= acc:  61.65413533834586
epochs:  200 ======= acc:  61.904761904761905
epochs:  225 ======= acc:  61.904761904761905
epochs:  250 ======= acc:  62.65664160401002
epochs:  275 ======= acc:  62.40601503759399
epochs:  300 ======= acc:  62.15538847117794
epochs:  325 ======= acc:  61.65413533834586
epochs:  350 ======= acc:  61.65413533834586
epochs:  375 ======= acc:  61.904761904761905
epochs:  400 ======= acc:  61.904761904761905
epochs:  425 ======= acc:  61.152882205513784
epochs:  450 ======= acc:  61.152882205513784
epochs:  475 ======= acc:  61.152882205513784
epochs:  500 ======= acc:  61.65413533834586
epochs:  525 ======= acc:  61.152882205513784
epochs:  550 ======= acc:  61.65413533834586
epochs:  575 ======= acc:  62.15538847117794
epochs:  600 ======= acc:  62.65664160401002
epochs:  625 ======= acc:  63.1578947368421
epochs:  650 ======= acc:  62.65664160401002
epochs:  675 ======= acc:  62.907268170426065
epochs:  700 ======= acc:  63.1578947368421
epochs:  725 ======= acc:  63.40852130325815
epochs:  750 ======= acc:  63.40852130325815
epochs:  775 ======= acc:  63.40852130325815
epochs:  800 ======= acc:  63.40852130325815
epochs:  825 ======= acc:  63.40852130325815
epochs:  850 ======= acc:  62.907268170426065
epochs:  875 ======= acc:  62.65664160401002
epochs:  900 ======= acc:  62.40601503759399
epochs:  925 ======= acc:  62.65664160401002
epochs:  950 ======= acc:  62.65664160401002
epochs:  975 ======= acc:  62.15538847117794
Training on  train2.csv  and testing with  test2.csv
Hidden Units:  250  | STEP SIZE:  0.001  | EPOCHS:  1000
epochs:  0 ======= acc:  49.87468671679198
epochs:  25 ======= acc:  51.127819548872175
epochs:  50 ======= acc:  53.63408521303258
epochs:  75 ======= acc:  55.88972431077694
epochs:  100 ======= acc:  56.14035087719298
epochs:  125 ======= acc:  56.14035087719298
epochs:  150 ======= acc:  57.393483709273184
epochs:  175 ======= acc:  58.39598997493734
epochs:  200 ======= acc:  58.39598997493734
epochs:  225 ======= acc:  59.14786967418546
epochs:  250 ======= acc:  59.3984962406015
epochs:  275 ======= acc:  59.899749373433586
epochs:  300 ======= acc:  60.6516290726817
epochs:  325 ======= acc:  60.40100250626567
epochs:  350 ======= acc:  61.40350877192983
epochs:  375 ======= acc:  61.40350877192983
epochs:  400 ======= acc:  61.40350877192983
epochs:  425 ======= acc:  61.65413533834586
epochs:  450 ======= acc:  60.902255639097746
epochs:  475 ======= acc:  61.65413533834586
epochs:  500 ======= acc:  61.904761904761905
epochs:  525 ======= acc:  61.40350877192983
epochs:  550 ======= acc:  61.40350877192983
epochs:  575 ======= acc:  61.904761904761905
epochs:  600 ======= acc:  61.904761904761905
epochs:  625 ======= acc:  61.65413533834586
epochs:  650 ======= acc:  61.65413533834586
```

```
epochs:  675 ======= acc:  61.40350877192983
epochs:  700 ======= acc:  61.904761904761905
epochs:  725 ======= acc:  61.904761904761905
epochs:  750 ======= acc:  61.65413533834586
epochs:  775 ======= acc:  62.15538847117794
epochs:  800 ======= acc:  63.40852130325815
epochs:  825 ======= acc:  63.65914786967418
epochs:  850 ======= acc:  64.16040100250626
epochs:  875 ======= acc:  64.16040100250626
epochs:  900 ======= acc:  64.41102756892231
epochs:  925 ======= acc:  64.41102756892231
epochs:  950 ======= acc:  64.16040100250626
epochs:  975 ======= acc:  64.66165413533834
```





```
Testing Accuracies:  [[46.365914786967416, 50], [47.86967418546366, 100], [52.6315789473
6842, 250]]
```