Assignment 7: Unsupervised learning (PCA, K-Means)

Introduction

In this assignment, you will need to compute the Principal Component Analysis and K-Means algorithms and use them on a dataset. The dataset is the set of images from MNIST database corresponding to the handwritten digit 7. Each image is $28px \times 28px$. The set is divided in a training set and a testing set of respective size 3133 and 3132.

As usual, the structure of the code is given to you and you need to fill the parts corresponding to the questions below.

Questions

PCA Starts here

Question 1 (25%) Complete the functions `pca(.)` and `pca_project(.)`. For information, the function `np.linalg.eigh` compute the eigenvalues and eigenvectors of a symmetric matrix. It returns two arrays, the first one contains the eigenvalues in ascending order and the second one the corresponding eigenvector.
Question 2 (15%) Use the function `pca(.)` to learn a decomposition on the **training set**. Then compute the reconstruction error E on the **testing set**, for a number of components varying from 1 to 100, as defined by:

$$E(D) = \frac{1}{N} \sum_{n=1}^{N} \|I_n - (\mu + \sum_{k=1}^{D} \omega_k^n \mathbf{u}_k)\|_2,$$

with I_n denoting the n-th image of the testing set, μ is the mean digit learnt from the training set, $\mathbf{u}_k$ is the eigenvector with the k-th largest eigenvalue, and ω_k^n is the expansion coefficient of the n-th image on the k-th eigenvector. Finally, $\|.\|_2$ denotes the L_2 norm. Numpy has the method `np.linalg.norm(.)` that computes norms (check out the documentation for more infos).
Question 3 (5%) Plot the evolution of the error E for $D = 1, \dots, 100$.

K-means Starts here

Question 4 (10%) Complete the function `distortion(.)` which computes the distortion cost F for a given clustering of the data:

$$F(m, c) = \frac{1}{N} \sum_{i=1}^{N} \|x^i - c^{m(i)}\|_2,$$

where N corresponds to the total number of images in the set and $m(i)$ denotes which cluster is assigned to the image x^i.
Question 5 (15%) Complete the functions `kmeans(.)` and `assign_cluster(.)`, make sure that it computes the distortion after each update. Then use the function on your training set, the number of cluster $k = 2$. Check that the distortion decreases as the algorithm progresses.
Question 6 (15%) In order to mitigate the local minima problem of K-Means, repeat the algorithm 10 times, and keep the solution that yields the smallest distortion at the end. Show the resulting digit clusters (centroids of your clusters) using the `plot_kmeans(.)` function given.
Question 7 (10%) Repeat the procedure of Question 6 for values of $k = 3, 4, 5, 10, 50, 100$ (Allow for ~10min). Plot the evolution of the distortion cost of the training and testing data. Remember to use the functions `select_clustering(.)`, `assign_cluster(.)`, and `distortion(.)` defined earlier.

Comparison Starts here

Question 8 (5%) Compare the results from PCA to the results of K-means on the **test set** by plotting on the same graph the reconstruction error $E(D)$ for $D = 3, 4, 5, 10, 50, 100$ and the distortion cost you just computed (remark that the two measures are simply L_2 norms thus the comparison is valid). To be clear, the first one measure the error in the reconstructed image from the projection on the components of PCA, the second measure the error between each image and the centroid of the cluster it is assigned to. Both correpond to the error made when approximating the original image to either its projection or its cluster's centroid.

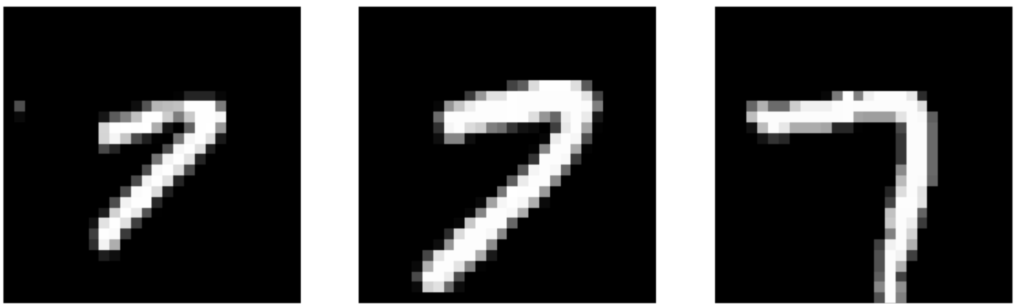
Importing necessary packages

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mnist import read, show

%matplotlib inline
```

Importing the data to form training and test sets

In [2]:
```
# Reads in the data from MNIST database
data = read()

# Retrieve the entries corresponding to the digit 7
samples = []
for sample in data:
    if sample[0] == 7:
        samples.append(sample[1].astype(float))

# Stack images in a tensor of size 28x28xnb_images
samples = np.stack(samples,axis=2)

# Defines training and testing set
train_set = samples[:,:,:3133]
test_set = samples[:,:,3133:]
print(train_set.shape, test_set.shape)

# Plot some images
fig, axes = plt.subplots(1,3,figsize=(15,100))
plt.rcParams['image.cmap'] = 'gray'
axes[0].imshow(train_set[:,:,1])
axes[1].imshow(train_set[:,:,100])
axes[2].imshow(train_set[:,:,1000])
axes[0].axis('off')
axes[1].axis('off')
axes[2].axis('off')

# Transform the data for processing, i.e. unroll the 28x28 images in vec
tors of size (28*28)x1
X = np.reshape(train_set,(28*28,3133)).T
Y = np.reshape(test_set,(28*28,3132)).T
```

```
(28, 28, 3133) (28, 28, 3132)
```



Principal Component Analysis

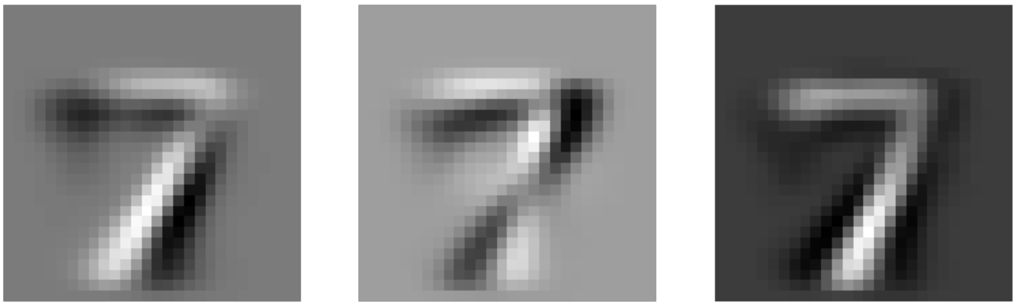
The data has now been initialised, everything is set to start on coding.

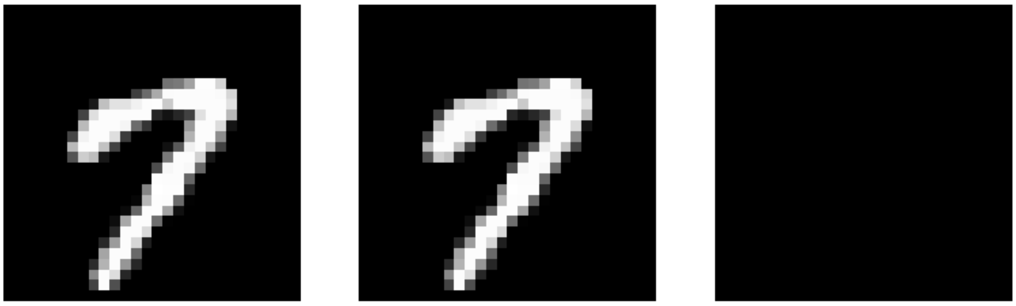
Question 1. Complete the functions `pca(.)` and `pca_project(.)`. For information, the function `np.linalg.eigh` compute the eigenvalues and eigenvectors of a symmetric matrix. It returns two arrays, the first one contains the eigenvalues in ascending order and the second one the corresponding eigenvector.

In [3]:
```
def pca(X,n_components = None):

    # If no number of component is specified, the function keeps them al
l
    if n_components is None:
        n_components = X.shape[1]

    ###################### TO DO QUESTION 1
################################

    # Compute mean digit and shift the data

    # Compute covariance of the data

    # Compute the eigenvector of the covariance matrix

    # Retrieve the eigenvectors to return

    ###################### TO DO QUESTION 1
################################

    # Returns the transformed data, the principal components, and the me
an digit
    return X_mean, components

def pca_project(Y,X_mean,components):
    # Compute the projection of the input data on the selected component
s

    ###################### TO DO QUESTION 1
################################

    # Compute the expansion coefficients of the data

    ###################### TO DO QUESTION 1
################################

    return X_mean + reconstruction
```

In [4]:
```
# This tests if your functions are correct, you should get the same outp
ut as we do
X_mean, components = pca(X,n_components=None)

# Reshapes the reconstructed data to have 28x28 pictures
comp_ = np.reshape(components,(28,28,784))

fig, axes = plt.subplots(1,3,figsize=(15,100))
plt.rcParams['image.cmap'] = 'gray'
axes[0].imshow(comp_[:,:,-1])
axes[1].imshow(comp_[:,:,-2])
axes[2].imshow(comp_[:,:,-3])
axes[0].axis('off')
axes[1].axis('off')
axes[2].axis('off')

X_projected = pca_project(X,X_mean,components)

X_ = np.reshape(X_projected.T,(28,28,3133))
fig, axes = plt.subplots(1,3,figsize=(15,100))
plt.rcParams['image.cmap'] = 'gray'
axes[0].imshow(train_set[:,:,0])
axes[1].imshow(X_[:,:,0])
axes[2].imshow(train_set[:,:,0]-X_[:,:,0]>10**(-12))
axes[0].axis('off')
axes[1].axis('off')
axes[2].axis('off')
```

Out[4]:
```
(-0.5, 27.5, 27.5, -0.5)
```





Testing PCA

You now have a (hopefully) working implementation of the Principal Component Analysis algorithm. Use it to fit your training set and observe the results on the testing set.

Question 2. Use the function `pca(.)` to learn a decomposition on the **training set**. Then compute the reconstruction error E on the **testing set**, for a number of components varying from 1 to 100, as defined by:

$$E(D) = \frac{1}{N} \sum_{n=1}^{N} \|I_n - (\mu + \sum_{k=1}^{D} \omega_k^n \mathbf{u}_k)\|_2,$$

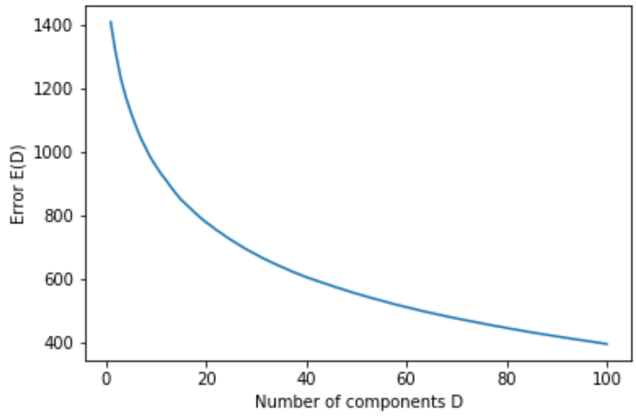
with I_n denoting the n-th image of the testing set, μ is the mean digit learnt from the training set, $\mathbf{u}_k$ is the eigenvector with the k-th largest eigenvalue, and ω_k^n is the expansion coefficient of the n-th image on the k-th eigenvector. Finally, $\|.\|_2$ denotes the L_2 norm. Numpy has the method `np.linalg.norm(.)` that computes norms (check out the documentation for more infos).

In [5]:
```
max_n_components = 100 # Max number of components to keep

xrange = range(1,max_n_components+1)
error = []


##################### TO DO QUESTION 2 ##############################

# Compute the Error for n_components between 1 and 100

##################### TO DO QUESTION 2 ##############################
```

Question 3. Plot the evolution of the error E for $D = 1, \ldots, 100$.

In [6]:
```
##################### TO DO QUESTION 3 ##############################

# Plot the error with respect to n_components

##################### TO DO QUESTION 3 ##############################
```

Out[6]:
```
Text(0,0.5,'Error E(D)')
```



K-Means

In this section, you will complete the implementation of the k-means algorithm.

Question 4. Complete the function `distortion(.)` which computes the distortion cost F for a given clustering of the data:

$$F(m, c) = \frac{1}{N} \sum_{i=1}^{N} \|x^i - c^{m(i)}\|_2,$$

where N corresponds to the total number of images in the set and $m(i)$ denotes which cluster is assigned to the image x^i.

In [7]:
```
def distortion(X, cluster_assignment, centroids):

    n_cluster, n_variables = centroids.shape
    distortion = 0

    #################### TO DO QUESTION 4
##############################

    # Compute distortion

    #################### TO DO QUESTION 4
##############################

    return distortion/X.shape[0]
```

Question 5. Complete the functions `K-means` and `assign_cluster(.)` , make sure that it computes the distortion after each update. Then use the function on your training set, the number of cluster $k = 2$. Check that the distortion decreases as the algorithm progresses.

In [8]:
```
def assign_cluster(centroids, X):

    n_observations, _ = X.shape
    # Initialise cluster_assignment to -1
    cluster_assignment = -1*np.ones((n_observations,))

    for i in range(n_observations):
        ##################### TO DO QUESTION 5
##############################


        ##################### TO DO QUESTION 5
##############################

    return cluster_assignment

def kmeans(X, n_clusters = 2, max_iter =1000, tol = 10**-10, verbose = F
alse):

    n_observations, n_variables = X.shape

    # Randomly initialise the centroids using the multivariate gaussian
computed from the data
    X_mean = np.mean(X,axis=0)
    X_cov = np.cov(X,rowvar=False)
    centroids = np.random.multivariate_normal(X_mean,X_cov,(n_cluster
s,))

    n_iter = 0
    distortion_scores = []
    # Loop as long as the number of iterations is below max_iter and if
the converging criteria has not be met
    while (n_iter < max_iter):
        n_iter += 1
        # Step 1: assign points to nearest center
        cluster_assignment = assign_cluster(centroids,X)

        # Step 2: compute distortion
        dist = distortion(X, cluster_assignment, centroids)
        distortion_scores.append(dist)
        if verbose:
            print("Iteration %s, distortion = %s" % (n_iter,dist))

        # Step 3: compute new centroids from the clusters
        new_centroids = np.zeros(centroids.shape)
        for j in range(n_clusters):

            ##################### TO DO QUESTION 5
##############################


            ##################### TO DO QUESTION 5
##############################

        # Step 4: break the loop if difference between previous centroid
s and new ones is small enough
        if np.linalg.norm(new_centroids-centroids)<tol:
            if verbose:
                print("Terminates with difference: %s\n" % np.linalg.nor
m(new_centroids-centroids))
            break
        else:
            centroids = new_centroids

    return cluster_assignment, centroids, distortion_scores
```

In [9]:
```
# If your implementation is correct you should get the same results as u
s here
np.random.seed(11) # DO NOT CHANGE THIS LINE, it ensures your random ini
tialisation is identical to ours

cluster_assignment, centroids, distortion_scores = kmeans(X,verbose=Tru
e)
```

```
Iteration 1, distortion = 2090.7481389717063
Iteration 2, distortion = 1452.3091110197047
Iteration 3, distortion = 1447.6490543130699
Iteration 4, distortion = 1446.005894465188
Iteration 5, distortion = 1444.893372279777
Iteration 6, distortion = 1444.208120556854
Iteration 7, distortion = 1443.8045936541398
Iteration 8, distortion = 1443.6018478205244
Iteration 9, distortion = 1443.5087750077328
Iteration 10, distortion = 1443.434478685969
Iteration 11, distortion = 1443.3888422890182
Iteration 12, distortion = 1443.365125904968
Iteration 13, distortion = 1443.3344999721164
Iteration 14, distortion = 1443.284897951173
Iteration 15, distortion = 1443.267950454885
Iteration 16, distortion = 1443.2627485947248
Terminates with difference: 0.0
```

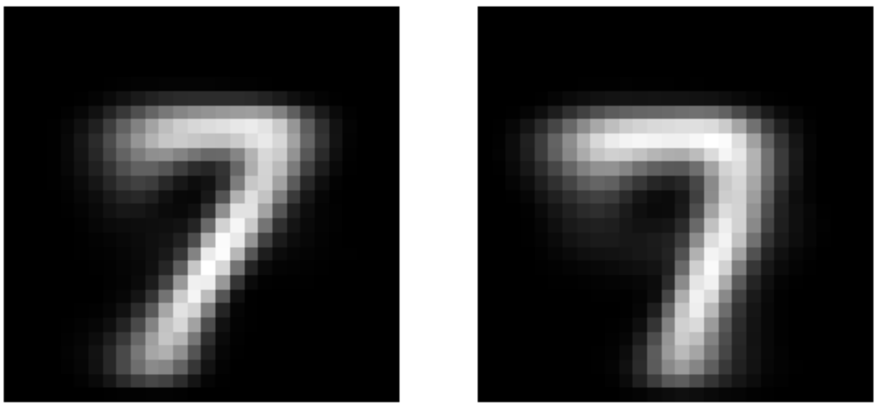
In [10]:
```
# Helper function to plot multiple images (non-graded)
def plot_kmeans(centroids, n = 4):

    k = centroids.shape[0]
    m = int(np.ceil(k/n))
    fig, axes = plt.subplots(m,n,figsize=(n*5,m*5))
    plt.rcParams['image.cmap'] = 'gray'
    for c in range(k):
        if m == 1:
            axes[c].imshow(np.reshape(centroids[c,:],(28,28)))
            axes[c].axis('off')
        else:
            i, j = int(c/n), int(c - i*n)
            axes[i,j].imshow(np.reshape(centroids[c,:],(28,28)))
            axes[i,j].axis('off')

    for c in range(k,m*n):
        if m == 1:
            axes[c].remove()
            axes[c].axis('off')
        else:
            i, j = int(c/n), int(c - i*n)
            axes[i,j].remove()
            axes[i,j].axis('off')
```

Testing K-means

Question 6. In order to mitigate the local minima problem of K-Means, repeat the algorithm 10 times, and keep the solution that yields the smallest distortion at the end. Show the resulting digit clusters (centroids of your clusters) using the `plot_kmeans(.)` function given.

In [11]:
```
def select_clustering(X,k=2,repeats=10):
    # Returns clustering with lowest distortion across "repeats" number of runs
    clustering = None
    print("Run k=%s:" % k,end=' ')


    for i in range(repeats):
        print("%s/%s.." % (i+1,repeats), end=' ')
        np.random.seed(i) # Do not change this line, it insures you get the same random initialisation as us

        #################### TO DO QUESTION 6 ##############################

        # Compute clusters and retrieve the one with lowest distortion

        #################### TO DO QUESTION 6 ##############################

    print("\n",end='')
    return clustering


clustering = select_clustering(X)

#################### TO DO QUESTION 6 ##############################

# Plot the centres of the clusters

#################### TO DO QUESTION 6 ##############################
```

```
Run k=2: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10.. 1
0/10..
```



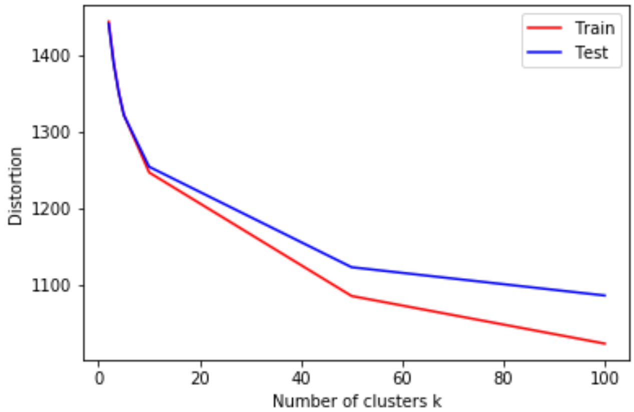
Question 7. Repeat the procedure of Question 6 for values of $k = 3, 4, 5, 10, 50, 100$ (Allow for ~10min). Plot the evolution of the distortion cost of the training and testing data. Remember to use the functions `select_clustering(.)` , `assign_cluster(.)` , and `distortion(.)` defined earlier.

In [12]:
```
train_distortions = []
test_distortions = []

ks = [2,3,4,5,10,50,100]


##################### TO DO QUESTION 7 ##############################


##################### TO DO QUESTION 7 ##############################
```

```
Run k=2: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10.. 1
0/10..
Run k=3: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10.. 1
0/10..
Run k=4: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10.. 1
0/10..
Run k=5: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10.. 1
0/10..
Run k=10: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10..
10/10..
Run k=50: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10..
10/10..
Run k=100: 1/10.. 2/10.. 3/10.. 4/10.. 5/10.. 6/10.. 7/10.. 8/10.. 9/10..
10/10..
```

In [13]:
```
# Plotting the evolution of distortion for train and test set
fig = plt.figure()

##################### TO DO QUESTION 7 ##############################


##################### TO DO QUESTION 7 ##############################

plt.xlabel("Number of clusters k")
plt.ylabel("Distortion")
plt.legend(['Train', 'Test'])
```

Out[13]: <matplotlib.legend.Legend at 0x10cd25f98>



Comparison

Question 8. Compare the results from PCA to the results of K-means on the **test set** by plotting on the same graph the reconstruction error $E(D)$ for $D = 3, 4, 5, 10, 50, 100$ and the distortion cost you just computed (remark that the two measures are simply L_2 norms thus the comparison is valid). To be clear, the first one measure the error in the reconstructed image from the projection on the components of PCA, the second measure the error between each image and the centroid of the cluster it is assigned to. Both correpond to the error made when approximating the original image to either its projection or its cluster's centroid.

In [14]:
```
# No need to recompute kmeans here, just compute PCA for the correspondi
ng values of k (similar to question 2/3)

n_components = [2,3,4,5,10,50,100] # Max number of components to keep

error = []

##################### TO DO QUESTION 8 ##############################


##################### TO DO QUESTION 8 ##############################
```

In [15]:
```
# Plotting the comparison

fig = plt.figure()

##################### TO DO QUESTION 8 ##############################


##################### TO DO QUESTION 8 ##############################

plt.xlabel("Number of clusters/components (k/D)")
plt.ylabel("Approximation errors")

plt.legend(['PCA', 'K-means'])
```

Out[15]:
```
<matplotlib.legend.Legend at 0x121bf7c18>
```