# BatchNormalizationTF\_BatchNormalized

import tensorflow as tf  
  
  
"""  
Normalized architechture   
"""  
  
  
def batch\_norm\_wrapper(inputs, is\_training):  
 ...  
 pop\_mean = tf.Variable(tf.zeros([inputs.get\_shape()[-1]]), trainable=False)  
 pop\_var = tf.Variable(tf.ones([inputs.get\_shape()[-1]]), trainable=False)  
  
 if is\_training:  
 mean, var = tf.nn.moments(inputs, [0])  
 ...  
 # learn pop\_mean and pop\_var here  
 ...  
 return tf.nn.batch\_normalization(inputs, batch\_mean, batch\_var, beta, scale, epsilon)  
 else:  
 return tf.nn.batch\_normalization(inputs, pop\_mean, pop\_var, beta, scale, epsilon)  
  
  
decay = 0.999 # use numbers closer to 1 if you have more data  
train\_mean = tf.assign(pop\_mean, pop\_mean\*decay + batch\_mean\*(1 - decay))  
train\_var = tf.assign(pop\_var, pop\_var\*decay + batch\_var\*(1 - decay))  
  
  
# this is a simpler version of Tensorflow's 'official' version. See:  
# https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/layers/python/layers/layers.py#L102  
def batch\_norm\_wrapper(inputs, is\_training, decay=0.999):  
 scale = tf.Variable(tf.ones([inputs.get\_shape()[-1]]))  
 beta = tf.Variable(tf.zeros([inputs.get\_shape()[-1]]))  
 pop\_mean = tf.Variable(tf.zeros([inputs.get\_shape()[-1]]), trainable=False)  
 pop\_var = tf.Variable(tf.ones([inputs.get\_shape()[-1]]), trainable=False)  
  
 if is\_training:  
 batch\_mean, batch\_var = tf.nn.moments(inputs, [0])  
 train\_mean = tf.assign(pop\_mean,  
 pop\_mean\*decay + batch\_mean\*(1 - decay))  
 train\_var = tf.assign(pop\_var,  
 pop\_var\*decay + batch\_var\*(1 - decay))  
 with tf.control\_dependencies([train\_mean, train\_var]):  
 return tf.nn.batch\_normalization(inputs,  
 batch\_mean, batch\_var, beta, scale, epsilon)  
 else:  
 return tf.nn.batch\_normalization(inputs,  
 pop\_mean, pop\_var, beta, scale, epsilon)  
  
  
def build\_graph(is\_training):  
 # Placeholders  
 x = tf.placeholder(tf.float32, shape=[None, 784])  
 y\_ = tf.placeholder(tf.float32, shape=[None, 10])  
  
 # Layer 1  
 w1 = tf.Variable(w1\_initial)  
 z1 = tf.matmul(x, w1)  
 bn1 = batch\_norm\_wrapper(z1, is\_training)  
 l1 = tf.nn.sigmoid(bn1)  
  
 # Layer 2  
 w2 = tf.Variable(w2\_initial)  
 z2 = tf.matmul(l1, w2)  
 bn2 = batch\_norm\_wrapper(z2, is\_training)  
 l2 = tf.nn.sigmoid(bn2)  
  
 # Softmax  
 w3 = tf.Variable(w3\_initial)  
 b3 = tf.Variable(tf.zeros([10]))  
 y = tf.nn.softmax(tf.matmul(l2, w3))  
  
 # Loss, Optimizer and Predictions  
 cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y))  
  
 train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)  
  
 correct\_prediction = tf.equal(tf.arg\_max(y, 1), tf.arg\_max(y\_, 1))  
 accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  
  
 return (x, y\_), train\_step, accuracy, y, tf.train.Saver()  
  
  
# Build training graph, train and save the trained model  
  
  
tf.reset\_default\_graph()  
(x, y\_), train\_step, accuracy, \_, saver = build\_graph(is\_training=True)  
  
acc = []  
with tf.Session() as sess:  
 sess.run(tf.global\_variables\_initializer())  
 for i in tqdm.tqdm(range(10000)):  
 batch = mnist.train.next\_batch(60)  
 train\_step.run(feed\_dict={x: batch[0], y\_: batch[1]})  
 if i%50 is 0:  
 res = sess.run([accuracy], feed\_dict={x: mnist.test.images, y\_: mnist.test.labels})  
 acc.append(res[0])  
 saved\_model = saver.save(sess, './temp-bn-save')  
  
print("Final accuracy:", acc[-1])  
  
tf.reset\_default\_graph()  
(x, y\_), \_, accuracy, y, saver = build\_graph(is\_training=False)  
  
predictions = []  
correct = 0  
with tf.Session() as sess:  
 sess.run(tf.global\_variables\_initializer())  
 saver.restore(sess, './temp-bn-save')  
 for i in range(100):  
 pred, corr = sess.run([tf.arg\_max(y, 1), accuracy],  
 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)