CS398 HW3: Convolution Network

Yanjun Guo (yanjung2)

In this assignment, I used stochastic gradient descent to train a convolution network model with multiple channels and reached 94.35% accuracy. Firstly, I generated K, b, W using randn() function. Then I set dimension of K to be 6, number of channels to be 8. I chose ReLU for σ, and constructed distribution function with the formula for derivatives from lecture, and upgrade new parameters each time I iterate. I used a learning rate of 0.005 because it has been good for stochastic gradient decent in previous homework. After 50000 iterations, I finally got 94.35% accuracy for test data.

(Code next page)

In [42]:

```
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 = 28
ky = 6
C = 8
dy = num_outputs
dim = d-ky+1
K = np.random.randn(ky, ky, C)
b = np.random.randn(dy)
W = np.random.randn(dy, dim, dim, C)

l_rate = 0.005
iteration_num = 50000

# def con_mult(x,k):
#     result = np.zeros(dim*dim).reshape(dim,dim)
#     for i in range(dim):
#         for j in range(dim):
#             for m in range(ky):
#                 for n in range(ky):
#                     result[i][j] = k[m][n]*x[i+m][j+n]
```

```
deriv = np.zeros(10)
U = np.zeros(dy)
Z = np.zeros(dim*dim*C).reshape(dim,dim,C)
H = np.zeros(dim*dim*C).reshape(dim,dim,C)
sigma = np.zeros(dim*dim*C).reshape(dim,dim,C)
de_W = np.zeros(dy*dim*dim*C).reshape(dy,dim,dim,C)
de_K = np.zeros(ky*ky*C).reshape(ky,ky,C)
i = 0
while (i < iteration_num):
    if (i%1000 == 0):
        print(i)
    idx = random.randint(0, len(x_train)-1)
    x_t = x_train[idx].reshape(28, 28)
    y_t = y_train[idx]
    for idx_z in range(C):
        for idx_x in range(dim):
            for idx_y in range(dim):
                Z[idx_x][idx_y][idx_z] = np.sum(x_t[idx_x:idx_x+ky,idx_y:idx_y+k
y]*K[:,:,idx_z])
#         H[:,:,idx_z] = relu(Z[:,:,idx_z])
    H = relu(Z)
    for idx in range(dy):
        U[idx] = np.sum(W[idx,:,:,:]*H) + b[idx]
    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]
    for idx_z in range(C):
        for idx_x in range(dim):
            for idx_y in range(dim):
                sigma[idx_x][idx_y][idx_z] = deriv@W[:,idx_x,idx_y,idx_z]
    for idx in range(dy):
        de_W[idx,:,:,:] = deriv[idx]*H
    W = W - l_rate*de_W
    b = b - l_rate*deriv
    der_Z = 1*(Z>1)
    temp = der_Z*sigma
    for idx_z in range(C):
        for idx_x in range(ky):
            for idx_y in range(ky):
                de_K[idx_x,idx_y,idx_z]= np.sum(x_t[idx_x:idx_x+ky,idx_y:idx_x+k
y]*temp[idx_x,idx_y,idx_z])
    K = K - l_rate*de_K

    i += 1

model['W'] = W
model['b'] = b
model['K'] = K


def forward(x, y, model):
    for idx_z in range(C):
        for idx_x in range(dim):
            for idx_y in range(dim):
                Z[idx_x][idx_y][idx_z] = np.sum(x[idx_x:idx_x+ky,idx_y:idx_y+ky]
```

```
*model['K'][:,:,idx_z])
#         H[:,:,idx_z] = relu(Z[:,:,idx_z])
    H = relu(Z)
    for idx in range(dy):
        U[idx] = np.sum(model['W'][idx,:,:,:]*H) + model['b'][idx]
    return softmax(U)

# #test data

total_correct = 0

for n in range( len(x_test)):
    if (n%1000 == 0):
        print(n)
    y = y_test[n]
    x = x_test[n][:].reshape(28,28)
    p = forward(x, y, model)
    prediction = np.argmax(p)
    if (prediction == y):
        total_correct += 1

print(total_correct/np.float(len(x_test)) )
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000
22000
23000
24000
25000
26000
27000
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000
43000
44000
45000
46000
47000
48000
49000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
0.9435
```