ML_Coursework_3

November 27, 2018

```
In [2]: %matplotlib notebook
        %matplotlib inline

        from matplotlib import pylab
        pylab.rcParams['figure.figsize'] = (10.0, 10.0)

        from tensorflow.examples.tutorials.mnist import input_data

        import matplotlib.pyplot as plt
        import numpy as np
        import tensorflow as tf
```

0.0.1 Initialisations

Let us write some helper functions to initialise weights and biases. We'll initialise weights as Gaussian random variables with mean 0 and variance 0.0025. For biases we'll initialise everything with a constant 0.1. This is because we're mainly going to be using ReLU non-linearities.

```
In [3]: def weight_variable(shape):
            initial = tf.truncated_normal(shape, stddev=0.05)
            return tf.Variable(initial)

        def bias_variable(shape):
            initial = tf.constant(0.1, shape=shape)
            return tf.Variable(initial)
```

0.0.2 Model

Let's define the model. The model is defined as follows:

- An input that is 728 dimensional vector.
- Reshape the input as 28x28x1 images (only 1 because they are grey scale)
- A convolutional layer with 25 filters of shape 12x12x1 and a ReLU non-linearity (with stride (2, 2) and no padding)
- A convolutional layer with 64 filters of shape 5x5x25 and a ReLU non-linearity (with stride (1, 2) and padding to maintain size)
- A max_pooling layer of shape 2x2
- A fully connected layer taking all the outputs of the max_pooling layer to 1024 units and ReLU nonlinearity

- A fully connected layer taking 1024 units to 10 no activation function (the softmax non-linearity will be included in the loss function rather than in the model)

```
In [4]: x = tf.placeholder(tf.float32, shape=[None, 784])
        x_ = tf.reshape(x, [-1, 28, 28, 1])
        y_ = tf.placeholder(tf.float32, shape=[None, 10])

        # Define the first convolution layer here
        # TODO
        W_conv1 = weight_variable([12, 12, 1, 25])
        b_conv1 = bias_variable([25])
        h_conv1 = tf.nn.relu(tf.nn.conv2d(x_, W_conv1,
                                          strides=[1, 2, 2, 1], padding='VALID'))

        # Define the second convolution layer here
        W_conv2 = weight_variable([5, 5, 25, 64])
        b_conv2 = bias_variable([64])
        h_conv2 = tf.nn.relu(tf.nn.conv2d(h_conv1,
                                          W_conv2, strides=[1, 1, 1, 1], padding='SAME'))

        # Define maxpooling
        h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1],
                                 strides=[1, 2, 2, 1], padding='VALID')

        # All subsequent layers will be fully connected ignoring geometry so we'll flatten the
        # Flatten the h_pool2_layer (as it has a multidimensiona shape)
        h_pool2_flat = tf.reshape(h_pool2, [-1, 4*4*64])

        # Define the first fully connected layer here
        W_fc1 = weight_variable([4 * 4 * 64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

        # Use dropout for this layer (should you wish)
        keep_prob = tf.placeholder(tf.float32)
        h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

        # The final fully connected layer
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])
        y_conv = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

Loss Function, Accuracy and Training Algorithm

- We'll use the cross entropy loss function. The loss function is called `tf.nn.cross_entropy_with_logits` in tensorflow
- Accuray is simply defined as the fraction of data correctly classified

- For training you should use the AdamOptimizer (read the documentation) and set the learning rate to be 1e-4. You are welcome, and in fact encouraged, to experiment with other optimisation procedures and learning rates.
- (Optional): You may even want to use different filter sizes once you are finished with experimenting with what is asked in this practial

In [5]:
```
# We'll use the cross entropy loss function
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=y_conv, labels=y_))

# And classification accuracy
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# And the Adam optimiser
train_step = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(
    cross_entropy)
```

```
WARNING:tensorflow:From <ipython-input-5-10e0b012a53a>:2: softmax_cross_entropy_with_logits (fi
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

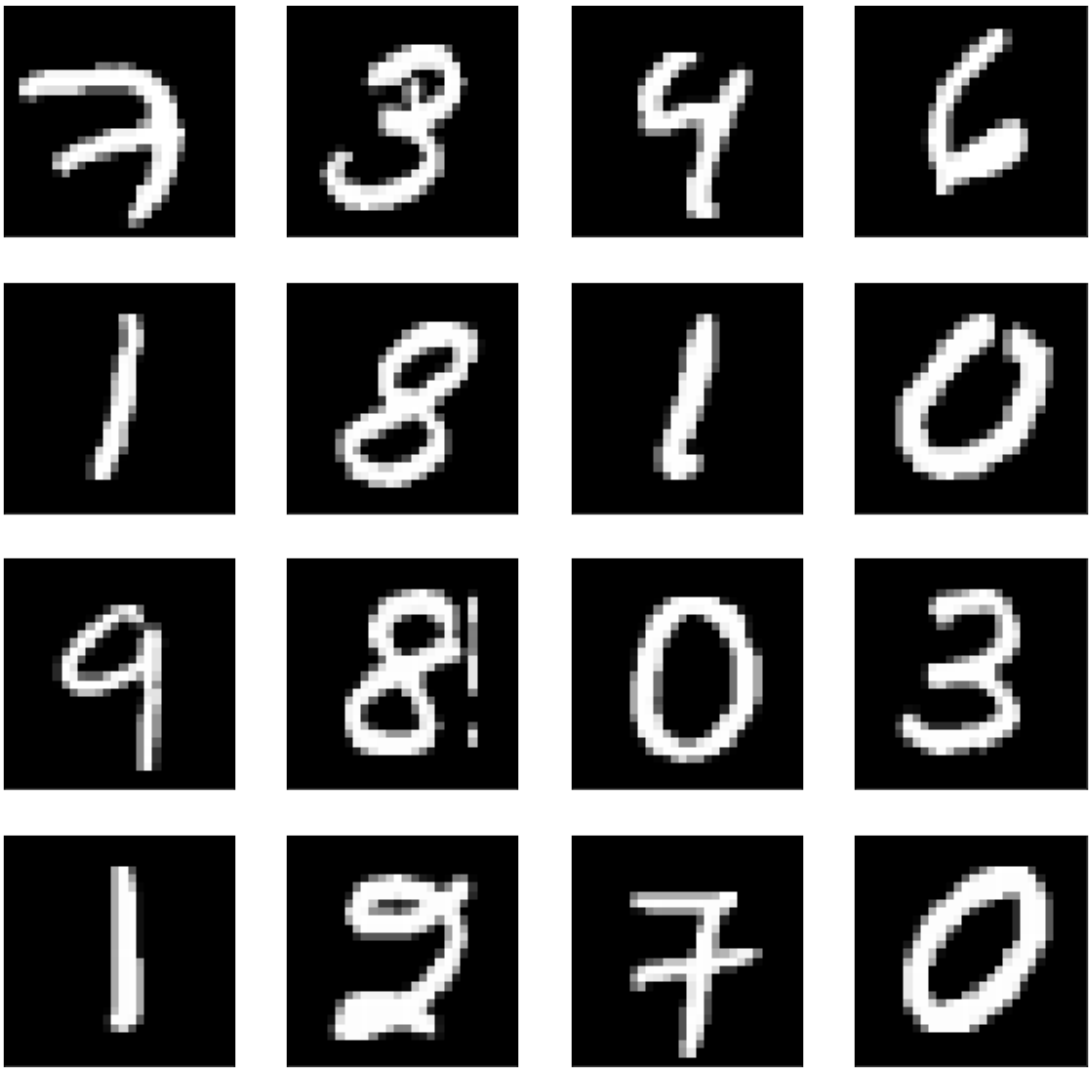
In [6]:
```
# Load the mnist data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

```
WARNING:tensorflow:From <ipython-input-6-19c914ec135d>:2: read_data_sets (from tensorflow.contr
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From /Users/lorcandelaney/anaconda3/lib/python3.6/site-packages/tensorflow/c
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From /Users/lorcandelaney/anaconda3/lib/python3.6/site-packages/tensorflow/c
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From /Users/lorcandelaney/anaconda3/lib/python3.6/site-packages/tensorflow/c
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From /Users/lorcandelaney/anaconda3/lib/python3.6/site-packages/tensorflow/c
Instructions for updating:
Please use tf.one_hot on tensors.
```

```
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /Users/lorcandelaney/anaconda3/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

In [7]:
```
# Let us visualise the first 16 data points from the MNIST training data

fig = plt.figure()
for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.imshow(mnist.train.images[i].reshape(28, 28), cmap='Greys_r')
```



In [10]:
```
# Start a tf session and run the optimisation algorithm
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(3000):
    batch = mnist.train.next_batch(50)
    if i%100 == 0:
        train_accuracy = accuracy.eval(session = sess, feed_dict={
            x:batch[0], y_: batch[1], keep_prob:1.0})
        print("Step %d, training accuracy %g"%(i, train_accuracy))
        print("Test accuracy: %g" % sess.run(accuracy, feed_dict={
            x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
    train_step.run(session = sess, feed_dict={x: batch[0],
                                              y_: batch[1], keep_prob: 0.5})
```

```
Step 0, training accuracy 0.02
Test accuracy: 0.1356
Step 100, training accuracy 0.64
Test accuracy: 0.7722
Step 200, training accuracy 0.86
Test accuracy: 0.8963
Step 300, training accuracy 0.96
Test accuracy: 0.9147
Step 400, training accuracy 0.96
Test accuracy: 0.9281
Step 500, training accuracy 0.92
Test accuracy: 0.939
Step 600, training accuracy 0.94
Test accuracy: 0.9454
Step 700, training accuracy 0.96
Test accuracy: 0.948
Step 800, training accuracy 0.92
Test accuracy: 0.9527
Step 900, training accuracy 0.98
Test accuracy: 0.9575
Step 1000, training accuracy 0.96
Test accuracy: 0.9555
Step 1100, training accuracy 1
Test accuracy: 0.9635
Step 1200, training accuracy 0.98
Test accuracy: 0.9675
Step 1300, training accuracy 0.98
Test accuracy: 0.9671
Step 1400, training accuracy 0.96
Test accuracy: 0.966
```

```
Step 1500, training accuracy 1
Test accuracy: 0.9704
Step 1600, training accuracy 0.96
Test accuracy: 0.9717
Step 1700, training accuracy 0.94
Test accuracy: 0.9726
Step 1800, training accuracy 1
Test accuracy: 0.9731
Step 1900, training accuracy 0.98
Test accuracy: 0.9744
Step 2000, training accuracy 1
Test accuracy: 0.9739
Step 2100, training accuracy 0.96
Test accuracy: 0.9759
Step 2200, training accuracy 0.92
Test accuracy: 0.9768
Step 2300, training accuracy 1
Test accuracy: 0.9752
Step 2400, training accuracy 0.96
Test accuracy: 0.9781
Step 2500, training accuracy 0.96
Test accuracy: 0.9787
Step 2600, training accuracy 0.98
Test accuracy: 0.9777
Step 2700, training accuracy 0.98
Test accuracy: 0.9764
Step 2800, training accuracy 0.98
Test accuracy: 0.9801
Step 2900, training accuracy 1
Test accuracy: 0.9802
```

```
In [12]: # Print accuracy on the test set
         x_test = mnist.test.images
         y_test = mnist.test.labels

         print ('Test accuracy: %g' % sess.run(accuracy, feed_dict={x: mnist.test.images, y_: m
```

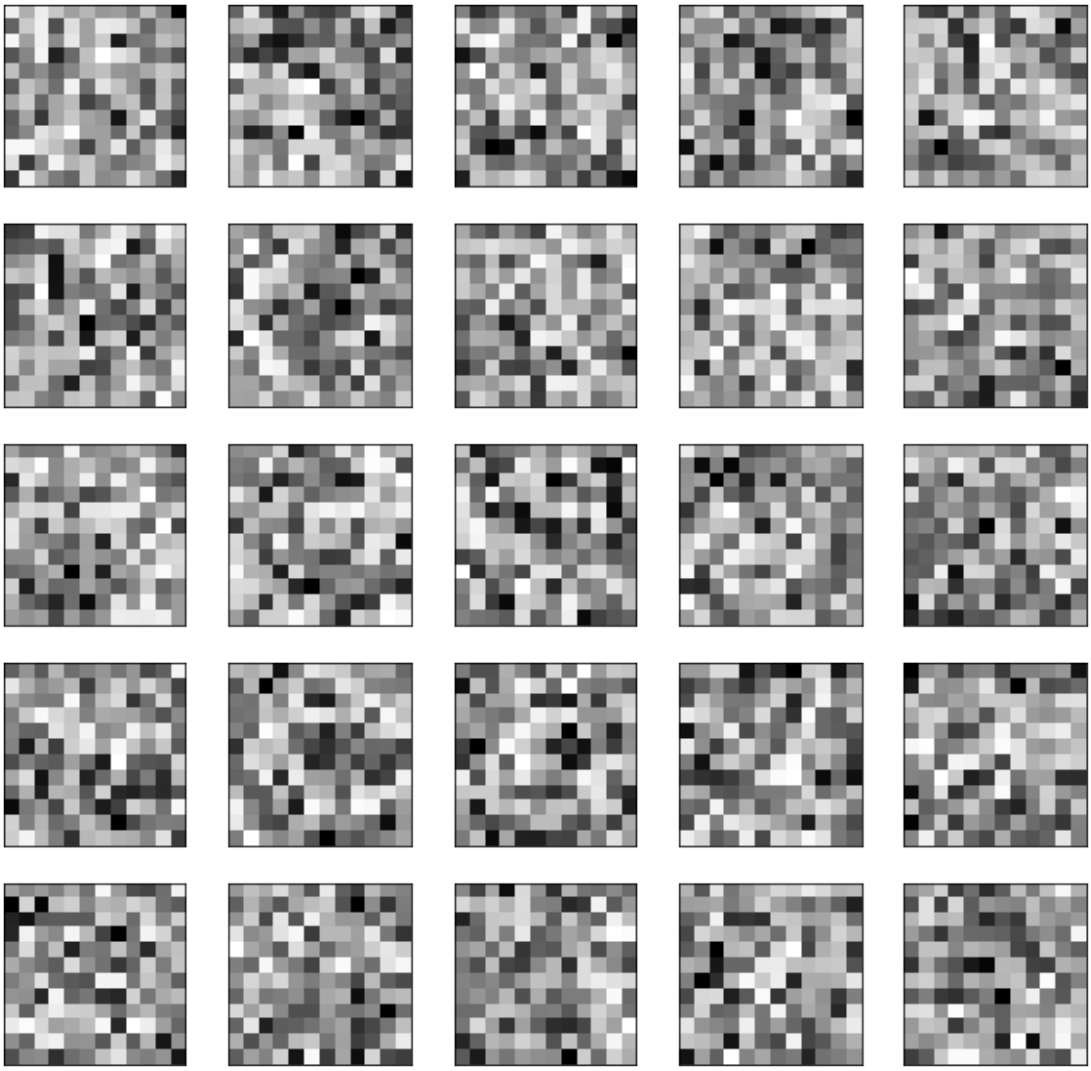
```
Test accuracy: 0.9794
```

Visualising the Filters We'll now visualise all the 32 filters in the first convolution layer. As they are each of shape 12x12x1, they may themselves be viewed as greyscale images. Visualising filters in further layers is more complicated and involves modifying the neural network. See the paper by Matt Zeiler and Rob Fergus if you are interested.

```
In [175]: # Visualise the filters in the first convolutional layer
          with sess.as_default():
```

```
W = W_conv1.eval()
transp_W = np.transpose(W, [3, 0, 1, 2])

fig = plt.figure()
for i in range(25):
    ax = fig.add_subplot(5, 5, i + 1)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.imshow(transp_W[i].reshape(12, 12), cmap='Greys_r')
plt.show()
```



Identifying image patches that activate the filters For this part you'll find the 12 patches in the test-set that activate each of the first 5 filters that maximise the activation for that filter.

In [176]:
```
H =  sess.run(h_conv1, feed_dict={x: mnist.test.images})
transp_outputs = np.transpose(H, [0, 3, 1, 2]) #NCHW

reshaped_test_images = mnist.test.images.reshape(10000, 28, 28)


best_act_values = [0.0 for _ in range(12)]
best_patches = [np.zeros(shape=(12, 12)) for _ in range(12)]

for i in range(len(transp_outputs)): # for all images in test dataset
  activations = transp_outputs[i]

  for j in range(5): # first 5 activations
    activation = activations[j]

    for (x,y), val in np.ndenumerate(activation):
      for k in range(len(best_act_values)):
        if val > best_act_values[k]:
          best_act_values[k] = val
          best_patches[k] = reshaped_test_images[i, (x*2):(x*2+12), (y*2):(y*2+12)]
          break

plt.figure(figsize=(10,10))
for i in range(12):
    plt.subplot(4,3,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(best_patches[i].reshape(12, 12), cmap='Greys_r')
plt.show()

# Add code to visualise patches in the test set that find the most result in
# the highest activations for filters 0, ... 4
```