project1_task3

November 7, 2017

```
In [1]: import tensorflow as tf
        from tensorflow.examples.tutorials.mnist import input_data
        import numpy as np
        import matplotlib.pyplot as plt
```

1 Training network on MNIST

Firstly, I trained mnist_conv1 model for 5 epoch until validation and training accuracy of 98% was achieved

Then the weights from the trained model were used to compute the hidden layer outputs, which are needed for PCA

```
In [2]: mnist = input_data.read_data_sets('./MNIST_data', one_hot=True)
        sess = tf.InteractiveSession()
        x = tf.placeholder(tf.float32, shape=[None, 784])
        y_ = tf.placeholder(tf.float32, shape=[None, 10])

        # Convolutional layer
        x_image = tf.reshape(x, [-1,28,28,1])
        W_conv = tf.Variable(tf.truncated_normal([5, 5, 1, 30], stddev=0.1))
        b_conv = tf.Variable(tf.constant(0.1, shape=[30]))
        h_conv = tf.nn.conv2d(x_image, W_conv, strides=[1, 1, 1, 1], \
                              padding='VALID')
        h_relu = tf.nn.relu(h_conv + b_conv)
        h_pool = tf.nn.max_pool(h_relu, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], \
                                padding='SAME')

        # Fully-connected layer
        W_fc1 = tf.Variable(tf.truncated_normal([12 * 12 * 30, 500], stddev=0.1))
        b_fc1 = tf.Variable(tf.constant(0.1, shape=[500]))
        h_pool_flat = tf.reshape(h_pool, [-1, 12*12*30])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

        # Output layer
        W_fc2 = tf.Variable(tf.truncated_normal([500, 10], stddev=0.1))
        b_fc2 = tf.Variable(tf.constant(0.1, shape=[10]))
        y_hat=tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)
```

```python
# Train and Evaluate the Model
cross_entropy = - tf.reduce_sum(y_*tf.log(y_hat))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_hat,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

sess.run(tf.global_variables_initializer())
print("=================================")
print("|Epoch\tBatch\t|Train\t|Val\t|")
print("|===============================|")
for j in range(5):
    for i in range(550):
        batch = mnist.train.next_batch(100)
        train_step.run(feed_dict={x: batch[0], y_: batch[1]})
        if i%50 == 49:
            train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_: batch[1]})
            val_accuracy = accuracy.eval(feed_dict=\
                {x: mnist.validation.images, y_:mnist.validation.labels})
            print("|%d\t|%d\t|%.4f\t|%.4f\t|"%(j+1, i+1, train_accuracy, val_accuracy))
print("|===============================|")
test_accuracy = accuracy.eval(feed_dict=\
    {x: mnist.test.images, y_:mnist.test.labels})
print("test accuracy=%.4f"%(test_accuracy))
```

```
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
=================================
|Epoch	Batch	|Train	|Val	|
|===============================|
|1	|50	|0.7100	|0.7624	|
|1	|100	|0.8600	|0.8600	|
|1	|150	|0.8600	|0.8858	|
|1	|200	|0.9500	|0.9058	|
|1	|250	|0.8800	|0.9140	|
|1	|300	|0.8800	|0.9200	|
|1	|350	|0.9300	|0.9232	|
|1	|400	|0.9100	|0.9336	|
|1	|450	|0.9300	|0.9278	|
|1	|500	|0.8900	|0.9320	|
|1	|550	|0.9100	|0.9384	|
|2	|50	|0.9600	|0.9460	|
|2	|100	|0.9600	|0.9430	|
|2	|150	|0.9700	|0.9488	|
|2	|200	|0.9900	|0.9548	|
|2	|250	|0.9100	|0.9502	|
```

```
|2        |300       |0.9400       |0.9534       |
|2        |350       |0.9400       |0.9570       |
|2        |400       |0.9500       |0.9582       |
|2        |450       |0.9800       |0.9580       |
|2        |500       |0.9400       |0.9614       |
|2        |550       |0.9300       |0.9590       |
|3        |50       |0.9400       |0.9630       |
|3        |100       |0.9600       |0.9662       |
|3        |150       |0.9800       |0.9634       |
|3        |200       |0.9800       |0.9658       |
|3        |250       |0.9600       |0.9688       |
|3        |300       |0.9400       |0.9696       |
|3        |350       |0.9800       |0.9696       |
|3        |400       |0.9600       |0.9700       |
|3        |450       |0.9900       |0.9712       |
|3        |500       |0.9600       |0.9740       |
|3        |550       |0.9600       |0.9732       |
|4        |50       |0.9700       |0.9710       |
|4        |100       |0.9800       |0.9760       |
|4        |150       |0.9900       |0.9754       |
|4        |200       |0.9600       |0.9740       |
|4        |250       |0.9700       |0.9764       |
|4        |300       |0.9900       |0.9754       |
|4        |350       |0.9900       |0.9778       |
|4        |400       |1.0000       |0.9760       |
|4        |450       |0.9700       |0.9774       |
|4        |500       |0.9500       |0.9792       |
|4        |550       |0.9800       |0.9736       |
|5        |50       |0.9800       |0.9760       |
|5        |100       |0.9900       |0.9782       |
|5        |150       |0.9900       |0.9772       |
|5        |200       |1.0000       |0.9798       |
|5        |250       |0.9800       |0.9810       |
|5        |300       |0.9600       |0.9778       |
|5        |350       |0.9900       |0.9798       |
|5        |400       |0.9800       |0.9794       |
|5        |450       |0.9700       |0.9804       |
|5        |500       |0.9700       |0.9770       |
|5        |550       |1.0000       |0.9812       |
|==============================|
test accuracy=0.9807
```

2 Perform PCA for the encoding of test dataset

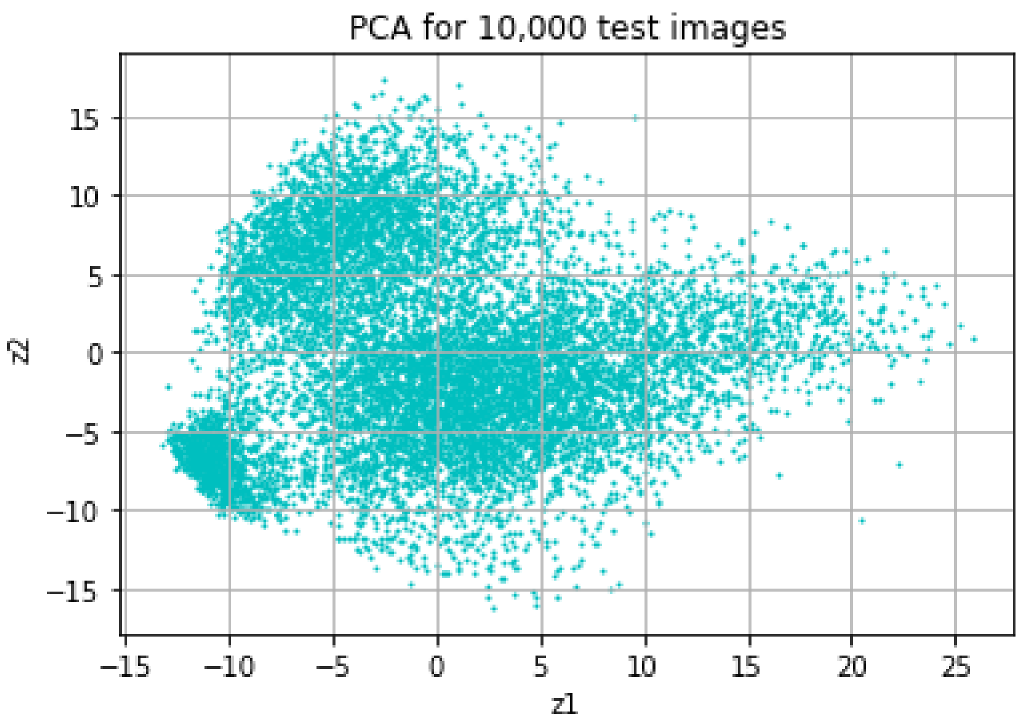
```
In [3]: psi = h_fc1.eval(feed_dict={x: mnist.test.images, y_:mnist.test.labels})
```

```
In [4]: phi = psi - np.mean(psi, axis=0)
```

```
    W, s_2, W_T = np.linalg.svd(np.matmul(np.transpose(phi), phi), \
                                                full_matrices=True)
```

```
In [5]: Z = np.matmul(phi, W)
```

```
In [19]: plt.title("PCA for 10,000 test images")
         plt.xlabel('z1')
         plt.scatter(Z[:,0],Z[:,1], c='c', edgecolor='c', s=1)#, label='label 0')
         #plt.axis([-8,8,-8,8])
         plt.ylabel('z2')
         #plt.legend(loc='best')
         plt.grid(True)
         plt.show()
```



2.1 Filter out 100 test images, 10 per each label

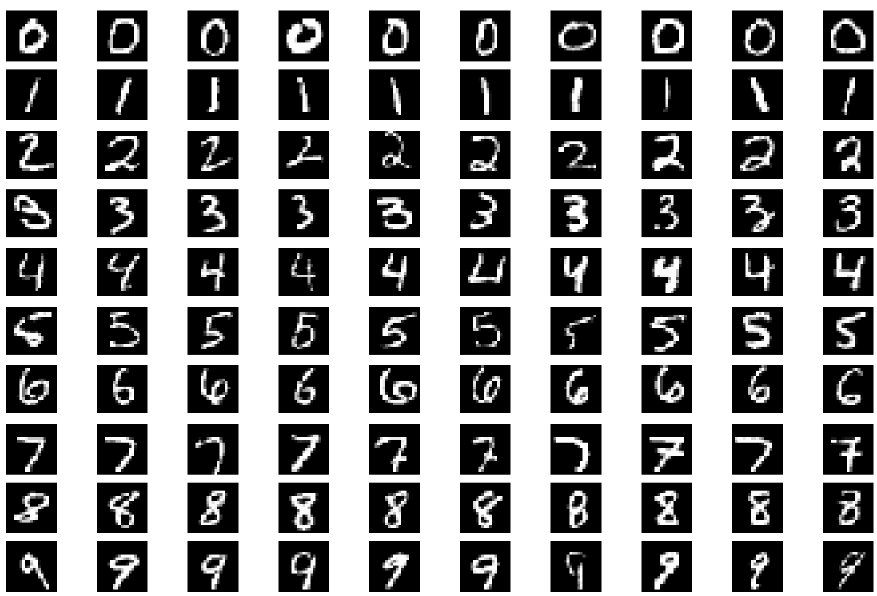
```
In [7]: mnist = input_data.read_data_sets('./MNIST_data', one_hot=False)
        test_i = []
        test_l = []
        for i in range(10):
            test_l += [mnist.test.labels[np.where(mnist.test.labels == i)][:10]]
            test_i += [mnist.test.images[np.where(mnist.test.labels == i)][:10]]

        test_i = np.reshape(np.array(test_i), (-1, mnist.test.images.shape[-1]))
```

```
test_lab = np.reshape(np.array(test_l), (-1,))
test_l = np.eye(10)[test_lab]
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
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
```

In [8]:
```
for i in range(10):
    for j in range(10):
        img=test_i[i*10+j]
        img.shape=(28,28)
        plt.subplot(10,10,i*10+j+1)
        plt.imshow(img,cmap='gray')
        plt.axis('off')
plt.savefig('mnist_train_images.png',dpi=300,bbox_inches='tight')
plt.show()
print(test_lab)
```



```
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3
 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 7 7 7 7
```

```
 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9]
```

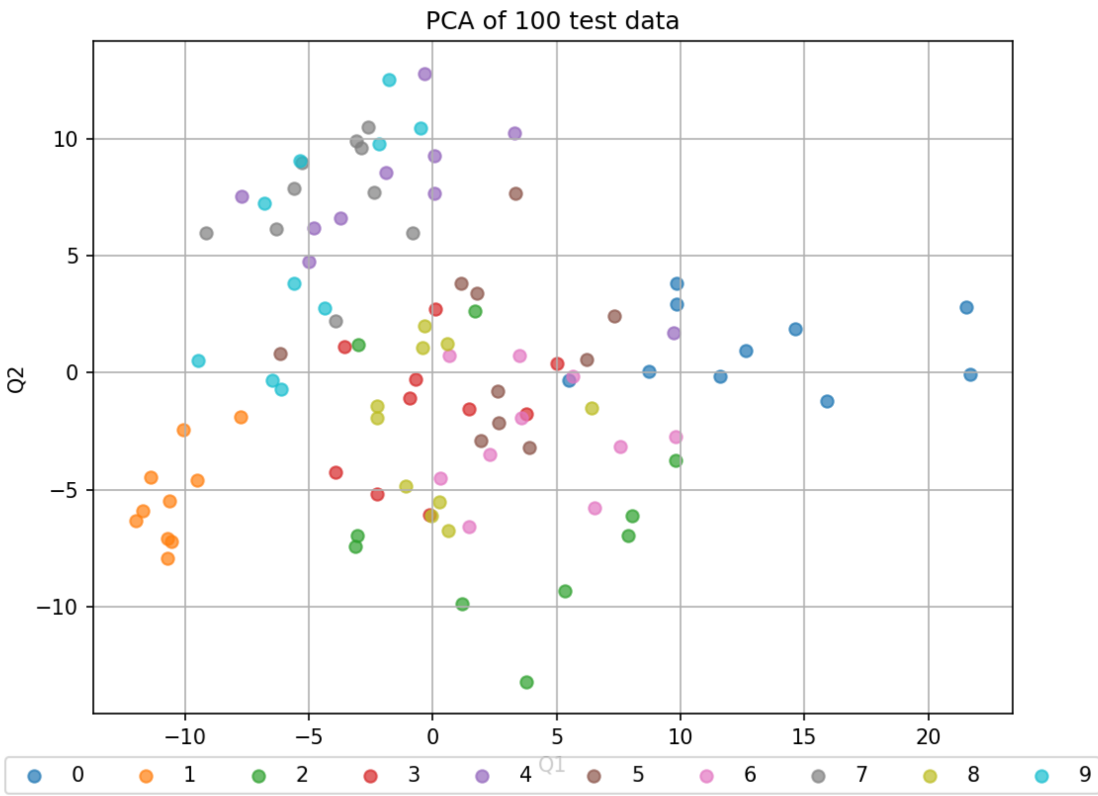
3 Perform PCA for the encoding of 100 test images

```
In [9]: omega = h_fc1.eval(feed_dict={x: test_i, y_:test_l})
        omega = omega - np.mean(psi, axis=0)
        Q = np.matmul(omega, W)
```

```
In [10]: plt.figure(num=None, figsize=(8, 6), dpi=150, facecolor='w', edgecolor='k')
         plt.title("PCA of 100 test data")
         plt.xlabel('Q1')
         plt.grid(True)
         colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', \
                   'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', \
                   'tab:olive', 'tab:cyan']

         for i in range(10):
             plt.scatter(Q[i*10:(i+1)*10,0],Q[i*10:(i+1)*10,1], c=colors[i], s=35, label='%d'%i,
         #plt.axis([-8,8,-8,8])
         plt.ylabel('Q2')
         plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),
                   fancybox=True, ncol=10)

         plt.show()
```

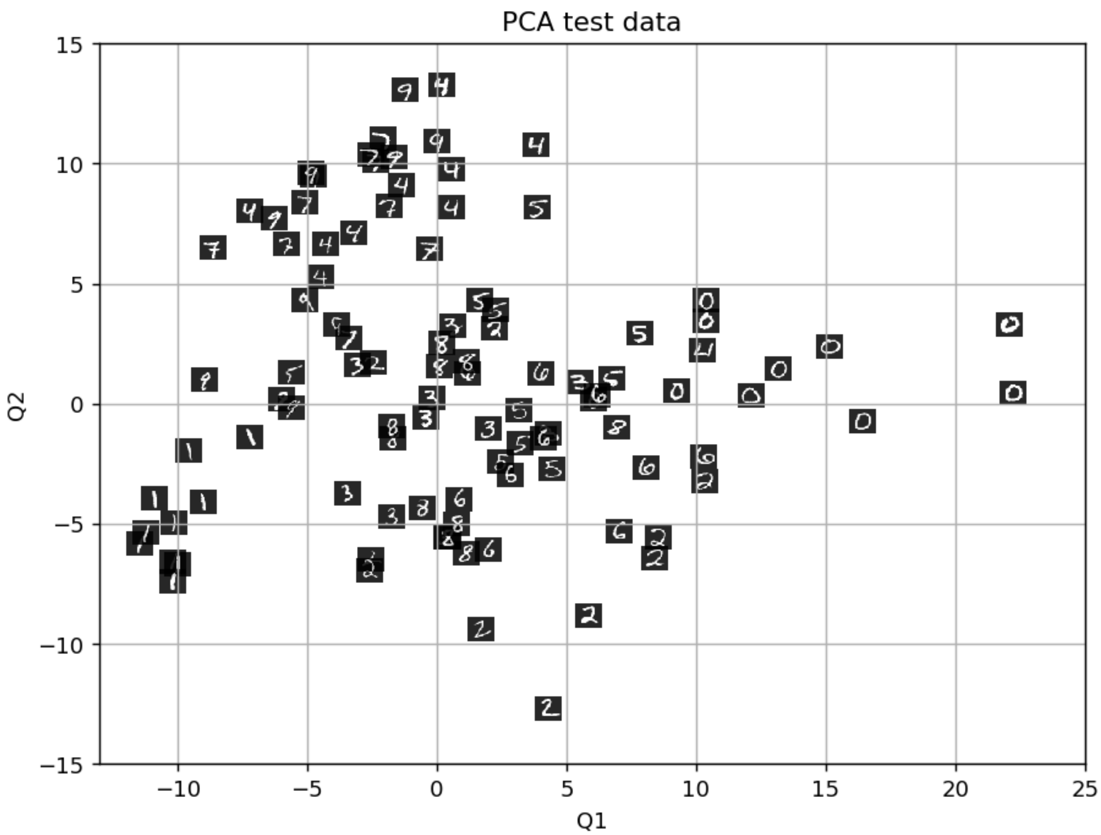



```
In [11]: from matplotlib.image import BboxImage
         from matplotlib.transforms import Bbox, TransformedBbox
```

```
In [22]: fig = plt.figure(num=None, figsize=(8, 6), dpi=130, facecolor='w', edgecolor='k')
         ax = fig.add_subplot(111)
         plt.title("PCA test data")
         plt.xlabel('Q1')
         plt.grid(True)
         colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',\
                   'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan']
         for i in range(10):
             for j in range(10):
                 img=test_i[i*10+j]
                 img.shape=(28,28)
                 bb = Bbox.from_bounds(Q[i*10+j,0],Q[i*10+j,1],1,1)
                 bb2 = TransformedBbox(bb,ax.transData)
                 bbox_image = BboxImage(bb2,
                                     norm = None,
                                     origin=None,
                                     clip_on=False, cmap='gray', alpha=0.85)

                 bbox_image.set_data(img)
```

```
        ax.add_artist(bbox_image)
#plt.axis([-8,8,-8,8])
plt.ylabel('Q2')
# Set the x and y limits
ax.set_ylim(-15,15)
ax.set_xlim(-13,25)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),
          fancybox=True, ncol=10)
plt.show()
```



4 Analysis of PCA results

PCA is used for projecting a high dimensional dataset into smaller subspace into the directions that maximize the variance in the dataset.

From the images above we can see in the projected space that MNIST classifier tried to cluster the image space, such that most similar digits are grouped together.

However, the main weakness of PCA is that it relies on the linear assumption. But if the data is not linearly correlated (e.g. in spiral, where x=t*cos(t) and y =t*sin(t)), PCA is not enough.

So in the image above not all digits are well grouped, those which have some similarities but belong to different classes are located closely. For instance digits "1" are separated from other classes, but "0" and "6" and sometimes "2" are highly overlapping. Since the classification accuracy on the test dataset is 98% it means that most of the times all digits are well disentangled, hence we can make a conclusion that PCA and linear assumption about the correlation are not amazingly good for finding clusters in low dimensinal space, because this classifier is a non-trivial high-dimensional structure, and these sorts of linear projections just aren't going to cut it.