

## Exercise 3.4.4

### Batch Normalization

#### 1. My entire code for Exercise 2.1 and 2.2

I attached to the archive my solution for exercise 2.1 and 2.2 because I had to do the batch normalization on the network I built for that exercise. I assume that you have a different solution for that exercise so you will have to adapt my batch normalization to your solution. Now I will highlight the code pieces that I changed in the network in order to add the batch visualization.

#### 2. Batch Normalization for the Convolutional layers and for the Densely Connected layer

I realized that I have to do a function to add the batch normalization in the convolutional layers and a different one for the densely connected layer because of my data encodings. I added these 2 functions before creating the layers. The only difference between these 2 functions is the dealing with different encodings for data.

```
def batch_norm_wrapper_convolutional(inputs, decay = 0.999):

    epsilon=0.00000001
    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)

    mean = tf.nn.moments(inputs,[0])[0][0][0][0]
    var = tf.nn.moments(inputs,[0])[1][0][0][0]
    train_mean = tf.assign(pop_mean,pop_mean*decay+mean*(1-decay))
    train_var = tf.assign(pop_var,pop_var*decay+var*(1-decay))

    with tf.control_dependencies([train_mean, train_var]):
        return tf.nn.batch_normalization(inputs,mean, var, beta, scale, epsilon)
```

```
def batch_norm_wrapper_densely(inputs, decay = 0.999):

    epsilon=0.00000001
    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)


    mean = tf.nn.moments(inputs,[0])[0]
    var = tf.nn.moments(inputs,[0])[1]
    train_mean = tf.assign(pop_mean,pop_mean*decay+mean*(1-decay))
    train_var = tf.assign(pop_var,pop_var*decay+var*(1-decay))

    with tf.control_dependencies([train_mean, train_var]):
        return tf.nn.batch_normalization(inputs,mean, var, beta, scale, epsilon)
```

### 3. Code replacement in the layers

#### *# First Convolutional Layer*


```
W_conv1 = new_weights([5, 5, 1, 32])
b_conv1 = new_biases([32])
x_image = tf.reshape(x, [-1, 28, 28, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```



```
W_conv1 = new_weights([5, 5, 1, 32])
b_conv1 = new_biases([32])
x_image = tf.reshape(x, [-1, 28, 28, 1])
h_conv1 = conv2d(x_image, W_conv1)
batched = batch_norm_wrapper_convolutional(h_conv1)
batched += b_conv1
h_conv1 = tf.nn.relu(batched)
h_pool1 = max_pool_2x2(h_conv1)
```

#### *# Second Convolutional Layer*


```
W_conv2 = new_weights([5, 5, 32, 64])
b_conv2 = new_biases([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```



```
W_conv2 = new_weights([5, 5, 32, 64])
b_conv2 = new_biases([64])
h_conv2 = conv2d(h_pool1, W_conv2)
batched = batch_norm_wrapper_convolutional(h_conv2)
batched += b_conv2
h_conv2 = tf.nn.relu(batched)
h_pool2 = max_pool_2x2(h_conv2)
```

### *# Densely Connected Layer*

```
W_fc1 = new_weights([7 * 7 * 64, 1024])
b_fc1 = new_biases([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```



```
W_fc1 = new_weights([7 * 7 * 64, 1024])
b_fc1 = new_biases([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.matmul(h_pool2_flat, W_fc1)
batched = batch_norm_wrapper_densely(h_fc1)
batched += b_fc1
h_fc1 = tf.nn.relu(batched)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

#### **4. My code after batch normalization**

I also attached my code after implementing the batch normalization. For me is working and I got some pretty nice results on the accuracy (>95%).

#### **5. How does it influence the training and testing loss? How does it influence convergence?**

The reason of batch normalization is to avoid noisy weight changes. The training is faster this way. The convergence is effectively accelerated and the loss starts to lower faster since the very first epoch.