Joanna Masikowska

B9TB1710

CAPS13_B9TB1710.m CAPS13_B9TB1710(2).m

```
#addpath('DeepLEarnToolbox/NN')
#addpath('DeepLEarnToolbox/util')

#loading data
fid=fopen('train-images-idx3-ubyte','r','b');
fread(fid,4,'int32')
train_img=fread(fid,[28*28,60000],'uint8');
train_img=train_img'; #60000x784
fclose(fid);

fid=fopen('train-labels-idx1-ubyte','r','b');
fread(fid,2,'int32')
train_lbl=fread(fid,60000,'uint8'); #60000x1
fclose(fid);

fid=fopen('t10k-images-idx3-ubyte','r','b');
fread(fid,4,'int32')
test_img=fread(fid,[28*28,10000],'uint8');
test_img=test_img'; #10000x784
fclose(fid);

fid=fopen('t10k-labels-idx1-ubyte','r','b');
fread(fid,2,'int32')
test_lbl=fread(fid,10000,'uint8');#10000x1
fclose(fid);

#standardization of data
mu=mean(train_img); #1x784
sigma = max(std(train_img),eps); #1x784
train_img_st = (train_img-mu)./sigma;#60000x784
test_img_st = (test_img-mu)./sigma; #10000x784
```

CAPS13_B9TB1710.m CAPS13_B9TB1710(2).m


```
test_img_st = (test_img-mu)./sigma; #10000x784
```



```

A=eye(10,10);
train_d=A(train_lbl+1,:); #target vector 60000x10
test_d=A(test_lbl+1,:); #target vector   10000x10

#nn with 784 input, 100 units intermediate layer, 10 units output
nn = nnsetup([784 100 10]);

opts.numepochs = 1; #number of full sweeps through data
opts.batchsize=100; #takes a mean gradient step over this many samples

pred=zeros(10000,10);
sums=zeros(1,10);

#training nn 10 times
for i=1:10
    [nn,L] =nntrain(nn,train_img_st,train_d,opts);
    pred(:,i)=nnpredict(nn,test_img_st);
    sums(1,i)=sum(pred(:,i).-1==test_lbl)/10000*100;
endfor

#plotting the accuracy of nn
xx=1:1:10;
plot(xx,sums(1,xx),"o")
set(gca,"fontsize",14);
xlabel("training counts");
ylabel("accuracy");

nn2 = nnsetup([784 30 30 10]);

```

CAPS13_B9TB1710.m CAPS13_B9TB1710(2).m


```

#plotting the accuracy of nn
xx=1:1:10;
plot(xx,sums(1,xx),"o")
set(gca,"fontsize",14);
xlabel("training counts");
ylabel("accuracy");

nn2 = nnsetup([784 30 30 10]);

```



```
pred2=zeros(10000,10); #predicted numbers
sums2=zeros(1,10); #accuracy

for i=1:10
    [nn2,L] =nntrain(nn2,train_img_st,train_d,opts);
    pred2(:,i)=nnpredict(nn2,test_img_st);
    sums2(1,i)=sum(pred2(:,i).-1==test_lbl)/10000*100;
endfor

#plotting accuracy of nn2
hold on
plot(xx,sums2(1,xx),"o")

#evaluating difference between nn and nn2
diff=sums-sums2;
figure
plot(xx,diff(1,xx),"o");
set(gca,"fontsize",14);
xlabel("training counts");
title("difference in accuracies between NNs");

```

I load data containing pictures of numbers and labels of the numbers. I use functions **fopen** and **fread** to open and read the files with data that I will be using. I explained them in details in my previous report.

I standardize the data to make its distribution uniform. This will help in training my Neural Network. I will make the mean of the data to be 0, and the variance to be 1 by linear transformation. In order to do that, I perform steps as below.

I calculate the mean *mu* of the *train_img* data (size 60000×784), which gives me a row vector (size 1×784) consisting of means of each column of the data. Then, I assign the standard deviations of the data to a matrix *sigma*, with the condition that any standard deviation equal to 0 will be converted to number **eps,** which is a very small number close to 0. Thanks to that, I can divide by sigma. *Sigma* has the same size as *mu*.

I subtract *mu* from each data sample, where a sample is one of the rows of the data. For instance, there are 60000 samples in *train_img*. Then, I divide each data point by the standard deviation of its column.

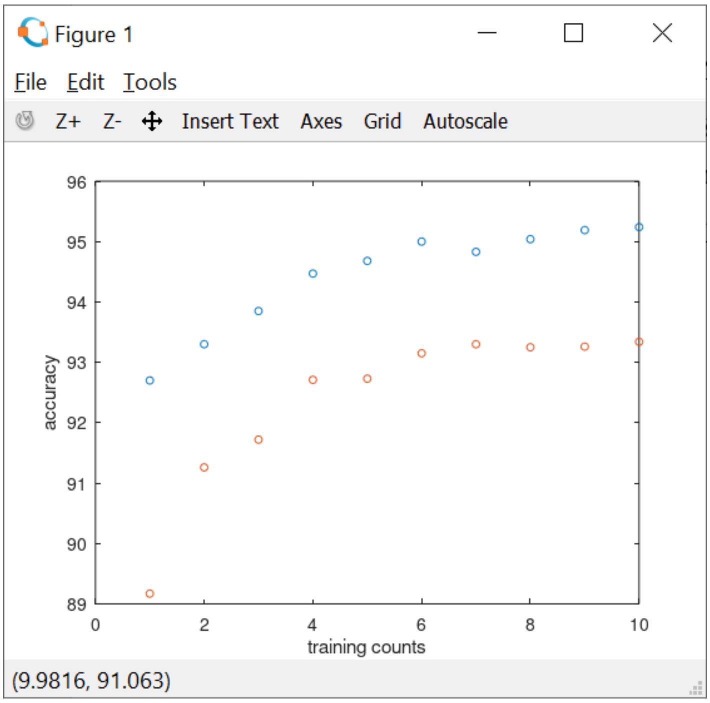
I use the mean and standard deviation of the training data to standardize the testing data. This is because I need to use the same parameters with which I trained the algorithm, in order to get unbiased validation of it. If I used mean and std of the testing data, the results would not represent the capability of the algorithm created on training data.

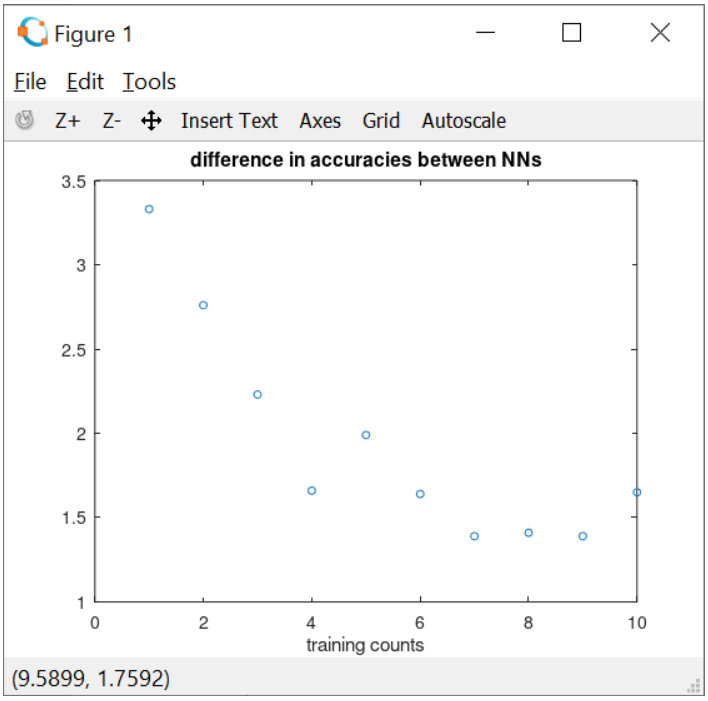
Next, I create target vectors for training data and testing data (with sizes 60000×10 and 10000×10). The number of rows of these correspond to the number of samples and the number of columns represent classes. Each class represents a digit between 0 and 9. In this case, the number of class j, $(1 \leq j \leq 10)$, equals to the digit in the image + 1. If the image i ($1 \leq i \leq 60000$ for training data, $1 \leq i \leq 10000$ for testing data) represents a digit k, then $target\ vector(i, k+1) = 1$ and for $j \neq k+1$, $target\ vector(i, j) = 0$

I add the path to ‘DeepLEarnToolbox/NN’ and DeepLEarnToolbox/util’.

```
>> addpath('DeepLEarnToolbox/NN')
>> addpath('DeepLEarnToolbox/util')
warning: function DeepLEarnToolbox/util\randp.m shadows a built-in function
warning: function DeepLEarnToolbox/util\zscore.m shadows a core library funct
```

Now, I am ready to train my Neural Network. I set up NN with 784 inputs, one intermediate layer with 100 units, and an output layer with 10 units (classes). I train the NN 10 times using the same training data. Each time, I make the NN recognize the numbers from test data, and I calculate its accuracy in percent. I plot it and the result is in the graph below, on the left (blue dots represent this NN). I can see that the more training was performed, the better the accuracy of the algorithm.





The red dots represent the accuracy of another NN which was trained on the same data but has two intermediate layer, each with 30 units. The plot on the right represents the differences between these two NNs. As can be seen, the accuracy of the more-layered NN is worse. However, the more training sessions are performed, the smaller the difference becomes.

Next, I will create 4 differently layered NNs.

```
##comparing differently layered NNs

acc=zeros(1,4);

nn1 = nnsetup([784 100 50 10]);
[nn1,L] =nntrain(nn1,train_img_st,train_d,opts);
pred1=nnpredict(nn1,test_img_st);
acc(1,1)=sum(pred1-1==test_lbl)/10000*100;

nn2 = nnsetup([784 50 50 50 50 50 50 10]);
[nn2,L] =nntrain(nn2,train_img_st,train_d,opts);
pred2=nnpredict(nn2,test_img_st);
acc(1,2)=sum(pred2-1==test_lbl)/10000*100;

nn3 = nnsetup([784 300 100 10]);
[nn3,L] =nntrain(nn3,train_img_st,train_d,opts);
pred3=nnpredict(nn3,test_img_st);
acc(1,3)=sum(pred3-1==test_lbl)/10000*100;

nn4 = nnsetup([784 400 10]);
[nn4,L] =nntrain(nn4,train_img_st,train_d,opts);
pred4=nnpredict(nn4,test_img_st);
acc(1,4)=sum(pred4-1==test_lbl)/10000*100;


#the accuracies of nn1,nn2,nn3,nn4
acc
```

```
epoch 1/1. Took 5.4675 seconds. Mini-batch mean squared error on training set
tch train err = 0.072134
epoch 1/1. Took 5.3503 seconds. Mini-batch mean squared error on training set
tch train err = 0.273458
epoch 1/1. Took 12.4891 seconds. Mini-batch mean squared error on training se
atch train err = 0.078220
epoch 1/1. Took 13.9155 seconds. Mini-batch mean squared error on training se
atch train err = 0.079298
acc =

   91.520   65.050   91.630   92.470
```

As can be seen from the data, the NNs (*nn2*) with the most layers (6 intermediate layers) but the smallest number of units had the worst performance (65.050%). It implies that many layers themselves do not guarantee good accuracy. *nn1*, one the other hand, has only 2 intermediate layers, but has two times more units in one of layers than *nn2*. Nevertheless, its accuracy is significantly better than *nn2* (91.520%) while it takes the about the same time (about 5.4s) to compute as *nn2*.

NNs with small number of intermediate layers but much bigger number of units (*nn3* and *nn4*) take noticeably more time to compute (12~14s) than the previous ones. Regardless of that, their accuracies do not differ significantly.

My conclusion is that for good accuracy, number of units cannot be to small. However, at the same time increasing number of units affects computation time. Thus, it is important to find the balanced number of units so that the computations does not take too much time. Also, increasing the number of layers is not necessarily efficient.