
# DD2424 Deep Learning

# Assignment 3

**Yunpeng Ma**

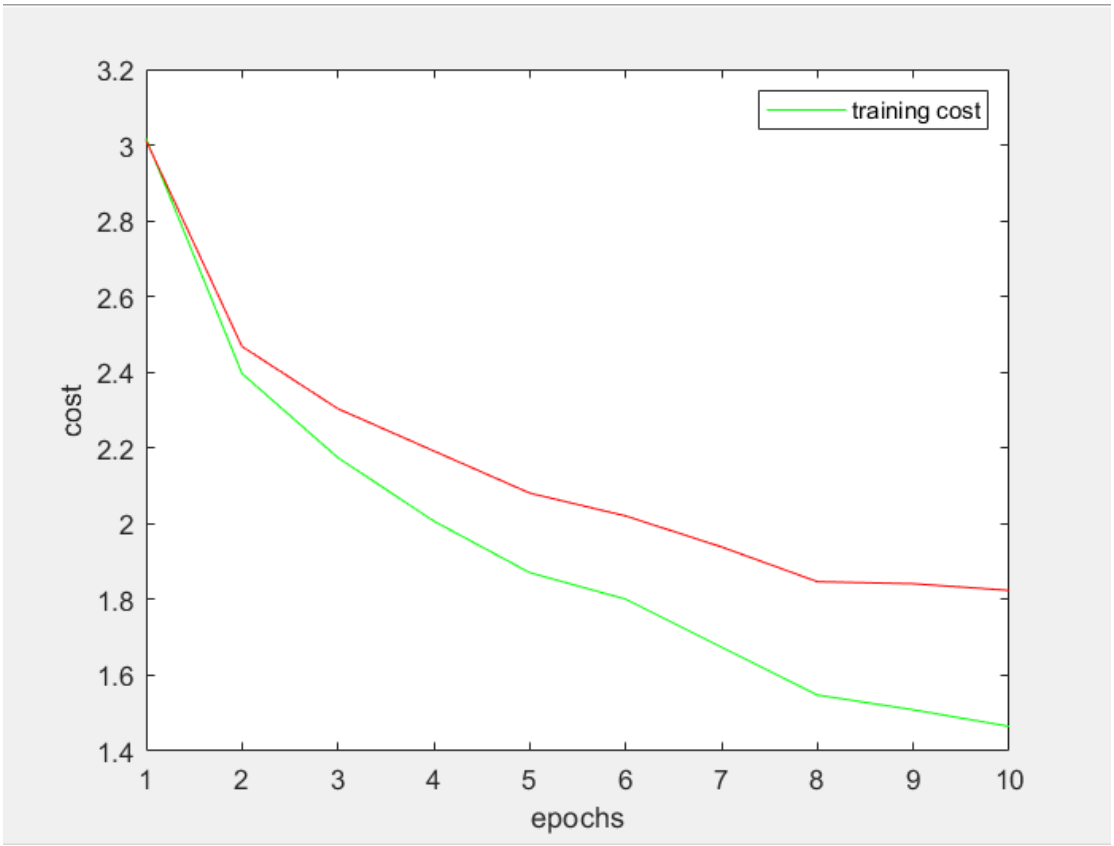
**Exercise 1: Upgrade Assignment 2 code to train & test k-layer networks**

In this part, I extended previous 2 layers' network code to k layers. After upgraded the code, I did gradient computations and check them numerically. The dimension of input data is reduced to 10 (d =10), lambda = 0. The maximum errors between analytical gradient and numerical gradient are as following:

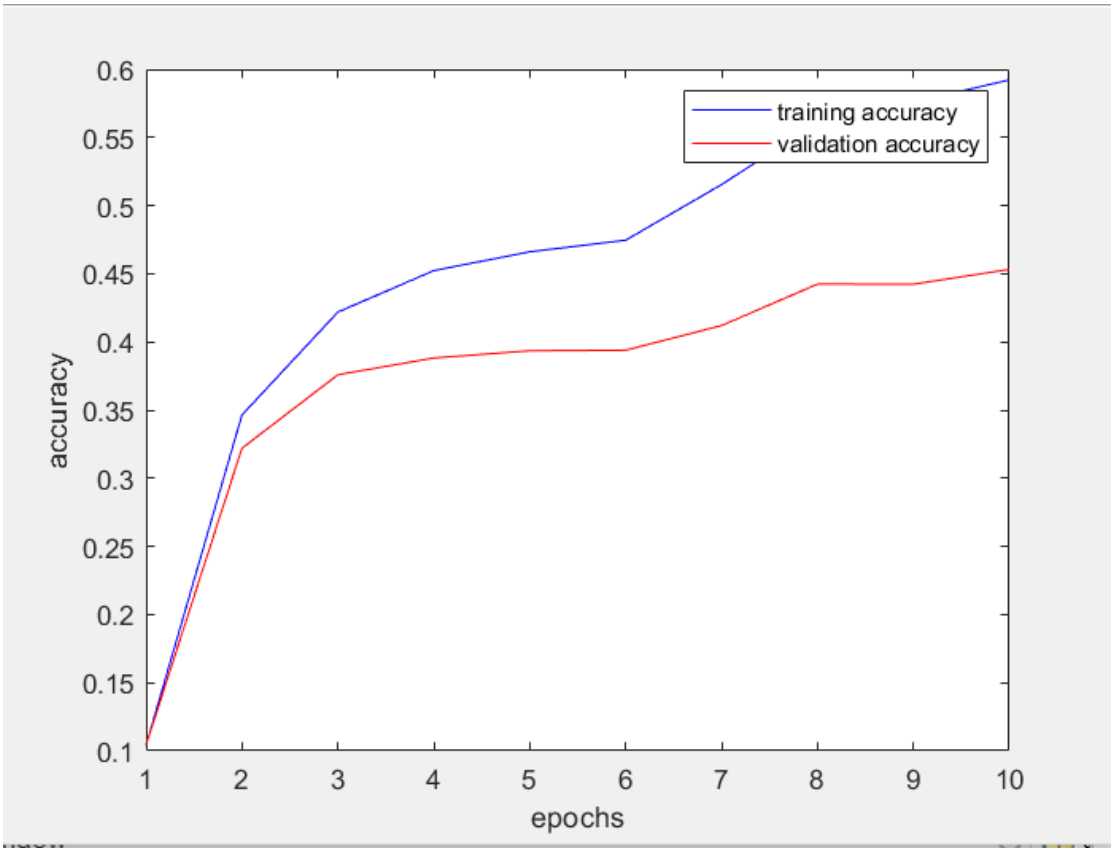
| Layers | Maximum error between analytical gradient and numerical gradient | | | |
|---|---|---|---|---|
| | W1 | W2 | W3 | W4 |
| 2-layers | 1.7719e-09 | 2.0195e-09 | - | - |
| 3-layers | 6.3024e-06 | 2.4288e-08 | 4.6167e-08 | - |
| 4-layers | 5.0664e-07 | 4.1845e-07 | 5.0973e-07 | 1.1359e-06 |

**Exercise 2: Can I train multi-layer networks?**

By using the same parameter with assignment 2, I trained 2-layer network with 50 nodes in the hidden layer using mini-batch gradient descent with a cyclic learning rate. By comparing the cost curve, training accuracy and test accuracy, we can see they have the same performance. The parameters are: n_batch=100; eta_min=1e-5, eta_max = 1e-1, n_s= 500, lambda=0.01, training for one cycle from t =1 until t = 2n_s, m =50. then I got training accuracy 60.52%, and test accuracy is 46.3%; while in assignment 2 the training accuracy is 59.52%, and test accuracy is 45.87%.

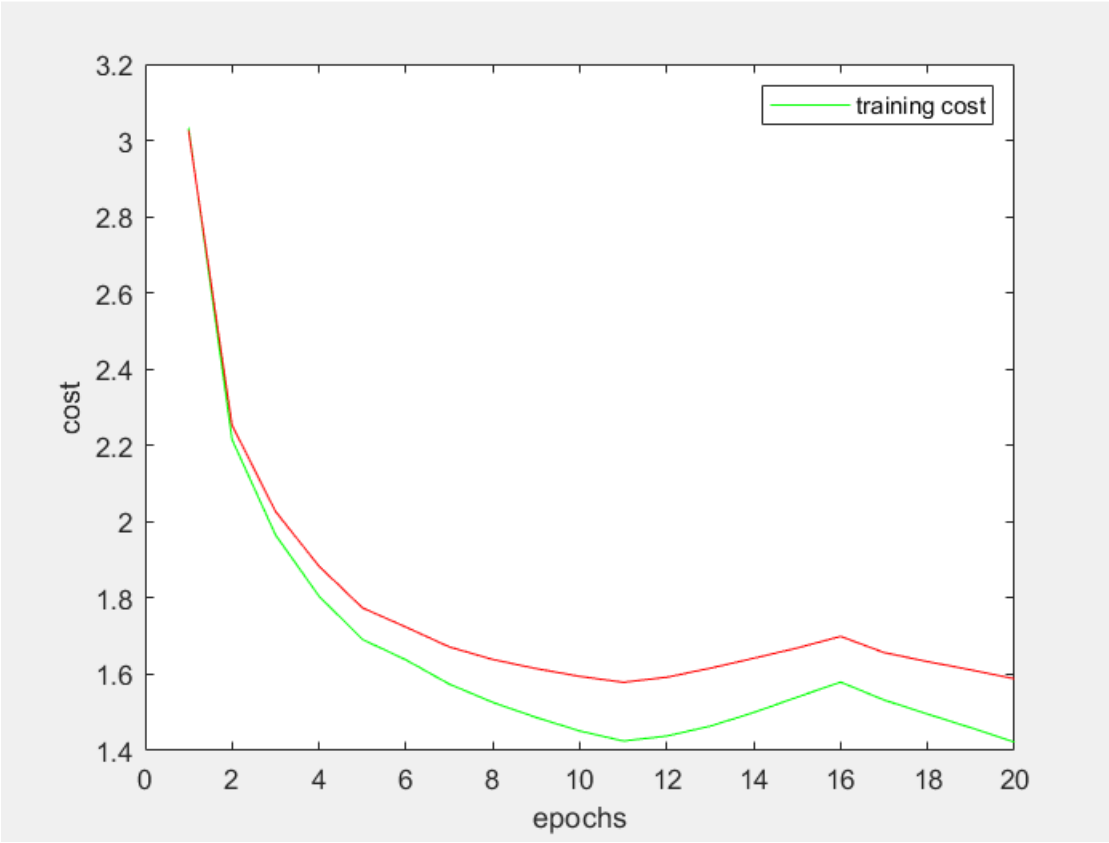



Cost Curve

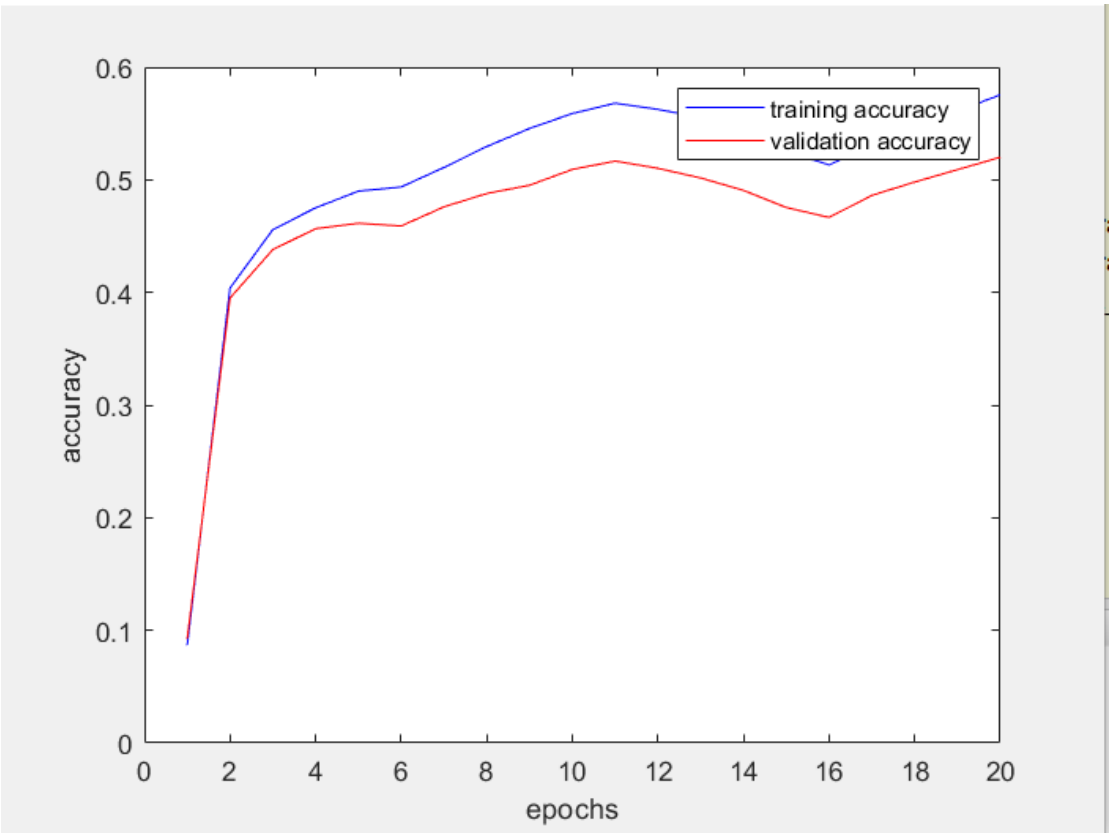



Accuracy Curve

Then I trained 3-layer network with 50 and 50 hidden layer nodes using the following parameters: n_batch = 100, eta_min=1e-5, eta_max =1e-1, lambda = 0.005, n_s = 5*45000/n_batch. I used Xavier initialization during this training process. I got training accuracy 58.6422%, and test accuracy 53.06%.

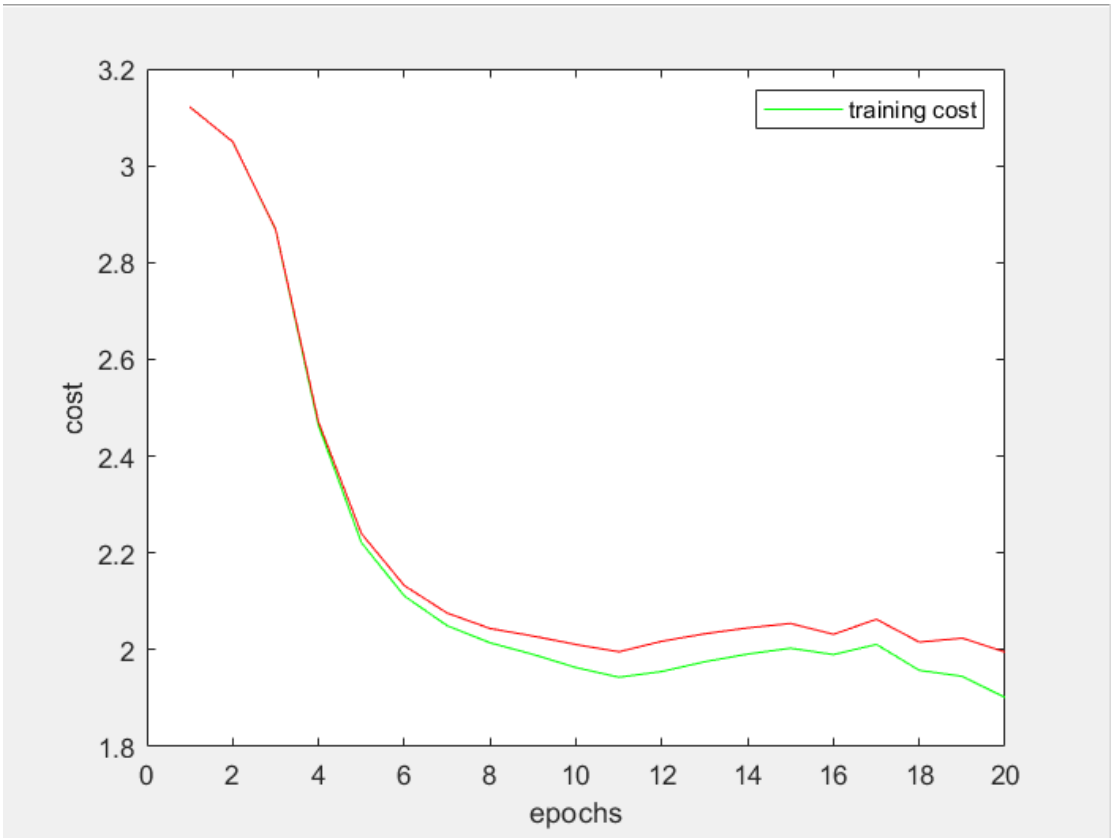



Cost Curve

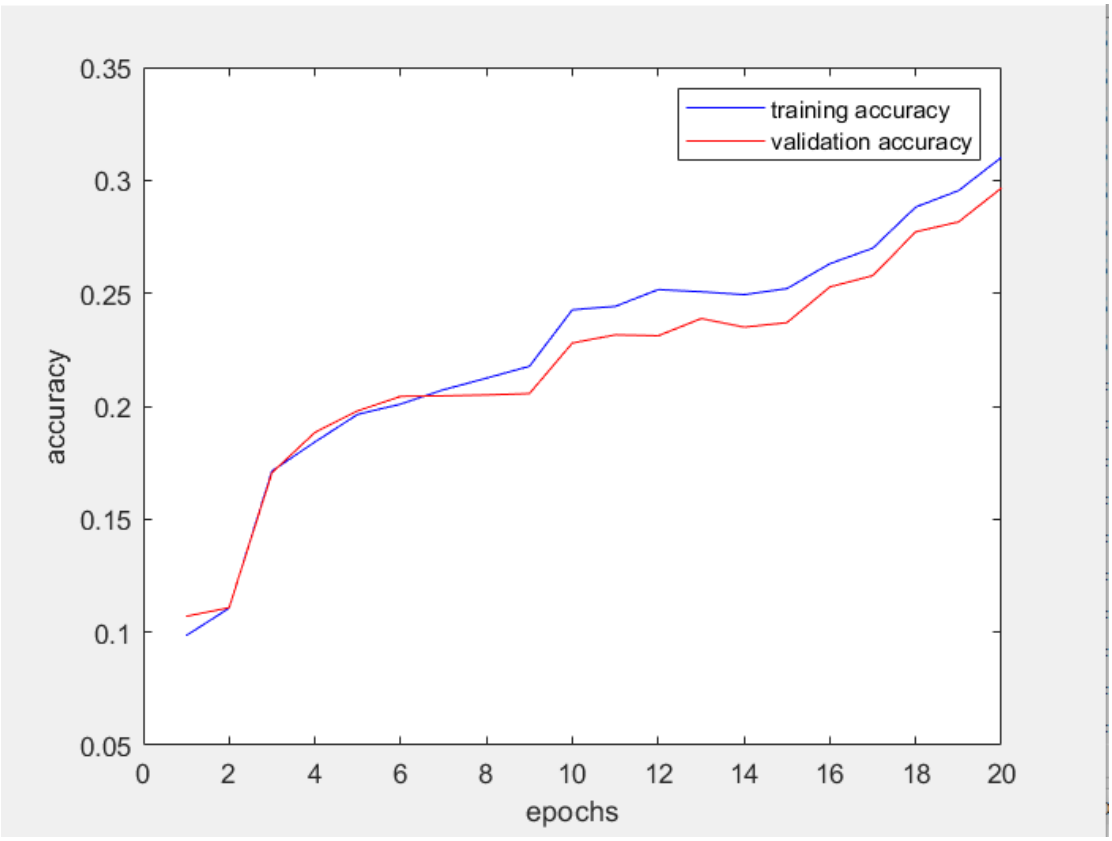



Accuracy Curve

Then I used the same parameters to train 9-layer network, the hidden layers are m = [50,30,20,20,10,10,10,10]. I got training accuracy 32.6556%, test accuracy 30.21%.




Cost Curve




Accuracy Curve

From the pictures above, we can see the performance dropped quite a bit. That is because for deeper network, it is difficult to train by using variants of mini-batch gradient descent and using a more standard decay of the learning rate. That is why we use batch normalization.

**Exercise 3: Implement batch normalization**

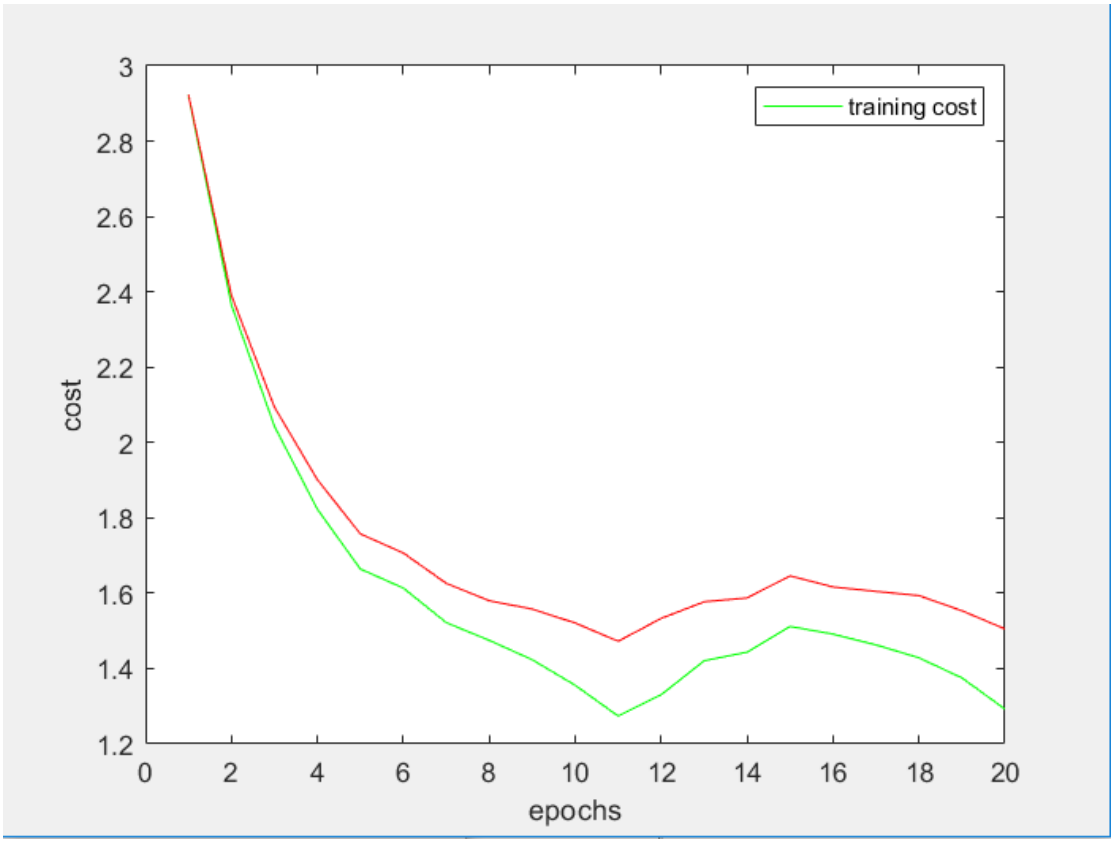
After applying batch normalization, I did gradient computations and check them numerically. The dimension of input data is reduced to 10 (d =10), lambda = 0. M =50 for two layers network, m = [50,50] for 3 layers network. The maximum errors between analytical gradient and numerical gradient are as following:

| Layers | Maximum error between analytical gradient and numerical gradient | | |
|---|---|---|---|
| | W1 | W2 | W3 |
| 2-layers | 5.6538e-08 | 7.6743e-08 | - |
| 3-layers | 1.5435e-07 | 2.6755e-08 | 1.6543e-08 |

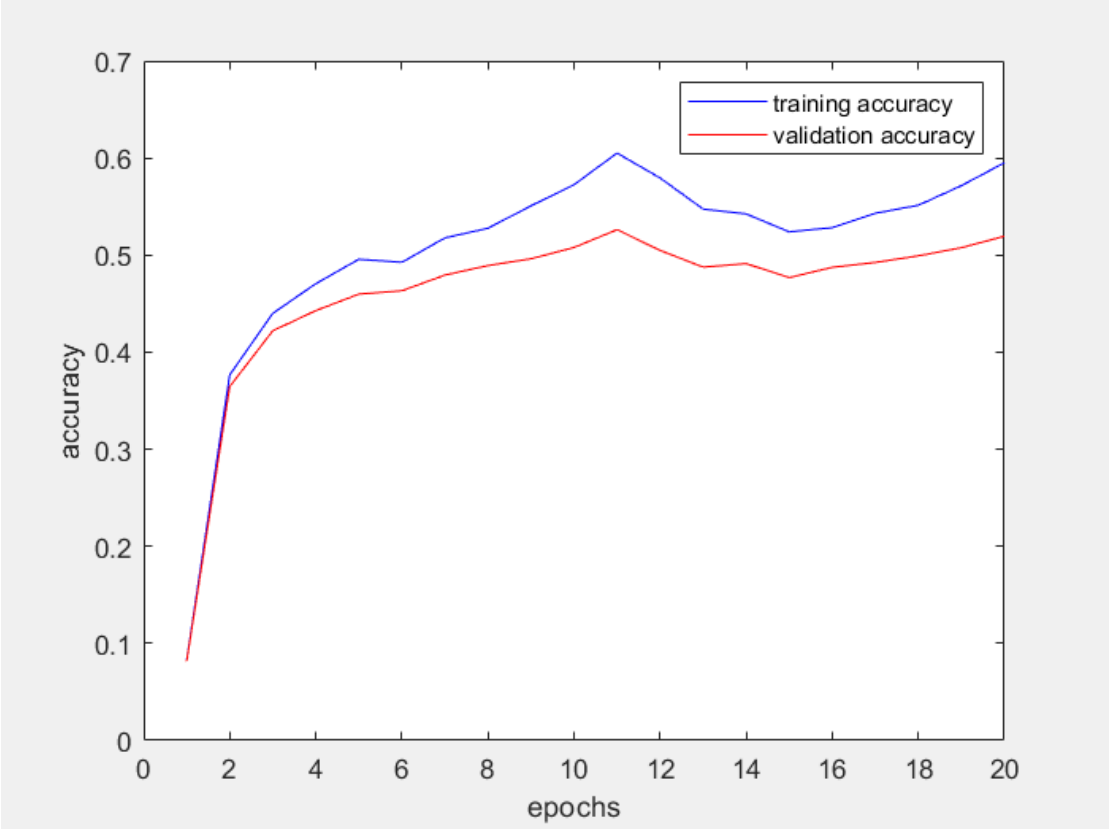
Then I keep an exponential moving average of the batch mean and variances for the un-normalized scores for each layer of network after each epoch.

Firstly, I trained a 3-layer network with 50 and 50 nodes in the hidden layers. I used Xavier initialization and using 5 batches of data, which 45000 sets of data for training,

5000 datasets for validation. The parameters are: n_batch = 100, eta_min=1e-5, eta_max =1e-1, lambda = 0.005, n_s = 5*45000/n_batch, Then I got training accuracy 62.9133%, test accuracy 53.37%.




Cost curve




Accuracy Curve

To find a proper lambda, I randomly get different performance with respect different lambda.

Coarse search (m = [50,50])

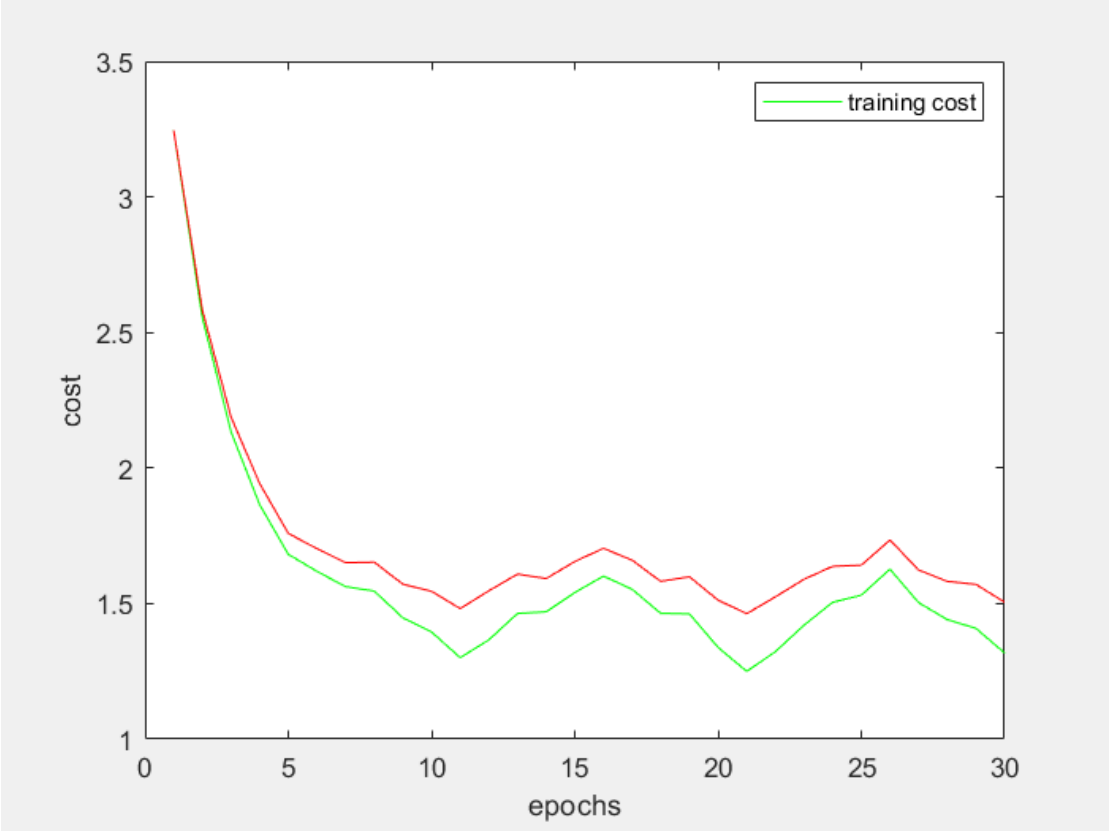
| No. | lambda | Training Accuracy | Test Accuracy |
|---|---|---|---|
| 1 | 0.00041622 | 65.82% | 51.53% |
| 2 | 0.078946 | 52.3444% | 49.81% |
| 3 | 0.032871 | 55.7533% | 52.3% |
| 4 | 0.014518 | 58.8822% | 53.28% |
| 5 | 7.9555e-05 | 65.9422% | 51.86% |
| 6 | 3.6394e-05 | 66.0556% | 51.66% |
| 7 | 6.9588e-05 | 66.2356% | 52.12% |
| 8 | 0.001741 | 65.2289% | 52.83% |
| 9 | 0.026629 | 56.5778% | 51.8% |
| 10 | 0.00022944 | 66.1111% | 51.4% |
| 11 | 0.00011881 | 66.1533% | 52.14% |
| 12 | 0.00095446 | 65.6089% | 52.63% |
| 13 | 0.032876 | 55.6178% | 51.34% |
| 14 | 0.0078223 | 61.1889% | 53.6% |
| 15 | 0.00088015 | 65.9244% | 52.23% |

From the data above, we can get lambda changes from 1e-3 to 4e-2, the test accuracy is the highest performance. Then we do the fine search.

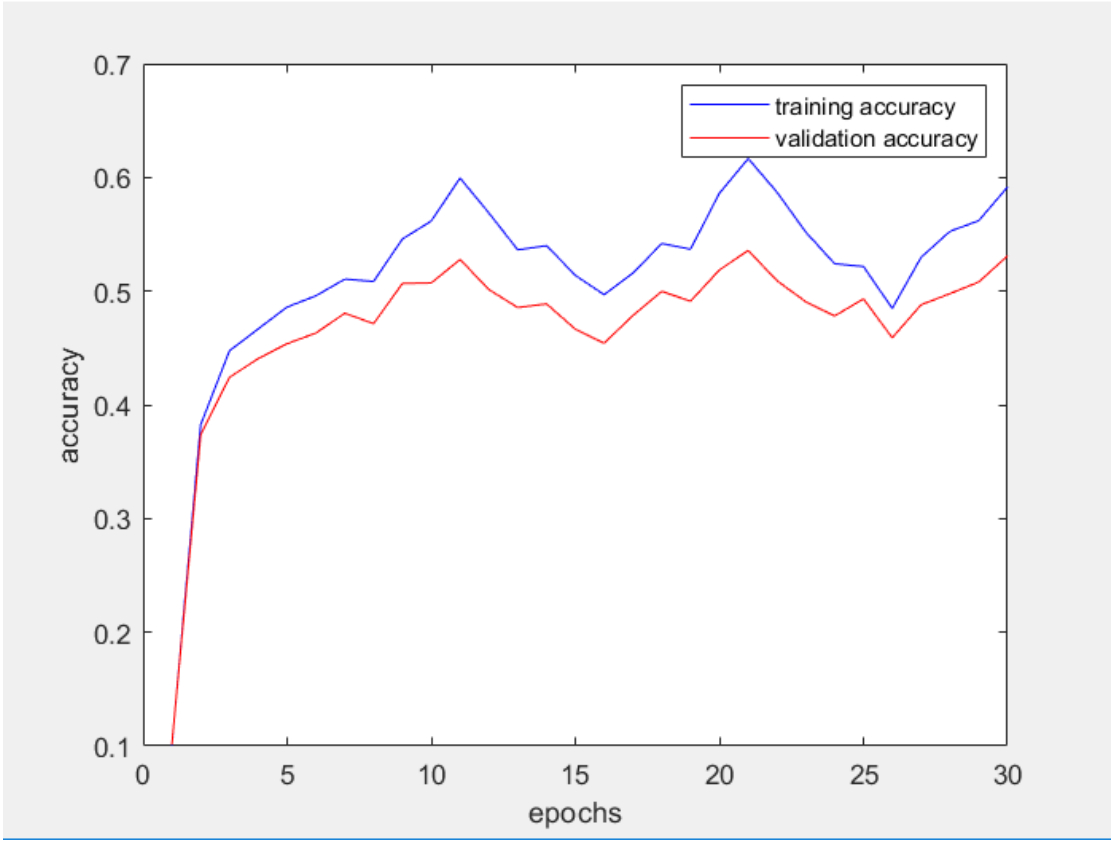
Fine search (m = [50,50])

| No. | lambda | Training Accuracy | Test Accuracy |
|---|---|---|---|
| 1 | 0.0019137 | 64.9689% | 53.3% |
| 2 | 0.0010514 | 65.64% | 52.69% |
| 3 | 0.0053956 | 62.5178% | 53.28% |
| 4 | 0.007253 | 61.5644% | 54.04% |
| 5 | 0.010821 | 60.2356% | 53.55% |
| 6 | 0.0028403 | 63.8511% | 53.41% |
| 7 | 0.017664 | 58.1178% | 52.79% |
| 8 | 0.0098151 | 60.0133% | 53.55% |
| 9 | 0.0039813 | 63.3644% | 52.94% |
| 10 | 0.0013086 | 65.4444% | 52.94% |

So I get the good value for lambda = 0.007253, then we use this parameter to train the network for 3 cycles, we get a test accuracy of 54.18%.

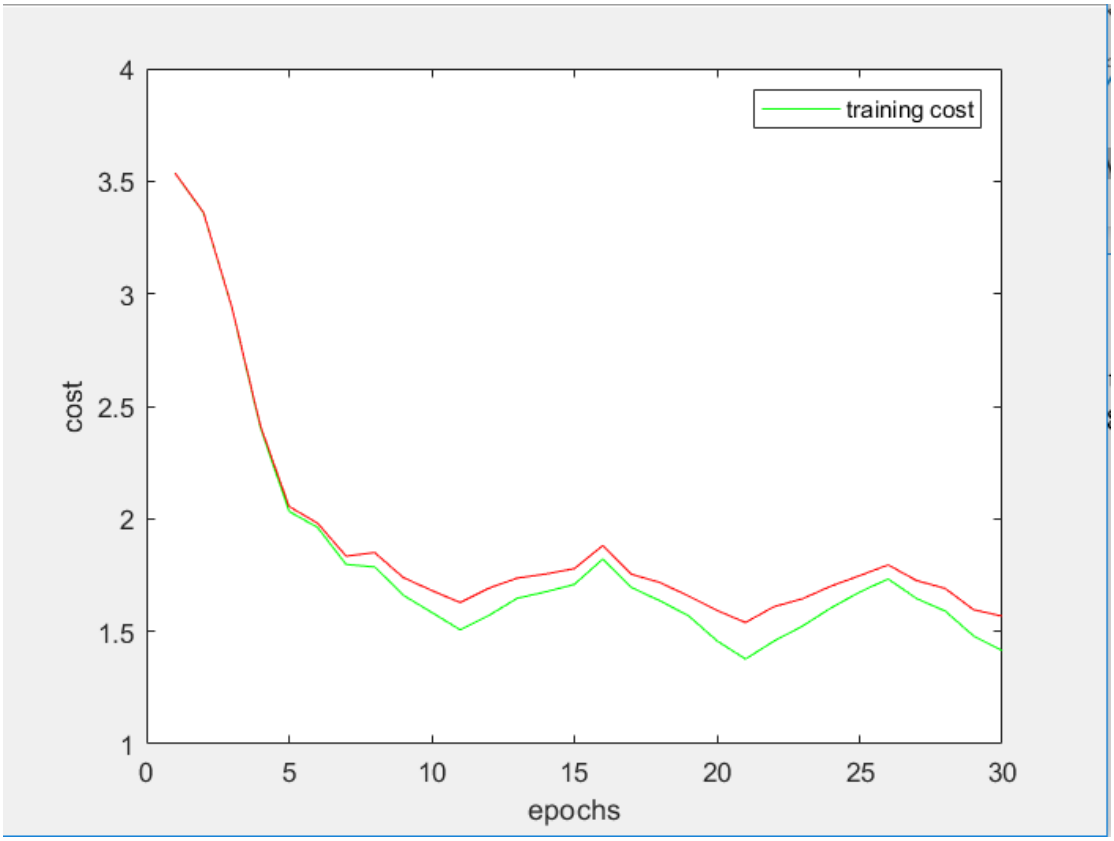



Cost Curve

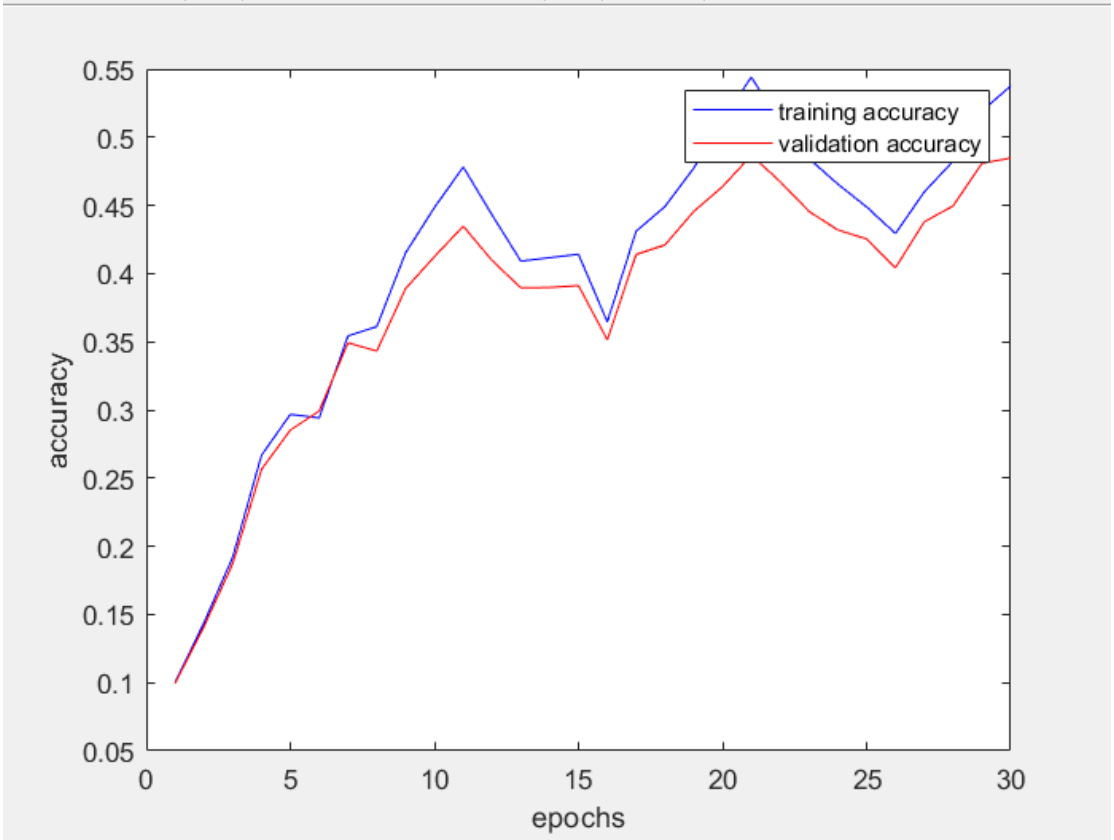



Accuracy Curve

Then I apply the parameters to a 9-layer network, m = [50,30,20,20,10,10,10,10]. I got a training accuracy57.5689% and test accuracy 51.07%.




Cost Curve




Accuracy Curve

**Sensitivity to initialization**

Now I use 3-layers network with 50 nodes at each hidden layer, and choose parameters eta_min = 1e-5, eta_max = 1e-1, n_s = 5 * 45000/ n_batch, ran two cycles from t =1 to t = 4n_s, lambda=0.005.

| sigmas | Test accuracy | |
|---|---|---|
| | Without BN | BN |
| 1e-1 | 53.03% | 54.17% |
| 1e-3 | 45.73% | 46.65% |
| 1e-4 | 43.63% | 44.86% |

From the data above, we can see that if batch normalization is applied, the network will be more robust than that without normalization.