Imports and data input

In [1]:

```
import tensorflow as tf
```

In [2]:

```
from tensorflow.examples.tutorials.mnist import input_data
```

In [4]:

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Helper Functions

In [5]:

```
#Function to initalize random weights for convolutional layers
def init_weights(shape):
    init_random_dist = tf.truncated_normal(shape,stddev=0.1)
    return tf.Variable(init_random_dist)
```

In [6]:

```
#Function to initalize biases for layers
def init_bias(shape):
    init_bias_vals = tf.constant(0.1,shape=shape)
    return tf.Variable(init_bias_vals)
```

In [7]:

```
#Create a 2D convolution
def conv2d(x,W):
    # x --> [batch,H,W,Channels]
    # W --> [filter H, filter W, Channels IN, Channels OUT]

    return tf.nn.conv2d(x,W,strides=[1,1,1,1],padding='SAME')
```

In [8]:

```
#Max pooling layer with 2x2 window
def max_pool_2by2(x):
    # x --> [batch,H,W,Channels]
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
```

In [9]:

```
#Returns a convolutional layer which uses ReLu activation
def convolutional_layer(input_x,shape):
    W = init_weights(shape)
    b = init_bias([shape[3]])
    return tf.nn.relu(conv2d(input_x,W)+b)
```

In [10]:

```
#Normal fully connected layer
def normal_full_layer(input_layer,size):
    input_size = int(input_layer.get_shape()[1])
    W = init_weights([input_size,size])
    b = init_bias([size])
    return tf.matmul(input_layer,W)+b
```

Placeholders

In [11]:

```
x = tf.placeholder(tf.float32,shape=[None,784])
y_true = tf.placeholder(tf.float32,shape=[None,10])
```

Layers

In [12]:

```
#reshape the image back into it's 28x28 shape
x_image = tf.reshape(x,[-1,28,28,1])
```

In [13]:

```
#First convolutional layer using 32 5x5 filters
convo_1 = convolutional_layer(x_image,shape=[5,5,1,32])
convo_1_pooling = max_pool_2by2(convo_1)
```

In [14]:

```
#Second convolutional layer using 64 5x5 filters from the first layers 32 filters
convo_2 = convolutional_layer(convo_1_pooling,shape=[5,5,32,64])
convo_2_pooling = max_pool_2by2(convo_2)
```

In [15]:

```
#Flattening the second convolutional layer.
#7*7*64 comes from the 28 horizontal and vertical pixels going through two pooling
#64 comes from the 64 filters in the second layer
convo_2_flat = tf.reshape(convo_2_pooling,[-1,7*7*64])
full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat, 1024))
```

In [16]:

```
#Dropout to prevent overfitting
hold_prob = tf.placeholder(tf.float32)
full_one_dropout = tf.nn.dropout(full_layer_one, keep_prob=hold_prob)
```

In [17]:

```
y_pred = normal_full_layer(full_one_dropout,10)
```

Loss Function

In [19]:

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_tr
```

Optimizer

In [20]:

```
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
train = optimizer.minimize(cross_entropy)
```

Initalize Variables

In [21]:

```
init = tf.global_variables_initializer()
```

Session

In [22]:

```
steps = 2000

with tf.Session() as sess:
    sess.run(init)

    for i in range(steps):

        #use 50 images per batch
        batch_x, batch_y = mnist.train.next_batch(50)

        #feed dictionary with x data, y data, and dropout probability of 50%
        sess.run(train,feed_dict={x:batch_x,y_true:batch_y,hold_prob:0.5})

        #report accuracy after every 100 steps
        if i%100 == 0:

            print("ON STEP: {}".format(i))
            print("Accuracy: ")

            matches = tf.equal(tf.argmax(y_pred,1),tf.argmax(y_true,1))

            acc = tf.reduce_mean(tf.cast(matches,tf.float32))

            print(sess.run(acc,feed_dict={x:mnist.test.images,y_true:mnist.test.la
            print('\n')
```

```
ON STEP: 0
Accuracy: 
0.1064


ON STEP: 100
Accuracy: 
0.9467


ON STEP: 200
Accuracy: 
0.9651


ON STEP: 300
Accuracy: 
0.9665


ON STEP: 400
Accuracy: 
0.9747


ON STEP: 500
Accuracy: 
0.9779
```

```
ON STEP: 600
Accuracy:
0.9795


ON STEP: 700
Accuracy:
0.9803


ON STEP: 800
Accuracy:
0.9811


ON STEP: 900
Accuracy:
0.9848


ON STEP: 1000
Accuracy:
0.9813


ON STEP: 1100
Accuracy:
0.9826


ON STEP: 1200
Accuracy:
0.9864


ON STEP: 1300
Accuracy:
0.9852


ON STEP: 1400
Accuracy:
0.9837


ON STEP: 1500
Accuracy:
0.9846


ON STEP: 1600
Accuracy:
0.9868


ON STEP: 1700
Accuracy:
```

```
0.9836


ON STEP: 1800
Accuracy:
0.9873


ON STEP: 1900
Accuracy:
0.9882
```

In [ ]:

```

```