Assignment One Analysis (Stochastic Gradient Descent)

by Haoyu Wang (haoyuw2)

Analysis:

By implementing the Stochastic Gradient Descent method, I train the logistic regression model to have 90.37% accuracy hitting the correct y_test data. To implement the model, I follow the steps proposed in the course note. First, I define a variable called *ite* to follow the iterations in the training process and a while loop for *max_ite* iterations to train a given random generated model *model['W1']*. During the training process, I create a random integer number every iteration and derive a randomly selected *x_rand* and *y_rand* data from *x_train* and *y_train*. By utilizing a self-defined function softMax, I put the soft max version *x_rand* into gradient functions (2.19 and 2.20) in the note, which results the deepest gradient descent for the current *x_rand.* Subtracting the multiplication of learning rate and the deepest gradient descent from *model['W1']*, one iteration is finished. Learning rate is defined as 0.0087 and max_ite is defined as 17500 by trial and error.

Code:

Attached in the next page.

Assignment_1_deep_learning

January 26, 2019

In [220]:
```
import numpy as np
import h5py
import time
import copy
from random import randint
import random
#load MNIST data
MNIST_data = h5py.File('data.hdf5', 'r')
x_train = np.float32(MNIST_data['x_train'][:] )
y_train = np.int32(np.array(MNIST_data['y_train'][:,0]))
x_test = np.float32( MNIST_data['x_test'][:] )
y_test = np.int32( np.array( MNIST_data['y_test'][:,0] ) )
MNIST_data.close()
################################################################################


#define softmax for f(x;theta)
def softMax(x):
    return np.exp(x)/np.sum(np.exp(x),axis =0)

#Implementation of stochastic gradient descent algorithm
#number of inputs
num_inputs = 28*28
#number of outputs
num_outputs = 10
model = {}
#model 'W1' is theta
model['W1'] = np.random.randn(num_outputs,num_inputs) / np.sqrt(num_inputs)
model_grads = copy.deepcopy(model)

################################start the stochastic gradient descent process##########
#iterate times
ite = 0
#define learning_rate to be 0.0085
learning_rate = 0.0087
max_ite = 17500
```

```
while(ite < max_ite):
    #pick a random point
    rand_num = random.randint(0,len(x_train)-1)
    x_rand = x_train[rand_num]
    y_rand = y_train[rand_num]

    #tranform into softmax version with theta @ x_rand
    soft_x = softmax(model['W1']@x_rand)

    #Initialize the Gradient and calculate the gradient
    G_grad = np.zeros(10*784).reshape(10,784)

    for k in range(10):
        #when k=y
        if(k == y_rand):
            G_grad[k] = [-(1-soft_x[k])*x_val for x_val in x_rand]
        else:
            G_grad[k] = [-(-soft_x[k])*x_val for x_val in x_rand]

    #get the next theta
    x = np.subtract(model['W1'] , learning_rate*G_grad)
    #increment the counter
    ite +=1

def forward(x, y, model):
    return model['W1']@x

#########################################################
#test data
total_correct = 0
for n in range(len(x_test)):
    y = y_test[n]
    x = x_test[n][:]
    p = forward(x, y, model)
    prediction = np.argmax(p)
    if (prediction == y):
        total_correct += 1
print(total_correct/np.float(len(x_test) ) )
```

0.9037