CSE152 HW2

November 5, 2019

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from torch import nn
        import torch
        from torchvision.datasets import MNIST
        import torchvision.transforms as transforms
        from sklearn.metrics import accuracy_score
        %matplotlib inline
```

For this homework you will be using `pytorch` and `torchvision` library for neural networks and datasets. You can install them with `pip install torch torchvision`.

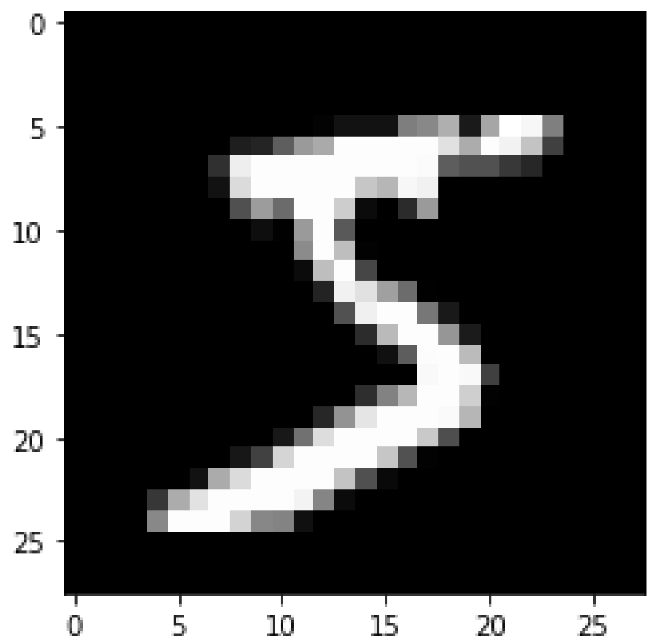
1 Question 1 Principal Component Analysis

This problem will guide you through the principal component analysis. You will be using a classical dataset, the MNIST hand written digit dataset.

```
In [2]: # Load the MNIST dataset
        mnist = MNIST('.', download=True)
        data = mnist.train_data.numpy()
        labels = mnist.train_labels.numpy()
        print('shapes:', data.shape, labels.shape)
        plt.imshow(data[0], cmap='gray')
        print('label:', labels[0])
```

```
shapes: (60000, 28, 28) (60000,)
label: 5
```

```
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/torchvision/datasets/mnis
  warnings.warn("train_data has been renamed data")
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/torchvision/datasets/mnis
  warnings.warn("train_labels has been renamed targets")
```



1.1 Question 1.1 Familiarize yourself with the data [5pt]

For this task, you will be using the torchvision package that provides the MNIST dataset. For each digit class(0-9), plot 1 image from the class and store those 10 images for each digit class in the array `digit_images`.

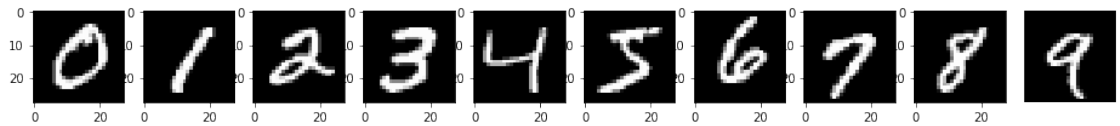
```
In [3]: digit_count = np.ones((10)) * -1
        digit_images = np.zeros([10, 28, 28])
        ### YOUR CODE HERE
        for image, label in zip(data, labels):
            if np.sum(digit_count) < 10:
                if digit_count[label] == 1:
                    continue
                else:
                    digit_images[label] = image
                    digit_count[label] = 1
            else:
                break

        fig = plt.figure(figsize = (16, 16))

        count = 1
        rows, cols = 1, 10

        for img in digit_images:
                fig.add_subplot(rows, cols, count)
```

```
            plt.imshow(img / 255, cmap='gray')
            count += 1
    plt.axis('off')
    plt.show()
    ### END OF CODE
```



1.2 Question 1.2 PCA

The following questions will guide you through the PCA algorithm.

1.2.1 Question 1.2.1 Centering the data [5pt]

For each image, flatten it to a 1-D vector. To perform PCA on the dataset, we first move the data points so they have 0 mean on each dimension. Store the centered data in variable `data_centered` and the mean of each dimension in variable `data_mean`.

```
In [4]: data_centered = None
        data_mean = None
        ### YOUR CODE HERE
        data = data.reshape(data.shape[0], 784)
        data_mean = np.mean(data, axis=0)
        data_centered = data - data_mean
        ### END OF CODE
```

1.2.2 Question 1.2.2 Compute the covariance matrix of the data [5pt]

You need to store the covariance matrix of the data in variable `data_covmat`. You may **not** use numpy.cov

```
In [5]: data_covmat = None
        ### YOUR CODE HERE
        data_covmat = (1 / data.shape[0]) * np.dot(data_centered.T, data_centered)
        ### END OF CODE
```

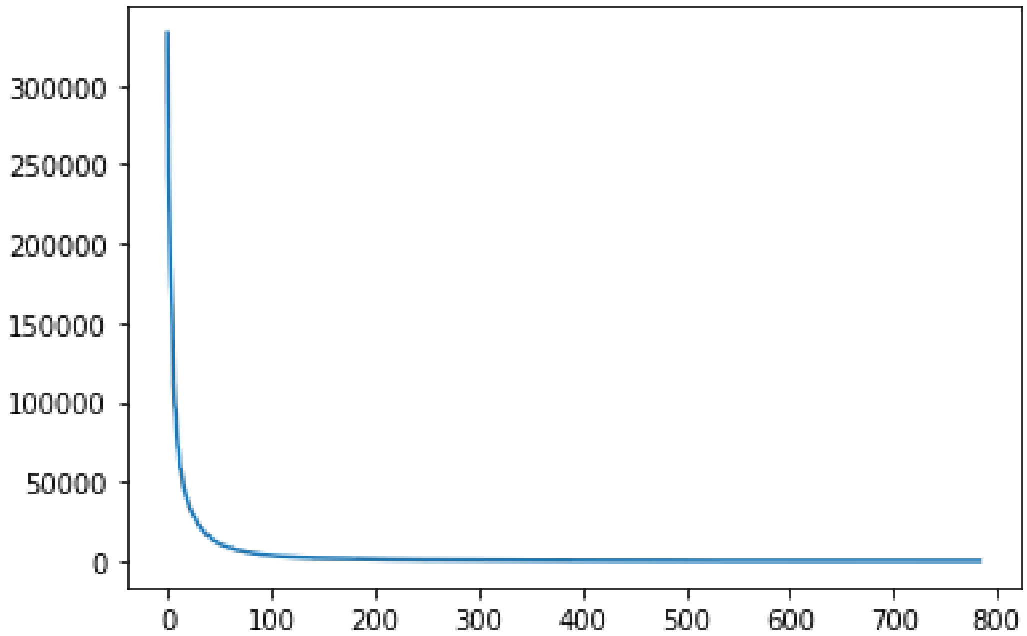
1.2.3 Question 1.2.3 Compute the eigenvalues of the covariance matrix [5pt]

You need to store the eigenvalues of the covariance matrix in variable `covmat_eig`, sorted in descending order. Then you need to plot the eigenvalues with `plt.plot`. You can use any numpy function.

```
In [6]: covmat_eig = None
        ### YOUR CODE HERE
        values, vectors = np.linalg.eig(data_covmat)
        plt.plot(values)
        ### END OF CODE
```

```
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/numpy/core/_asarray.py:85
  return array(a, dtype, copy=False, order=order)
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x143f7e610>]
```



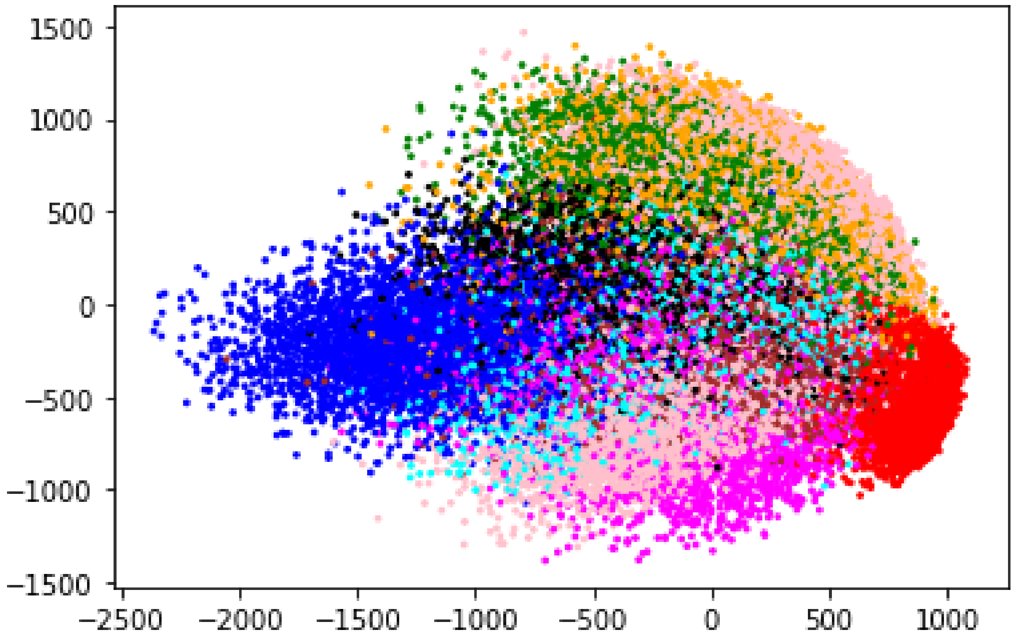
1.2.4 Question 1.2.4 Project data onto the first 2 principal components [5pt]

Now you need to project the centered data on the 2D space formed by the eigenvectors corresponding to the 2 largest eigenvalues. Create a 2D scatter plot where you need to assign a unique color to each digit class.

```
In [7]: ### YOUR CODE HERE
        colors = ['blue', 'red', 'magenta', 'pink', 'green', 'cyan', 'black', 'pink', 'brown',
        colors_plot = [colors[label] for label in labels]
        pca_proj = np.dot(data_centered, vectors[: , :2])
        plt.scatter(pca_proj[:, 0], pca_proj[:, 1], c=colors_plot, s=3)
        ### END OF CODE
```

```
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/numpy/core/_asarray.py:13
  return array(a, dtype, copy=False, order=order, subok=True)
```

```
Out[7]: <matplotlib.collections.PathCollection at 0x15c4da990>
```



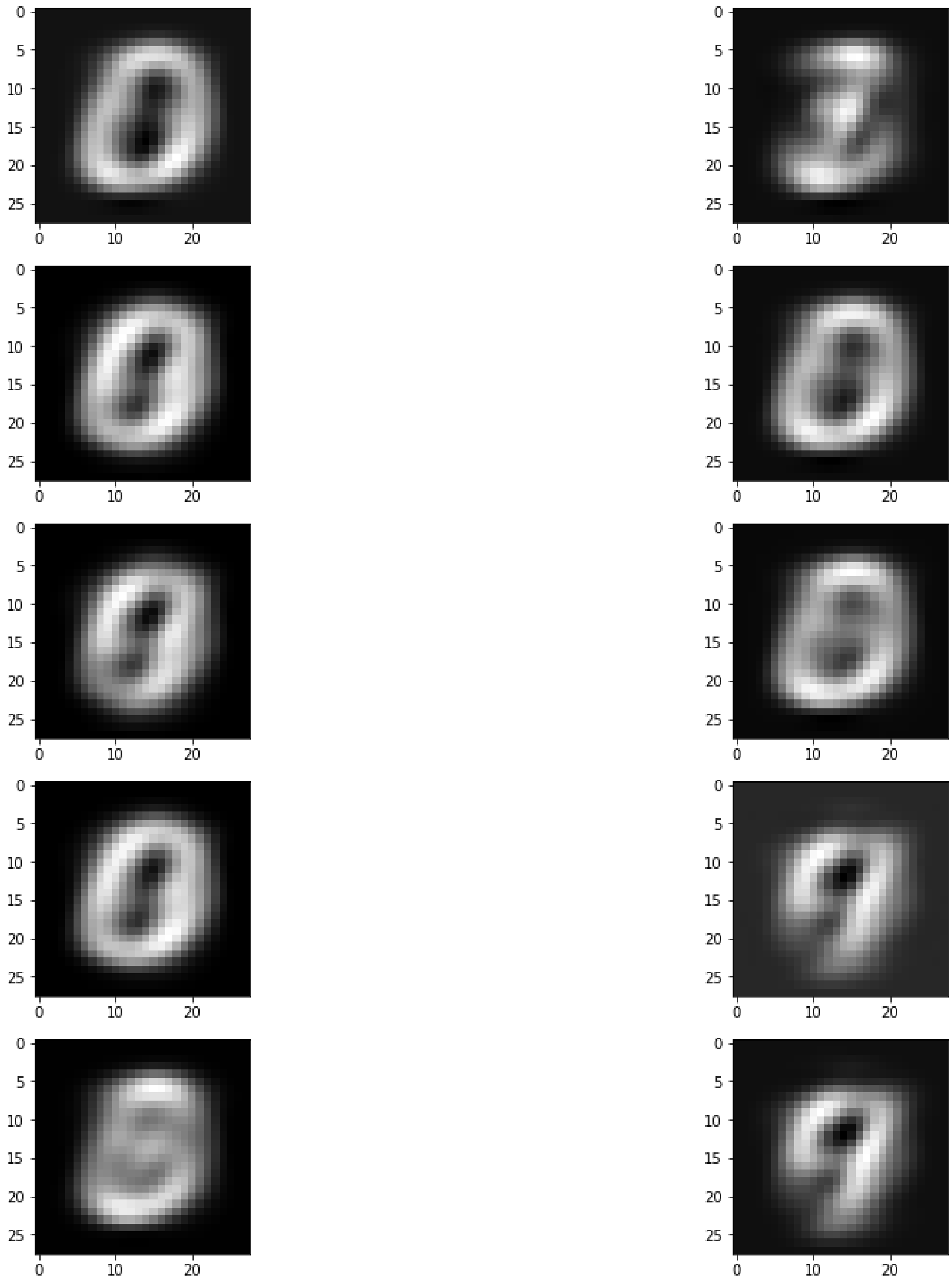
1.2.5 Question 1.2.5 Unproject data back to high dimensions [10pt]

For this question, you need to project the 10 images you plotted in **1.1** on the first 2 principal components, and then unproject the "compressed" 2-D representations back to the original space. Plot the "compressed" digit (the reconstructed digit). Do they look similar to the original images?

```
In [8]: ### YOUR CODE HERE
        def project_imgs(imgs, k_dims):
            """
            imgs: (N x 28 x 28)
            k_dims: number of dimensions to project data onto
            """
            proj_imgs = np.dot(np.dot(imgs.reshape(-1, 784), vectors[:, :k_dims]), vectors[:,
            fig = plt.figure(figsize=(16, 16))
            rows, cols = 5, 2
            for idx, img in enumerate(proj_imgs):
                fig.add_subplot(rows, cols, idx + 1)
                plt.imshow((img.astype(float) + data_mean).reshape(28, 28) / 255, cmap='gray')
            plt.show()

        project_imgs(digit_images, 2)
        ### END OF CODE
```

```
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/ipykernel_launcher.py:12
  if sys.path[0] == '':
```

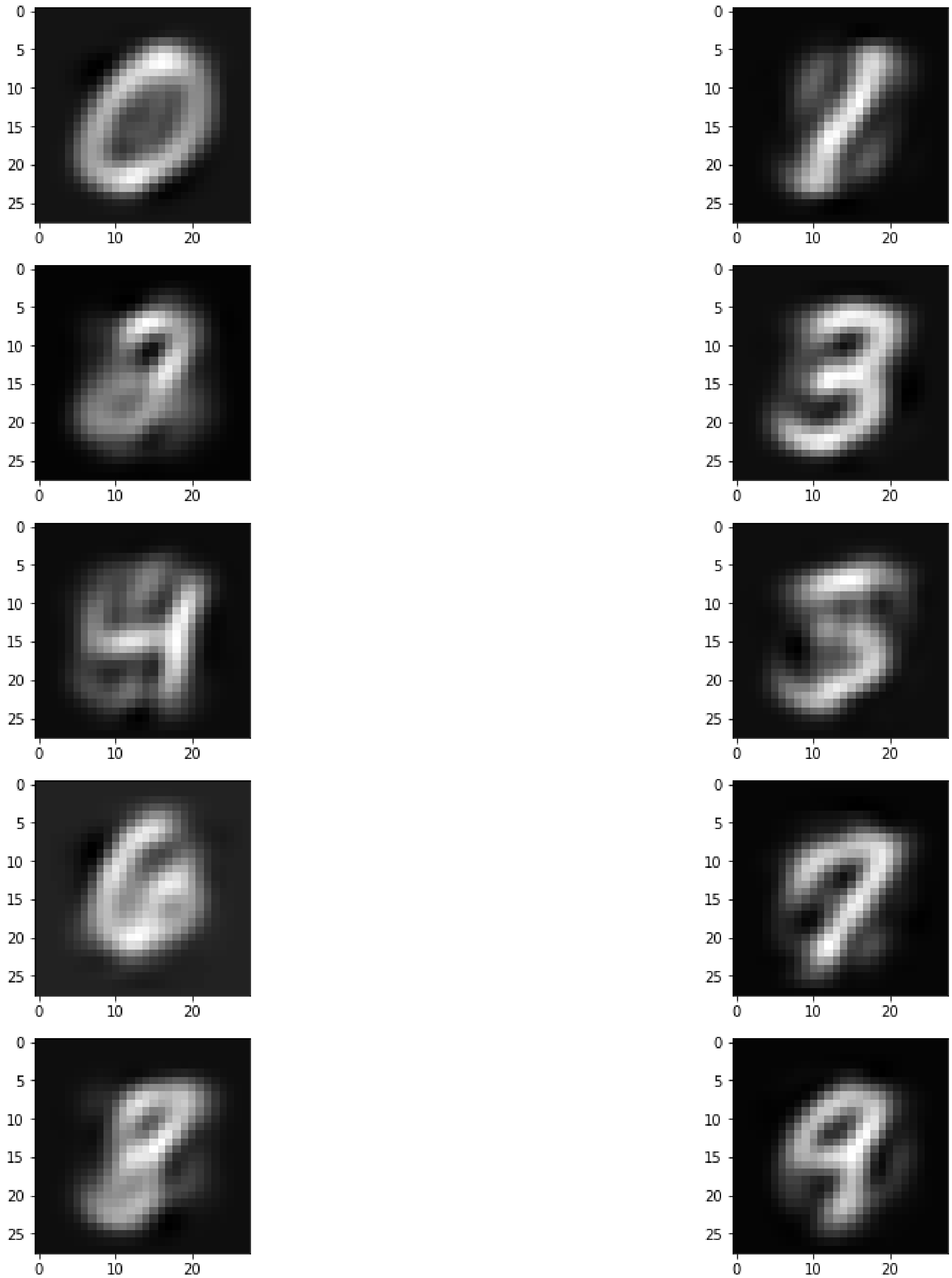


1.2.6 Question 1.2.6 Choose a better low dimension space. [5pt]

Do the previous problem with more dimensions (e.g. 3, 5, 10, 20, 50, 100). You only need to show results for one of them. Answer the following questinos. How many dimensions are required to represent the digits reasonably well? How are your results related to **question 1.2.3**?

```
In [9]: ### YOUR CODE HERE
        project_imgs(digit_images, 20)
        ### END OF CODE
```

```
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/ipykernel_launcher.py:12
  if sys.path[0] == '':
```



Depending on what reasonable is, you can represent the images with only 20 dimensions. The results are related to question 1.2.3 because each eigenvector in order of descending eigenvalues encodes less and less information about the images. In this regard, the eigenvectors relating to the higher magnitude eigenvalues contain more general information (closer to the mean) while each extra dimension adds another layer of data to the images.

1.3 Question 1.3 Harris Corner and PCA [10pt]

Recall Harris corner detector algorithm: 1. Compute x and y derivatives (I_x, I_y) of an image 2. Compute products of derivatives (I_x^2, I_y^2, I_{xy}) at each pixel 3. Compute matrix M at each pixel, where

$$M(x_0, y_0) = \sum_{x,y} w(x - x_0, y - y_0) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Here, we set weight $w(x, y)$ to be a box filter of size 3×3 (the box is placed centered at (x_0, y_0)).

In this problem, you need to show that Harris Corner detector is really just principal component analysis in the gradient space. Your explanation should answer the following quesions. 1. As we know, PCA is performed on data points. What are the data points in Harris corner detector when we think of it as a PCA? 2. What is the covariance matrix used in Harris corner detector and why it is a covariance matrix? 3. What are the principal components in Harris corner detector? 4. Briefly explain how principal components imply "cornerness".

(1) The data points in the Harris corner detector are the gradient values of the image in the horizontal and vertical directions. (2) The covariance matrix is used to find the components/directions with the most variance. (3) Because we are using the covariance matrix of the gradients, the principal components are directions with the highest variance. (4) The principal components imply "cornerness" because they represent the directions with the highest variance, thus, components with low magnitudes (small eigenvalues) will show little to no change in a direction, while ones with higher magnitudes show a large change in the direction. By projecting the data into 2 dimensions in the gradient space, we can see whether the data has "cornerness" through it's eigenvalues (aka magnitudes).

2 Question 2 KNN, Softmax Regression

In [10]:
```
train_dataset = MNIST(root='.', train=True, transform=transforms.ToTensor, download=T
test_dataset = MNIST('.', train=False, transform=transforms.ToTensor())
train_X = train_dataset.data.numpy()  # training data, uint8 type to reduce memory an
train_y = train_dataset.targets.numpy()  # training label
test_X = test_dataset.data.numpy() # testing data, uint8 to reduce memory and compari
test_y = test_dataset.targets.numpy()  # testing label
```

In [11]:
```
train_X = train_X.reshape((train_X.shape[0], -1))
test_X = test_X.reshape((test_X.shape[0], -1))
```

2.1 Question 2.1 K-Nearest Neighbor [10pt]

In this problem you will be implementing the KNN classifier. Fill in the functions in the starter code below. You are are allowed to use `scipy.spatial.KDTree` and `scipy.stats.mode` (in case of a tie, pick any one). Please avoid `sklearn.neighbors.KDTree` as it appears extremely slow. You are **not** allowed to use a library KNN function that directly solves the problem.

If you do not know what a KD-tree is, please read the documentation for `scipy.spatial.KDTree` to understand how you can use it.

Note: if you run into memory issues or neighbor queries run for more than 10 minutes, you are allowed to reduce the data size, and explain what you have done to the training data.

In [12]:
```
from scipy.spatial import KDTree
from scipy.stats import mode
```

In [13]:
```
class KNNClassifier:
    def __init__(self, num_neighbors):
        """
        construct the classifier
        Args:
            num_centers: number of neighbors
        """
        ### YOU CODE HERE
        self.k = num_neighbors
        self.kdt = None
        ### END OF CODE

    def fit(self, X, y):
        """
        train KNN classifier
        Args:
            X: training data, numpy array with shape (Nxk) where N is number of data p
            y: training labels, numpy array with shape (N)
        """
        ### YOU CODE HERE
        self.kdt = KDTree(X)
        self.X = X
        self.y = y
        ### END OF CODE
        return self

    def predict(self, X):
        """
        predict labels
        Args:
            X: testing data, numpy array with shape (Mxk) where M is number of data p
        Return:
            y: predicted labels, numpy array with shape (N)
        """
        pred = np.zeros((len(X)))
        ### YOU CODE HERE
        dists, inds = self.kdt.query(X, self.k)
        for idx, n_ind in enumerate(inds):
            neighbors = np.array([self.y[i] for i in n_ind])
            unique, counts = np.unique(neighbors, return_counts=True)
            pred[idx] = np.asarray((unique, counts)).T[0][0]
        ### END OF CODE
        return pred
```

```
In [14]: from sklearn.metrics import accuracy_score
         knn= KNNClassifier(3).fit(train_X, train_y)
         pred_y = knn.predict(test_X)
         print('KNN accuracy:', accuracy_score(test_y, pred_y))
```

```
/Users/sniradi/Documents/ucsd/152/env_152/lib/python3.7/site-packages/scipy/spatial/kdtree.py:3
  sd[node.split_dim] = np.abs(node.split-x[node.split_dim])**p


KNN accuracy: 0.4858
```

2.2 Question 2.2 Softmax Regression

In this problm, you will be implementing the softmax regression(multi-class logistic regression). Here is a brief recap of several important concepts. In the following explanation, I will use x for data vector, y' for ground truth label, and y for predicted label.

Suppose we have a problem where we need to classify data points into m classes.

1. Softmax function S normalize a vector to have sum 1. (it turns any vector into a probability distribution)

$$S(x) = [\frac{e^{x_1}}{\sum_{j=1}^{m} e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^{m} e^{x_j}}, ..., \frac{e^{x_m}}{\sum_{j=1}^{m} e^{x_j}}]$$

2. Cross entropy loss J is the multiclass logistic regression loss.

$$J(y', y) = -\sum_{i=1}^{m} y'_i \log y_i$$

   where y' is the one-hot ground truth label and y is the predicted label distribution.

3. Softmax regression is the following optimization problem.

$$\min_{W,b} \sum_{(X,y')\in\{\text{training set}\}} J(y', S(Wx+b))$$

   where W has shape $(m \times k)$ where k is the number of features in a data point; b is a m dimensional vector.

4. This objective is optimized with gradient descent. Let

$$L = \sum_{(x,y')\in\{\text{training set}\}} J(y', S(Wx+b))$$

   Update W and b with $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$.