Bonus report assignment 2:

Shiva Besharat Pour
shivabp@kth.se

Exercise 1:

The purpose of this exercise was to optimize the network implemented in the default assignment. For this purpose, I chose the 3 methods:

- Exhaustive random search to find a good lambda
- Explore effects of more hidden nodes
- Applying noise to data

In subsections below, I will present the results I achieved for each of the above optimizations.

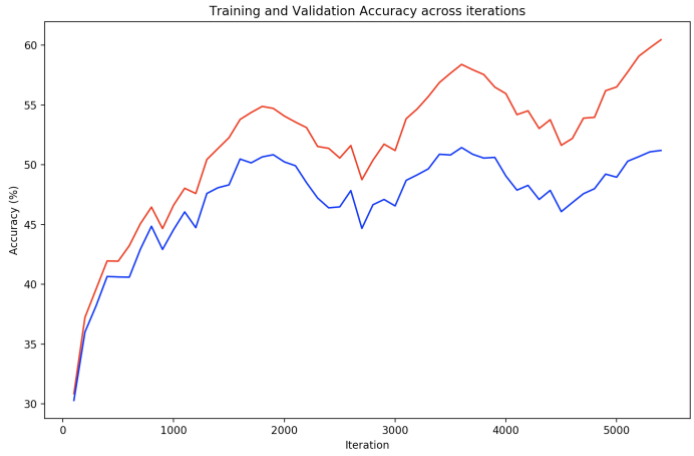
Exhaustive random search for lambda:

For this optimization, I just increased the number of iterations through which I searched for lambda from 15 to 20.

```
n_lambda = 20
n_cycles = 3
```

After 20 iterations, the results suggested a lambda value of: 0.000432 compared to the previous value of 0.002866 as found in the default assignment.

```
Lamda: 0.000432  n_s: 900  total iterations of: 5400  total batches of 450  for 12  epochs of training:
Best accuracy achieved on training set:  60.282222
Best accuracy achieved on validation set:  52.520000
Final test accuracy  51.350000
```




```
Test accuracy:  50.949999999999996
```

The test accuracy has not exactly increased which would be perhaps due to unluckiness. Since the calculations for lambda values included random numbers, this particular search was just unlucky compared to the search performed in the default assignment where a test accuracy of 51.91 was achieved.

Effects of hidden nodes:

I tried different number of hidden nodes with 50, 60, 80 and 100 nodes and the results are as follows:

50 hidden nodes:

```
Lamda: 0.000032  n_s: 900  total iterations of: 3600  total batches of 450  for 8  epochs of training:
Best accuracy achieved on training set:  58.033333
Best accuracy achieved on validation set:  51.560000
Final test accuracy  51.090000

 Test accuracy:  50.64999999999999
```

60 hidden nodes:

```
Lamda: 0.000023  n_s: 900  total iterations of: 3600  total batches of 450  for 8  epochs of training:
Best accuracy achieved on training set:  59.831111
Best accuracy achieved on validation set:  52.420000
Final test accuracy  51.860000

 Test accuracy:  50.99
```

80 hidden nodes:

```
Lamda: 0.002316  n_s: 900  total iterations of: 3600  total batches of 450  for 8  epochs of training:
Best accuracy achieved on training set:  60.260000
Best accuracy achieved on validation set:  52.700000
Final test accuracy  52.750000

 Test accuracy:  52.61
```

100 hidden nodes:

```
Lamda: 0.002063  n_s: 900  total iterations of: 3600  total batches of 450  for 8  epochs of training:
Best accuracy achieved on training set:  61.344444
Best accuracy achieved on validation set:  53.820000
Final test accuracy  53.130000

Test accuracy:  52.980000000000004
```

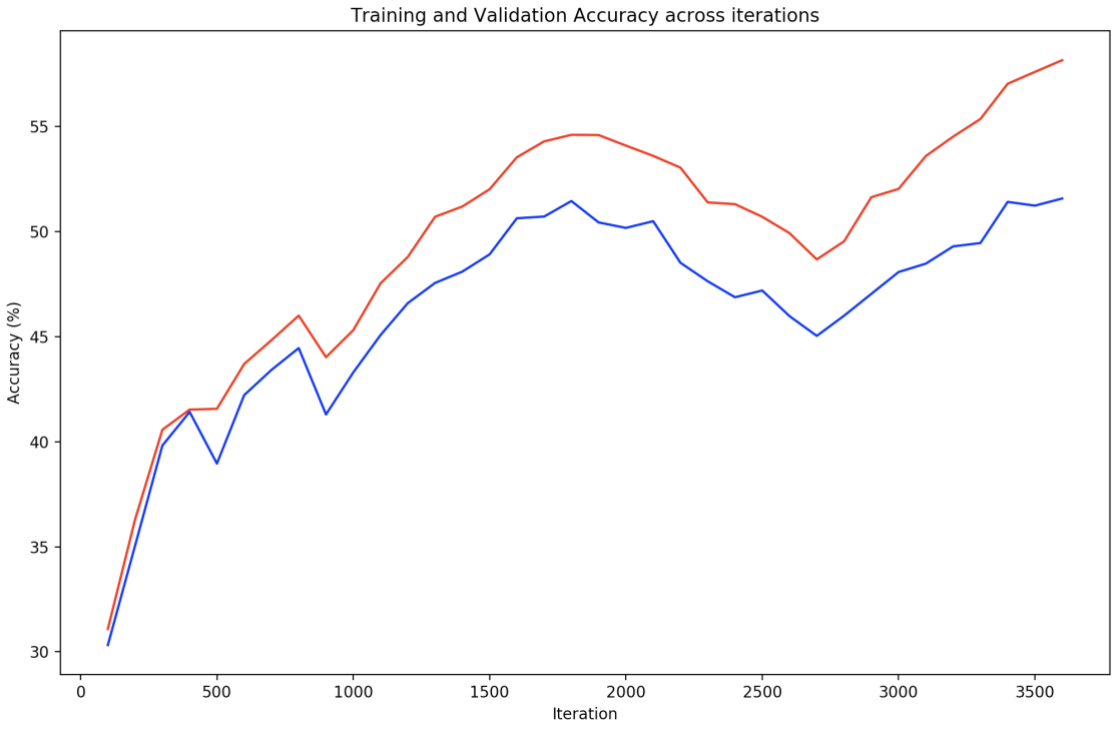
It is important to note that the rest of the parameters were kept constant except for the number of hidden nodes for the sake of reliability. Increasing number of nodes did indeed help improve accuracy which would perhaps be because the original number of nodes being 50 was too little. This may have caused the network being undertrained. I could keep increasing the number of hidden nodes and I think it would continue to improve the accuracy a little further. However, the runtime was getting slower and slower and I did arrive at a conclusion that increasing number of nodes was effective.

Applying noise:

After creating the batches, a random noise was added to the X_batch.

```
    X_batch = X[: , j_start: j_end]
    Y_batch  = Y[: , j_start:j_end]
    noise = np.random.normal(0 , 0.01 , (X_batch.shape[0] ,X_batch.shape[1] ))
    X_batch = np.add(X_batch   , noise)
```

The effects of applying noise to data turned out as follows:




```
Test accuracy:  50.74999999999999
```

The final accuracy seems to have decreased rather than increase. Previous accuracy from the default assignment was ~52%. This could perhaps confirm the conclusion from increasing number of hidden nodes that 50 may have been too few. Noise is usually effective when the network has started to over fit the data. However, given that the number of hidden nodes being 50 originally was too few, the network would be under fit at this point and thereby adding noise, will make the network rather more under fit.

From the optimization methods chose, increasing number of nodes seemed the most relevant and effective one since my hypothesis is that 50 nodes were too few to train the network and network was under trained.

Exercise 2:

The purpose of this exercise was to perform a grid search in order to find more optimal values to define the range of eta-values, the learning rates. For this purpose, I implemented a greed search function similar to the one implemented for lambda. The greed search searches through a range of possible eta_min values with the static eta_max being 0.1. I then search for the most optimal eta_min value chosen randomly.

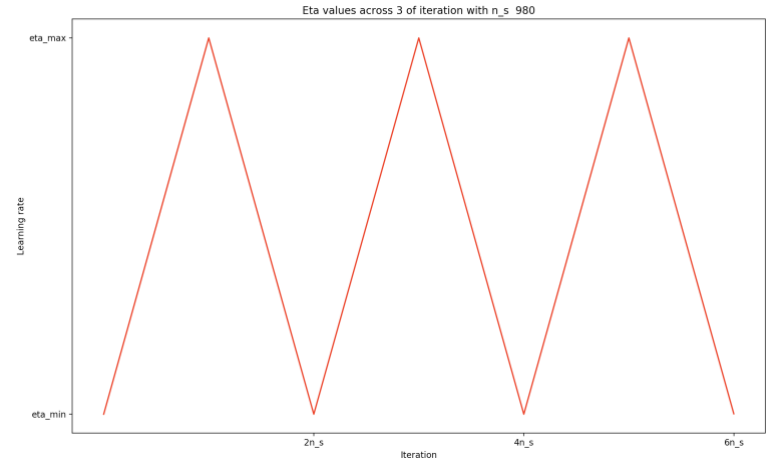
```
def coarseToFine():
    lamdaValues = list()
    for i in range(n_lambda):
        lamda = cycleLambda()
        lamdaValues.append(lamda)
    path = "/Users/shivabp/Desktop/DD2424/Labs/Lab 2/Results/bonus1_search/Coarse-to-fine" + str(n_lambda) +".txt"
    file = open(path , "w+" )
    file.write("Results for %i lambda values within the range [ %i , %i ] with batch size of %i :\n\n" %(n_lambda , l_min , l_max , n_batch) )
    for lamda in lamdaValues:
        W1, W2 , b1, b2 = initParams()
        train = miniBatchGradientDescent(lamda , eta_min ,  W1, W2  , b1, b2 )
        params , iters,  accuracyValues  , accuracyValValues ,  testAcc = train
        bestValidResults = np.max(accuracyValValues)
        bestTrainResults = np.max(accuracyValues)
        file.write("Lamda: %f  n_s: %i  total iterations of: %i  total batches of %i  for %i  epochs of training:\n" %(lamda , params[0],params[1] , params[2]
        file.write("Best accuracy achieved on training set:  %f\n" %(bestTrainResults)  )
        file.write("Best accuracy achieved on validation set:  %f\n" %(bestValidResults) )
        file.write("Final test accuracy  %f\n\n\n" %(testAcc) )
    file.close()
```

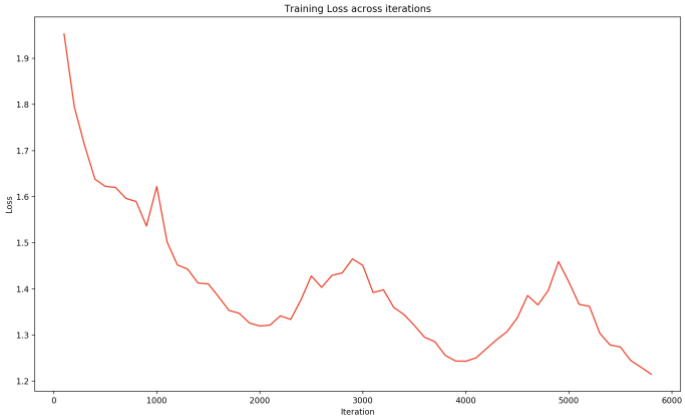
The static parameters used for the search except for eta are taken from the parameters that trained the best performing network from previous exercises.

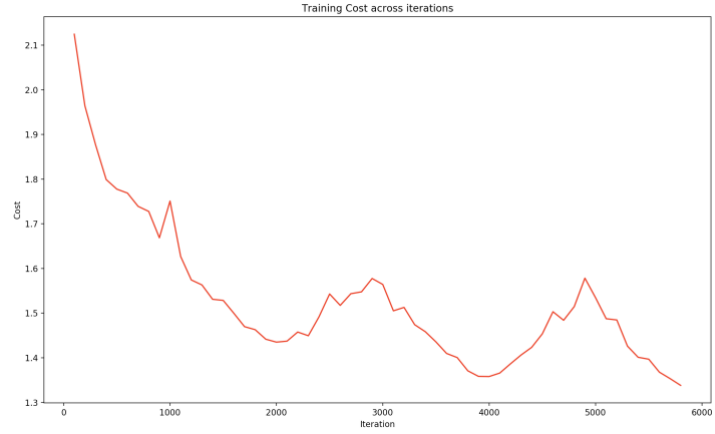
```
d = 3072
k = 10
m = 50
mu = 0
sigma1 =  1 / math.sqrt(d)
sigma2 = 1 / math.sqrt(m)
h = 1e-5
lamda = 0.002866
eta_max = 1e-1
eta_min = 1e-5
n_eta = 10
n_batch = 100
n_cycles = 3
```

n_eta is the number of iterations for the search. As requested by the assignment instructions, I present the training loss and cost plots for both before and after using the optimal eta_min value found.

Initial performance:







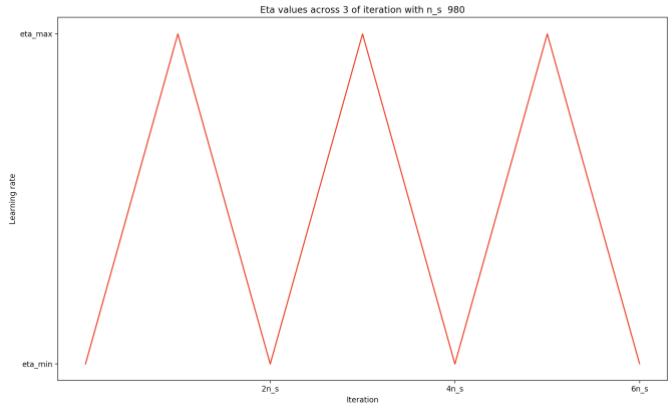

```
Test accuracy:  52.129999999999995
```

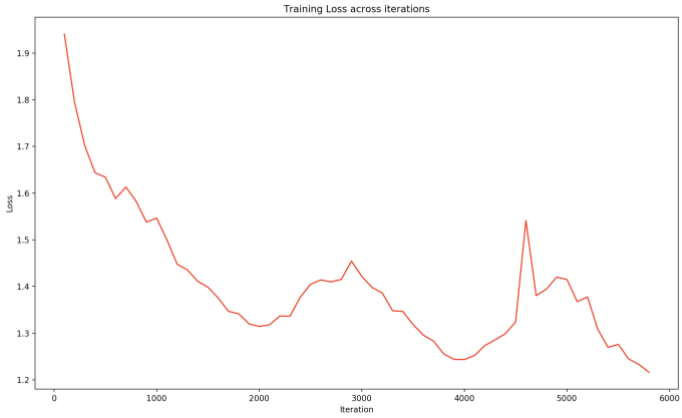
Result of the search:

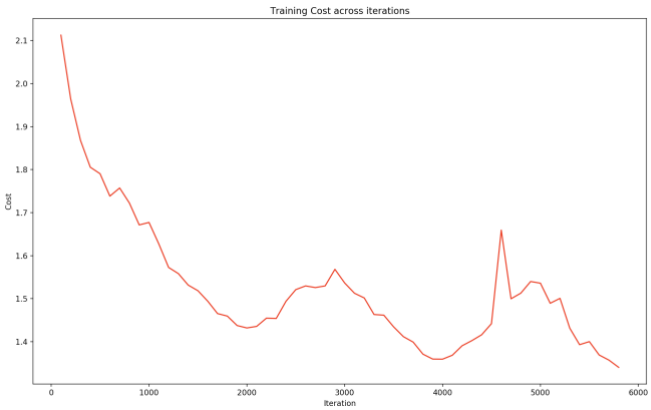
```
Results for 10 eta values within the range [ 0.000017 , 0.100000 ] with batch size of 100 :

Results for total iterations of: 5880  total batches of 490  for 12  epochs of training:
Final test accuracy  52.490000
```

Final performance:








```
Test accuracy:  52.0
```

The grid search retuned the optimal eta_min very close the original value of 1e-5 as suggested by the instructions. Therefore, it is reasonable that the cost/loss as well as final classification accuracy are very similar. I suppose perhaps modifying the range and performing more exhaustive searches could help. I did try doing so and I did get some improvements however the min_value retuned and final accuracy as well as training loss/cost remain very similar before and after. Also, a further improvement could be to perform the grid search for combinations of both eta_min and eta_max. However, such search would be a bit more complex and requires more running time.