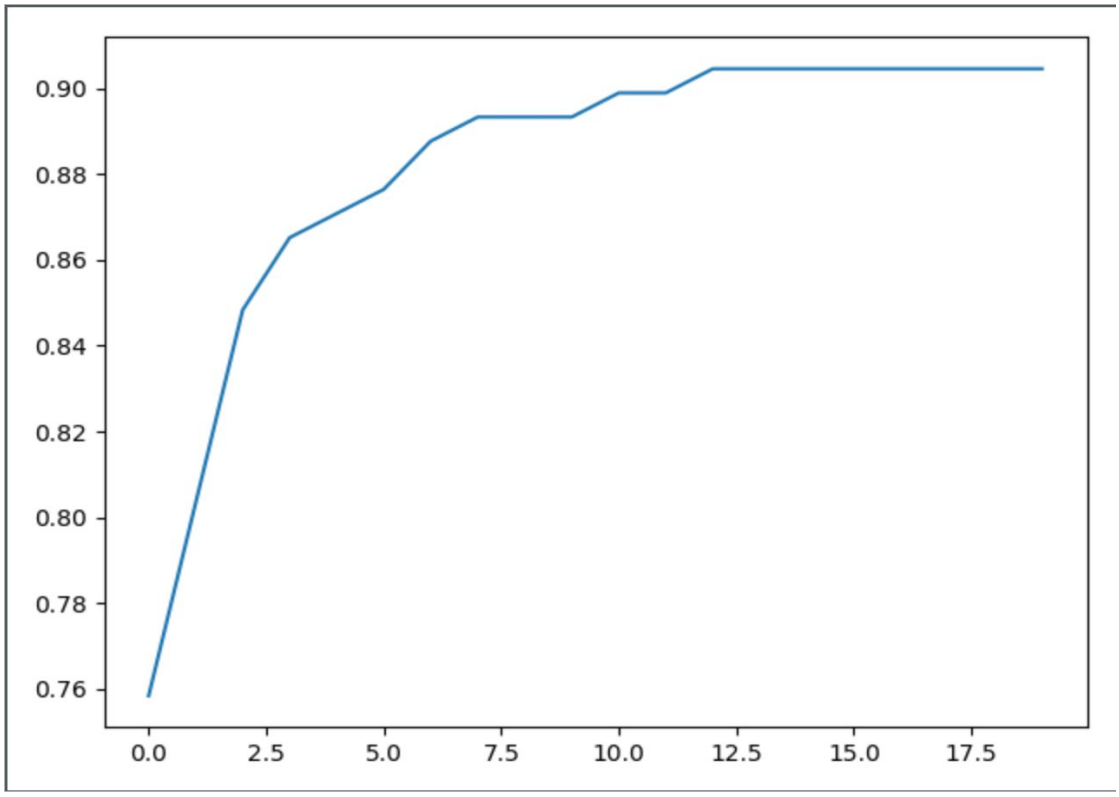
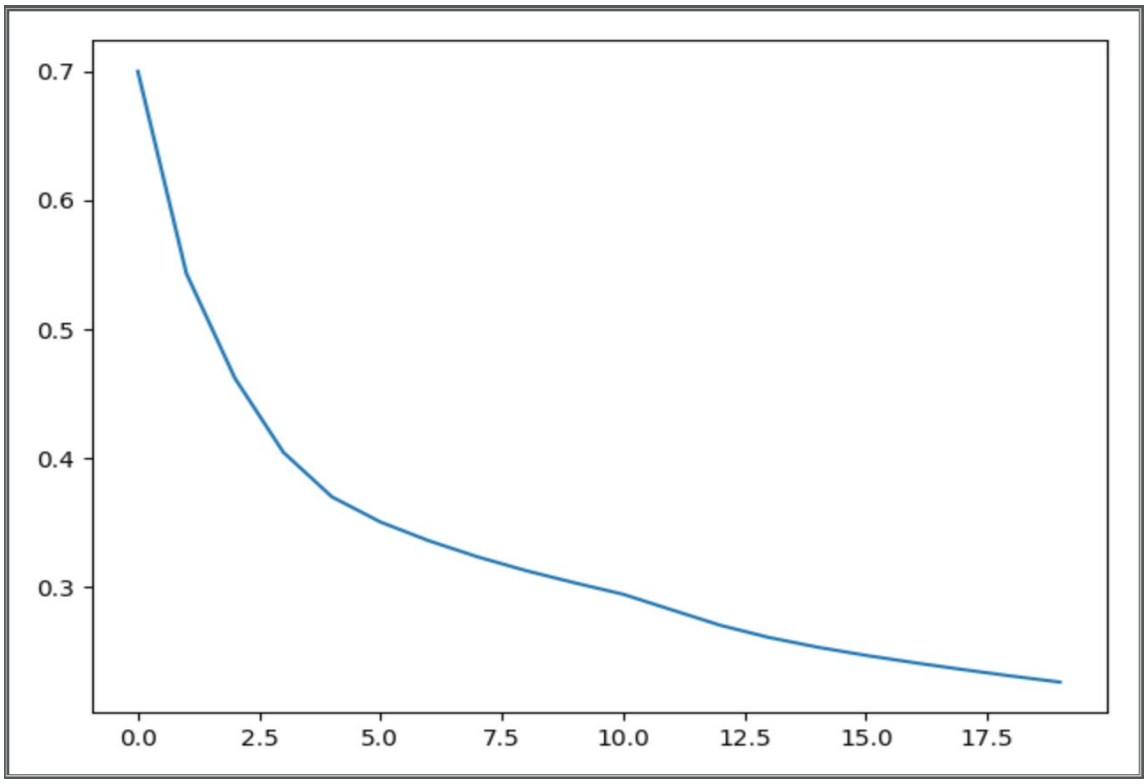
3)
Part1

```
[epoch= 4 ]: Loss= 0.7001817533340978
[epoch= 4 ]: acuracy= 0.7584269662921348
[epoch= 9 ]: Loss= 0.5433530249469045
[epoch= 9 ]: acuracy= 0.8033707865168539
[epoch= 14 ]: Loss= 0.46205285006488256
[epoch= 14 ]: acuracy= 0.848314606741573
[epoch= 19 ]: Loss= 0.40448746743403946
[epoch= 19 ]: acuracy= 0.8651685393258427
[epoch= 24 ]: Loss= 0.3699790388165232
[epoch= 24 ]: acuracy= 0.8707865168539326
[epoch= 29 ]: Loss= 0.3504468187993429
[epoch= 29 ]: acuracy= 0.8764044943820225
[epoch= 34 ]: Loss= 0.3358727523346474
[epoch= 34 ]: acuracy= 0.8876404494382022
[epoch= 39 ]: Loss= 0.3234389632263957
[epoch= 39 ]: acuracy= 0.8932584269662921
[epoch= 44 ]: Loss= 0.3127147251505344
[epoch= 44 ]: acuracy= 0.8932584269662921
[epoch= 49 ]: Loss= 0.3032990874940174
[epoch= 49 ]: acuracy= 0.8932584269662921
[epoch= 54 ]: Loss= 0.29440181405971877
[epoch= 54 ]: acuracy= 0.898876404494382
[epoch= 59 ]: Loss= 0.282337238446515
[epoch= 59 ]: acuracy= 0.898876404494382
[epoch= 64 ]: Loss= 0.27035784232760324
[epoch= 64 ]: acuracy= 0.9044943820224719
[epoch= 69 ]: Loss= 0.2609239704437733
[epoch= 69 ]: acuracy= 0.9044943820224719
[epoch= 74 ]: Loss= 0.2533746282600144
[epoch= 74 ]: acuracy= 0.9044943820224719
[epoch= 79 ]: Loss= 0.24700536078582486
[epoch= 79 ]: acuracy= 0.9044943820224719
[epoch= 84 ]: Loss= 0.24130391547134378
[epoch= 84 ]: acuracy= 0.9044943820224719
[epoch= 89 ]: Loss= 0.23598427584276796
[epoch= 89 ]: acuracy= 0.9044943820224719
[epoch= 94 ]: Loss= 0.230949695931474
[epoch= 94 ]: acuracy= 0.9044943820224719
[epoch= 99 ]: Loss= 0.22622396611444776
[epoch= 99 ]: acuracy= 0.9044943820224719
```

Accuracy Graph: Ascending, converge to 1



Error Graph: Descending, converge to 0



4)
set environment variables in the notebook using os.environ.
initializing TensorFlow to limit TensorFlow to first GPU.

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = ""
```

Load the MNIST images

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
full_data_x = mnist.train.images
```

We define the hyperparameters here.

```
num_steps = 50 # Total steps to train
batch_size = 1024 # The number of samples per batch
k = 25 # The number of clusters
num_classes = 10 # The 10 digits
num_features = 784 # Each image is 28x28 pixels
```

Define placeholder for inputs:

```
X = tf.placeholder(tf.float32, shape=[None, num_features])
Y = tf.placeholder(tf.float32, shape=[None, num_classes])
```

Initialize kmeans with its parameters:

```
kmeans = KMeans(inputs=X, num_clusters=k, distance_metric='cosine',
                use_mini_batch=True)
```

Build that kmeans:

```
training_graph = kmeans.training_graph()
```

According to our kmeans size we extract some features from it.

```
if len(training_graph) > 6: # Tensorflow 1.4+
  (all_scores, cluster_idx, scores, cluster_centers_initialized, cluster_centers_var, init_op, train_op) =
training_graph
else:
  (all_scores, cluster_idx, scores, cluster_centers_initialized, init_op, train_op) = training_graph
```

Define the average distance operation

```
cluster_idx = cluster_idx[0] # fix for cluster_idx being a tuple
avg_distance = tf.reduce_mean(scores)
```

Start a tf session

```
sess = tf.Session()
sess.run(init_vars, feed_dict={X: full_data_x})
sess.run(init_op, feed_dict={X: full_data_x})
```

Train the kmeans using inputs and compute the summary

```
for i in range(1, num_steps + 1):
    _, d, idx = sess.run([train_op, avg_distance, cluster_idx],
                         feed_dict={X: full_data_x})
    if i % 10 == 0 or i == 1:
        print("Step %i, Avg Distance: %f" % (i, d))
```

Result:

```
Step 1, Avg Distance: 0.341471
Step 10, Avg Distance: 0.221609
Step 20, Avg Distance: 0.220328
Step 30, Avg Distance: 0.219776
Step 40, Avg Distance: 0.219419
Step 50, Avg Distance: 0.219154
```

Each cluster map to a label

```
counts = np.zeros(shape=(k, num_classes))
for i in range(len(idx)):
    counts[idx[i]] += mnist.train.labels[i]
# Assign the most frequent label to the centroid
labels_map = [np.argmax(c) for c in counts]
labels_map = tf.convert_to_tensor(labels_map)
```

Compute accuracy

```
correct_prediction = tf.equal(cluster_label, tf.cast(tf.argmax(Y, 1), tf.int32))
accuracy_op = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Compute accuracy of our model on test data using the defined accuracy

```
test_x, test_y = mnist.test.images, mnist.test.labels
print("Test Accuracy:", sess.run(accuracy_op, feed_dict={X: test_x, Y: test_y}))
```

Result :

Test Accuracy: 0.7127