CS398 HW2: Neural Network

Yanjun Guo (yanjung2)

In this assignment, I used stochastic gradient descent to train a single layer neural network model and reached 97.65% accuracy. Firstly, I generated W, b1, b2, C using randn() function. Then I obtained training data randomly by generating random integers (dimension of hidden layer = 500). I chose ReLU for σ, and constructed distribution function with the formula from lecture, and upgrade new parameters each time I iterate. I used a learning rate of 0.005 because it is the most accurate under lower iterations. After 500000 iterations, I finally got 97.65% accuracy for test data.

(Code next page)

In [2]:

```
import numpy as np

import h5py
import time
import copy
import random
from random import randint


#load MNIST data
MNIST_data = h5py.File('MNISTdata.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()

################################################################################
####
#Implementation of stochastic gradient descent algorithm
def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis=0)
def relu(x):
    return x * (x > 0)
def reluDerivative(x):
    x_new = np.copy(x)
    x_new[x_new <= 0] = 0
    x_new[x_new > 0] = 1
    return x_new

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


######################################################
d = num_inputs
K = num_outputs
dH = 500

W = np.random.randn(dH, d)/np.sqrt(num_inputs)
b1 = np.random.randn(dH)
b2 = np.random.randn(K)
C = np.random.randn(K, dH)

l_rate = 0.005
iteration_num = 500000

deriv = np.zeros(10)

i = 0
while (i < iteration_num):
    idx = random.randint(0, len(x_train)-1)
    x_t = x_train[idx]
    y_t = y_train[idx]
    Z = W@x_t + b1
    H = relu(Z)
    U = C@H + b2
    fun_x = softmax(U)

    for k in range(10):
        if k == y_t:
            deriv[k] = -(1-fun_x[k])
        else:
            deriv[k] = fun_x[k]
    sigma = C.T@deriv

    C = C - l_rate*np.outer(deriv,H)
    b2 = b2 - l_rate*deriv
    b1 = b1 - l_rate*np.multiply(sigma,reluDerivative(H))
    W = W - l_rate*np.outer(np.multiply(sigma,reluDerivative(H)),x_t)

    i += 1

model['W'] = W
model['b1'] = b1
model['b2'] = b2
model['C'] = C


def forward(x, y, model):
    Z = model['W']@x + model['b1']
    H = relu(Z)
    U = model['C']@H + model['b2']
    return softmax(U)

# #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) ) )
```

```
0.9765
```

```
In [ ]:
```