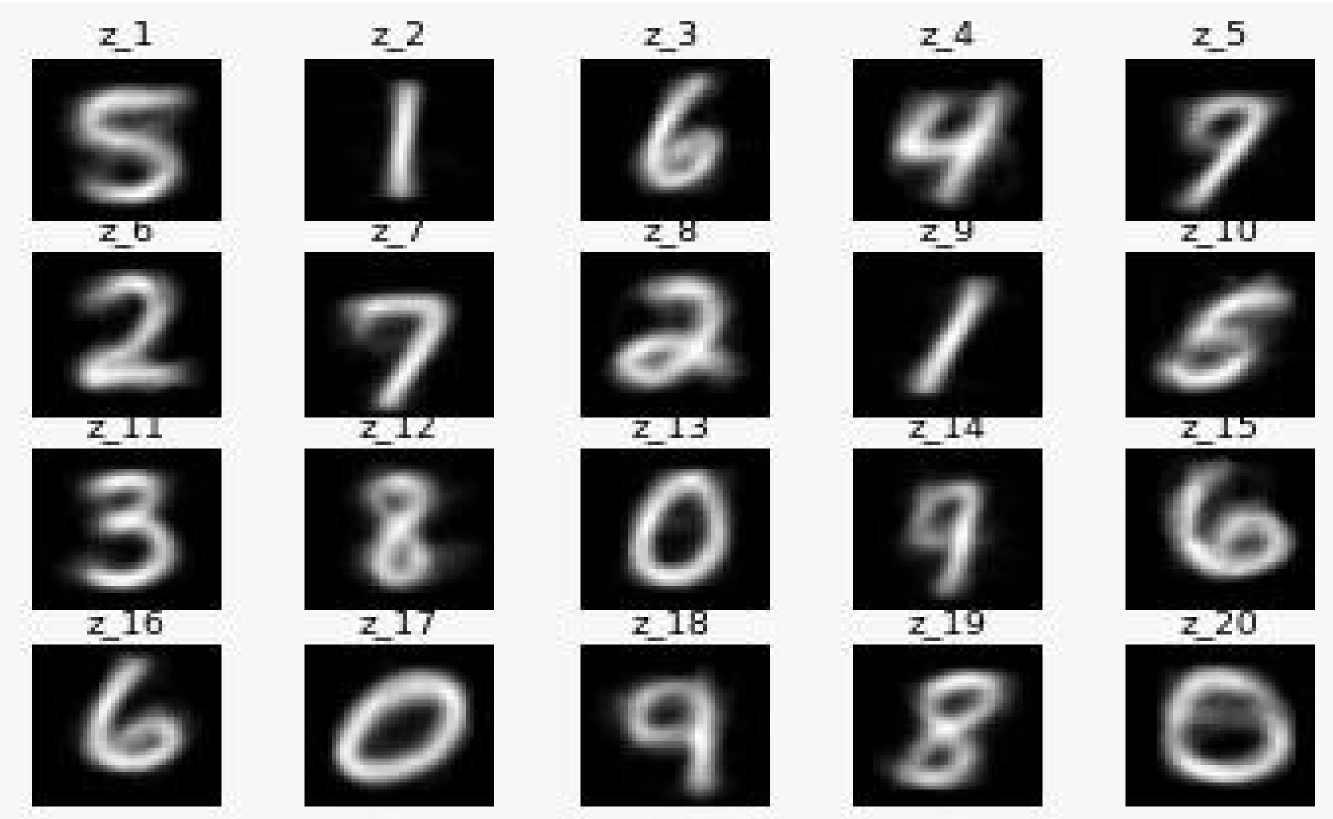
AI61003 LINEAR ALGEBRA FOR AI AND ML
ASSIGNMENT 01 - PROBLEM 09

N (the number of training sample) : **1000**
n (the dimension of vectors to be clustered) : **784**

CASE 1 Random Initialization of Cluster Representatives



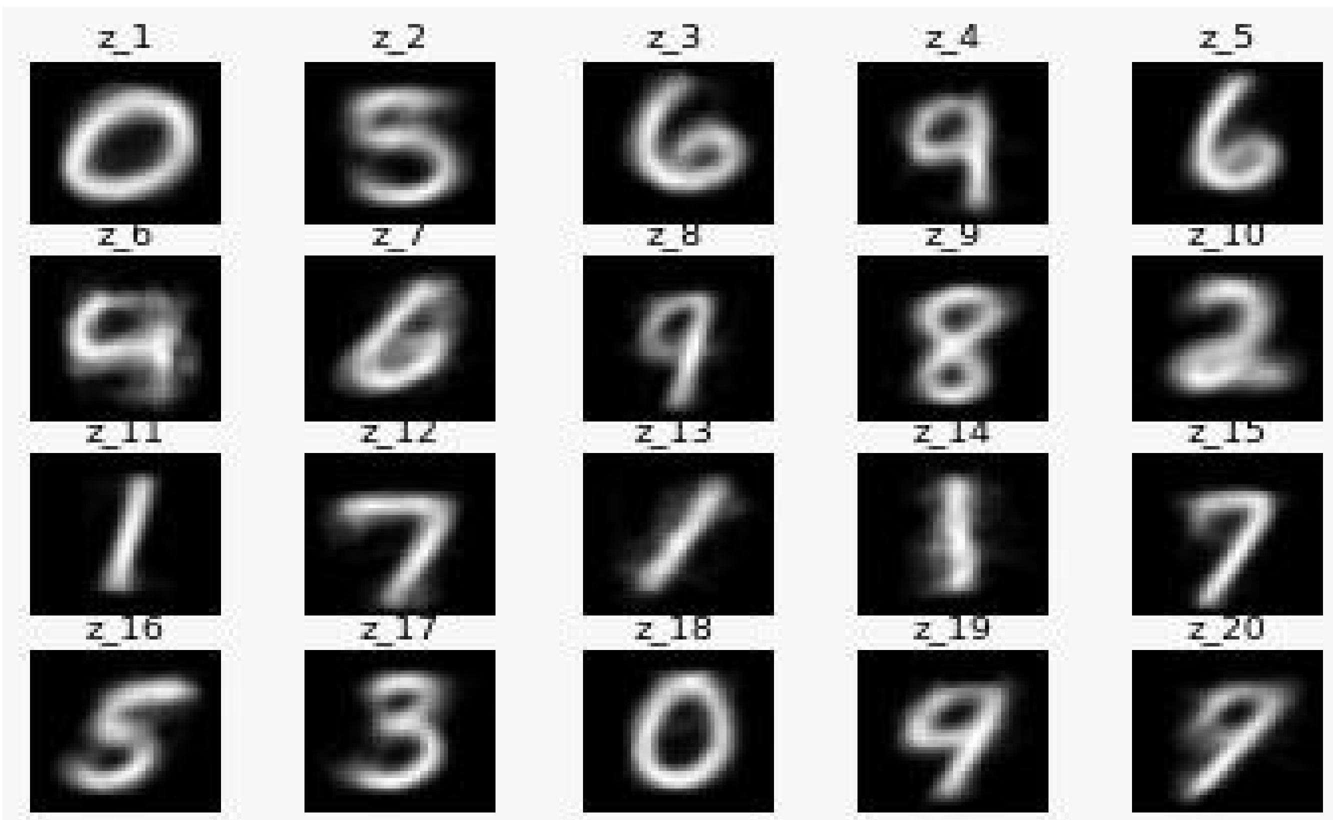
Number of iterations taken for the algorithm to converge : **18**
Accuracy of cluster assignment for a randomly selected set of 50 images : **68.00%**

Number of clusters (k)	J_{clust}
5	5.8968
6	5.8798
7	5.8824
8	5.8515
9	5.8549
10	5.8671
11	5.8752
12	5.9129
13	5.8592

14	5.8509
15	5.8954
16	5.8937
17	5.8979
18	5.8635
19	5.8869
20	5.8572

The optimal number of clusters is the one that gives the least value of the objective function J_{clust}. Therefore, **14** is the optimal cluster size.

CASE 2 Choosing Cluster Representatives from Given Dataset



Number of iterations taken for the algorithm to converge : **16**
Accuracy of cluster assignment for a randomly selected set of 50 images : **72.00%**

Number of clusters (k)	J_{clust}
5	6.5726
6	6.4847
7	6.4099
8	6.3861

9	6.3043
10	6.2270
11	6.1661
12	6.1165
13	6.1421
14	6.0424
15	6.0164
16	5.9788
17	5.9403
18	5.9260
19	5.9095
20	5.8953

The optimal number of clusters is the one that gives the least value of the objective function J_{clust}. Therefore, **20** is the optimal cluster size.

The choice of initial condition definitely has an effect on the performance of the clustering algorithm.

- Firstly the algorithm converges faster in the second case than in the first case. The converged value of inertia (or *within-cluster sum-of-squares criterion*) in the first case (for k = 20) was 35668 and in the second case was 35658. It shows that not only does the number of iterations get affected by the initial condition, but in the second case the algorithm may tend to find a more optimal clustering (converges towards global minima) than in the first case (might converge towards local minima). Having said this, though both the number of iterations and the converged criterion value are better for *data-dependent initialization* of cluster representatives, the relative difference between these values for the two cases is quite small. So, random initialization of cluster representatives can also give a good clustering of objects.
- The accuracy of cluster assignments was evaluated in both the cases on the same test set (constructed by randomly choosing 50 images from MNIST dataset that were not seen in training). It is clear that the accuracy of cluster assignment in the second case is 4% higher than in the first case. This shows that the initial condition can also have an impact on the performance of the produced clusters on the unseen test samples.
- If you analyze the change in the converging value of J_{clust} with the number of clusters, you would find that the converging value decreases (almost) monotonically in the second case, whereas in the first case there is a very irregular pattern. As a result of this, their optimal cluster sizes also come out to be very different (14 for random initialization and 20 for data-dependent initialization).

In [1]:

```
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
from random import shuffle, randint
import sklearn.cluster as Clustering_Algorithm
```

In [2]:

```
mnist_data = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist_data.load_data()
train_data = list(zip(x_train, y_train))
shuffle(train_data)
x_train, y_train = list(zip(*train_data))
```

In [3]:

```
train_images = [ [] for digit in range(10) ]
sample_count_per_digit = 100
for i, label in enumerate(y_train) :
    if ( len(train_images[label]) == sample_count_per_digit ) : continue
    train_images[label].append(x_train[i].flatten().astype(np.float32) / 255)

def Get_Random_Test_Data ( test_samples_count = 50 ) :
    global x_test, y_test

    test_data = list(zip(x_test, y_test))
    shuffle(train_data)
    shuffle(test_data)
    x_train, y_train = list(zip(*train_data))
    x_test_shuff, y_test_shuff = list(zip(*test_data))

    test_images = [ [] for digit in range(10) ]
    count = 0
    for i, label in enumerate(y_test_shuff) :
        if count == test_samples_count : break
        count += 1
        test_samples_count
        test_images[label].append(x_test_shuff[i].flatten().astype(np.float32) / 255)
    return test_images
```

In [4]:
```python
train = []
do = [train.extend(img) for img in train_images]
N = len(train)
n = train_images[0][0].shape[0]
print ('N (number of training samples) :', N)
print('n (dimension of vectors) :', n)
```

```
N (number of training samples) : 1000
n (dimension of vectors) : 784
```

In [5]:
```python
def Get_Distance ( a , b ) :
    return np.linalg.norm(b-a)
```

In [6]:
```python
def J_clust ( train , kmeans ) :
    s = 0
    for img in train :
        cluster_id = kmeans.predict(img.astype(np.float).reshape(1,-1))[0]
        s += Get_Distance(img, kmeans.cluster_centers_[cluster_id])
    return s / len(train)
```

CASE I : Random Initialization of Cluster Representatives

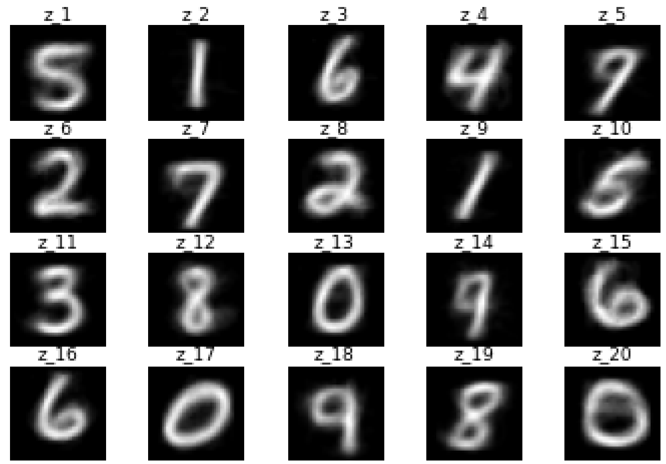
In [7]:
```
k = 20
kmeans = Clustering_Algorithm.KMeans(n_clusters = k, init='random', verbose=True, n_init=1)
kmeans.fit(train)
```

```
Initialization complete
Iteration 0, inertia 60732.524707579985
Iteration 1, inertia 38050.91288994666
Iteration 2, inertia 36886.65874578197
Iteration 3, inertia 36505.343118868186
Iteration 4, inertia 36233.53763940629
Iteration 5, inertia 36055.34772788261
Iteration 6, inertia 35925.409068793735
Iteration 7, inertia 35836.56467969832
Iteration 8, inertia 35781.09028613918
Iteration 9, inertia 35733.11039512983
Iteration 10, inertia 35708.260845345685
Iteration 11, inertia 35688.57368506976
Iteration 12, inertia 35678.89407890276
Iteration 13, inertia 35675.8489286663
Iteration 14, inertia 35672.49733949574
Iteration 15, inertia 35671.12267063208
Iteration 16, inertia 35669.257571344424
Iteration 17, inertia 35668.07054899751
Converged at iteration 17: strict convergence.
```

Out[7]:
```
KMeans(init='random', n_clusters=20, n_init=1, verbose=True)
```

In [8]:

```python
for cluster_id, rep in enumerate(kmeans.cluster_centers_) :
    plt.subplot(4, 5, cluster_id+1)
    cluster_rep_img = (rep*255).astype(np.int32).reshape((28,28))
    plt.imshow(cluster_rep_img, cmap='gray')
    plt.title('z_' + str(cluster_id+1), fontsize=9, pad=4)
    plt.axis('off')
plt.savefig('cluster_representative_plots_part_1.png')
```



In [9]:

```python
digits_represented_by_clusters = [5, 1, 6, 4, 9, 2, 7, 2, 1, 5, 3, 8, 0, 9, 6, 6, 0, 9, 8, 0]
```

In [10]:

```python
correctly_classified = 0
test_images_count = 50
test_images = Get_Random_Test_Data(test_images_count)
for dig in range(10) :
    for img in test_images[dig] :
        cluster_id = kmeans.predict(img.astype(np.float).reshape(1,-1))[0]
        correctly_classified += (digits_represented_by_clusters[cluster_id] == dig)

accuracy = 100 * correctly_classified / test_images_count
print('ACCURACY OF CLUSTER ASSIGNMENT :', round(accuracy, 3))
```

```
ACCURACY OF CLUSTER ASSIGNMENT : 68.0
```

In [11]:

```
least_J_value = float('inf')
opt_cluster_size = -1
for cluster_size in range(5, 21) :
    kmeans = Clustering_Algorithm.KMeans(n_clusters = k, init='random', verbose=False, n_init=1)
    kmeans.fit(train)
    J_value = J_clust(train, kmeans)
    print('CLUSTER SIZE :', cluster_size, '\tJ_clust :', round(J_value, 5))

    if J_value < least_J_value :
        least_J_value = J_value
        opt_cluster_size = cluster_size

print('OPTIMAL CLUSTER SIZE :', opt_cluster_size)
```

```
CLUSTER SIZE : 5        J_clust : 5.89679
CLUSTER SIZE : 6        J_clust : 5.87985
CLUSTER SIZE : 7        J_clust : 5.8824
CLUSTER SIZE : 8        J_clust : 5.85147
CLUSTER SIZE : 9        J_clust : 5.8549
CLUSTER SIZE : 10       J_clust : 5.86706
CLUSTER SIZE : 11       J_clust : 5.8752
CLUSTER SIZE : 12       J_clust : 5.91291
CLUSTER SIZE : 13       J_clust : 5.85923
CLUSTER SIZE : 14       J_clust : 5.85086
CLUSTER SIZE : 15       J_clust : 5.89545
CLUSTER SIZE : 16       J_clust : 5.8937
CLUSTER SIZE : 17       J_clust : 5.89795
CLUSTER SIZE : 18       J_clust : 5.86347
CLUSTER SIZE : 19       J_clust : 5.88693
CLUSTER SIZE : 20       J_clust : 5.85719
OPTIMAL CLUSTER SIZE : 14
```

CASE II : Data Dependent Initialization of Cluster Representatives

In [12]:

```
def Data_Dependent_Cluster_Representatives_Init ( train , k ) :
    clusters = []
    for i in range(k) : clusters.append([])
    for img in train :
        clusters[randint(0, k-1)].append(img)

    cluster_representatives = []
    for i in range(k) :
        rep = sum(clusters[i]) / len(clusters[i])
        cluster_representatives.append(rep)

    cluster_representatives = np.array(cluster_representatives)
    return cluster_representatives
```

In [13]:

```python
k = 20
kmeans = Clustering_Algorithm.KMeans(n_clusters = k, verbose=True, n_init=1,
                                     init=Data_Dependent_Cluster_Representatives_Init(train, k))
kmeans.fit(train)
```

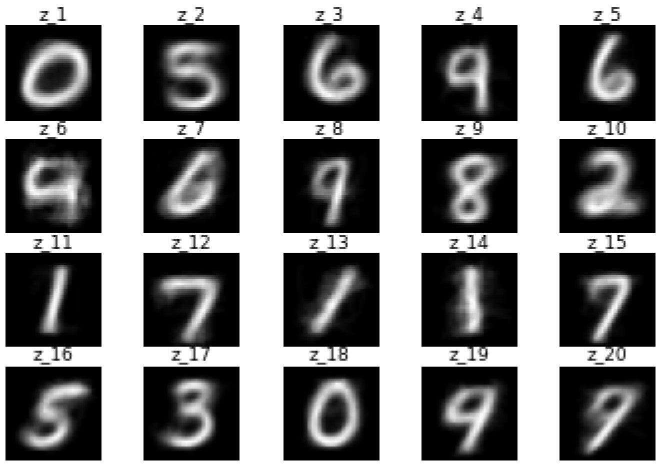
```
Initialization complete
Iteration 0, inertia 49601.48328326748
Iteration 1, inertia 39424.78061696993
Iteration 2, inertia 37262.66951403245
Iteration 3, inertia 36526.05086918052
Iteration 4, inertia 36129.293411810526
Iteration 5, inertia 35995.34874088044
Iteration 6, inertia 35928.17828729691
Iteration 7, inertia 35885.53611488616
Iteration 8, inertia 35829.41673297419
Iteration 9, inertia 35792.76307584839
Iteration 10, inertia 35752.53303179572
Iteration 11, inertia 35721.684346373004
Iteration 12, inertia 35687.63237248872
Iteration 13, inertia 35672.999806197186
Iteration 14, inertia 35664.05230663018
Iteration 15, inertia 35658.41160480236
Converged at iteration 15: strict convergence.
```

Out[13]:

```
KMeans(init=array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32),
       n_clusters=20, n_init=1, verbose=True)
```

In [15]:

```python
for cluster_id, rep in enumerate(kmeans.cluster_centers_) :
    plt.subplot(4, 5, cluster_id+1)
    cluster_rep_img = (rep*255).astype(np.int32).reshape((28,28))
    plt.imshow(cluster_rep_img, cmap='gray')
    plt.title('z_' + str(cluster_id+1), fontsize=9, pad=4)
    plt.axis('off')
plt.savefig('cluster_representative_plots_part_2.png')
```



In [16]:

```python
digits_represented_by_clusters = [0, 5, 6, 9, 6, 9, 6, 9, 8, 2, 1, 7, 1, 1, 7, 5, 3, 0, 9, 9]
```

In [17]:

```python
correctly_classified = 0
test_images_count = 50
for dig in range(10) :
    for img in test_images[dig] :
        cluster_id = kmeans.predict(img.astype(np.float).reshape(1,-1))[0]
        correctly_classified += (digits_represented_by_clusters[cluster_id] == dig)

accuracy = 100 * correctly_classified / test_images_count
print('ACCURACY OF CLUSTER ASSIGNMENT :', round(accuracy, 3))
```

```
ACCURACY OF CLUSTER ASSIGNMENT : 72.0
```

In [18]:

```
least_J_value = float('inf')
opt_cluster_size = -1
for cluster_size in range(5, 21) :
    kmeans = Clustering_Algorithm.KMeans(n_clusters = cluster_size, verbose=False, n_init=1,
                                         init=Data_Dependent_Cluster_Representatives_Init(train, cluster_size))
    kmeans.fit(train)
    J_value = J_clust(train, kmeans)
    print('CLUSTER SIZE :', cluster_size, '\tJ_clust :', round(J_value, 5))

    if J_value < least_J_value :
        least_J_value = J_value
        opt_cluster_size = cluster_size

print('OPTIMAL CLUSTER SIZE :', opt_cluster_size)
```

```
CLUSTER SIZE : 5        J_clust : 6.57261
CLUSTER SIZE : 6        J_clust : 6.48468
CLUSTER SIZE : 7        J_clust : 6.40986
CLUSTER SIZE : 8        J_clust : 6.38606
CLUSTER SIZE : 9        J_clust : 6.30426
CLUSTER SIZE : 10       J_clust : 6.22705
CLUSTER SIZE : 11       J_clust : 6.16606
CLUSTER SIZE : 12       J_clust : 6.11653
CLUSTER SIZE : 13       J_clust : 6.14211
CLUSTER SIZE : 14       J_clust : 6.04239
CLUSTER SIZE : 15       J_clust : 6.0164
CLUSTER SIZE : 16       J_clust : 5.97875
CLUSTER SIZE : 17       J_clust : 5.94029
CLUSTER SIZE : 18       J_clust : 5.92599
CLUSTER SIZE : 19       J_clust : 5.90952
CLUSTER SIZE : 20       J_clust : 5.8953
OPTIMAL CLUSTER SIZE : 20
```