A2

February 27, 2017

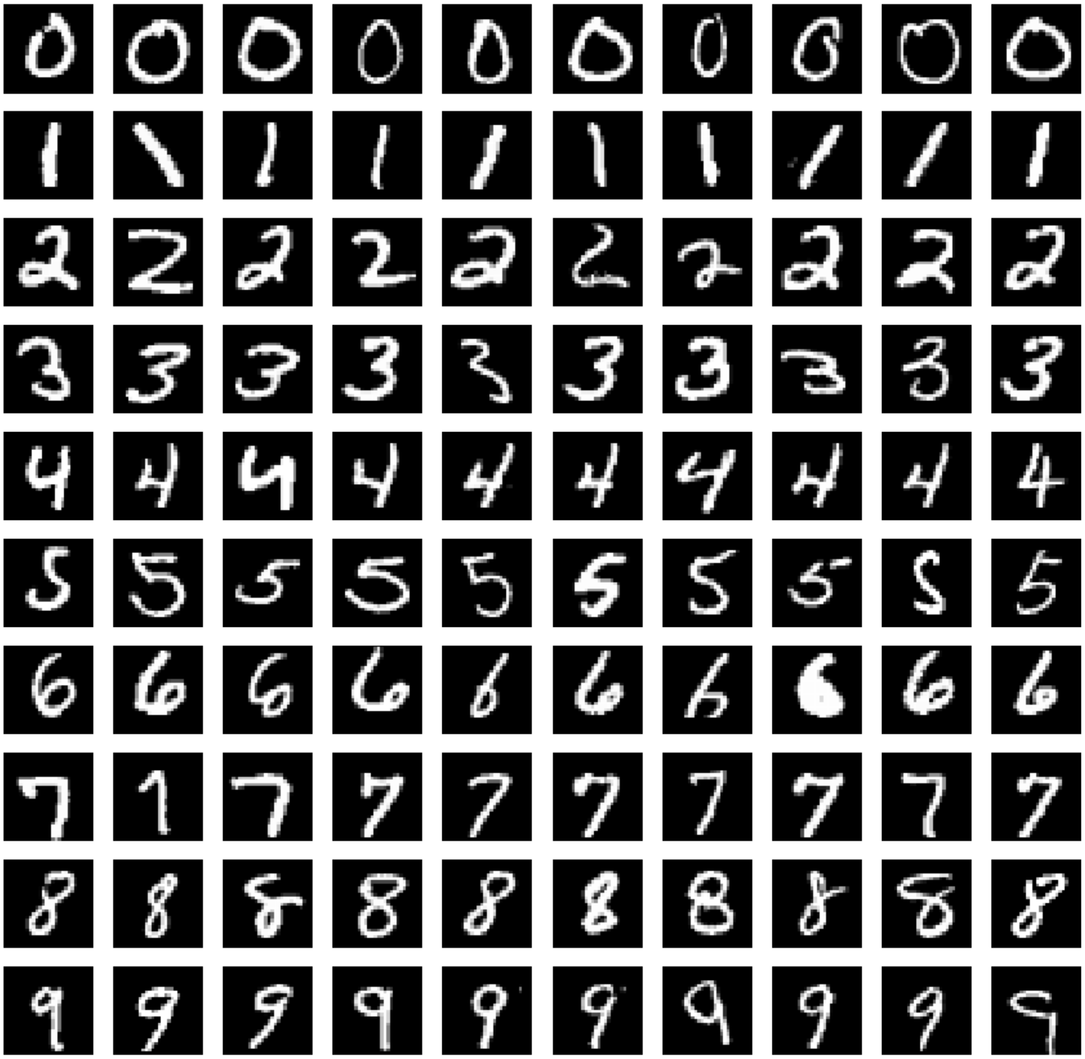
```
In [10]: %matplotlib inline
         from pylab import *
         import numpy as np
         import matplotlib.pyplot as plt
         import matplotlib.cbook as cbook
         import time
         from scipy.misc import imread
         from scipy.misc import imresize
         import matplotlib.image as mpimg
         from scipy.ndimage import filters
         import urllib
         from numpy import random

         import cPickle

         import os
         from scipy.io import loadmat

         #Load the MNIST digit data
         M = loadmat("mnist_all.mat")

         #Display the 150-th "5" digit from the training set
         #imshow(M["train5"][150].reshape((28,28)), cmap=cm.gray)
         #show()


         def softmax(y):
             '''Return the output of the softmax function for the matrix of output
             is an NxM matrix where N is the number of outputs for a single case, a
             is the number of cases'''
             return exp(y)/tile(sum(exp(y),0), (len(y),1))

         def tanh_layer(y, W, b):
             '''Return the output of a tanh layer for the input matrix y. y
             is an NxM matrix where N is the number of inputs for a single case, an
             is the number of cases'''
             return tanh(dot(W.T, y)+b)
```

```
def forward(x, W0, b0, W1, b1):
    L0 = tanh_layer(x, W0, b0)
    L1 = dot(W1.T, L0) + b1
    output = softmax(L1)
    return L0, L1, output

def cross_entropy(y, y_):
    return -sum(y_*log(y))

def deriv_multilayer(W0, b0, W1, b1, x, L0, L1, y, y_):
    '''Incomplete function for computing the gradient of the cross-entropy
    cost function w.r.t the parameters of a neural network - NOT WORKING'
    dCdL1 =  y - y_
    dCdW1 =  dot(L0, dCdL1.T )
    dCdobydodh = dot(W1, dCdL1)
    one_minus_h_sq = 1-L0**2

    dCdW0 = tile(dCdobydodh, 28*28).T * dot(x, (one_minus_h_sq.T))
    dCdb1 = dCdL1
    dCdb0 = dCdobydodh * one_minus_h_sq

    return dCdW1, dCdb1, dCdW0, dCdb0
```

0.1 Part 1 - Dataset Description

- variety of angles
- different styles of handwriting
- gaps between continuous lines
- different thickness levels

```
In [2]: f, sub = plt.subplots(10, 10,figsize=(15,15))
        for i in range(10):
            for j in range(10):
                sub[i][j].axis('off')
                sub[i][j].imshow(M['train' + str(i)][j+20].reshape((28,28)), cmap=
```



```
In [14]: #Load sample weights for the multilayer neural network
         f = open("snapshot50.pkl","rb")
         snapshot = cPickle.load(f)
         W0 = snapshot["W0"]
         b0 = snapshot["b0"].reshape((300,1))
         W1 = snapshot["W1"]
         b1 = snapshot["b1"].reshape((10,1))

         #Load one example from the training set, and run it through the
         #neural network
         x = M["train5"][148:149].T
         L0, L1, output = forward(x, W0, b0, W1, b1)
         #get the index at which the output is the largest
         y = argmax(output)
```

```
    print(y)

    y_true = array([[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]]).T
    print(deriv_multilayer(x, L0, L1, output, y_true))
    ########################################################################
    #Code for displaying a feature from the weight matrix mW
    #fig = figure(1)
    #ax = fig.gca()
    #heatmap = ax.imshow(mW[:,50].reshape((28,28)), cmap = cm.coolwarm)
    #fig.colorbar(heatmap, shrink = 0.5, aspect=5)
    #show()
    ########################################################################
```

```
6
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-14-3ce848074a24> in <module>()
         16
         17 y_true = array([[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]]).T
    ---> 18 print(deriv_multilayer(x, L0, L1, output, y_true))
         19 ########################################################################
         20 #Code for displaying a feature from the weight matrix mW


        TypeError: deriv_multilayer() takes exactly 9 arguments (5 given)
```

0.2 Part 2 - network function

```
In [4]: def network_function(x,theta):
            return dot(x,theta)
```

0.3 Part 3 - negative log-probabilities of all the training cases as the cost function

cost function $\sum_j y_j * log(p_j)$ refer to the slides for the gradient