Bonus Report assignment 1

Shiva Besharat Pour
shivabp@kth.se

Bonus Assignment 1:

In this assignment, the network was to be optimized. I chose the following 3 methods to optimize the network performance with. For all the following plots, blue line corresponds to the validation data and the red line corresponds to the training data.

a) Eta decay

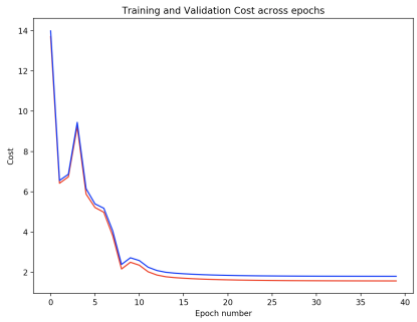
For this purpose, I implemented the following function which decays the eta value:

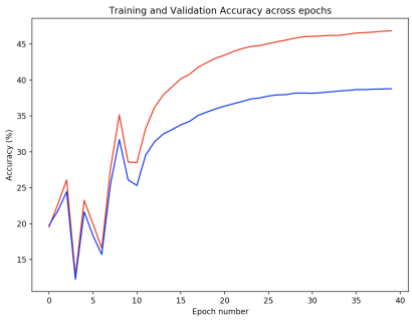
```
def etaDecay(eta):
    eta = eta * 0.9
    return eta
```

I call this function at the end of each epoch in my mini-batch gradient descent in order to update the value of eta that is to be used in the next epoch. I chose the 4 different test cases suggested by the assignment instructions to change parameters and experiment with the network. All other optimizations were turned off for this test. The results are as follows:
Batch size: 100
Number of epochs: 40

Test 1: **ETA= 0.1** **Lambda= 0**

Test accuracy before optimization: 25.71

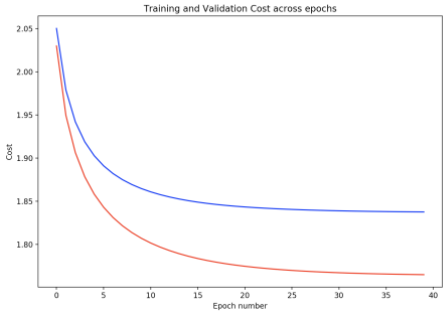





```
Test accuracy:  38.6
```

Test 2: **ETA= 0.01** **Lambda=0**

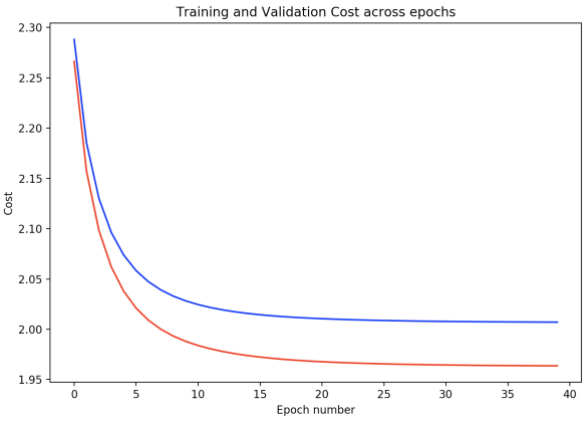
Test accuracy before optimization: 36.87

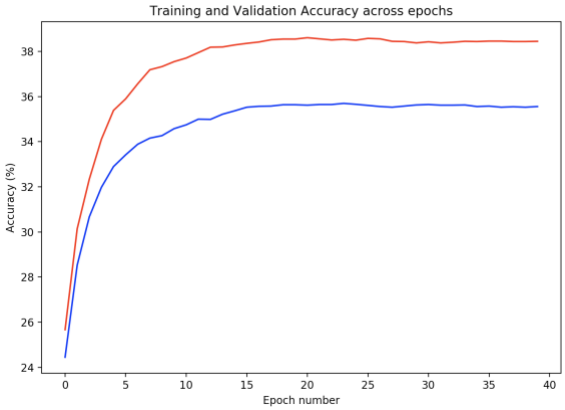



```
Test accuracy:  37.46
```

Test 3: ETA= 0.01 Lambda=0.1

Test accuracy before optimization: 33.39

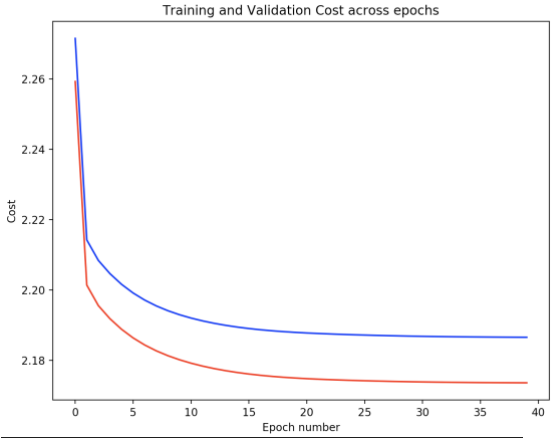





```
Test accuracy:  36.49
```

Test 4: ETA= 0.01 Lambda=1

Test accuracy before optimization: 21.87




```
Test accuracy:  31.77
```

b) Shuffling input for each epoch

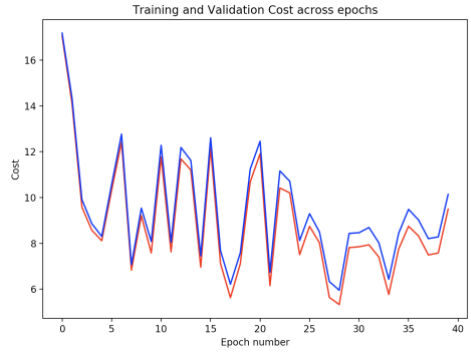
The following function was used to shuffle the input as well as corresponding labels whilst taking care of dimensions:

```
def shuffle(X , Y):
    transposeX = X.T
    transposeY = Y.T
    elementIndices = np.arange(X.shape[1])
    np.random.shuffle(elementIndices)
    shuffledX  =  transposeX[elementIndices]
    shuffledY  =  transposeY[elementIndices]
    X = shuffledX.T
    Y = shuffledY.T
    return X, Y
```

For each epoch, the training samples and their corresponding labels were rearranged using this function. I turned off all other optimization experiments for this test to get an overview of how much does only shuffling samples affects performance. The test cases are similar to above with fixed parameters:
Number of epochs: 40
Batch size: 100

Test 1: ETA= 0.1 Lambda= 0

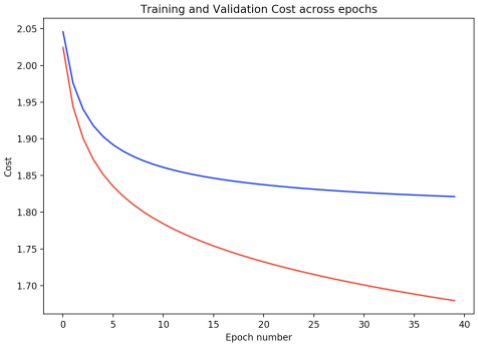
Test accuracy before optimization: 25.71

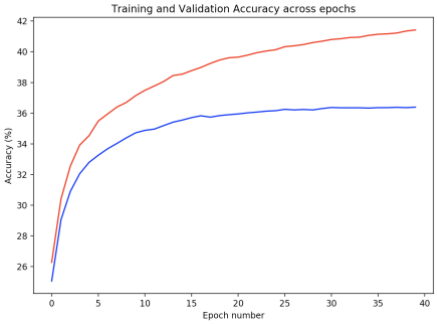



```
Test accuracy:  20.48
```

Test 2: ETA= 0.01 Lambda=0

Test accuracy before optimization: 36.87

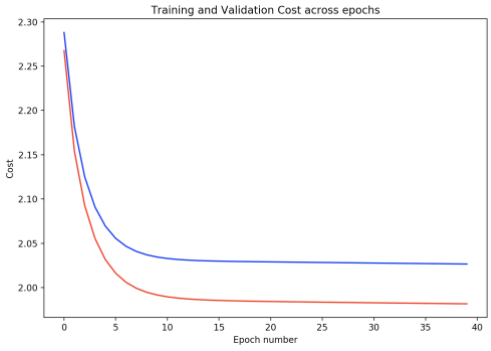


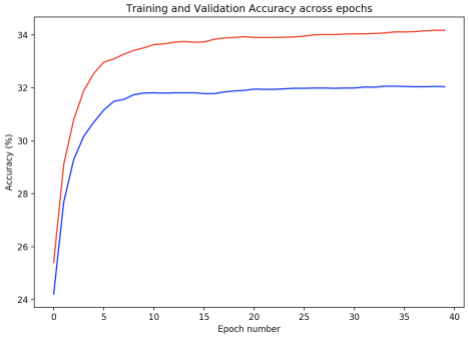



```
Test accuracy:  36.980000000000004
```

Test 3: ETA= 0.01 Lambda=0.1

Test accuracy before optimization: 33.39

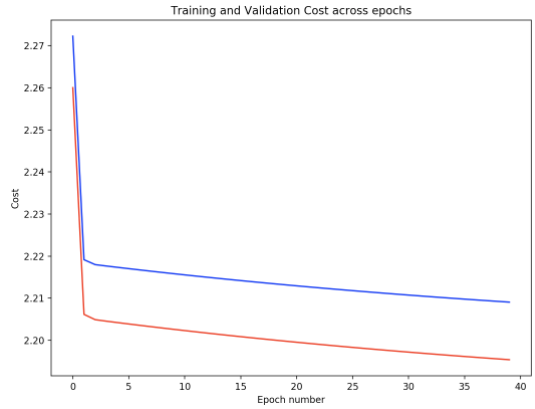


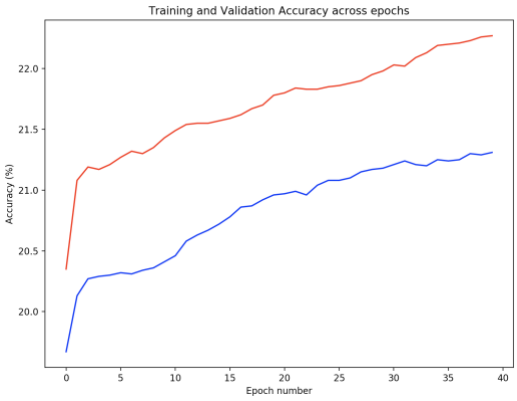



```
Test accuracy:  33.37
```

Test 4: ETA= 0.01 Lambda=1

Test accuracy before optimization: 21.87






```
Test accuracy:  21.92
```

c) Train for longer time until overfit

For this test, I just increased the number of epochs to 300 and then I implemented the following function that will return the highest accuracy achieved on the validation set and the corresponding epoch number.

```
def choosebestModel(ValidationAccuracy):
    maximumAccuracy = np.max(ValidationAccuracy)
    index = np.argmax(ValidationAccuracy)
    print("The best validation accuracy achieved is:    " , maximumAccuracy , "occured at epoch:    " , index , ".\n")
```

Because ETA = 0.1 was originally too large of a learning rate for the network, I just experimented with 3 test cases where I change lambda values. I then train the network with the returned perfect epoch number right before overfitting to see how it has affected test accuracy.

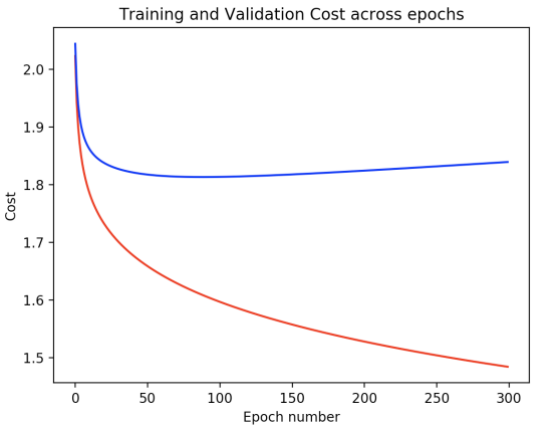
Number of epochs: 40

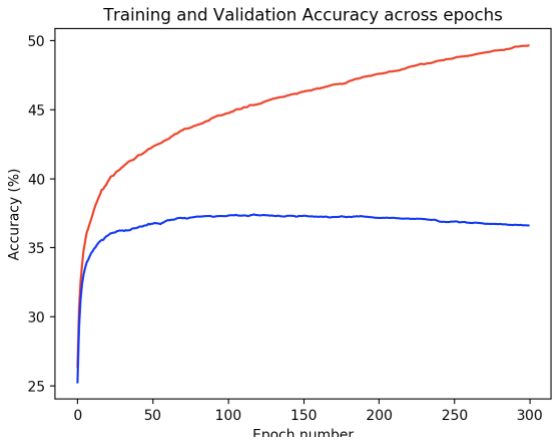
Batch size: 100

ETA = 0.01

Test 1: Lambda=0

Test accuracy before optimization: 36.87



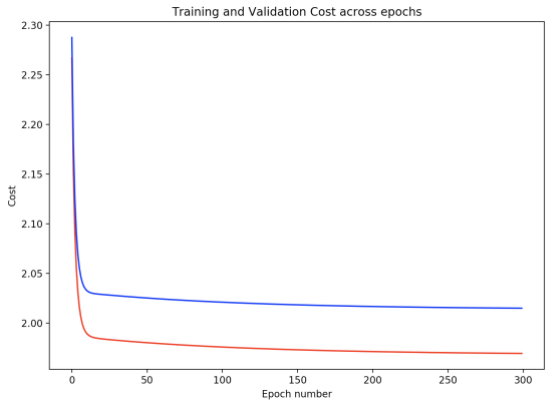


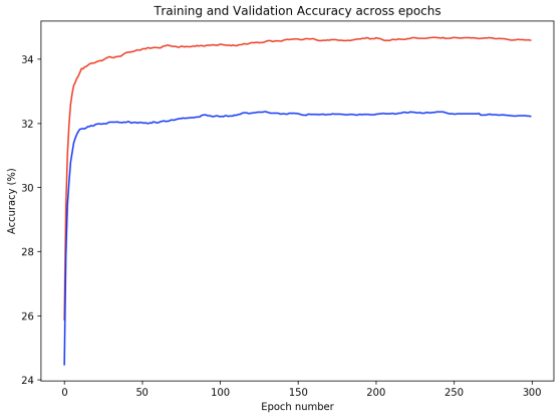
Highest validation accuracy achieved is 37.46 at epoch 106.

Test accuracy after 106 epochs: 37.18

Test 2: Lambda=0.1

Test accuracy before optimization: 33.39

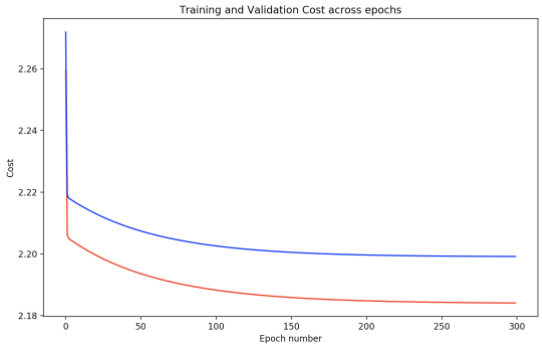


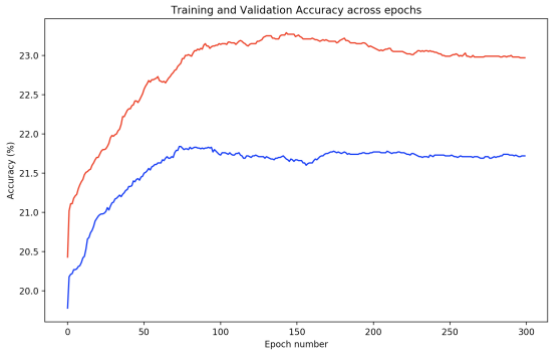


Highest validation accuracy achieved is 32.39 at epoch 219.
Test accuracy after 219 epochs: 33.63

Test 3: Lambda=1

Test accuracy before optimization: 21.87





Highest validation accuracy achieved is 21.84 at epoch 219.
Test accuracy after 88 epochs: 22.55

Generally, I realized the eta_decay had the best results in terms of accuracy improvement. Shuffling samples was not really effective and at some points it made the results worse. The highest accuracy the network achieved was by eta decay and parameters corresponding to that of test case 1 where eta is initialized at 0.1 and lambda is 0. The test accuracy achieved is 38.6. Using longer training, it turned out that the test accuracy could have been improved further by increasing epochs from 40 to 100 and above without risking overfitting.

Bonus Assignment 2:

For this assignment, I implemented SVM model trained with mini batch gradient descent. the following functions were the core of the implementation:

```
def classify(X, W):
    scores = np.dot(X, W)
    return scores

def svmLoss(X, W, y):
    # lecture 2
    gradW = np.zeros((W.shape[0] , W.shape[1]))
    loss = 0
    scores = classify(X, W)
    correctScores = scores[np.arange(X.shape[0]), y].reshape((X.shape[0] , 1))
    margins = np.maximum(0, scores - correctScores[y] + delta)
    margins[np.arange(X.shape[0]), y] = 0
    loss = np.sum(margins) /X.shape[0]
    loss += 0.5 * lamda * np.sum(np.square(W))
    #samples with margin greater than 0
    # https://math.stackexchange.com/questions/2572318/derivation-of-gradient-of-svm-loss
    temp = np.zeros((X.shape[0] , k))
    temp[margins > 0] = 1
    numTemp = np.sum(temp,axis=1)
    temp[np.arange(X.shape[0]),y] = - numTemp
    gradW = np.dot(X.T , temp )
    gradW /= X.shape[0]
    gradW += lamda*W
    return gradW
```

I called SVMLoss in the gradient descent function instead of previously implemented analytic gradients in order to calculate gradients used to update the weights. I initialized the bias vector as well as the weight matrix with random Gaussian numbers with mean 0 and standard deviation 0.01. I then concatenated the bias vector as an extra dimension (d+1) to both the training data matrix as well as the weight vector. The test cases used were similar to that of the original assignment. The accuracy comparison results are presented below. Generally, the comparison of results in this case is very parameter-dependent. But it seems like SVM contributed to the most improvement in the case that had too large learning rate for cross entropy loss training (0.1) and 0 lambda. Generally lower learning of 0.01 was not so effective in terms of test accuracy with SVM loss training.

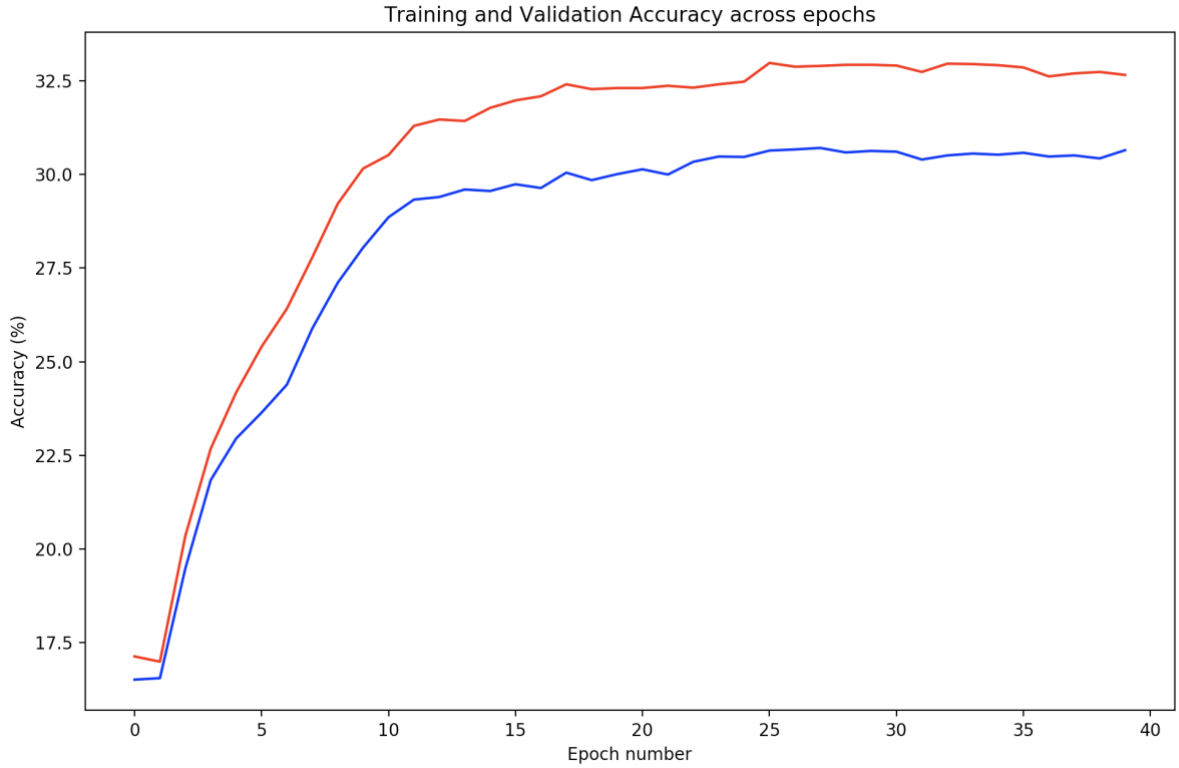
Number of epochs: 40

Batch size = 100

Test 1: ETA= 0.1 Lambda= 0

Test accuracy with cross entropy loss: 25.71

Test accuracy with SVM loss: 30.49



Test 2: ETA= 0.01 Lambda= 0

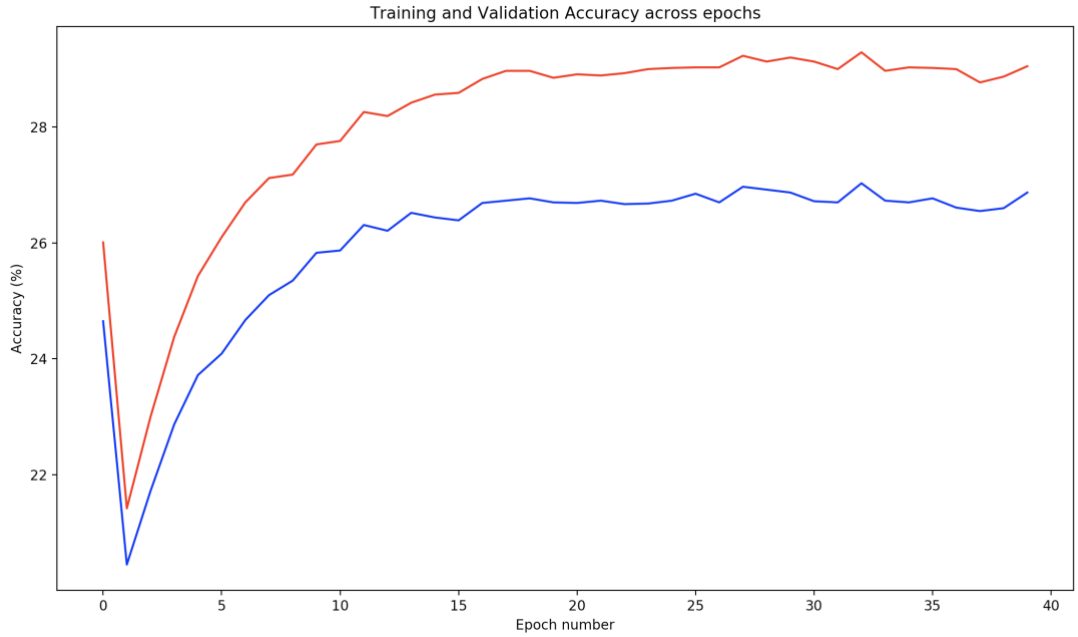
Test accuracy with cross entropy loss: 36.87

Test accuracy with SVM loss: 31.26

Test 3: ETA= 0.01 Lambda= 0.1

Test accuracy with cross entropy loss: 33.39

Test accuracy with SVM loss: 27.28



Test 4: ETA= 0.01 Lambda= 1

Test accuracy with cross entropy loss: 21.87

Test accuracy with SVM loss: 25.24