# CSE 190 - Neural Network - Homework 2

Justin Liu

October 30, 2015

## 1 Multi-layer neural network

### 1.1 Problem 1

For one example, $argmin(\theta)(f(x,\theta)-t)^2$
For N examples, $\sum_{i=1}^{N} argmin(\theta)(f(x^i,\theta)-t^i)^2$

### 1.2 Problem 2a

$\delta_k = -\frac{\partial E}{\partial a_k}$
$\delta_k = -\frac{\partial E}{\partial y_k}\frac{\partial y_K}{\partial a_k}$
$\delta_k = -\frac{\partial E}{\partial y_k}y'_k$
$\delta_k = t_k - y_k$

$\delta_j = -\frac{\partial E}{\partial a_j}$
$\delta_j = -\sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial a_j}$
$\delta_j = -\sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial y_j}\frac{\partial y_j}{\partial a_j}$
$\delta_j = \sum_k \delta_k w_{j,k} y'_j$
$\delta_j = y'_j \sum_k w_{j,k}\delta_k$

### 1.3 Problem 2b

$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial a_j}{\partial w_{i,j}}\frac{\partial E}{\partial a_j}$
$\frac{\partial E}{\partial w_{i,j}} = -x_i\delta_j$
$w_{i,j} = w_{i.j} - \alpha\frac{\partial E}{\partial w_{i,j}}$
$w_{i,j} = w_{i.j} + \alpha x_i\delta_j$

$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial a_k}{\partial w_{j,k}}\frac{\partial E}{\partial a_k}$
$\frac{\partial E}{\partial w_{j,k}} = -y_j\delta_k$
$w_{j,k} = w_{j,k} - \alpha\frac{\partial E}{\partial w_{j,k}}$
$w_{j,k} = w_{j,k} + \alpha y_j\delta_k$

## 1.4 Problem 2c

$W_{H,O} = W_{j,k} + \alpha y_j \delta_k$
$W_{I,H} = W_{i,j} + \alpha x_i \delta_j$
Where $y_j \delta_k, x_i \delta_j$ are outer products

## 1.5 Problem 2d

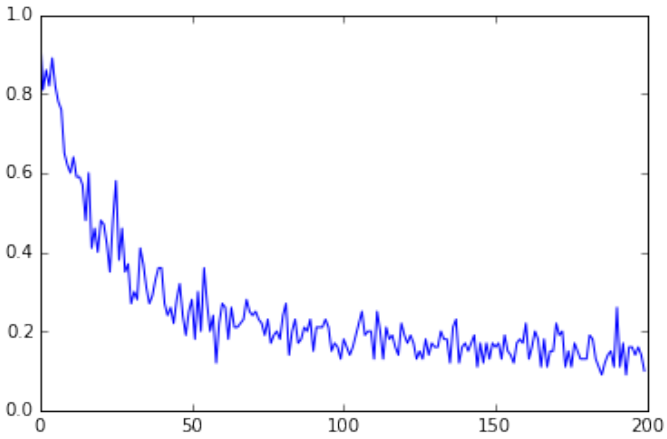
For results on numerical gradient vs backprop, please see the "txt" file in email attachment.

First split training, validation, and testing set = 50000, 10000, 10000
Using 200 hidden nodes, 200 epoches, with a constant learning rate of 0.1, mini-batch size = 100 (in random order), I trained a normal multi-layer net as follows:

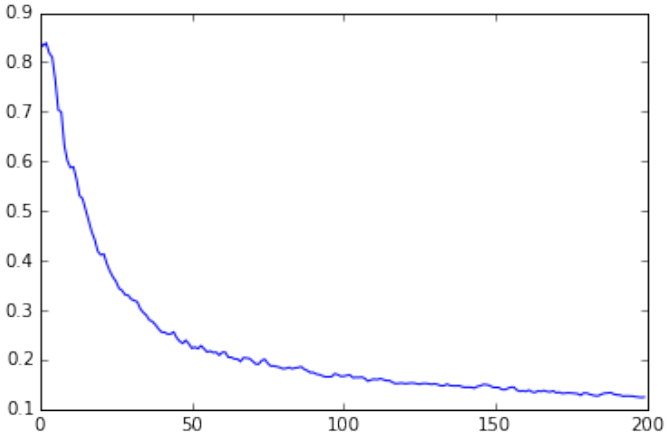
foreach epoch:
—-foreach training sample in mini-batch:
———propagate forward to get prediciton
———propagate backward to update the weights
—-append training error
—-validation, append validation error
—-test, append testing error

I obtained the follow plots.

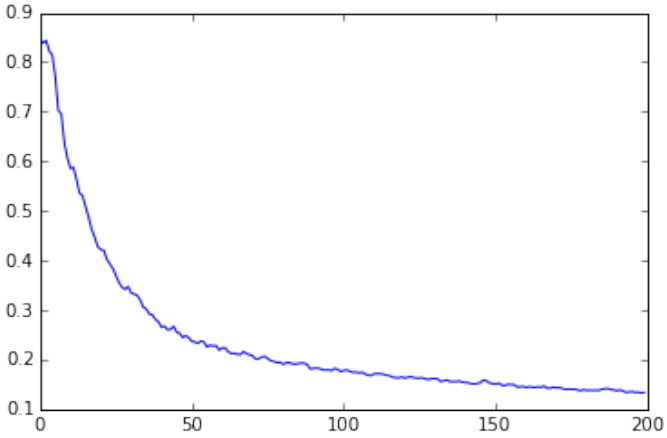
Training error rate




Validation error rate



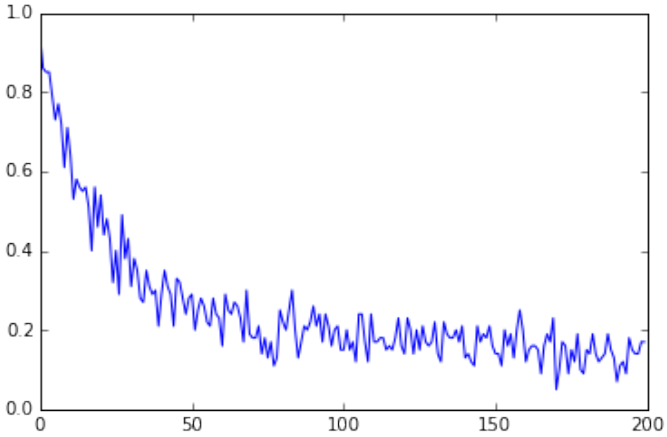

Testing error rate



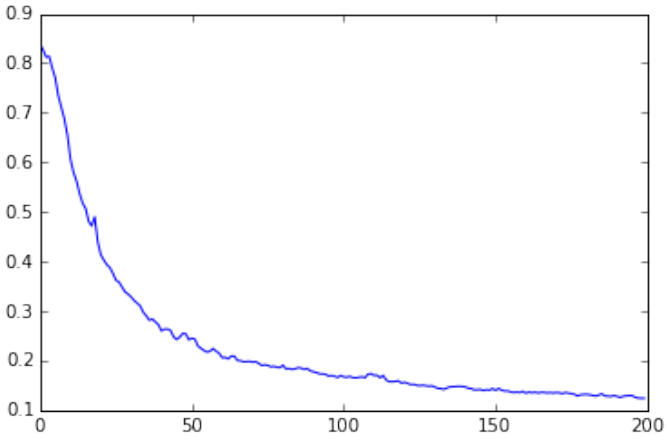

## 1.6 Problem 2e - Regularization

Using the same parameter set (200 hidden nodes, 200 epoches, 0.1 learning rate, mini-batch = 100) but adding regularization in the update rule: in this case, simply multiple previous weights by $1 - r * regularization_rate/trainingsize$, which is usually about 0.99

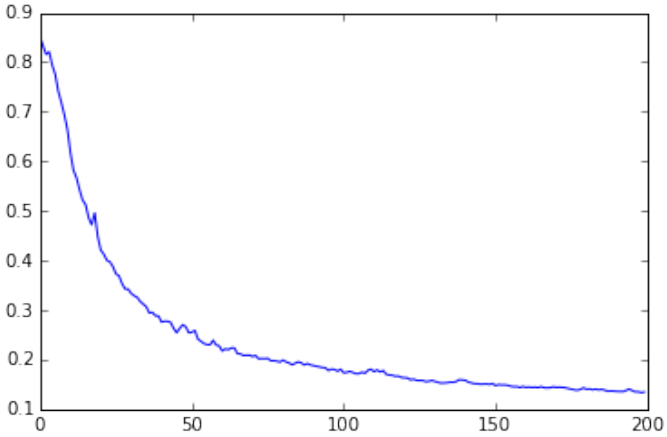
Training error rate




Validation error rate

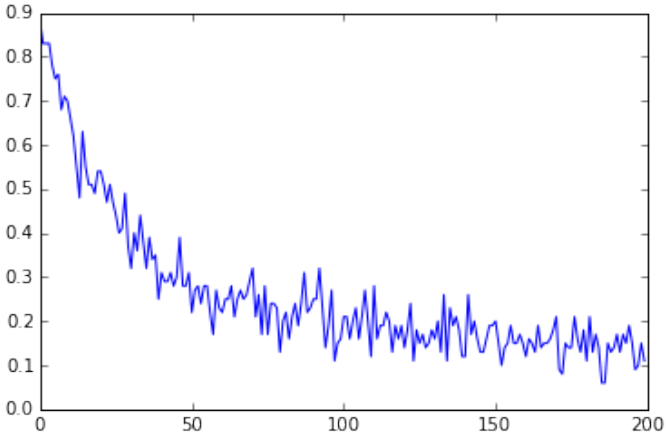



Testing error rate

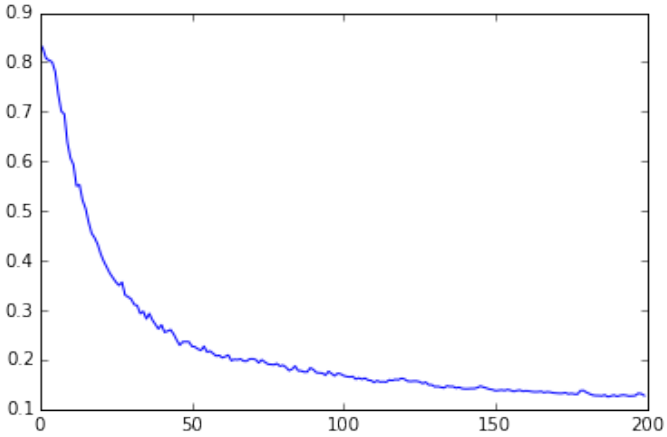



Changing regularization parameter to 0.0001, obtain the following plots:

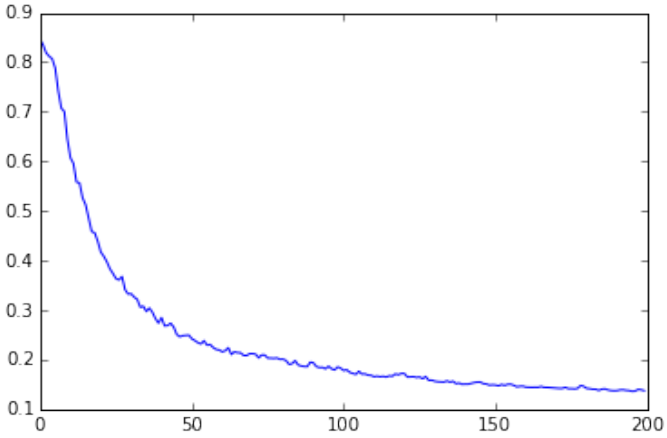
Training error rate




Validation error rate




Testing error rate

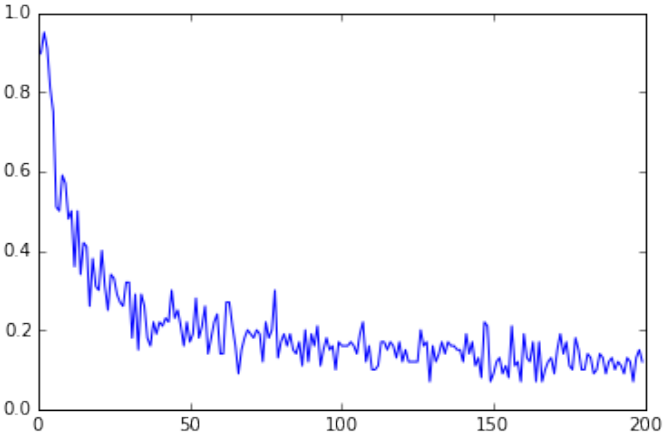



It is interesting to see these two set of plots produce very similar results compare to a normal multiple layer network. Because regularization is used to prevent overfitting, in our case, there is no evidence pf overfitting in our normal testing plot, so it is hard to observe it given only 200 epoches

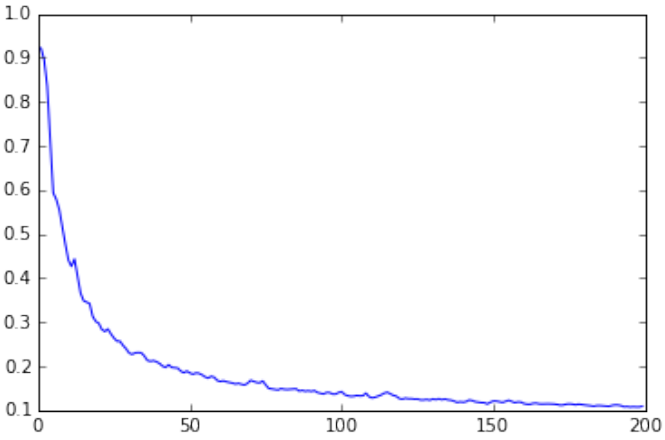
## 1.7 Problem 2f - Momentum

Adding momentum is also straightforward, just simply add a fraction (in this case 0.9) of the previous weight changes to the new weight, along with the new weight changes.

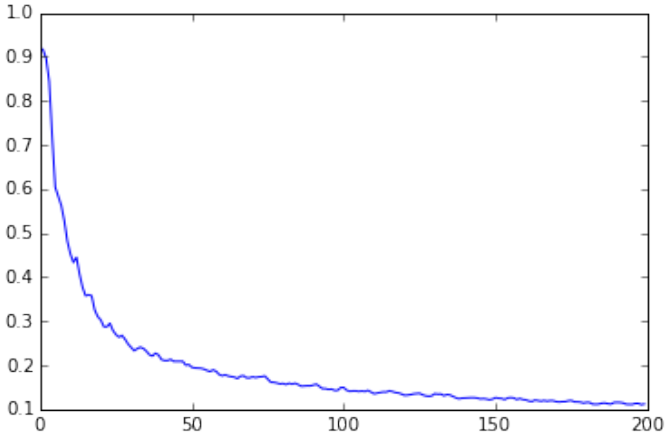
Training error rate




Validation error rate




Testing error rate

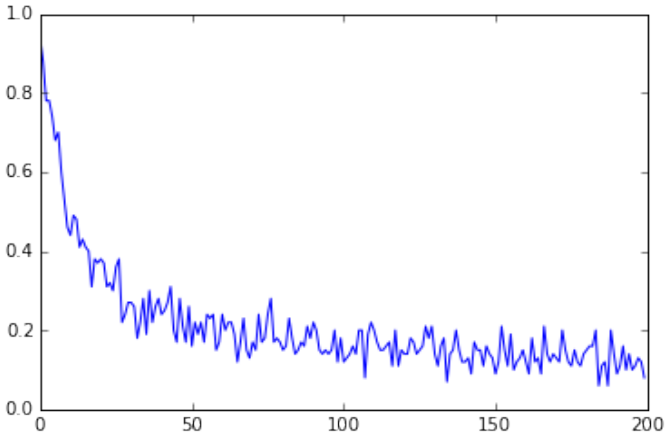



Clearly, both given 200 epoches, momentum produces better accuracy compare to the normal case. Also, notice the training error drops a little bit significantly compre to the normal case. (converge a bit faster in the first few epoch)

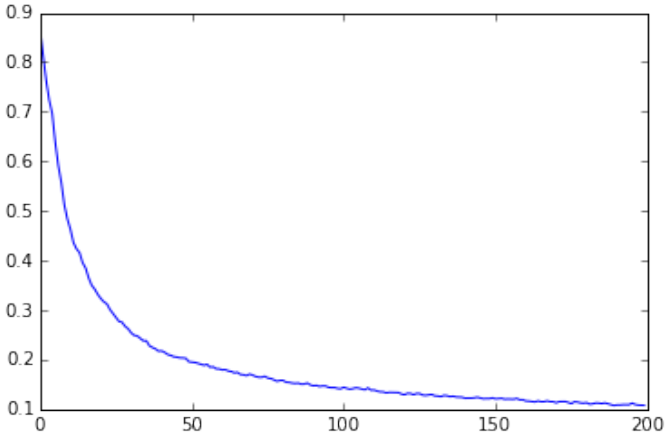
## 1.8 Problem 2g - Different Activation

For using different activation functions, we simply add if statements in our code, and noticing the gradient of tanh is $1 - tanh^2$ and the gradient of reul is the sigmoid function. Using same parameters, results are shown below.

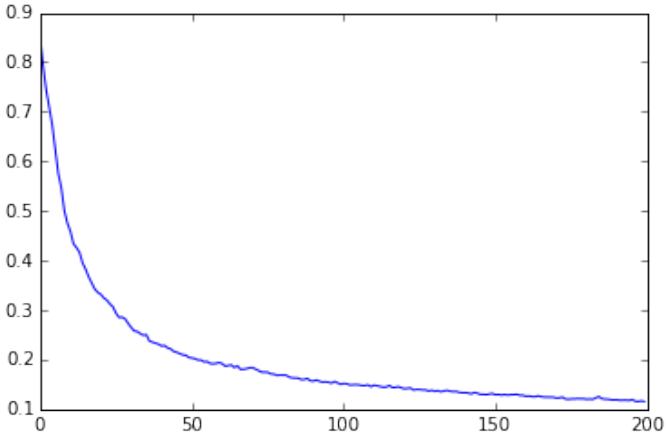
tanh:
Training error rate



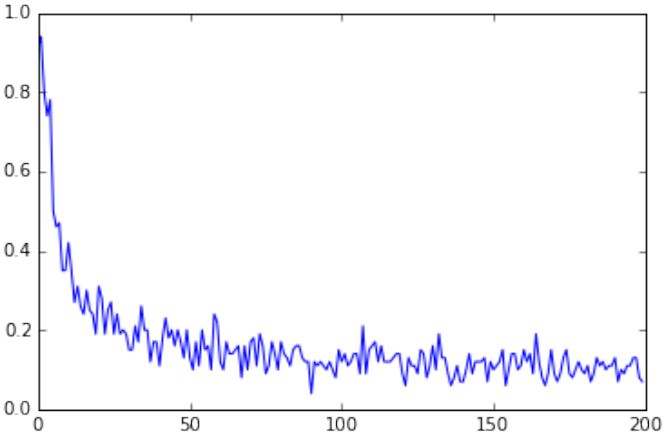

Validation error rate



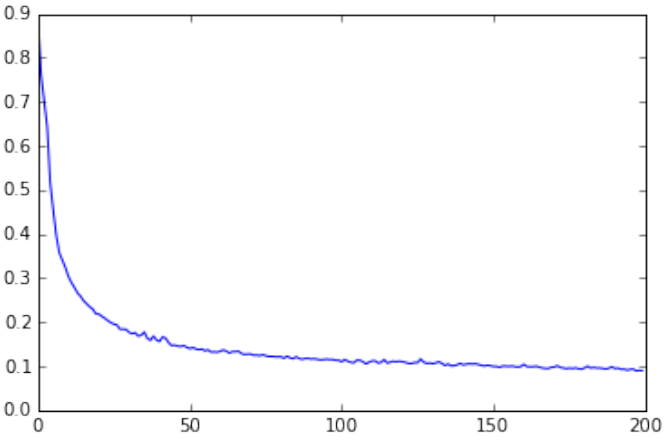

Testing error rate



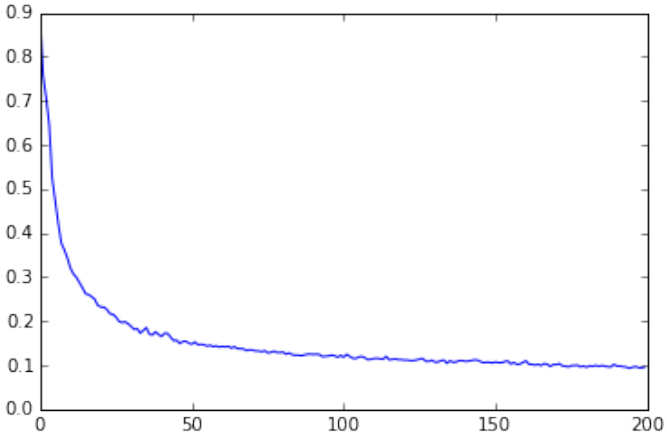

Relu:
Training error rate




Validation error rate



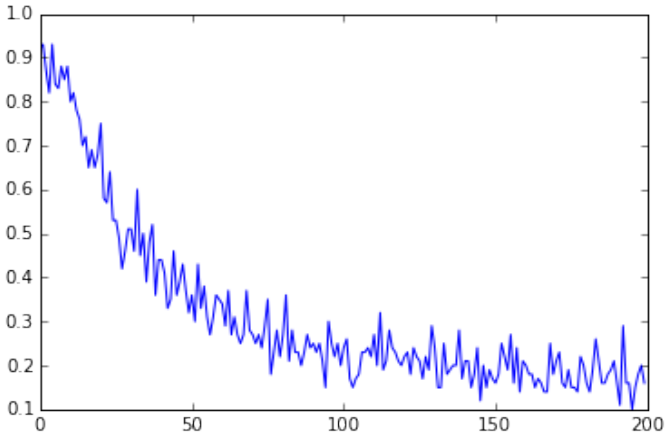

Testing error rate



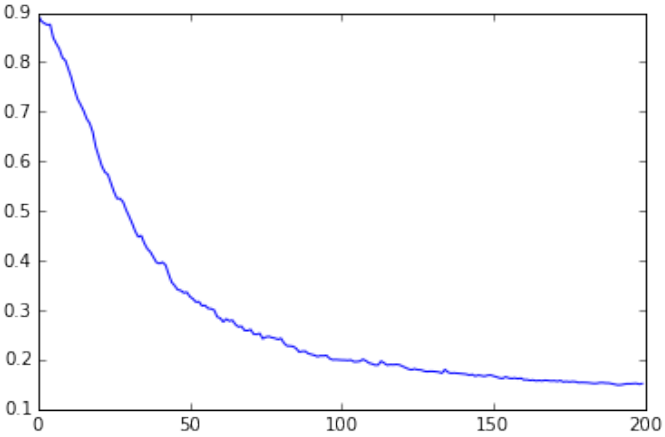

Some of the things worth noticing:
1.both tanh and relu produce better training accuracy, or converge faster than sigmod. Ex: at epoch = 25, sigmoid gives 0.4 training error while tanh and relu give 0.2.
2.tanh gives a slightly better test accuracy (0.12), and relu gives the best out of all three (0.1) testing error rate.
3. tanh and relu both are faster.

## 1.9 Problem 2h - Network Topology

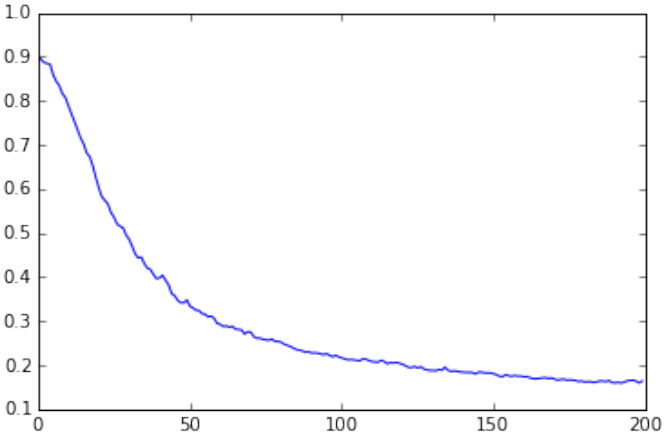
Halve the hidden layer size:
Training error rate




Validation error rate

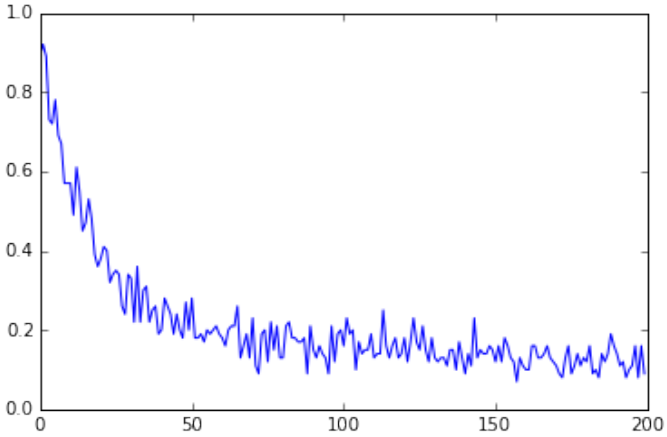



Testing error rate

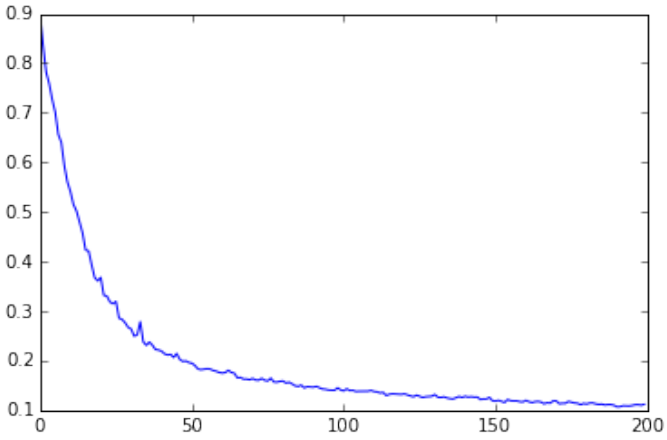



One observation I can say is that shrinking the hidden layer size gives slightly worse testing accuracy, in this case, it is about 5 percent more than the normal case. And it is converging slower than the one with double the number of hidden nodes. (See below)

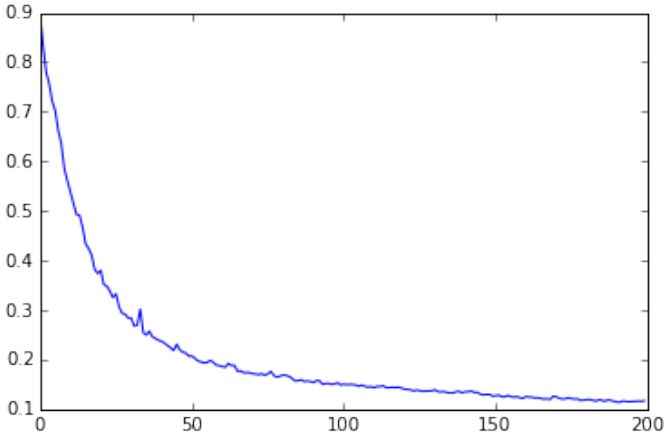
Double the hidden layer size:
Training error rate




Validation error rate

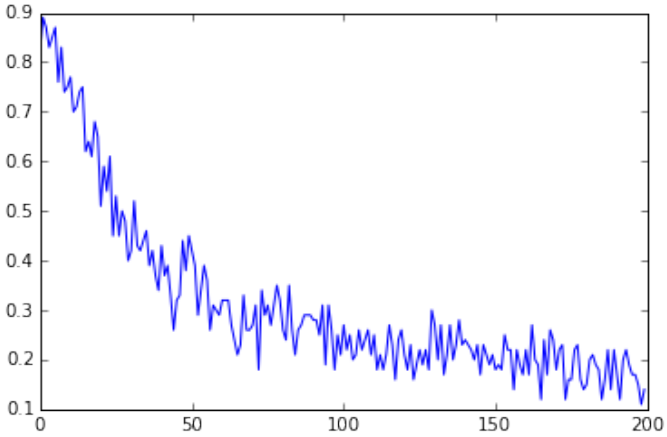



Testing error rate

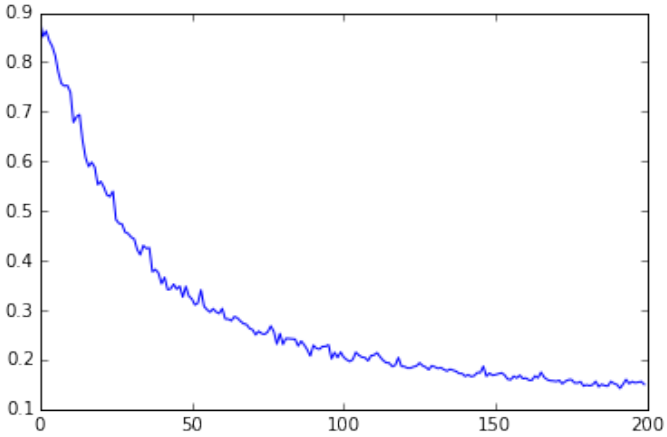



In addition, here are the plots after adding an extra layer, everything else keeps the same.

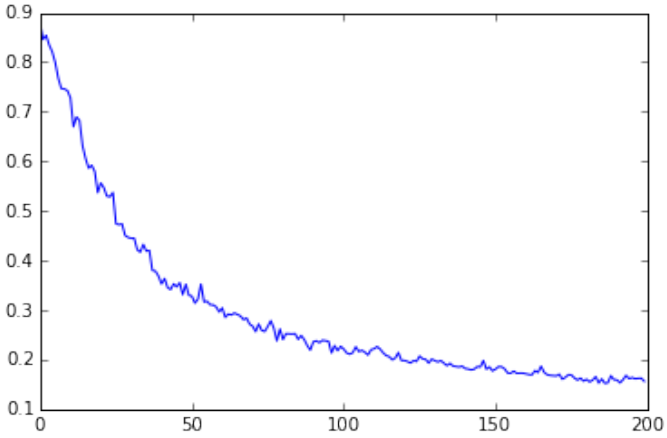
Training error rate




Validation error rate




Testing error rate




One interesting thing we can say from the 2 hidden layers architecture is that there are more "spikes" in the plots, and the accuracy is slightly worse than the one with only one hidden layer.
In general, choosing the best number of layers and number of hidden nodes depends on the problem and the data, so there is no best answer.

# 2 Conclusion

In this homework, we are introduced to multi-layer neural network and trained digit classifier using backpropgation. A multi-layer network clearly takes more time to train than a typical perceptron or logistic regression which doesn't have hidden layer.

In addition, several techniques are also introduced: regularization, momentum, different activation functions, different hidden layer size and number of hidden layers...etc, which either help us speed up training or improve test accuracy.

# Appendices