# reference/BN/BatchNormalizationTF/main.py

import numpy as np, tensorflow as tf, tqdm  
from tensorflow.examples.tutorials.mnist import input\_data  
import matplotlib.pyplot as plt  
  
mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)  
  
  
"""  
No batch normalization   
"""  
# Generate predetermined random weights so the networks are similarly initialized  
w1\_initial = np.random.normal(size=(784,100)).astype(np.float32)  
w2\_initial = np.random.normal(size=(100,100)).astype(np.float32)  
w3\_initial = np.random.normal(size=(100,10)).astype(np.float32)  
  
# Small epsilon value for the BN transform  
epsilon = 1e-3  
  
# Placeholders  
x = tf.placeholder(tf.float32, shape=[None, 784])  
y\_ = tf.placeholder(tf.float32, shape=[None, 10])  
  
# Layer 1 without BN  
w1 = tf.Variable(w1\_initial)  
b1 = tf.Variable(tf.zeros([100]))  
z1 = tf.matmul(x,w1)+b1  
l1 = tf.nn.sigmoid(z1)  
  
# Layer 1 with BN  
w1\_BN = tf.Variable(w1\_initial)  
  
# Note that pre-batch normalization bias is ommitted. The effect of this bias would be  
# eliminated when subtracting the batch mean. Instead, the role of the bias is performed  
# by the new beta variable. See Section 3.2 of the BN2015 paper.  
z1\_BN = tf.matmul(x,w1\_BN)  
  
# Calculate batch mean and variance  
batch\_mean1, batch\_var1 = tf.nn.moments(z1\_BN,[0])  
  
# Apply the initial batch normalizing transform  
z1\_hat = (z1\_BN - batch\_mean1) / tf.sqrt(batch\_var1 + epsilon)  
  
# Create two new parameters, scale and beta (shift)  
scale1 = tf.Variable(tf.ones([100]))  
beta1 = tf.Variable(tf.zeros([100]))  
  
# Scale and shift to obtain the final output of the batch normalization  
# this value is fed into the activation function (here a sigmoid)  
BN1 = scale1 \* z1\_hat + beta1  
l1\_BN = tf.nn.sigmoid(BN1)  
  
# Layer 2 without BN  
w2 = tf.Variable(w2\_initial)  
b2 = tf.Variable(tf.zeros([100]))  
z2 = tf.matmul(l1,w2)+b2  
l2 = tf.nn.sigmoid(z2)  
  
# Layer 2 with BN, using Tensorflows built-in BN function  
w2\_BN = tf.Variable(w2\_initial)  
z2\_BN = tf.matmul(l1\_BN,w2\_BN)  
batch\_mean2, batch\_var2 = tf.nn.moments(z2\_BN,[0])  
scale2 = tf.Variable(tf.ones([100]))  
beta2 = tf.Variable(tf.zeros([100]))  
BN2 = tf.nn.batch\_normalization(z2\_BN,batch\_mean2,batch\_var2,beta2,scale2,epsilon)  
l2\_BN = tf.nn.sigmoid(BN2)  
  
# Softmax  
w3 = tf.Variable(w3\_initial)  
b3 = tf.Variable(tf.zeros([10]))  
y = tf.nn.softmax(tf.matmul(l2,w3)+b3)  
  
w3\_BN = tf.Variable(w3\_initial)  
b3\_BN = tf.Variable(tf.zeros([10]))  
y\_BN = tf.nn.softmax(tf.matmul(l2\_BN,w3\_BN)+b3\_BN)  
  
# Loss, optimizer and predictions  
cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y))  
cross\_entropy\_BN = -tf.reduce\_sum(y\_\*tf.log(y\_BN))  
  
train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)  
train\_step\_BN = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy\_BN)  
  
correct\_prediction = tf.equal(tf.arg\_max(y,1),tf.arg\_max(y\_,1))  
accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,tf.float32))  
correct\_prediction\_BN = tf.equal(tf.arg\_max(y\_BN,1),tf.arg\_max(y\_,1))  
accuracy\_BN = tf.reduce\_mean(tf.cast(correct\_prediction\_BN,tf.float32))  
  
zs, BNs, acc, acc\_BN = [], [], [], []  
  
sess = tf.InteractiveSession()  
sess.run(tf.global\_variables\_initializer())  
for i in tqdm.tqdm(range(40000)):  
 batch = mnist.train.next\_batch(60)  
 train\_step.run(feed\_dict={x: batch[0], y\_: batch[1]})  
 train\_step\_BN.run(feed\_dict={x: batch[0], y\_: batch[1]})  
 if i % 50 is 0:  
 res = sess.run([accuracy,accuracy\_BN,z2,BN2],feed\_dict={x: mnist.test.images, y\_: mnist.test.labels})  
 acc.append(res[0])  
 acc\_BN.append(res[1])  
 zs.append(np.mean(res[2],axis=0)) # record the mean value of z2 over the entire test set  
 BNs.append(np.mean(res[3],axis=0)) # record the mean value of BN2 over the entire test set  
  
zs, BNs, acc, acc\_BN = np.array(zs), np.array(BNs), np.array(acc), np.array(acc\_BN)  
  
fig, ax = plt.subplots()  
  
ax.plot(range(0,len(acc)\*50,50),acc, label='Without BN')  
ax.plot(range(0,len(acc)\*50,50),acc\_BN, label='With BN')  
ax.set\_xlabel('Training steps')  
ax.set\_ylabel('Accuracy')  
ax.set\_ylim([0.8,1])  
ax.set\_title('Batch Normalization Accuracy')  
ax.legend(loc=4)  
plt.show()  
  
fig, axes = plt.subplots(5, 2, figsize=(6,12))  
fig.tight\_layout()  
  
for i, ax in enumerate(axes):  
 ax[0].set\_title("Without BN")  
 ax[1].set\_title("With BN")  
 ax[0].plot(zs[:,i])  
 ax[1].plot(BNs[:,i])  
  
  
  
predictions = []  
correct = 0  
for i in range(100):  
 pred, corr = sess.run([tf.arg\_max(y\_BN,1), accuracy\_BN],  
 feed\_dict={x: [mnist.test.images[i]], y\_: [mnist.test.labels[i]]})  
 correct += corr  
 predictions.append(pred[0])  
print("PREDICTIONS:", predictions)  
print("ACCURACY:", correct/100)