Exercise 13.1 (June 12, 2020)

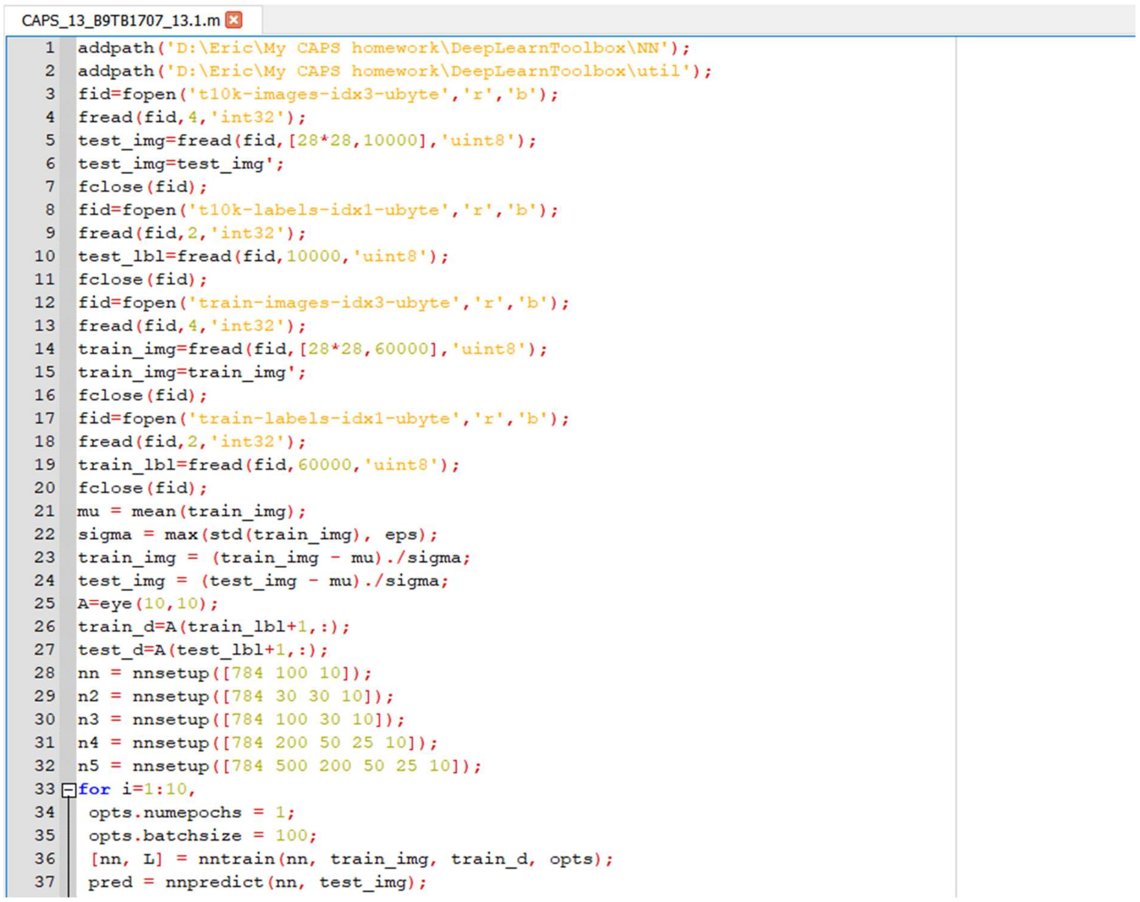
-B9TB1707

Question:

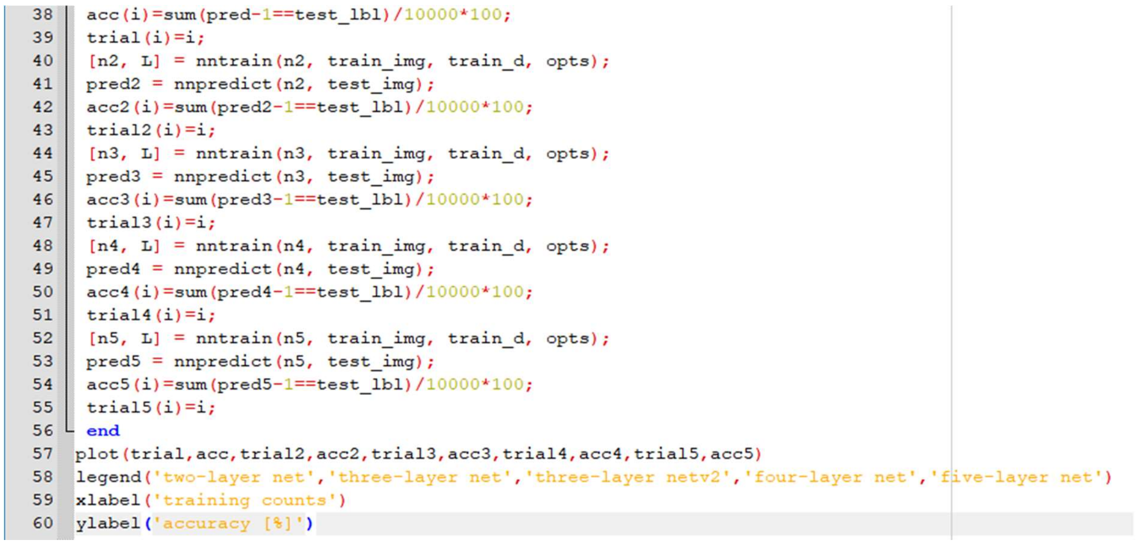
1. Repeat training for, say, 10 steps, from initialization and evaluate performance of the net at each step; plot 'training counts'-vs-'accuracy'
2. Design a three-layer NN, *for instance*, having two intermediate layers with 30 units each, and train it; and evaluate the difference in performance from the earlier two-layer net
3. Try increasing the number of layers in NN, and varying the number of units in the intermediate layers a few more times. How do their properties change?

Solution:

My code for the solution is as follows:

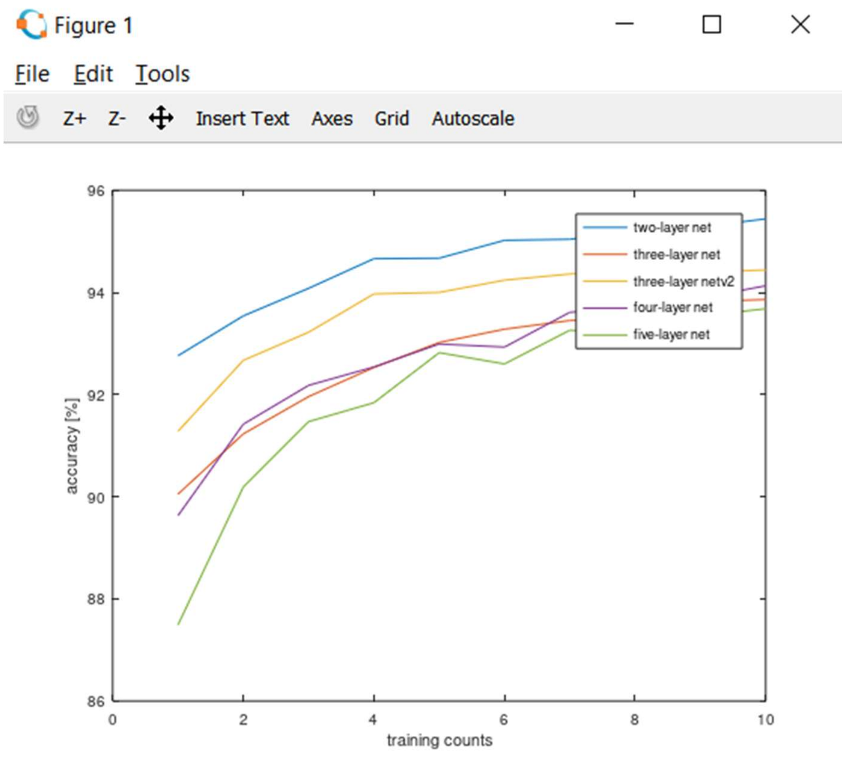


```
addpath('D:\Eric\My CAPS homework\DeepLearnToolbox\NN');
addpath('D:\Eric\My CAPS homework\DeepLearnToolbox\util');
fid=fopen('t10k-images-idx3-ubyte','r','b');
fread(fid,4,'int32');
test_img=fread(fid,[28*28,10000],'uint8');
test_img=test_img';
fclose(fid);
fid=fopen('t10k-labels-idx1-ubyte','r','b');
fread(fid,2,'int32');
test_lbl=fread(fid,10000,'uint8');
fclose(fid);
fid=fopen('train-images-idx3-ubyte','r','b');
fread(fid,4,'int32');
train_img=fread(fid,[28*28,60000],'uint8');
train_img=train_img';
fclose(fid);
fid=fopen('train-labels-idx1-ubyte','r','b');
fread(fid,2,'int32');
train_lbl=fread(fid,60000,'uint8');
fclose(fid);
mu = mean(train_img);
sigma = max(std(train_img), eps);
train_img = (train_img - mu)./sigma;
test_img = (test_img - mu)./sigma;
A=eye(10,10);
train_d=A(train_lbl+1,:);
test_d=A(test_lbl+1,:);
nn = nnsetup([784 100 10]);
n2 = nnsetup([784 30 30 10]);
n3 = nnsetup([784 100 30 10]);
n4 = nnsetup([784 200 50 25 10]);
n5 = nnsetup([784 500 200 50 25 10]);
for i=1:10,
 opts.numepochs = 1;
 opts.batchsize = 100;
 [nn, L] = nntrain(nn, train_img, train_d, opts);
 pred = nnpredict(nn, test_img);
```



```
38      acc(i)=sum(pred-1==test_lbl)/10000*100;
39      trial(i)=i;
40      [n2, L] = nntrain(n2, train_img, train_d, opts);
41      pred2 = nnpredict(n2, test_img);
42      acc2(i)=sum(pred2-1==test_lbl)/10000*100;
43      trial2(i)=i;
44      [n3, L] = nntrain(n3, train_img, train_d, opts);
45      pred3 = nnpredict(n3, test_img);
46      acc3(i)=sum(pred3-1==test_lbl)/10000*100;
47      trial3(i)=i;
48      [n4, L] = nntrain(n4, train_img, train_d, opts);
49      pred4 = nnpredict(n4, test_img);
50      acc4(i)=sum(pred4-1==test_lbl)/10000*100;
51      trial4(i)=i;
52      [n5, L] = nntrain(n5, train_img, train_d, opts);
53      pred5 = nnpredict(n5, test_img);
54      acc5(i)=sum(pred5-1==test_lbl)/10000*100;
55      trial5(i)=i;
56    end
57    plot(trial,acc,trial2,acc2,trial3,acc3,trial4,acc4,trial5,acc5)
58    legend('two-layer net','three-layer net','three-layer netv2','four-layer net','five-layer net')
59    xlabel('training counts')
60    ylabel('accuracy [%]')
```

And the output is as follows:



How it works:

1. Line 1 and 2 adds the deep learn toolbox to the path.
2. Lines 3 through 20 imports and initializes all the training and testing images all with their labels into arrays.
3. Lines 21 through 24 attempts to normalize the data set. We first subtract the respective means and divide by the respective standard deviation. Line 21 computes the mean and Line 22 computes the standard deviation. It also replaces all standard deviation value of zero with a really small number using the eps. Lines 23 and 24 applies the transformation onto to the data set. We use normalization so that any no attribute has any unfair advantage over

the other due to its scale. We use the mean and standard deviation of our training data, because that is the data set we want to normalize. It does not make sense to apply any other transform than that which is required. Using the mean of the testing set will normalize our testing set, but not our training set. This is not what we want. We use the mean and standard deviation of the training set on the testing set because we need to apply the same transform on both our training and testing set.

4. Line 25 through 27 creates targets vectors to train the network and test the output. The array in Line 25 is an identity matrix of size ten because there are 10 digits. By multiplying with testing or training data set we get the results.
5. Line 28 establishes a two layer neural network.
6. Line 29 establishes a three layer neural network.
7. Line 30 establishes a three layer neural network but with layers of different sizes from the previous one.
8. Line 31 establishes a four layer neural network.
9. And Line 32 establishes a five layer neural network
10. Line 33 establishes the for loop to train the network 10 times.
11. Line 34 dictates the number of times the net sees each sample during training.
12. Line 35 dictates how many times the weights are updated once per number of samples (in this case hundred).
13. Line 36 trains the neural net using the function nntrain().
14. Line 37 predicts the each of the test cases using the neural net and the function nnpredict()
15. Line 38 stores the accuracy of the prediction for the given number of trainings in an array.
16. And Line 39 stores the training number.
17. Line 40 to 55 repeats the process for other neural nets.
18. Line 56 ends the for block.
19. Line 57 plots all the data.
20. Line 58 marks the legend.
21. And Line 59 through 60 labels the axes.

Conclusion:

Compared to the method with SVM's used last class, this method is far superior in accuracy. Even a five layer net is almost as accurate as the previous method on its first training and more than 93% accurate on its tenth training. Though one drawback to this method is that, it is very computational intensive. The growth of accuracy to number of trainings seems to be logarithmic. Also the accuracy of the predictions goes up when the number of layers are reduced. Within the same number of layers it seems that evenly decreasing the size from 784 to 10 produces better accuracy. I have always wanted to experiment with neural networks and I have finally been able to do so. This was a fun and interesting exercise to end a fun and interesting course.

Thank you.