CS189–Spring 2016 — Homework 8Solutions

Andy Chu, SID 23042499, PUT SOMETHING HERE

1 Problem

$z2 \equiv X \cdot V^T$
$a2 \equiv tanh(z2)$
$z3 = a2 \cdot W^T$
sig is sigmoid function.
$\hat{y} \equiv sig(z3)$
$y \equiv y_k$

$J = \frac{1}{2}\Sigma_{k=1}^{n_{out}}(y - \hat{y})^2$

$\frac{\partial J}{\partial W} = (y - \hat{y}) \cdot \frac{\partial \hat{y}}{\partial W}$ (ignore summation for now)
$\frac{\partial J}{\partial W} = (y - \hat{y}) \cdot \frac{\partial \hat{y}}{\partial z3}\frac{\partial z3}{\partial W}$
$\frac{\partial J}{\partial W} = \Sigma_{k=1}^{n_{out}}(y - \hat{y})(sig(z3)(1 - sig(z3))a2$

$J = \frac{1}{2}\Sigma_{k=1}^{n_{out}}(y - \hat{y})^2$

$\frac{\partial J}{\partial V} = (y - \hat{y}) \cdot \frac{\partial \hat{y}}{\partial V}$ (ignore summation for now)
$\frac{\partial J}{\partial V} = \Sigma_{k=1}^{n_{out}}(y - \hat{y}) \cdot \frac{\partial \hat{y}}{\partial z3}\frac{\partial z3}{\partial a2}\frac{\partial a2}{\partial z2}\frac{\partial z2}{\partial V}$
$\frac{\partial J}{\partial V} = \Sigma_{k=1}^{n_{out}}(y - \hat{y}) \cdot sig(z3)(1 - sig(z3))W(sech(z3))X$

$J = -\Sigma_{k=1}^{n_{out}}(yln\hat{y} + (1 - y)ln(1 - \hat{y}))$

$J = -(yln\hat{y} + (1 - y)ln(1 - \hat{y}))$ (ignore summation for now)

$\frac{\partial J}{\partial W} = -(\frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z3}\frac{\partial z3}{\partial W} + \frac{1 - y}{1 - \hat{y}} \cdot -\frac{\partial \hat{y}}{\partial z3}\frac{\partial z3}{\partial W})$
$\frac{\partial J}{\partial W} = -(\frac{y}{\hat{y}} \cdot \dot{sig}(z3) \cdot a2 + \frac{1 - y}{1 - \hat{y}} \cdot (-\dot{sig}(z3) \cdot a2))$
$\frac{\partial J}{\partial W} = -(\frac{y}{\hat{y}} \cdot (1 - sig(z3)) \cdot sig(z3) \cdot a2 + \frac{1 - y}{1 - \hat{y}} \cdot -((1 - sig(z3)) \cdot sig(z3) \cdot a2))$
$\frac{\partial J}{\partial W} = -(y \cdot (1 - sig(z3)) \cdot a2 + -(1 - y) \cdot sig(z3) \cdot a2))$
$\frac{\partial J}{\partial W} = -((y - sig(z3)) \cdot a2)$
$\frac{\partial J}{\partial W} = \Sigma_{k=1}^{n_{out}}(-y + \hat{y}) \cdot a2$
$J = -\Sigma_{k=1}^{n_{out}}(yln\hat{y} + (1 - y)ln(1 - \hat{y}))$

$J = -(yln\hat{y} + (1 - y)ln(1 - \hat{y}))$ (ignore summation for now)

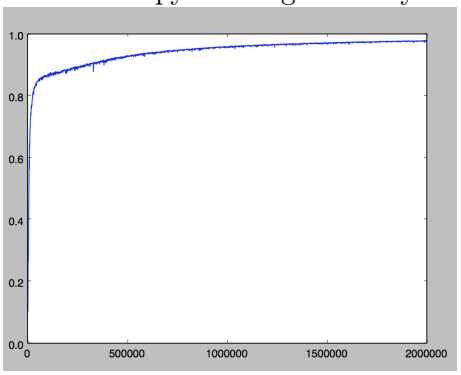
$\frac{\partial J}{\partial V} = -(\frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z3}\frac{\partial z3}{\partial a2}\frac{\partial a2}{\partial z2}\frac{\partial z2}{\partial V} + \frac{1 - y}{1 - \hat{y}} \cdot -\frac{\partial \hat{y}}{\partial z3}\frac{\partial z3}{\partial a2}\frac{\partial a2}{\partial z2}\frac{\partial z2}{\partial V})$

$$\frac{\partial J}{\partial V} = -(\frac{y}{\hat{y}} \cdot sig(z3)(1 - sig(z3))W(sech(z3))X + \frac{1-y}{1-\hat{y}} \cdot (-sig(z3)(1 - sig(z3))W(sech(z3)X))$$
$$\frac{\partial J}{\partial V} = -(y \cdot (1 - sig(z3))W(sech(z3)X + (1-y) \cdot (-sig(z3))W(sech(z3))X)$$
$$\frac{\partial J}{\partial V} = \Sigma_{k=1}^{n_{out}}(\hat{y} - y)sig(z3))W(sech(z3)X)$$

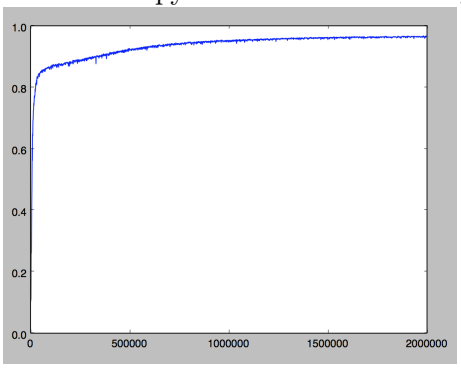
2 Problem

The learning rate i used is 0.008. The stopping criterion is to stop after 2 million iterations of stochastic gradient descent. The weights were generated from a random normal distribution in python. Training accuracy is 0.97834 and validation accuracy is 0.967 for cross entropy loss . Training accuracy is 0.9046 and validation accuracy is 0.8934 for mean square loss . Running time is 4 hours for each loss.

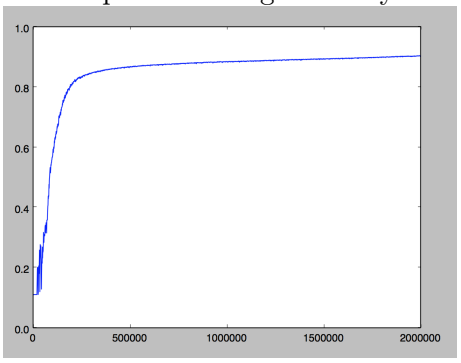
Cross Entropy training accuracy iteration vs. accuracy plot.



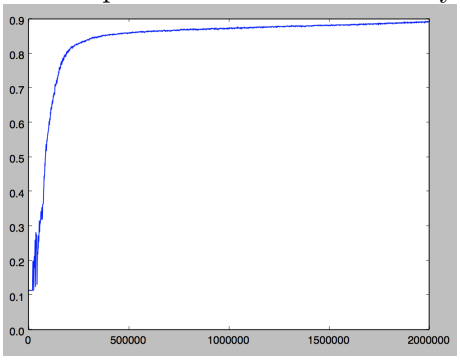
Cross Entropy classification accuracy iteration vs. accuracy plot.



Mean square training accuracy iteration vs. accuracy plot.



Mean square classification accuracy iteration vs. accuracy plot.



The cross entropy loss function performs better than the mean square loss function. Kaggle score with cross entropy loss function is 0.96880.

3 Appendix

n.py runs crossEntropy

```
import scipy.io as sio
import random
import numpy as np
from sklearn.preprocessing import normalize
import matplotlib.pyplot as plt
import math

DEBUG = 0



train_contents = sio.loadmat('train.mat')
X = train_contents['train_images']
Y = train_contents['train_labels']
indice = np.arange(60000)
np.random.shuffle(indice)
train_num = 50000
valid_num = 10000
# save the shuffled indice, so can compare if improve or not
training = indice[0:train_num]   # first 50,000 to train
indice = indice[train_num:]      #truncating list
train_images = X[:,:,training]
train_label = Y[training, :]
validation = indice[0:valid_num]
valid_images = X[:,:,validation]
valid_label = Y[validation, :]
pixel_images = train_images[0,:,:]
validation_images = valid_images[0,:,:]

#if (DEBUG):
  #print "before reshape"
  #print "train_images", train_images.shape, "train_label", train_label.shape, "
    validation", valid_images.shape


for i in range(1, 28):

    pixel_images = np.vstack([pixel_images, train_images[i,:,:]])
    validation_images = np.vstack([validation_images, valid_images[i,:,:]])
pixel_images = pixel_images.T
validation_images = validation_images.T

if (DEBUG):
  print "validation_images", validation_images.shape, "pixel_image", pixel_images.
    shape, "train_label", train_label.shape, "valid_label", valid_label.shape

#normalize the data and test
validation_images = normalize(validation_images)
pixel_images = normalize(pixel_images)
if (DEBUG):
  print "mean validation_images", np.mean(validation_images), "mean pixel_images",
    np.mean(pixel_images)
  print "now add a column or row of ones to validation", validation_images.shape, "
    pixel_images", pixel_images.shape
```

```
51
52 #stack = np.array([np.ones(784), 1])
53 stack = np.ones([train_num, 1])
54 stackv = np.ones([valid_num,1])
55
56 if (DEBUG):
57   print "shape before stack", validation_images.shape, pixel_images.shape
58   print stack.shape, "stack shape", stackv.shape
59 validation_images = np.hstack((validation_images, stackv))
60 pixel_images = np.hstack((pixel_images, stack))
61
62 if (DEBUG):
63   print "validation shape after stac", validation_images.shape, pixel_images, "check
       if have column of 1's", pixel_images[:,784], validation_images[:,784]
64
65
66 class Neural_Network(object):
67   def __init__(self):
68     #Define HyperParameters
69     self.inputLayerSize = 785
70     self.outputLayerSize = 10
71     self.hiddenLayerSize = 200
72
73     #Weights (parameters)
74
75     self.W1 = np.random.normal(0, 0.01, (self.inputLayerSize, \
76                 self.hiddenLayerSize))
77     self.W2 = np.random.normal(0, 0.01, (self.hiddenLayerSize + 1, \
78                 self.outputLayerSize))
79
80     #load weights to continue
81     # self.W1 = np.loadtxt("W1900000")
82     # self.W2 = np.loadtxt("W2900000")
83
84   def sigmoid(self, z):
85     #Apply sigmoid activation function
86     return 1/(1+np.exp(-z))
87
88
89   def forward(self, X):
90     #Propogate inputs through network
91     D = 0
92     if (D):
93       print "X", X.shape, "W1", self.W1.shape
94     self.z2 = np.dot(X, self.W1)
95     self.a2 = np.tanh(self.z2)
96     if (D):
97       print "a2", self.a2.shape, "W2", self.W2.shape
98     self.a2 = np.hstack([self.a2, np.ones([self.a2.shape[0], 1])])
99     if (D):
100      print "a2 after stacking", self.a2.shape
101    self.z3 = np.dot(self.a2, self.W2)
102
103    if (D):
104      print "z3", self.z3.shape, self.z3
105    yHat = self.sigmoid(self.z3)
106    if (DEBUG):
107      print "yHat", yHat.shape, "X", X.shape
108    return yHat
```

```
109
110
111
112 #test sigmoid
113 #testInput = np.arange(-6,6,0.01)
114 #plt.plot(testInput, sigmoid(testInput), linewidth=2)
115 #plt.show()
116
117   def sigmoidPrime(self, z):
118     #Derivative of Sigmoid Function
119     return np.exp(-z)/((1+np.exp(-z))**2)
120
121 #test sigmoidPrime
122 #testValues = np.arange(-5, 5, 0.01)
123 #plt.plot(testValues, sigmoidPrime(testValues), linewidth=2)
124 #plt.show()
125
126   def costFunctionPrime(self, X, y):
127     #compute derivative with respect to W1 and W2
128     #for mean loss
129     # self.yHat = self.forward(X)
130     # yVec = np.zeros(self.yHat.shape)
131     # yVec[0,y] = 1
132
133
134     # delta3 = np.multiply(-(yVec-self.yHat), self.sigmoidPrime(self.z3))
135     # dJdW2 = np.dot(self.a2.T, delta3)
136
137     # #delta2 = np.dot(delta3, self.W2.T)*self.sigmoidPrime(self.z2)
138     # if (DEBUG):
139     #   print "delta3", delta3.shape, "W2.T", self.W2.T[:,:200].shape
140     # delta2 = np.dot(delta3, self.W2.T[:, :200])*(1 - np.tanh(self.z2)**2)
141     # dJdW1 = np.dot(X.T, delta2)
142     # return dJdW1, dJdW2
143
144
145     # # # #for cross entropy
146     self.yHat = self.forward(X)
147     yVec = np.zeros(self.yHat.shape)
148     yVec[0,y] = 1
149     #print "yHat ", self.yHat, "yVec", yVec
150     dJdW2 = np.multiply(-(yVec - self.yHat).T, self.a2)
151     #print dJdW2.shape
152
153
154     delta2 = np.dot(-(yVec - self.yHat), self.W2.T[:, :200])*(1 - np.tanh(self.z2)
    **2)
155     dJdW1 = np.dot(X.T, delta2)
156     return dJdW1, dJdW2.T
157
158
159
160     def train(self, train, label, valid, vlabel):
161       i = 0
162       #i = 1000000 #because loading this
163       #stop = 1001
164       #stop = 100
165
166       #stop = 50000
```

```
167       #stop = 1000
168       stop = 2000000
169       alpha = 0.008
170       trainPlot = []
171       classPlot = []
172       x = np.arange(0, stop +1000, 1000)
173       while (i < stop):
174         #data = np.array(random.choice(train))
175         r = np.random.randint(50000)
176         if (DEBUG):
177           print "train", train.shape, "label", label.shape
178         dtrain = np.array([train[r,:]])
179         dlabel = np.array([label[r, :]])
180         if (DEBUG):
181           print "dtrain", dtrain.shape, "dlabel", dlabel.shape
182         dJdW1, dJdW2 = self.costFunctionPrime(dtrain, dlabel)
183         #dJdW1, dJdW2 = self.costFunctionPrime(data, label)
184       #print self.W1, self.W2, "before"
185       self.W1 = self.W1 - alpha*dJdW1
186       self.W2 = self.W2 - alpha*dJdW2
187       #print self.W1, self.W2, "after"
188       if ((i%1000) == 0):
189         # if ((i%100000)==0):
190           # np.savetxt("W1" + str(i), self.W1)
191           # np.savetxt("W2" + str(i), self.W2)
192           # print "write", i
193         trainError = self.error(train, label)
194         classError = self.error(valid, vlabel)
195         trainPlot.append(trainError)
196         classPlot.append(classError)
197         print  i, "trainAccuracy", trainError, "classAccuracy", classError
198 
199       i+=1
200     print "display trainPlot"
201     #print "x", x, "trainPlot", trainPlot, "classPlot", classPlot
202     plt.plot(x, trainPlot, 'ro')
203     plt.show()
204     print "display classPlot"
205     plt.plot(x, classPlot, 'ro')
206     plt.show()
207     #print "training complete"
208 
209   def error(self, train, label):
210     result = self.forward(train)
211     #pred = np.array((np.argmax(result, axis=1)))
212     error = 0
213 
214     for i in range(label.shape[0]):
215       #print np.where(result[i,:] == max(result[i,:]))[0], result[i,:], label.shape
    [0]
216       p = np.where(result[i,:] == max(result[i,:]))[0]
217       if (len(p) > 1):
218         p = p[0]
219       if (p != label[i][0]):
220         error += 1
221       #if (DEBUG):
222       #print "p is ", p, "label is ", label[i][0], "p not same as label", p != label
    [i][0]
223     #if (DEBUG):
```

```
224     return 1 - error/float(label.shape[0])
225 
226   def predict(self, test):
227     self.W1 = np.loadtxt("W11900000")
228     self.W2 = np.loadtxt("W21900000")
229     result = self.forward(test)
230     #pred = np.array((np.argmax(result, axis=1)))
231 
232     pred = []
233 
234     #print "result", result.shape, "result[i, :]", result[0,:], "argmax", np.argmax(
    result[0,:])
235     #print self.a2.shape, self.a2[:,200], "self.a2"
236     for i in range(test.shape[0]):
237       #print np.where(result[i,:] == max(result[i,:]))[0], result[i,:], label.shape
    [0]
238       p = np.where(result[i,:] == max(result[i,:]))[0]
239       if (len(p) > 1):
240         p = p[0]
241       pred.append(p)
242     f = open("digit.csv", 'w')
243     f.write('Id,Category\n')
244     for i in range(len(pred)):
245       label = pred[i][0]
246       image_id = i + 1
247       f.write(str(image_id) + ',' + str(label) + '\n')
248     f.close()
249 
250 
251 
252 
253 
254 
255 
256 
257 nn = Neural_Network()
258 nn.train(pixel_images, train_label, validation_images, valid_label)
259 
260 #nn.predict(validation_images, valid_label)
261 #change to test data and write to a csv file
262 #load the weights from files
263 
264 #test error, by loading W1 and W2
265 
266 # nn = Neural_Network()
267 # nn.W1 = np.loadtxt("W11900000")
268 # nn.W2 = np.loadtxt("W21900000")
269 
270 
271 # #training accuracy
272 # print "training accuracy"
273 # nn.error(pixel_images, train_label)
274 # print "validation accuracy"
275 # nn.error(validation_images, valid_label)
276 
277 #validation accuracy
278 
279 
280 
```

```
281
282 # nn.error(pixel_images, train_label)
283 # train_contents = sio.loadmat('test.mat')
284 # test = train_contents['test_images']
285 # print "shape train", test.shape
286 # #test_images = test[0,:,:]
287 # test_images = test[:,0,:]
288 # for i in range(1, 28):
289
290 #     #test_images = np.vstack([test_images, test[i,:,:]])
291 #     #test_images = np.vstack([test_images, test[:,i,:]])
292 #     test_images = np.hstack([test_images, test[:,i,:]])
293 # #print "reshaped to", test_images.shape
294
295 # #normalize test
296 # #then stack ones
297 # test_images = normalize(test_images)
298 # stack = np.ones([test.shape[0],1])
299 # test_images = np.hstack((test_images, stack))
300 # print "stack ones", test_images.shape, "see if have ones", test_images[:,784]
301 # nn.predict(test_images)
302
303 #nn.train(pixel_images, train_label, validation_images, valid_label)
```

mean.py runs mean square loss

```
1  import scipy.io as sio
2  import random
3  import numpy as np
4  from sklearn.preprocessing import normalize
5  import matplotlib.pyplot as plt
6  import math
7
8  DEBUG = 0
9
10
11
12 train_contents = sio.loadmat('train.mat')
13 X = train_contents['train_images']
14 Y = train_contents['train_labels']
15 indice = np.arange(60000)
16 np.random.shuffle(indice)
17 train_num = 50000
18 valid_num = 10000
19 # save the shuffled indice, so can compare if improve or not
20 training = indice[0:train_num]   # first 50,000 to train
21 indice = indice[train_num:]      #truncating list
22 train_images = X[:,:,training]
23 train_label = Y[training, :]
24 validation = indice[0:valid_num]
25 valid_images = X[:,:,validation]
26 valid_label = Y[validation, :]
27 pixel_images = train_images[0,:,:]
28 validation_images = valid_images[0,:,:]
29
30 #if (DEBUG):
31   #print "before reshape"
```

```
 #print "train_images", train_images.shape, "train_label", train_label.shape, "
    validation", valid_images.shape


for i in range(1, 28):

    pixel_images = np.vstack([pixel_images, train_images[i,:,:]])
    validation_images = np.vstack([validation_images, valid_images[i,:,:]])
pixel_images = pixel_images.T
validation_images = validation_images.T

if (DEBUG):
  print "validation_images", validation_images.shape, "pixel_image", pixel_images.
    shape, "train_label", train_label.shape, "valid_label", valid_label.shape

#normalize the data and test
validation_images = normalize(validation_images)
pixel_images = normalize(pixel_images)
if (DEBUG):
  print "mean validation_images", np.mean(validation_images), "mean pixel_images",
    np.mean(pixel_images)
  print "now add a column or row of ones to validation", validation_images.shape, "
    pixel_images", pixel_images.shape

#stack = np.array([np.ones(784), 1])
stack = np.ones([train_num, 1])
stackv = np.ones([valid_num,1])

if (DEBUG):
  print "shape before stack", validation_images.shape, pixel_images.shape
  print stack.shape, "stack shape", stackv.shape
validation_images = np.hstack((validation_images, stackv))
pixel_images = np.hstack((pixel_images, stack))

if (DEBUG):
  print "validation shape after stac", validation_images.shape, pixel_images, "check
     if have column of 1's", pixel_images[:,784], validation_images[:,784]


class Neural_Network(object):
  def __init__(self):
    #Define HyperParameters
    self.inputLayerSize = 785
    self.outputLayerSize = 10
    self.hiddenLayerSize = 200

    #Weights (parameters)

    self.W1 = np.random.normal(0, 0.01, (self.inputLayerSize, \
                self.hiddenLayerSize))
    self.W2 = np.random.normal(0, 0.01, (self.hiddenLayerSize + 1, \
                self.outputLayerSize))

    #load weights to continue
    # self.W1 = np.loadtxt("W1900000")
    # self.W2 = np.loadtxt("W2900000")

  def sigmoid(self, z):
    #Apply sigmoid activation function
```

```
    return 1/(1+np.exp(-z))


  def forward(self, X):
    #Propogate inputs through network
    D = 0
    if (D):
      print "X", X.shape, "W1", self.W1.shape
    self.z2 = np.dot(X, self.W1)
    self.a2 = np.tanh(self.z2)
    if (D):
      print "a2", self.a2.shape, "W2", self.W2.shape
    self.a2 = np.hstack([self.a2, np.ones([self.a2.shape[0], 1])])
    if (D):
      print "a2 after stacking", self.a2.shape
    self.z3 = np.dot(self.a2, self.W2)

    if (D):
      print "z3", self.z3.shape, self.z3
    yHat = self.sigmoid(self.z3)
    if (DEBUG):
      print "yHat", yHat.shape, "X", X.shape
    return yHat



#test sigmoid
#testInput = np.arange(-6,6,0.01)
#plt.plot(testInput, sigmoid(testInput), linewidth=2)
#plt.show()

  def sigmoidPrime(self, z):
    #Derivative of Sigmoid Function
    return np.exp(-z)/((1+np.exp(-z))**2)

#test sigmoidPrime
#testValues = np.arange(-5, 5, 0.01)
#plt.plot(testValues, sigmoidPrime(testValues), linewidth=2)
#plt.show()

  def costFunctionPrime(self, X, y):
    #compute derivative with respect to W1 and W2
    #for mean loss
    self.yHat = self.forward(X)
    yVec = np.zeros(self.yHat.shape)
    yVec[0,y] = 1


    delta3 = np.multiply(-(yVec-self.yHat), self.sigmoidPrime(self.z3))
    dJdW2 = np.dot(self.a2.T, delta3)

    #delta2 = np.dot(delta3, self.W2.T)*self.sigmoidPrime(self.z2)
    if (DEBUG):
      print "delta3", delta3.shape, "W2.T", self.W2.T[:,:200].shape
    delta2 = np.dot(delta3, self.W2.T[:, :200])*(1 - np.tanh(self.z2)**2)
    dJdW1 = np.dot(X.T, delta2)
    return dJdW1, dJdW2


```

```
    ####for cross entropy
    # self.yHat = self.forward(X)
    # yVec = np.zeros(self.yHat.shape)
    # yVec[0,y] = 1
    # #print "yHat ", self.yHat, "yVec", yVec
    # dJdW2 = np.multiply(-(yVec - self.yHat).T, self.a2)
    # #print dJdW2.shape


    # delta2 = np.dot(-(yVec - self.yHat), self.W2.T[:, :200])*(1 - np.tanh(self.z2)
    **2)
    # dJdW1 = np.dot(X.T, delta2)
    # return dJdW1, dJdW2.T



    def train(self, train, label, valid, vlabel):
      i = 0
      #i = 1000000 #because loading this
      #stop = 1001
      #stop = 100

      #stop = 50000
      #stop = 1000
      stop = 2000000
      alpha = 0.008
      trainPlot = []
      classPlot = []
      x = np.arange(0, stop +1000, 1000)
      while (i < stop):
        #data = np.array(random.choice(train))
        r = np.random.randint(50000)
        if (DEBUG):
          print "train", train.shape, "label", label.shape
        dtrain = np.array([train[r,:]])
        dlabel = np.array([label[r, :]])
        if (DEBUG):
          print "dtrain", dtrain.shape, "dlabel", dlabel.shape
        dJdW1, dJdW2 = self.costFunctionPrime(dtrain, dlabel)
        #dJdW1, dJdW2 = self.costFunctionPrime(data, label)
      #print self.W1, self.W2, "before"
      self.W1 = self.W1 - alpha*dJdW1
      self.W2 = self.W2 - alpha*dJdW2
      #print self.W1, self.W2, "after"
      if ((i%1000) == 0):
        # if ((i%100000)==0):
          # np.savetxt("W1" + str(i), self.W1)
          # np.savetxt("W2" + str(i), self.W2)
          # print "write", i
        trainError = self.error(train, label)
        classError = self.error(valid, vlabel)
        trainPlot.append(trainError)
        classPlot.append(classError)
        print  i, "trainAccuracy", trainError, "classAccuracy", classError

      i+=1
    print "display trainPlot"
    #print "x", x, "trainPlot", trainPlot, "classPlot", classPlot
    plt.plot(x, trainPlot, 'ro')
```

```
    plt.show()
    print "display classPlot"
    plt.plot(x, classPlot, 'ro')
    plt.show()
    #print "training complete"

  def error(self, train, label):
    result = self.forward(train)
    #pred = np.array((np.argmax(result, axis=1)))
    error = 0

    for i in range(label.shape[0]):
      #print np.where(result[i,:] == max(result[i,:]))[0], result[i,:], label.shape
    [0]
      p = np.where(result[i,:] == max(result[i,:]))[0]
      if (len(p) > 1):
        p = p[0]
      if (p != label[i][0]):
        error += 1
      #if (DEBUG):
      #print "p is ", p, "label is ", label[i][0], "p not same as label", p != label
    [i][0]
    #if (DEBUG):
    return 1 - error/float(label.shape[0])

  def predict(self, test):
    self.W1 = np.loadtxt("W11900000")
    self.W2 = np.loadtxt("W21900000")
    result = self.forward(test)
    #pred = np.array((np.argmax(result, axis=1)))

    pred = []

    #print "result", result.shape, "result[i, :]", result[0,:], "argmax", np.argmax(
    result[0,:])
    #print self.a2.shape, self.a2[:,200], "self.a2"
    for i in range(test.shape[0]):
      #print np.where(result[i,:] == max(result[i,:]))[0], result[i,:], label.shape
    [0]
      p = np.where(result[i,:] == max(result[i,:]))[0]
      if (len(p) > 1):
        p = p[0]
      pred.append(p)
    f = open("digit.csv", 'w')
    f.write('Id,Category\n')
    for i in range(len(pred)):
      label = pred[i][0]
      image_id = i + 1
      f.write(str(image_id) + ',' + str(label) + '\n')
    f.close()









nn = Neural_Network()
```

```
nn.train(pixel_images, train_label, validation_images, valid_label)

#nn.predict(validation_images, valid_label)
#change to test data and write to a csv file
#load the weights from files

#test error, by loading W1 and W2

# nn = Neural_Network()
# nn.W1 = np.loadtxt("W11900000")
# nn.W2 = np.loadtxt("W21900000")


# #training accuracy
# print "training accuracy"
# nn.error(pixel_images, train_label)
# print "validation accuracy"
# nn.error(validation_images, valid_label)

#validation accuracy




# nn.error(pixel_images, train_label)
# train_contents = sio.loadmat('test.mat')
# test = train_contents['test_images']
# print "shape train", test.shape
# #test_images = test[0,:,:]
# test_images = test[:,0,:]
# for i in range(1, 28):

#     #test_images = np.vstack([test_images, test[i,:,:]])
#     #test_images = np.vstack([test_images, test[:,i,:]])
#     test_images = np.hstack([test_images, test[:,i,:]])
# #print "reshaped to", test_images.shape

# #normalize test
# #then stack ones
# test_images = normalize(test_images)
# stack = np.ones([test.shape[0],1])
# test_images = np.hstack((test_images, stack))
# print "stack ones", test_images.shape, "see if have ones", test_images[:,784]
# nn.predict(test_images)

#nn.train(pixel_images, train_label, validation_images, valid_label)
```