CS 535 Deep Learning, Assignment 2 –Aashish Adhikari

Q1.Answer

In my implementation, I have two functions.

The first function calculate_misclassification_rate() calculates the misclassifications that the network makes after it gets trained. It has been shown below.

```
def calculate_misclassification_rate(self, y, y_cap):

    y_cap = self.conversion_y_cap(y_cap)
    check = (y_cap == y)
    misclassifications = len((np.where(check==False))[0]) #shortcut
    return misclassifications
```

The second function cross_entropy_loss_calculation() calculates the cross entropy loss plus the L2 regularization loss and returns an average of the batch. It can be seen below and has been implemented in the code.

```
def cross_entropy_loss_calculation(self, y, y_cap, l2_penalty_factor):


    total_cost = L2_regularization_cost + Cross_Entropy_loss
            #using two costs for better performance
    cross_entropy_loss_batch_average = (-1.0)*np.sum(cumulative_cross_entropy)/len(y)


    return cross_entropy_loss_batch_average
```

Sub-gradients

The subgradients necessary for the task have been defined in the implementation itself as

dE_by_dW2 = self.hidden_to_output_layer.backward(dE_by_dA2, dA2_by_dW2)

where dE_by_dA2 is the gradient outout from next layer fed to this layer and dA2_by_dW2 is in fact the input to the layer(here Z1 itself).

dE_by_dW1 = self.input_to_hidden_layer.backward(dE_by_dA1, x_batch)

where dE_by_dA1 = gradient output from the next layer fed to this layer and x_batch is the input of this layer itself.

Q2.Answer

Stochastic mini-batch gradient descent has been implemented as my code in the program itself.

Please note that the question does not require us to shuffle the data after each epoch.

The momentum approach has also been used which takes into account not only the changes that come from the gradient of the error with respect to error and regularization term, but also the direction of the previous gradient.

Q3Answer

I started with only learning rates for a vanilla Neural Network. I got 78.00% accuracy on 0.001 learning rate much earlier than from a learning rate of 0.0008. It took about 200 epochs for 0.0008 to give this accuracy while 0.001 gave this accuracy before 100 epochs.
Then, taking the number of hidden nodes as 50, I started changing the momentum values to get the best momentum. 0.9 and 0.8 were stuck at an accuracy of 50%. I believe it was because the original weights were themselves small and taking a large momentum meant restricting a change in the weights when the weight changes were already small. The momentum of 0.7,0.6,0.5,0.55,0.575,0.59,and 0.7 gave the accuracies of 75.05,75.75,76.55,77.50,77.75,78.10,and 76.55. However, these relative accuracies with respect to the momentum values did not persist once I introduced changes in other hyper-parameters.

After several combination checks for different hyper-parameters, I set the momentum of 0.8 which produced the result that was not converging in a reasonable time. So I chose 0.7. Since the professor mentioned that we do not have to tune the momentum as it is always taken a standard between 0.7 and 0.8, I kept it the same after this step.

Then I looked for the variation in the accuracy with different L2 penalty factors and obtained the best testing accuracy of 79.65% for the penalty factor of 0.0001. The values of different hyper-parameter combinations are given in the questions that follow. Note that the neural network overfits on the training data beyond this testing accuracy and we see an increase in the loss for testing data. The training accuracy goes beyond 96 percent.

Q4.Answer

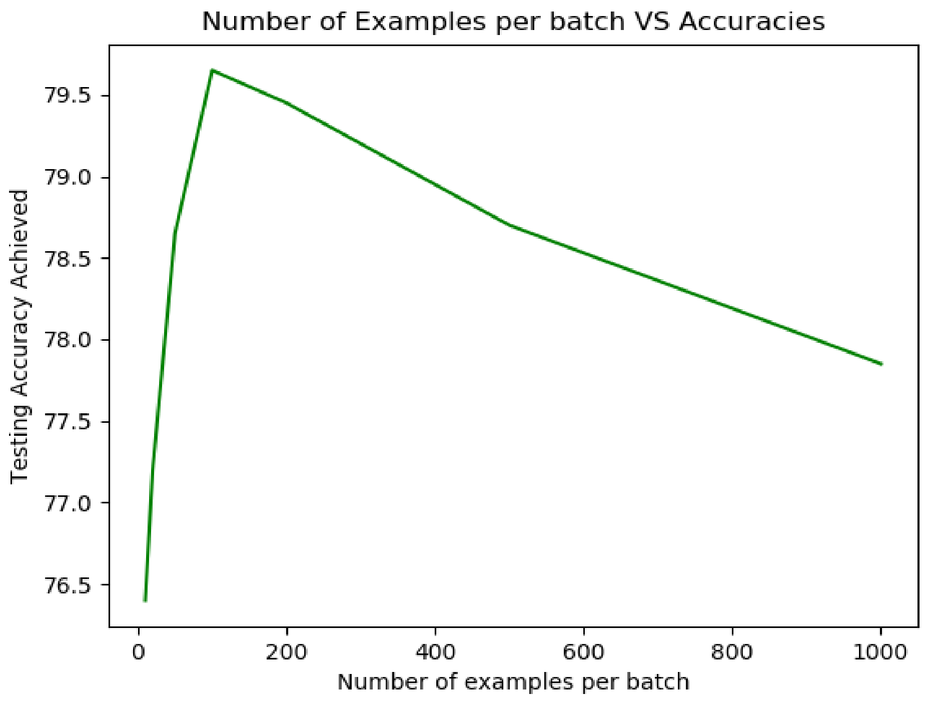
The answer for this question has been implemented and the program prints training objective, testing objective, training misclassification error, and testing misclassification error on the console. Note that the loss has been scaled down for a batch. Also, it can be seen that the loss decreases with the decrease in misclassifications. However, once the model starts to overfit on the training data, we get an increasing loss on the testing set.

Q5.Answer

a.**Test Accuracy with different batch sizes**

The parameters used for this data are Learning Rate = 0.0001 Batch Size:100, L2 penalty factor= 0.0001, Momentum = 0.7

Number of Examples per batch	Testing Accuracy in %
10	76.40
20	77.20
50	78.65
100	79.85
200	79.45
500	78.70
1000	77.85



b.

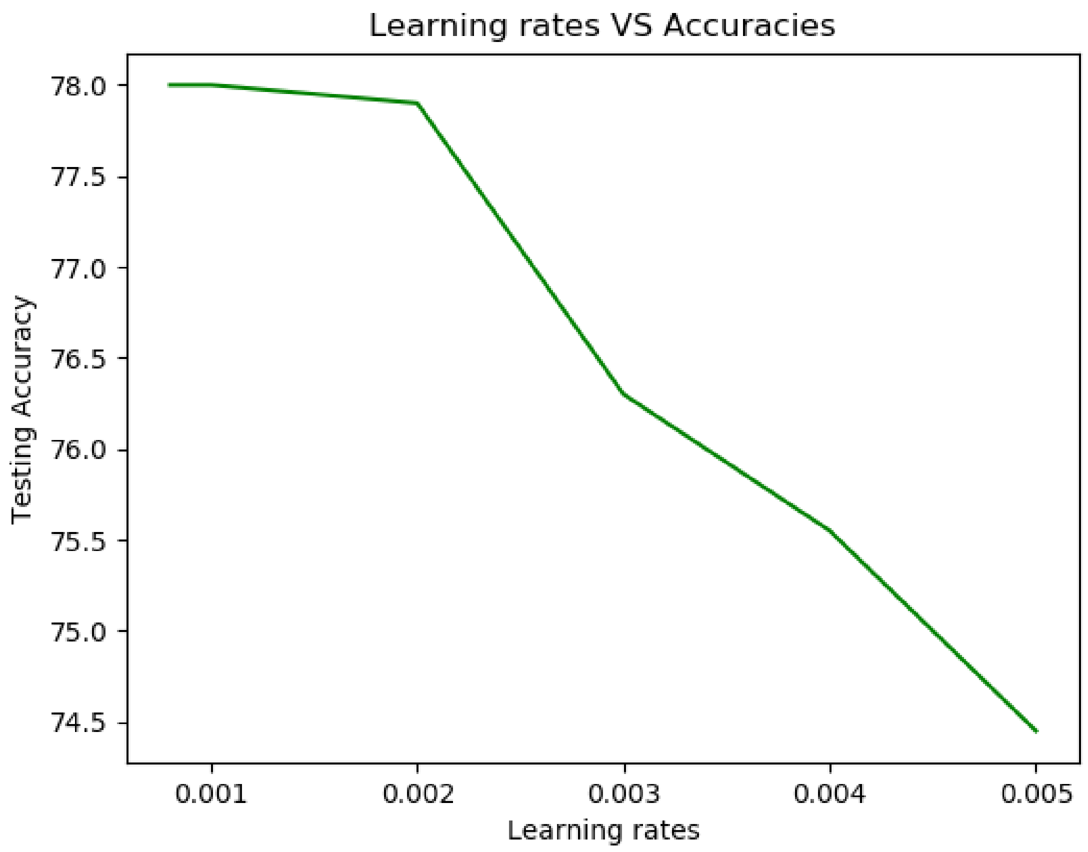
The graph below shows the relationship between the learning rate and the testing accuracy.

The corresponding values can be seen on the table as well. Please note that the set points are not equidistant but instead taken according to the judgement during the experiment.

The parameters used for this data are Batch Size: 1000, Number of hidden Units: 50, L2 penalty factor= 0, Momentum = 0

Learning Rate	Testing Accuracy in %
0.005	74.45
0.004	75.55(increasing)
0.003	76.30(increasing)
0.002	77.90(increasing)
0.001	78.00(maximum test accuracy achieved)
0.0008	78.00 (max test accuracy within 200 epochs)
0.00075	77.25(max test accuracy within 200 epochs)
0.0005	77.45(max test accuracy within 200 epochs)
0.0001	76.20(max test accuracy within 200 epochs)

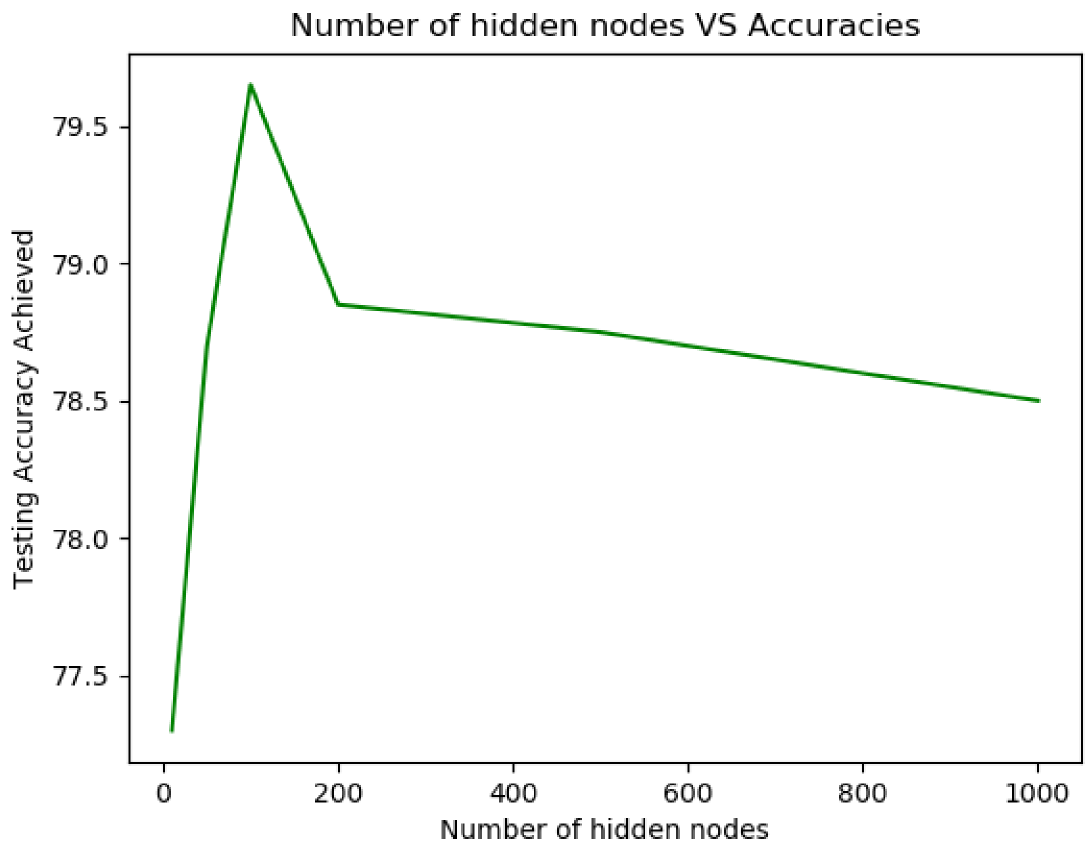
Please note that the values less than 0.001 achieved those accuracies below 150 epochs and weight updates were very slow for them to continue. Thus 0.001 was chosen. However, using other factors into account to train the model, 0.0001 gives a better accuracy and that was used as a reference for other graphs. **The graph for this question does not include all the data in this table for the sake of the trend of data.**



c.

Testing accuracy with different number of hidden units

Number of Hidden Nodes	Testing Accuracy in %
10	77.30
50	78.70
100	79.65
200	78.85
500	78.75
1000	78.50



.Answer.

My neural network gives a training accuracy that reaches beyond 96% for training data. However, while doing so, it overfits on the training data and a decrease in the testing accuracy can be seen after 79.65%. I would like to believe that the network should have performed better. However, despite several attempts and writing two separate programs to make sure I did not make any silly mistake, I still got similar results. I tried the following to increase the accuracy:

Increase the number of the hidden nodes

Decrease the batch size

Try a different learning rate

Try a different coefficient of momentum

Try a different penalty factor

But 79.65% was the best achievable accuracy for the network even after more than 500 epochs.

I tried debugging the code and everything looks fine there. So I assume that since different initializations start from a different state, a network's architecture can show varying results. I verified it by running the model for the same set of hyper-parameters and I could see some variation in the results for each different run.

As seen in the figure, the best accuracy is given by 100 hidden nodes and the accuracy decreases for batch sizes less or greater than that.

0.001 is the best learning rate for the vanilla Neural Network. Other rates smaller than this also achieved the same accuracy but in a time that was comparably way longer than the time taken by 0.001. If we introduce other hyper-parameters, other learning rates perform better. For example, in my case, it was 0.0001.

Also, a batch size of 100 gave the best accuracy as we can see in the graph. I had the feeling that smaller the batch size, better the accuracy but the results showed otherwise. I checked with some other students and many have received similar observations.