CSI 873 Fall 2017 Midterm

Bruce Goldfeder

January 3, 2018

1 Code Artificial Neural Net

I coded the Artificial Neural Net (ANN) in python following the algorithm for stochastic gradient descent in TABLE 4.2 from the text on page 98. I implemented the python using looping structures for the main parts of the algorithm and matrix mathematics using the python numpy library. My results were were consistent with the graphs shown in the text on page 110. I provided the following set of parameters that are adjustable at the command line when invoking the application.

```
**Network Parameters**
input: number of input units
hidden: number of hidden units
output: number of output units

**Training Parameters**
train: number of training images to use
valid: number of validation images to use
test: number of test images to use
epochs: number of full training set run throughs

**Hyperparameters**
lrnrate: this is the factor for learning rate
momentum: this is the momentum term added to weight update
stop: This is the stopping criteria a threshold of percent drop in validation
error per iteration
```

```
**Future implementation**
rate_decay: used for annealing the learning rate using a 1/t decay using
            the mathematical form alpha=alpha0/a0+(a0+a0*rate_decay)
```

I also included a parameter that I will be adding in at a future date called ratedecay. I am hoping that this will have a limiting effect on training set overfitting after a large number of weight updates. I researched various methodologies to improve performance such as this annealing of the learning rate using a 1/T or a step methodology. The ANN algorithm is based directly from the Tom Mitchell text. Additional research for mitigating overfitting is from the Stanford Unsupervised Feature Learning Deep Learning web site ($http://deeplearning.stanford.edu/wiki/index.php/UFLDL\_Tutorial$)

I also researched various algorithms for the sigmoid unit and settled on the sigmoid presented in the text.

$$f(z) = \frac{1}{1 + \exp(-z)}$$

The other function that I explored was the tanh() function. This is a little tricky but easily overcome in that it has a range of [-1,1] while the sigmoid range is [0,1]. The tanh function is shown below:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

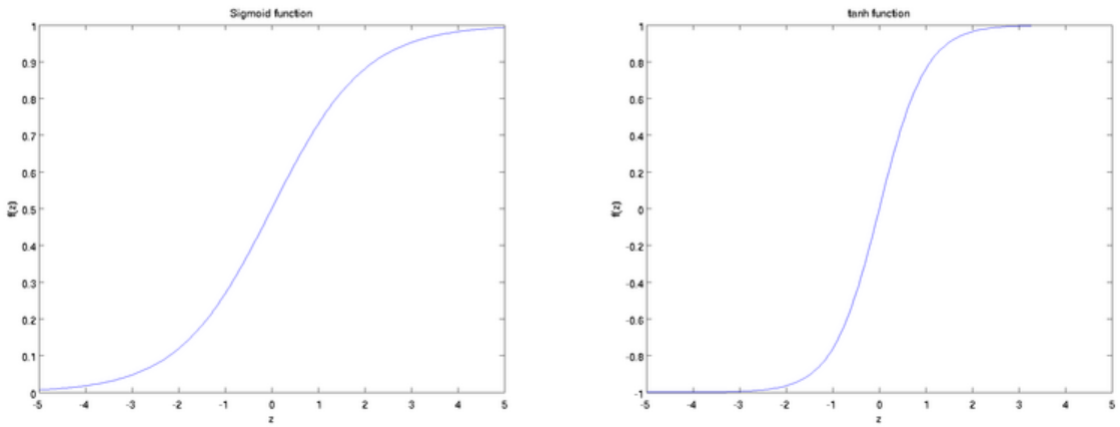


Figure 1: The Sigmoid and tanh functions

Both of these functions have easy to derive and easy to code derivatives:
Sigmoid Derivative

$$f'(z) = f(z)(1 - f(z))$$

tanh Derivative

$$f'(z) = 1 - (f(z))^2$$

For the midterm implementation, I leveraged the standard sigmoid function and derivative. I also coded a more robust version to make this function a bit more robust as I would have a tendency to run into overflow errors as the exponential term reached very large negative numbers. In later iterations of the code this was not needed as I had better control on the weights. The code that corrects these issues is shown below:

```
def sigmoid(z):
    try:
        if (z < 0):
            sig = 1.0 - 1.0/(1.0 + math.exp(z))
        else:
            sig = 1.0 / (1.0 + math.exp(-z))
    except OverflowError:
        if (z > 0):
            sig = 0.0000001
        else:
            sig = 1000000
    return sig
```

The code contained in the class NeuralNet contains four major functions matching the four steps defined in the text on Table 4-2 on page 98. The driver code reads in the data files, executes the methods first for training, then for validation, and finally for running the test set through for final results.

2 How to run the code

The python version is the current version of Anaconda which is Python 3.6.2. The code uses command line arguments that are optional and are passed into the program using the following flags:

An example to run trial number one is:

```
python csi873bgoldfeder.py -f C:/csi873/csi873midterm/data -e 30 -t 4000 -v 500
    -x 890
    -l 0.1 -m 0.1 -s 0.001
```

Table 1: Arguments for the program csi873bgoldfeder.py

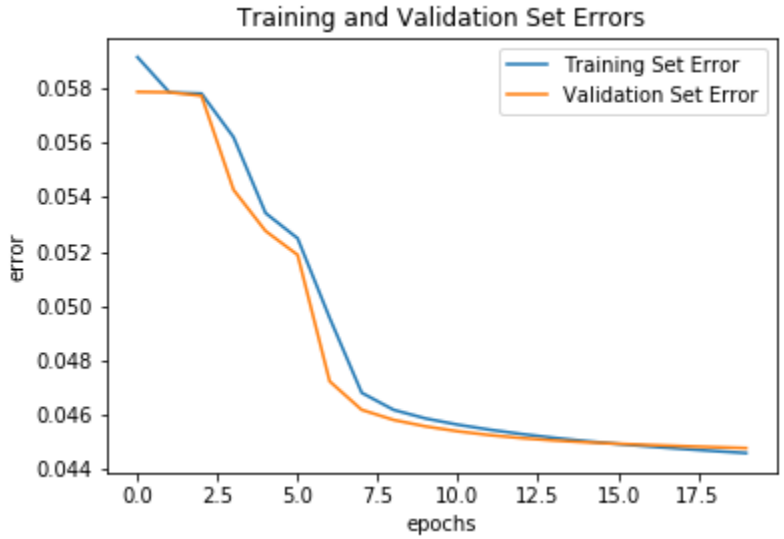
flag	**description**	**example**
-f	path to data	C:/csi873/data
-i	# hidden nodes	3
-e	# epochs to run	10
-t	# of training images per written number	100
-v	# validation images per written number	50
-x	# test images per written number	3
-l	learning rate	0.3
-m	momentum	0.3
-s	stop criteria	0.0005

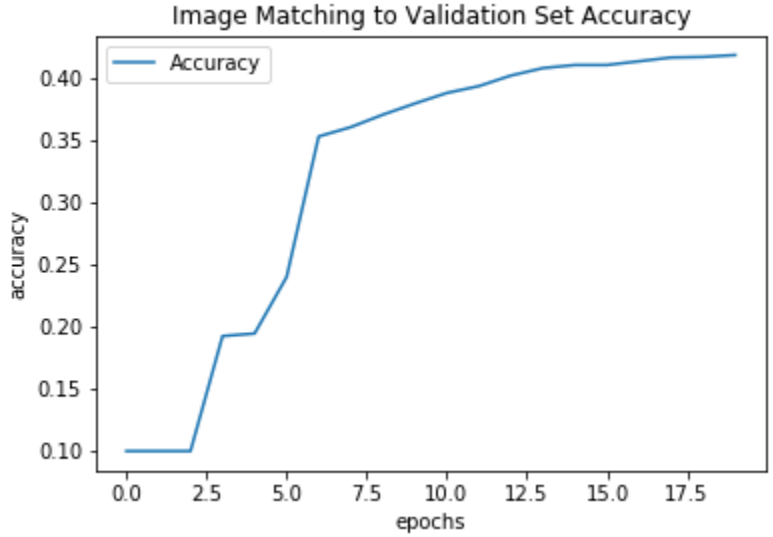
3 Successful Trial Run with limited data set

A successful trial run that indicated to me the ANN was running well is shown below using the following set of input parameters:

```
input      ==>        784
hidden     ==>          3
output     ==>         10
number of training images ==>       3000
number of validation images ==>       2000
number of test images ==>       2000
epochs     ==>         20
learn_rate ==>       0.10
momentum   ==>       0.10
```

Although the final accuracy rate was not the greatest, the behavior shown in both sets of output graphs validated that the ANN was iteratively reducing both the training set and the validation set sum of squared errors per epoch. The accuracy for this first trial came to 39.9% accuracy for the test set. The plots for the error and accuracy during training are shown below. Even with this small sample we are observation the positive reduction of training errors and iterative increase in accuracy of the validation set per epoch. I even noticed a slight start of overfitting in the error plot where the training error starting falling below the validation error.



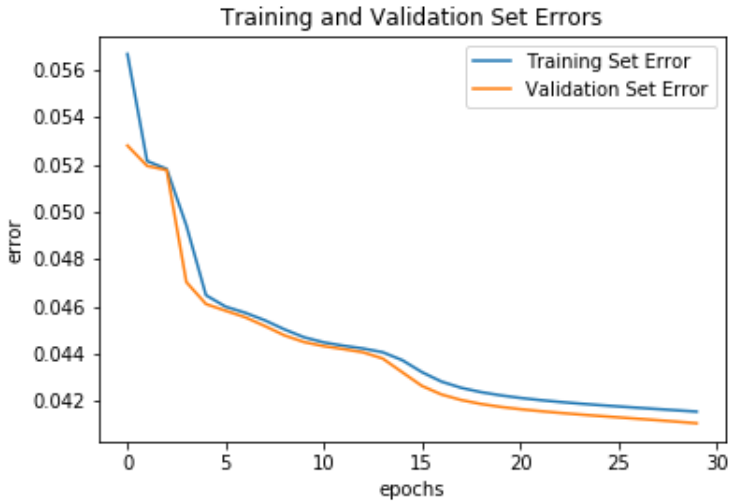


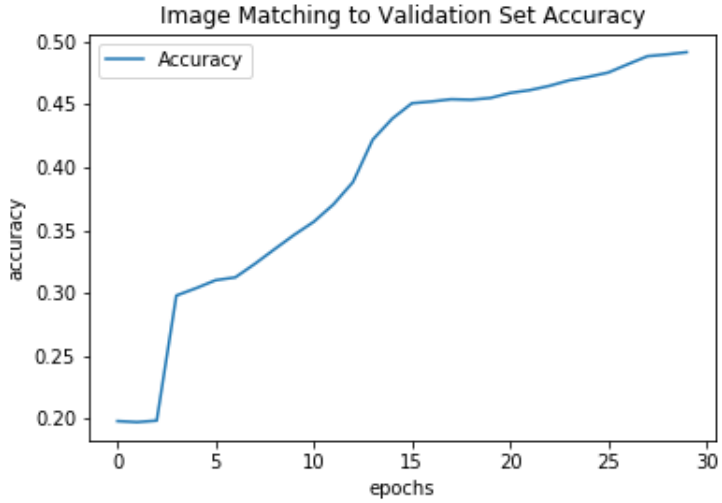
4 Trial One - 3 hidden nodes

To make this and later trials, for 2 and 3 nodes respectively, I increased the training sample size to 4000, the validation set size to 500, and the test set size to 890. I increased the number of epochs to 30. In addition, the initial runs went very quickly to overfitting, to counter this I lowered the learning rate to 0.03. This is due to the fact that the number of input images increased by a factor of 10, I inferred that the impact of these parameters would need to be mitigated proportionately. The output is as follows:

```
input       ==>         784
```

```
hidden     ==>          3
output     ==>         10
number of training images ==>      40000
number of validation images ==>       5000
number of test images ==>       8900
epochs     ==>         30
learn_rate ==>       0.03
momentum   ==>       0.10
Final Test results of  4301  out of  8900  accuracy is  0.4832584269662921
```





This provided an improvement over the smaller data set providing 48.3% accuracy over the test set. Lowering the learning rate had the positive effect of mitigating the overfitting, but I observe that I can raise it a little or I can add epochs of training.

4.a 95% Confidence Interval

The result set was 4301 correct responses out of 8900 images tested. The 95% confidence interval can be calculated using the formulas from chapter 5:

$$\sigma_{error_s(h)} = \sqrt{\frac{p(1-p)}{n}}$$

4301 correct responses out of 8900 = .4833

$$\sigma_{error_s(h)} = \sqrt{\frac{.4833(.5167)}{8900}} = .005297$$

Applying the formula

$$\mu \pm z_N\sigma; 1.96 \times .005297 = .01038$$

giving .4833 ± .01038; a 95% confidence interval of [.4729,.4937]

5 Trial Two - Two hidden nodes

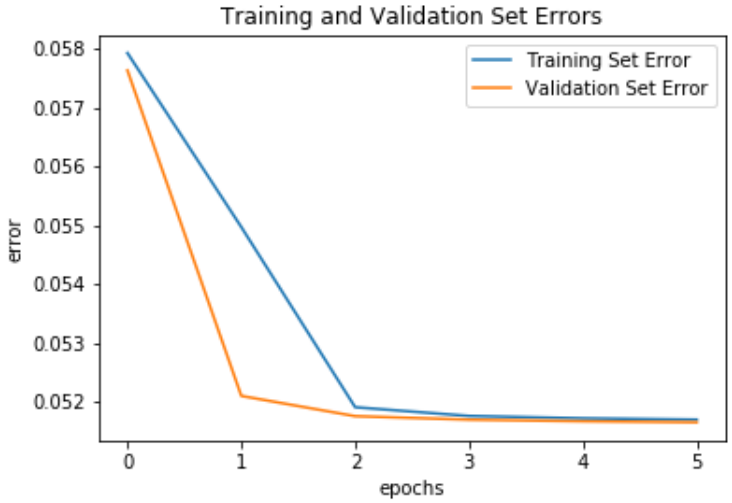
```
csi873bgoldfeder.py -f C:/csi873/csi873midterm/data -i 2 -e 30 -t 4000 -v 500 -x 890
-l 0.05 -m 0.1
```

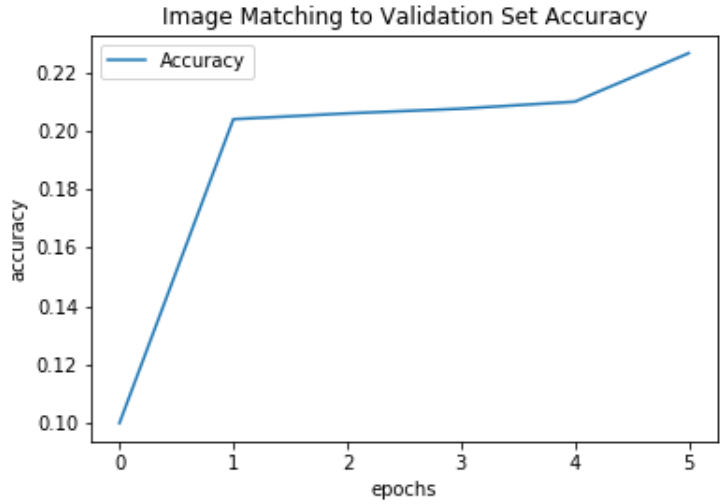
In this trial the number of hidden nodes was reduced from three to two hidden nodes. I also bumped up the learning rate from .03 to .05 to see if that accelerates the learning rate and I can observe the cross-over of the training and the validation errors indicating overfitting.
The parameters used are shown below:

```
input      ==>        784
hidden     ==>          2
output     ==>         10
number of training images ==>      40000
number of validation images ==>       5000
number of test images ==>       8900
epochs     ==>         30
learn_rate ==>       0.05
```

```
momentum    ==>        0.10
stopping criteria ==>     0.00050
```

By limiting the number of hidden units down to two, the output seemed to become limited due to the lack of expression ability from just the two hidden units. This trial ended very early by hitting the stopping criteria on the 5th iteration. Understandably the accuracy of 22.49% was about half of the prior trial using three hidden units. The graphs are shown below:





5.a 95% Confidence Interval

The result set was 2002 correct responses out of 8900 images tested. The 95% confidence interval can be calculated using the formulas from chapter 5:

$$\sigma_{error_s(h)} = \sqrt{\frac{p(1-p)}{n}}$$

2002 correct responses out of 8900 = .2249

$$\sigma_{error_s(h)} = \sqrt{\frac{..2249(.7751)}{8900}} = .004426$$

Applying the formula

$$\mu \pm z_N\sigma; 1.96 \times .004426 = .008675$$

giving .2249 ± .008675; a 95% confidence interval of [.2163,.2336]

6 Trial 3: Four Hidden Units

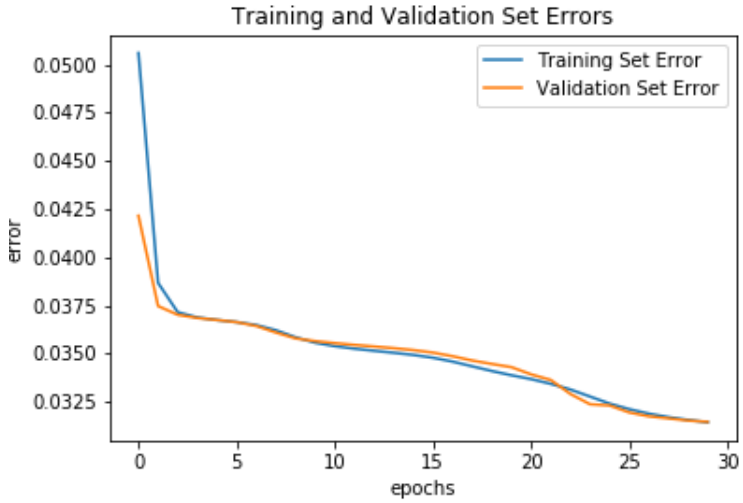
```
csi873bgoldfeder.py -f C:/csi873/csi873midterm/data -i 4 -e 30 -t 4000 -v 500 -x 890
-l 0.05 -m 0.1
```

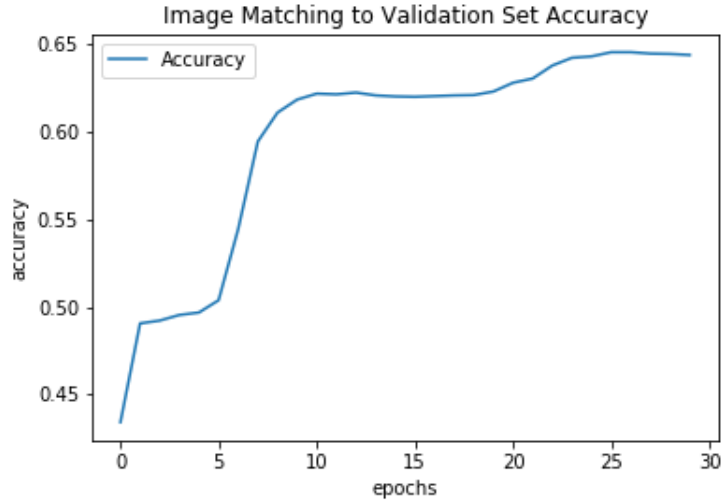
For this last trial the number of hidden units is increased to four. I left all other parameters the same to include the stopping criteria of .05% drop in Validation Error rate (this is the default setting).
The following parameters were used:

```
input      ==>        784
hidden     ==>          4
output     ==>         10
number of training images ==>      40000
number of validation images ==>       5000
number of test images ==>       8900
epochs     ==>         30
learn_rate ==>       0.05
momentum   ==>       0.10
stopping criteria ==>    0.00050
```

The addition of two additional hidden nodes from the previous trial nearly tripled the test accuracy. An interesting observation is that the stopping criteria did not get triggered using the threshold percentage

drop of .05% decrease in validation error. A higher learning rate could impact this or an increase in the number of epochs. From the graphs below it appears that additional accuracy could have been gained with either of these approaches. The momentum appears to have popped the validation set accuracy in two locations on the graph where major leaps in accuracy occurred leading to the final more gradual climb at the end. The graphs are shown below:





6.a 95% Confidence Interval

The result set was 5689 correct responses out of 8900 images tested. The 95% confidence interval can be calculated using the formulas from chapter 5:

$$\sigma_{error_s(h)} = \sqrt{\frac{p(1-p)}{n}}$$

5689 correct responses out of 8900 = .6392

$$\sigma_{error_s(h)} = \sqrt{\frac{.6392(.7751)}{8900}} = .005090$$

Applying the formula

$$\mu \pm z_N\sigma; 1.96 \times .005090 = .009976$$

giving .6392 ± .009976; a 95% confidence interval of [.6292,.6492]

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 18 18:29:16 2017

@author:   Bruce Goldfeder
           CSI 873
           Fall 2017
           Midterm
"""

import os,sys
import math
import numpy as np
import matplotlib.pyplot as plt

def sigmoid2(z):
    try:
        if (z < 0):
            sig = 1.0 - 1.0/(1.0 + math.exp(z))
        else:
            sig = 1.0 / (1.0 + math.exp(-z))
    except OverflowError:
        if (z > 0):
            sig = 0.01
        else:
            sig = 100
    return sig

def sigmoid(z):
    sig = 1.0 / (1.0 + math.exp(-z))
    return sig

def ReadInFiles(path,trnORtst):
    # This reads in all the files from a directory filtering on what the file
    # starts with
    fullData = []
    fnames = os.listdir(path)
    for fname in fnames:
        if fname.startswith(trnORtst):
            print (fname)
            data = np.loadtxt(path + "\\" + fname)
            fullData.append(data)
    #numFiles = len (fullData)
    #print(numFiles)

    return fullData

```

```
48 def ReadInOneList(fullData,maxRows):
49     # This function combines all of the data into one array for ease of use
50     # It contains a capping ability to configure how many results to use
51     allData = []
52     numFiles = len (fullData)
53     for j in range (numFiles):
54         # allows for smaller data set sizes
55         numRows = len (fullData[j])
56         #print('numrows,maxrows ',numRows,maxRows)
57         if (maxRows < numRows):
58             numRows = maxRows
59 
60         for k in range(numRows):
61             allData.append(fullData[j][k])
62     return np.asarray(allData)
63 
64 def ReadInValidList(fullData,start,end):
65     # This function combines all of the data into one array for ease of use
66     # It contains a capping ability to configure how many results to use
67     allData = []
68     numFiles = len (fullData)
69     for j in range (numFiles):
70         # allows for smaller data set sizes
71         numRows = len (fullData[j])
72         if numRows < start + end :
73             print('Starting at ' + str(start) + ' there are not ' + str(end) + \
74                   ' images in the data set. Please retry')
75             sys.exit()
76         for k in range(start,end):
77             allData.append(fullData[j][k])
78     return np.asarray(allData)
79 
80 
81 def HeatMap(numberIn):
82     #heat map to show numbers
83     plt.matshow(numberIn.reshape(28,28))
84     plt.colorbar()
85     plt.show()
86 
87 class NeuralNet(object):
88 
89     def __init__(self, input, hidden, output, inNum, valNum, tstNum, epochs, lrn_rate,
        momentum,stop):
90         """
91         **Network Parameters**
92         input: number of input units
93         hidden: number of hidden units
```

```
94          output: number of output units
95
96          **Training Parameters**
97          inNum: number of images to run
98          epochs: number of full training set run throughs
99
100         **Hyperparameters**
101         lrn_rate: this is the factor for learning rate
102         momentum: this is the momentum term added to weight update
103         rate_decay: used for annealing the learning rate using a 1/t decay using
104                     the mathematical form alpha=alpha0/a0+(a0+a0*rate_decay)
105
106         """
107         # Training Parameters
108         self.inNum = inNum
109         self.valNum = valNum
110         self.tstNum = tstNum
111         self.epochs = epochs
112
113         # Hyperparameters
114         self.lrn_rate = lrn_rate
115         self.momentum = momentum
116
117         # initialize arrays
118         self.input = input # add 1 for bias node
119         self.hidden = hidden # This bias node add causes trouble
120         self.output = output
121
122         # set up arrays for the outputs of the nodes
123         self.ai = np.ones(self.input) # removes the threshold input
124         #print('ai shape ',self.ai.shape)
125         self.ah = np.ones(self.hidden) # removes the threshold input
126         self.ao = np.ones(self.output)
127
128         # create random weights between -0.05 and 0.05 as per text on pg. 98
129         # plus one for the bias/threshold unit
130         self.wHidThresh = np.random.uniform(-0.05, 0.05, size = (self.hidden))
131         self.wOutThresh = np.random.uniform(-0.05, 0.05, size = (self.output))
132
133         self.wi = np.random.uniform(-0.05, 0.05, size = (self.input, self.hidden))
134         self.wo = np.random.normal(-0.05, 0.05, size = (self.hidden, self.output))
135
136         # temporary arrays to hold the numbers to be updated each iteration
137         self.ci = np.zeros((self.input, self.hidden))
138         self.co = np.zeros((self.hidden, self.output))
139
140         # Error array to capture the output array for each training output unit
```

```
141         self.outUnitErr = np.zeros((self.inNum,self.output))
142 
143         # Error array to capture the output array for each training output unit
144         self.outValErr = np.zeros((self.valNum,self.output))
145 
146         # Error array to capture the output array for each training output unit
147         self.outTstErr = np.zeros((self.tstNum,self.output))
148 
149         # String to append to files to identify experiment parameters
150         self.expName = 'in' + str(self.input) + 'hi' + str(self.hidden) + \
151                  'out' + str(self.output) + 'lr' + str(self.lrn_rate) + \
152                  'mo' + str(self.momentum) + 'trN' + str(self.inNum) + \
153                  'vN' + str(self.valNum) + \
154                  'tsN' + str(self.tstNum) + 'ep' + str(self.epochs)
155 
156         # Set the stopping criteria which is the percent drop in Validation
157         # from the current iteration to the previous
158         self.stop = stop
159 
160         print("name " + self.expName)
161 
162     def print_params(self):
163 
164         print ('%-10s ==> %10d' % ('input', self.input))
165         print ('%-10s ==> %10d' % ('hidden', self.hidden))
166         print ('%-10s ==> %10d' % ('output', self.output))
167         print ('%-10s ==> %10d' % ('number of training images', self.inNum))
168         print ('%-10s ==> %10d' % ('number of validation images', self.valNum))
169         print ('%-10s ==> %10d' % ('number of test images', self.tstNum))
170         print ('%-10s ==> %10d' % ('epochs', self.epochs))
171         print ('%-10s ==> %10.2f' % ('learn_rate', self.lrn_rate))
172         print ('%-10s ==> %10.2f' % ('momentum', self.momentum))
173         print ('%-10s ==> %10.5f' % ('stopping criteria', self.stop))
174 
175     def makeTargetArray(self,answer):
176         tk = int(answer)
177         #print('tk is ', tk)
178         # Make the trained output 0.1 for 0 and 0.9 for 1 as per the text
179         # from the first paragraph on the top of page 115
180         tkArray = np.add(np.zeros(10),0.1)
181         tkArray[tk]= 0.9
182         #print("answer array is ",tkArray)
183 
184         return tkArray
185 
186     def feedForward(self,image,answer):
187 
```

```
188         #print (input,len(inputs))
189         """
190         The feedforward algorithm propogates the input forward through the network
191         It calculates the net by summing from one to n the weights (including bias
192         w0) times the value sigma term for each node in the hidden and output
193         layers. This is then used as the negative exponent value in the sigmoid
194         function
195             net = Sigma[i=0,n]wi*xi
196             ah = sigmoid(net) = 1/(1 + e**-net)
197         """
198         #self.ai[0] = 1.0 # This is the threshold for every hidden unit
199         self.ai = image # remember to account for actual value in 0th index
200 
201         # hidden activations
202 
203         for j in range(self.hidden):
204             sum = self.wHidThresh[j]
205             for i in range(self.input):
206                 prodAW = self.ai[i] * self.wi[i][j]
207                 sum += prodAW
208             #print ('sum is ',sum)
209             self.ah[j] = sigmoid(sum) #what about using tanh here?
210 
211         # output activations
212         for k in range(self.output):
213             sum = self.wOutThresh[k]
214             for j in range(self.hidden):
215                 prodAWO = self.ah[j] * self.wo[j][k]
216                 sum += prodAWO
217             self.ao[k] = sigmoid(sum)
218 
219     def calcDeltaKO(self,answer):
220         """
221         Calculates the delta_k for each output unit using the formula from the text
222         delta_k = ouput_k * (1-ouput_k)*(target_k - output_k)
223         """
224         tkArray= self.makeTargetArray(answer)
225 
226         delta_k = np.zeros(self.output)
227 
228         # Use the derivative of sigmoid times actual - observed
229         for y in range(self.output):
230             delta_k[y] = self.ao[y] * (1-self.ao[y]) * (tkArray[y] - self.ao[y])
231             #print ('tkarray[y] ',tkArray[y],' ao[y] ',self.ao[y],' sub ',tkArray[y] - self.
                ao[y])
232 
233 
```

```
234         return delta_k
235
236     def calcDeltaKH(self,d_k):
237         """
238         Calculates the delta_h for each hidden unit using the forumula from the text
239         delta_h = output_h * (1-ouput_h) * Sum(weight_kh * delta_ko)
240         ah is the output of hidden units
241         w_kh is the weight of hidden units or the array in main 'wi'
242         d_k is the delta_k returned from the function 'calcDeltaKO' for output nodes
243         """
244
245         # Use derivative term for sigmoid applied to the hidden outputs
246         delta_h1 = np.zeros(self.hidden) # first part of equation
247         delta_h = np.zeros(self.hidden) # output of equation
248         # corrected - switched rows and columns
249         for hid in range(self.hidden):
250             sum = 0.0  # question should this be the threshold value?
251             delta_h1[hid] = self.ah[hid] * (1 - self.ah[hid])
252             #print('y is ',y)
253             # This calculates the second part of equation summing over this hidden
254             # node the product of the weight_kh*delta_k of the output node
255             for out in range(self.output):
256                 #print('x is ',x)
257
258                 # the product of the weight_kh*delta_k of the output node
259                 delta_h2 = self.wo[hid,out] * d_k[out]
260                 # Sum up all weight*dk for all output this hidden node touches
261                 sum += delta_h2
262                 #print(sum)
263
264             # Calculate the derivative times the sum of the w_kh * delta_k
265             dh = delta_h1[hid] * sum
266             #print('delta h is ',dh)
267             delta_h[hid] = dh
268
269         #print('delta_h is: ',delta_h,' shape is ',delta_h.shape)
270         return delta_h
271
272     def updateWeights(self,answer,d_ko,d_kh):
273         """
274         The weights are updated using the forumula from the text
275         w_ji <-- w_ji + Delta(w_ji)
276
277         where
278         Delta(w_ji) = n*delta_j*x_ji
279         """
280         # update the weights connecting hidden to output
```

```
281         # the co array represents the n-1 or prior iteration delta value
282         for j in range(self.hidden):
283             for k in range(self.output):
284                 delta = self.lrn_rate * d_ko[k] * self.ah[j] + (self.momentum * self.co[j][k
                        ])
285                 #print('delta is ',delta)
286                 self.wo[j][k] += delta
287                 self.co[j][k] = delta
288 
289         # update the weights connecting input to hidden
290         for i in range(self.input):   # add in w0 threshold term
291             for j in range(self.hidden): # add in w0 threshold term
292                 delta = self.lrn_rate * d_kh[j] * self.ai[i] + (self.momentum * self.ci[i][j
                        ])
293                 self.wi[i][j] += delta
294                 self.ci[i][j] = delta
295 
296     # to save time and complexity these are 3 different functions
297     #TODO refactor to one function
298     def calculateError(self,num,answer):
299 
300         tkArray = self.makeTargetArray(answer)
301         for out in range(self.output):
302             self.outUnitErr[num][out] = (tkArray[out] - self.ao[out])**2
303 
304     def calculateValErr(self,num,answer):
305 
306         tkArray = self.makeTargetArray(answer)
307         for out in range(self.output):
308             self.outValErr[num][out] = (tkArray[out] - self.ao[out])**2
309 
310     def calculateTstErr(self,num,answer):
311 
312         tkArray = self.makeTargetArray(answer)
313         for out in range(self.output):
314             self.outTstErr[num][out] = (tkArray[out] - self.ao[out])**2
315 
316     def plotErrorperNum(self):
317 
318         plt.imshow(self.outUnitErr[:,:])
319         plt.xlabel('iterations')
320         plt.ylabel('error')
321         plt.title('Sum of squared errors for each output unite')
322         plt.grid(True)
323         plt.savefig("test.png")
324         plt.show()
325 
```

```
326     def plotErrList(self,errTrn,errTst):
327         plt.figure()
328         plt.ylabel('error')
329         plt.xlabel('epochs')
330         ax = plt.subplot(111)
331         ax.plot(errTrn,label='Training Set Error')
332         ax.plot(errTst,label='Validation Set Error')
333 
334         plt.title('Training and Validation Set Errors')
335         ax.legend()
336         plt.savefig('pics/errPlots_' + self.expName + '.png')
337         plt.show()
338 
339     def plotAccList(self,accList):
340         plt.figure()
341         plt.ylabel('accuracy')
342         plt.xlabel('epochs')
343         ax = plt.subplot(111)
344         ax.plot(accList,label='Accuracy')
345 
346         plt.title('Image Matching to Validation Set Accuracy')
347         ax.legend()
348         plt.savefig('pics/accPlot_' + self.expName + '.png')
349         plt.show()
350 
351 def driver(dpath,inNodes,outNodes,hidNodes,epochs,trnNum,valNum,tstNum,lrnRate,momentum,stop
        ):
352 
353     # Read in the Training data first
354     dataset = ReadInFiles(dpath,'train')
355     my_data = ReadInOneList(dataset,trnNum)
356 
357     # Convert the 0-255 to 0 through 1 values in data
358     my_data[:,1:] /= 255.0
359     #HeatMap(my_data[40,1:])
360 
361     # randomize the rows for better training
362     np.random.shuffle(my_data)
363     inNum,cols = my_data.shape
364     just_img_data = my_data[:,1:]
365     answer = my_data[:,0]
366 
367     # Create the Validation data
368     my_valid = ReadInValidList(dataset,tstNum,tstNum+valNum)
369     my_valid[:,1:] /= 255.0
370     valNum,valCols = my_valid.shape
371     #print('val num is ',valNum)
```

```
372     just_valid_data = my_valid[:,1:]
373     answerValImg = my_valid[:,0]
374     #print('array of answerws to follow')
375     #print(answerValImg)
376 
377     # Read in the test data
378     #dpath2 = os.getcwd()+'\data3'
379     dataset2 = ReadInFiles(dpath,'test')
380     my_test = ReadInOneList(dataset2,tstNum)
381 
382     tstNum,cols = my_test.shape
383     #print('num rows ',tstNum)
384 
385     # Convert the 0-255 to 0 through 1 values in data
386     my_test[:,1:] /= 255.0
387 
388     just_test_data = my_test[:,1:]
389     answerImg = my_test[:,0]
390 
391     myNet = NeuralNet(inNodes, hidNodes, outNodes, inNum, valNum, tstNum, epochs,lrnRate,
            momentum,stop)
392     myNet.print_params()
393 
394     trnErrorList = []
395     trnValErrList = []
396     tstErrList = []
397     accList = []
398 
399     # Iterate over the number of epochs of data to run
400     for eps in range(myNet.epochs):
401         print("Training epoch ",eps)
402         # Iterate over the range of total images for training
403         for imgNum in range(inNum):
404             myNet.feedForward(just_img_data[imgNum,:],answer[imgNum])
405 
406             # Calculate the error term deltaKO for each output unit
407             deltaKO = myNet.calcDeltaKO(answer[imgNum])
408             #print('deltaKO is: ',deltaKO,' shape is ',deltaKO.shape)
409 
410             # Calculate the error term deltaKH for each hidden unit
411             deltaKH = myNet.calcDeltaKH(deltaKO)
412             #print ('delta h is ',deltaKH)
413 
414             # Update the weights for output and hidden weight sets
415             myNet.updateWeights(answer[imgNum],deltaKO,deltaKH)
416 
417             # Calculate the error per image per output unit
```

```
            myNet.calculateError(imgNum,answer[imgNum])

        # Output the training set error
        errPerEpoch = np.sum(myNet.outUnitErr,dtype='float')
        trnErrorList.append(errPerEpoch/(inNum*10.0))
        print("Total Training Error for epoch ",eps," is ",errPerEpoch/(inNum*10.0))

        accuracyList = []

        # Run this epochs trained model against the Validation Set of data
        for imgNum in range(valNum):

            myNet.feedForward(just_valid_data[imgNum,:],answerValImg[imgNum])

            valAnswer = myNet.ao.argmax(axis=0)
            #print('Val Answer is ',valAnswer, ' image answer is ',answerValImg[imgNum])
            if (valAnswer - answerValImg[imgNum] == 0):
                accuracyList.append(1)
            else:
                accuracyList.append(0)

            # Calculate the error for the validation images per output unit
            myNet.calculateValErr(imgNum,answerValImg[imgNum])

        # Output the Validation set error
        errValEpoch = np.sum(myNet.outValErr,dtype='float')
        trnValErrList.append(errValEpoch/(valNum*10.0)) # for the ten digits
        print("Total Validation Error for epoch ",eps," is ",errValEpoch/(valNum*10.0))

        # Output the Validation set accuracy
        right = sum(accuracyList)
        total = len(accuracyList)
        print('Results of ',right,' out of ',total,' accuracy is ',right/total)
        accList.append(right/total)

        # for every epoch iteration need to save off weights
        weights = [myNet.wHidThresh,myNet.wOutThresh,myNet.wi,myNet.wo]
        np.savez('output/weightEpoch_' + str(eps) + '_' + myNet.expName + '.npz', \
                                            wHidThresh=weights[0], \
                                            wOutThresh=weights[1], \
                                            wi=weights[2], \
                                            wo=weights[3])

        # Check for the stopping criteria on Validation Test Set
        criteriaMet = False
        if len(trnValErrList) > 1:
            currErr = trnValErrList[-1]
```

```
            prevErr = trnValErrList[-2]
            diffErrRatio = (prevErr - currErr) / prevErr
            print("{0:.4f}%".format(100.0 * diffErrRatio))
            if (diffErrRatio < myNet.stop):
                print("Stopping criteria of ",str(stop)," is more than ",str(diffErrRatio))
                criteriaMet = True

        if criteriaMet:
            break

    # Need to run the Test set of data
    # First find the optimal set of weights from Validation
    # Find the epoch with the lowest validation set error and then
    # set the weights in the ANN to those weights for testing
    # This should only not be the last set if there are very large stopping
    # criteria which would make the error start to go back up due to
    # overtraining for example
    npValErr = np.asarray(trnValErrList)
    minEpoch = npValErr.argmin(axis=0)
    print("The epoch with lowest validation error is ",str(minEpoch))
    optWt = np.load('output/weightEpoch_' + str(minEpoch) + '_' + myNet.expName + '.npz')
    myNet.wHidThresh = optWt['wHidThresh']
    myNet.wOutThresh = optWt['wOutThresh']
    myNet.wi = optWt['wi']
    myNet.wo = optWt['wo']
    optWt.close()
    #print('Weight arrays wHidThresh', myNet.wHidThresh,'\n' \
    #      'wOutThresh ',myNet.wOutThresh, '\n' \
    #      'shape of wi ',myNet.wi.shape, '\n' \
    #      'shape of wo ',myNet.wo.shape)

    # Then run the test images through using the optimal weights
    tstAccList = []
    for imgNum in range(tstNum):

        myNet.feedForward(just_test_data[imgNum,:],answerImg[imgNum])

        tstAnswer = myNet.ao.argmax(axis=0)
        #print('Val Answer is ',valAnswer, ' image answer is ',answerValImg[imgNum])
        if (tstAnswer - answerImg[imgNum] == 0):
            tstAccList.append(1)
        else:
            tstAccList.append(0)

    # Output the Test set error
    errTstEpoch = np.sum(myNet.outTstErr,dtype='float')
    tstErrList.append(errTstEpoch/(tstNum*10.0)) # for the ten digits
```

```
512     print("Final Testing Error is ",errTstEpoch/(tstNum*10.0))
513 
514     # Output the Validation set accuracy
515     right = sum(tstAccList)
516     total = len(tstAccList)
517     testAccuracy = right/total
518     print('Final Test results of ',right,' out of ',total,' accuracy is ',testAccuracy)
519 
520 
521     # Plot output and save plot data to file
522     myNet.plotErrList(trnErrorList,trnValErrList)
523     myNet.plotAccList(accList)
524 
525     np.savez('output/plotData_' + myNet.expName + '.npz', \
526                             trnErrorList=trnErrorList, \
527                             trnValErrList=trnValErrList, \
528                             accList=accList, \
529                             testAccuracy=testAccuracy)
530 
531 if __name__ == "__main__":
532 
533     # Parse commjand line options filename, epsilon, and maximum iterations
534     from optparse import OptionParser
535 
536     parser = OptionParser()
537     parser.add_option("-f", "--file", dest="filepath", help="Folder path for data")
538     parser.add_option("-i", "--hid", dest="hidNodes", help="Number of Hidden Nodes")
539     parser.add_option("-e", "--epochs", dest="epochs", help="Number of Epochs")
540     parser.add_option("-t", "--train", dest="trnNum", help="Number of Training Images per
            Number")
541     parser.add_option("-v", "--valid", dest="valNum", help="Number of Validation Images per
            Number")
542     parser.add_option("-x", "--test", dest="tstNum", help="Number of Test Images per Number
            ")
543     parser.add_option("-l", "--learn", dest="lrnRate", help="Number of Test Images per
            Number")
544     parser.add_option("-m", "--momentum", dest="momentum", help="Number of Test Images per
            Number")
545     parser.add_option("-s", "--stop", dest="stop", help="Validation Stopping Criteria
            Percentage")
546 
547 
548     options, args = parser.parse_args()
549 
550     if not options.filepath :
551         print("Used default of data" )
552         filepath = os.getcwd()+'\data'
```

```
553     else: filepath = options.filepath
554 
555     if not options.hidNodes :
556         print("Used default hidden nodes of 3" )
557         hidNodes = 3
558     else: hidNodes = int(options.hidNodes)
559 
560     if not options.epochs :
561         print("Used default epochs = 30" )
562         epochs = 30
563     else: epochs = int(options.epochs)
564 
565     if not options.trnNum :
566         print("Used default trnNum = 1000" )
567         trnNum = 4500
568     else: trnNum = int(options.trnNum)
569 
570     if not options.valNum :
571         print("Used default valNum = 500" )
572         valNum = 500
573     else: valNum = int(options.valNum)
574 
575     if not options.tstNum :
576         print("Used default tstNum = 500" )
577         tstNum = 890
578     else: tstNum = int(options.tstNum)
579 
580     if not options.lrnRate :
581         print("Used default lrnRate = 0.5" )
582         lrnRate = 0.5
583     else: lrnRate = float(options.lrnRate)
584 
585     if not options.momentum :
586         print("Used default momentum = 0.5" )
587         momentum = 0.5
588     else: momentum = float(options.momentum)
589 
590     if not options.stop :
591         print("Used default stop = 0.05%" )
592         stop = 0.0005
593     else: stop = float(options.stop)
594 
595     inNodes = 784
596     outNodes = 10
597 
598     driver(filepath,inNodes,outNodes,hidNodes,epochs,trnNum,valNum,tstNum,lrnRate,momentum,
        stop)
```