MLPy Workshop 2

Alexandru Girban (S2148980), Hariaksh Pandya (S2692608)

January 22, 2025

1 Week 2: Principal Component Analysis

In this workshop, we will work through a set of problems on dimensionality reduction – a cannonical form of unsupervised learning. Within the machine learning pipeline, dimensionality reduction is an important tool, which can used in EDA to understand patterns in the data, feature engineering to create a low-dimensional representation of the inputs, and/or in the final phase when you are presenting and visualizing your solution.

As usual, the worksheets will be completed in teams of 2-3, using **pair programming**, and we have provided cues to switch roles between driver and navigator. When completing worksheets:

- You will have tasks tagged by (CORE) and (EXTRA).
- Your primary aim is to complete the (CORE) components during the WS session, afterwards you can try to complete the (EXTRA) tasks for your self-learning process.
- Look for the as cue to switch roles between driver and navigator.
- In some Exercises, you will see some beneficial hints at the bottom of questions.

Instructions for submitting your workshops can be found at the end of worksheet. As a reminder, you must submit a pdf of your notebook on Learn by 16:00 PM on the Friday of the week the workshop was given.

As you work through the problems it will help to refer to your lecture notes (navigator). The exercises here are designed to reinforce the topics covered this week. Please discuss with the tutors if you get stuck, even early on!

1.1 Outline

1. Problem Definition and Setup
2. Principal Component Analysis

a. Examining the Basis Vectors and Scores

b. Selecting the Number of Components

c. Other Digits

2 Problem Definition and Setup

2.1 Packages

First, lets load in some packages to get us started.

[1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

2.2 Data

Our dataset will be the famous MNIST dataset of handwritten digits, which we will download from sklearn. The dataset consists of a set of greyscale images of the numbers 0-9 and corresponding labels. Usually the goal is to train a classifier (i.e. given an image, what digit does it correspond to?). Here we will throw away the labels and focus on the images themselves. Specifically, we will use dimensionality reduction to explore the images and underlying patterns and find a low-dimensional representation.

First, load the data:

[2]:
```
import tensorflow_datasets as tfds

(Xtrain, ytrain), (Xtest, ytest) = tfds.as_numpy(tfds.load(
    'mnist',
    split=['train','test'],
    batch_size=-1,
    as_supervised=True
))

X = np.vstack((Xtrain,Xtest))
y = np.concatenate((ytrain,ytest))
X = X.reshape(X.shape[0],X.shape[1]*X.shape[2])
```

```
2025-01-21 18:18:15.971504: E
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to
cuInit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
2025-01-21 18:18:16.097728: I
tensorflow/core/kernels/data/tf_record_dataset_op.cc:376] The default buffer
size is 262144, which is overridden by the user specified `buffer_size` of
8388608
```

2.2.1 Exercise 1 (CORE)

What is stored in `X` and `y` in the command above? What is the shape/datatype etc if an array?

[3]:
```
X.shape
```

[3]:
```
(70000, 784)
```

[4]:
```
pd.Series(y).value_counts()
```

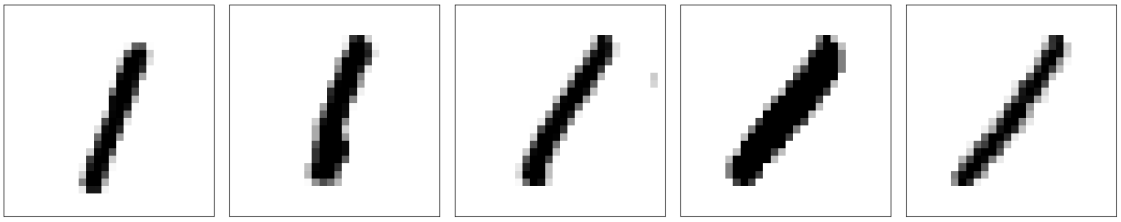
[4]:
```
1    7877
7    7293
3    7141
2    6990
9    6958
0    6903
6    6876
8    6825
4    6824
5    6313
Name: count, dtype: int64
```

Now, let's create a dictionary, with the digit classes (0-9) as keys, where the correponding values are the set of all images corresponding to that particular label.

[5]:
```
digits_dict = {}
count = 0

for label in y:
  if label in digits_dict:
    digits_dict[label] += [X[count]]
  else:
    digits_dict[label] = [X[count]]
  count += 1
```

Next let's visualize some of the images. We will start by picking a label and plotting a few images from within the dictionary. Note that each image contains a total of 784 pixels (28 by 28) and we will need to `reshape` the image to plot with `imshow(...,cmap='gray_r')`. Try also changing the label to view different digits.

[6]:
```
mylabel = 1
n_images_per_label = 5

fig = plt.figure(figsize=(4*n_images_per_label, 4))
for j in range(n_images_per_label):
    ax_number = 1 + j
    ax = fig.add_subplot(1, n_images_per_label, ax_number)
    ax.imshow(digits_dict[mylabel][j].reshape((28,28)), cmap='gray_r')
    ax.set_xticks([])
    ax.set_yticks([])
fig.tight_layout()
```



2.2.2 Exercise 2 (EXTRA)

Edit the code above to plot a few images for multiple labels.

Hint

Create a vector of labels and add additional for loop in the code above.

[ ]:

2.2.3 Exercise 3 (CORE)

Now focus on the 3s only and create a data matrix called `X_threes`. Define also `N` (# datapoints) and `D` (# features).

What are the features in this problem? How many features and data points are there?

[7]:
```
X_threes = np.asarray(digits_dict[3])
N, D = X_threes.shape
```

[8]:
```
print("Datapoints:")
print(N)
print("Features:")
print(D)
```

```
Datapoints:
7141
Features:
784
```

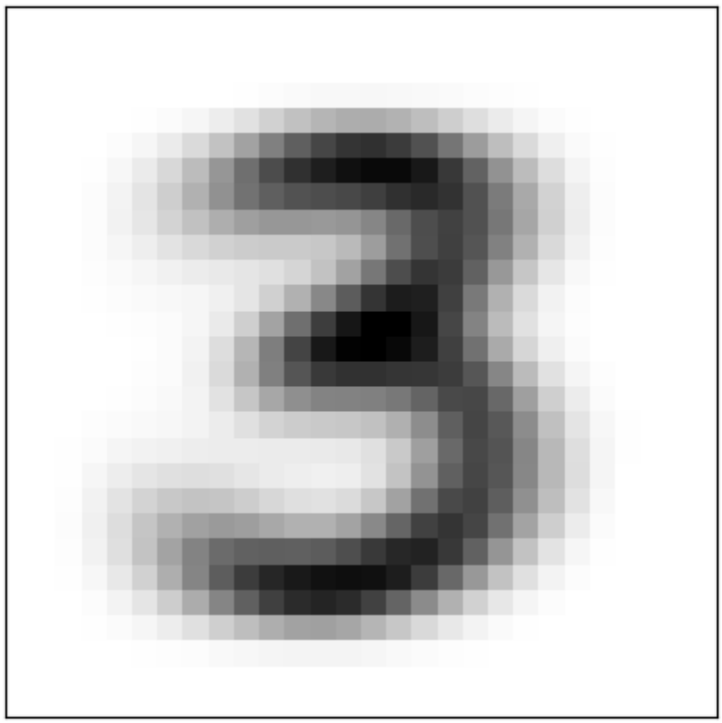
There are 7141 datapoints, and there are 784 features

2.2.4 Exercise 4 (CORE)

Now compute and plot the mean image of three.

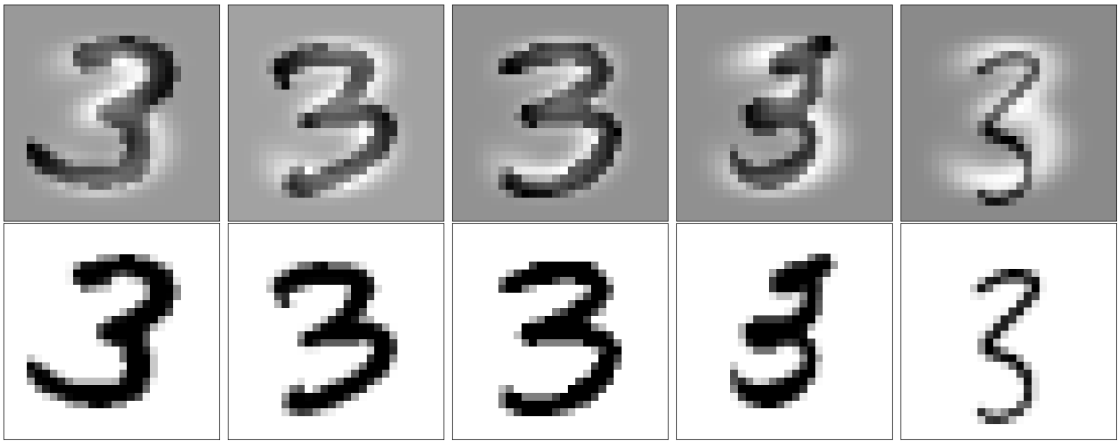
[11]:
```
X_three_mean = np.mean(X_threes, axis=0)
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(1,1,1)
ax.imshow(X_three_mean.reshape((28,28)), cmap='gray_r')
ax.set_xticks([])
```

```
ax.set_yticks([])
fig.tight_layout()
```



Run the following code to first create a new data matrix that centers the data by subtracting the mean image, and then visualise some of the images and compare to the original data. Note: you will need to replace `X_three_mean` with the name you gave the mean image in the computation above.

[12]:
```
X_three_centred = X_threes - X_three_mean

n_images = 5

fig = plt.figure(figsize=(4*n_images, 4*2))
for j in range(n_images):
  ax = fig.add_subplot(2, n_images, j+1)
  ax.imshow(X_three_centred[j,:].reshape((28,28)), cmap='gray_r')
  ax.set_xticks([])
  ax.set_yticks([])

  ax = fig.add_subplot(2, n_images, j+1+n_images)
  ax.imshow(X_threes[j,:].reshape((28,28)), cmap='gray_r')
  ax.set_xticks([])
  ax.set_yticks([])
fig.tight_layout()
```



2.2.5 Exercise 5 (CORE)

Comment on whether or not the images need to be standardized before using PCA

The features we are operating with are pixels. We should aim to standardize our data because performing PCA on standardized data is much easier than on original data.

Now, is a good point to switch driver and navigator

3 PCA

Now, we will perform PCA to summarize the main patterns in the images. We will use the `PCA()` transform from the `sklearn.decomposition` package:

- We can specify the number of components with the option `n_components`. If omitted, all components are kept.
- Note that by default the `PCA()` transform centers the variables to have zero mean (but does not scale them). After fitting, we can access the mean through the attribute `mean_`. If we also want to standardize to have not only zero mean but also unit variance, we can set `whiten=True`.
- We can access the basis vectors (principal components) through the `components_` attribute.
- We can call `fit()` to fit the model, followed by `transform` to obtain the low-dimensional representation (or also `fit_transform`).

First, let's create the PCA transform and call `fit()`:

[13]:
```
pca_threes = PCA(n_components = 200)
pca_threes.fit(X_threes)
```

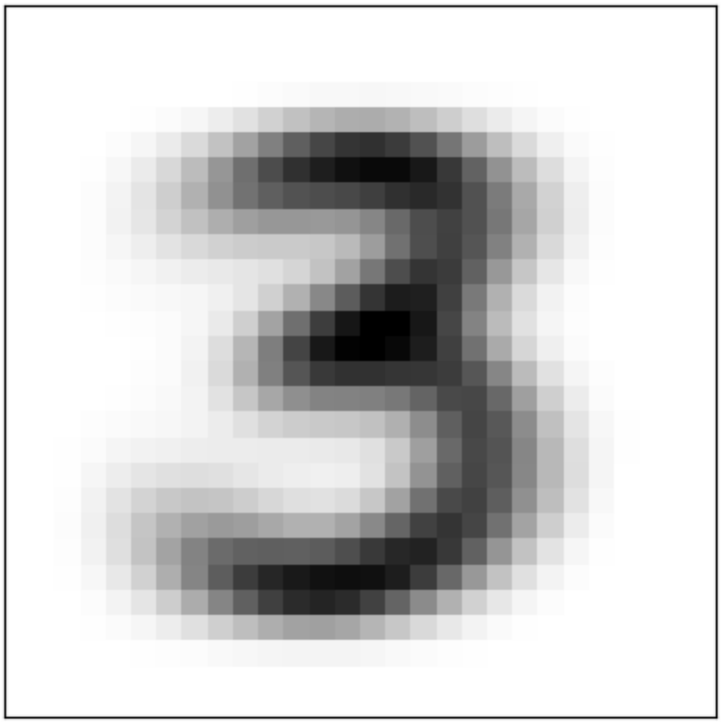
[13]:
```
PCA(n_components=200)
```

3.1 Examining the Basis Vectors and Scores

3.1.1 Exercise 6 (EXTRA)

Plot the mean image by accessing the `mean_` attribute and check that it is the same as above.

[15]:
```
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(1,1,1)
ax.imshow(pca_threes.mean_.reshape((28,28)), cmap='gray_r')
ax.set_xticks([])
ax.set_yticks([])
fig.tight_layout()
```



This is indeed the same as the previous mean image, as we expected, since we are taking a global mean of the images (average should be the same after standardization including in the PCA average).

3.1.2 Exercise 7 (CORE)

Plot the the first ten basis vectors as images by accessing the `components_` attribute. Overall, what patterns do they seem describe?

[16]:
```
n_images = 10

fig = plt.figure(figsize=(4*n_images, 4))
```

```
for j in range(n_images):
  ax = fig.add_subplot(1, n_images, j+1)
  ax.imshow(pca_threes.components_[j].reshape((28,28)), cmap='gray_r')
  ax.set_xticks([])
  ax.set_yticks([])

fig.tight_layout()
```

From the images above we can broadly see that the first 10 basis vectors broadly try to measure some form of "curlyness in the centre of the image", which is accurate with the geometry of the digit 3.

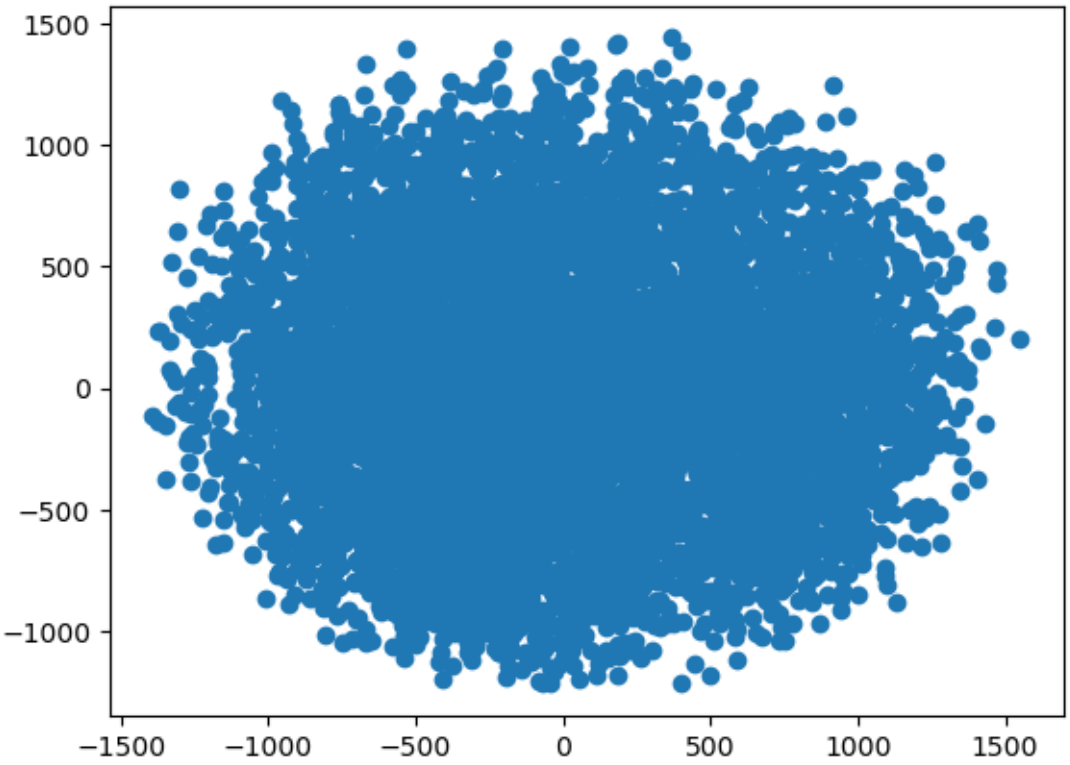
3.1.3 Exercise 8 (CORE)

a) Use the `transform()` method to compute the PCA scores and save them in an object called `scores`. Then, plot the data points in the low-dimensional space spanned by the first two principal components.

[20]:
```
scores = pca_threes.transform(X_threes)
scores.shape
```

[20]:
```
(7141, 200)
```

[27]:
```
plt.plot(scores[:,0], scores[:,1], 'o')
plt.show()
```



To better interpret the latent dimensions, let's look at some projected points along each dimension and the corresponding images. Specifically, run the following code to:

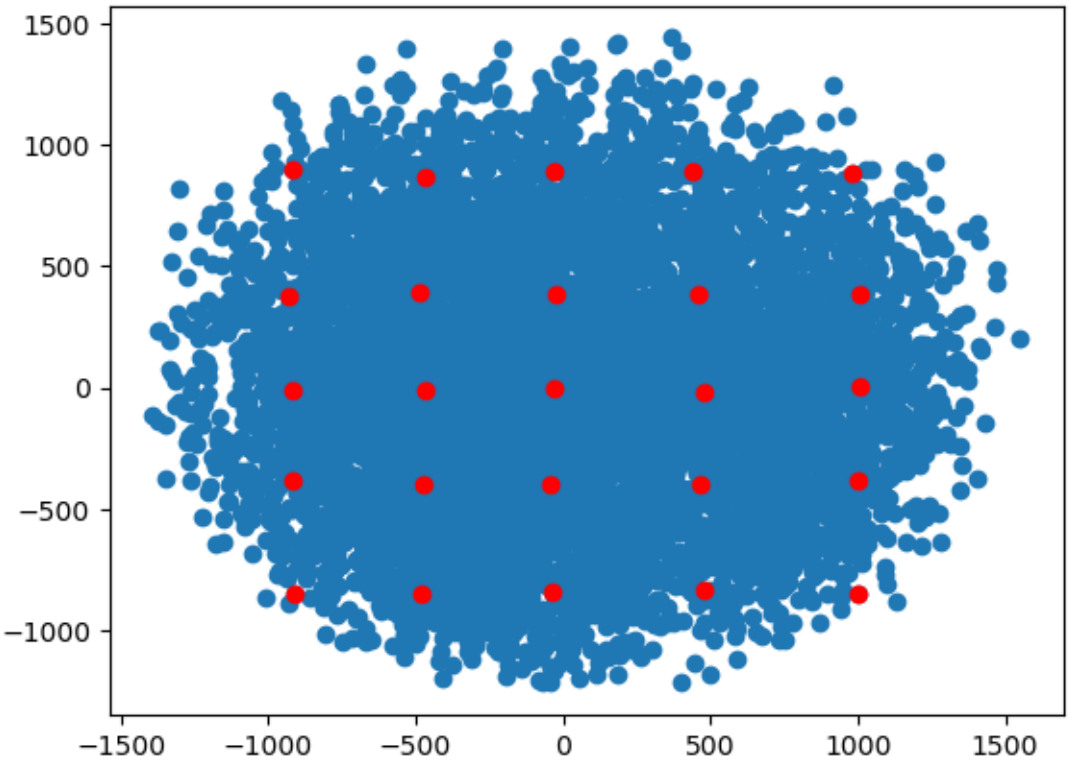
- first compute the $5, 25, 50, 75, 95\%$ quantiles of the scores for the first two dimensions
- then find the data point whose projection is closest to each combination of quantiles.

[19]:
```
s1q = np.quantile(scores[:,0],[.05,.25,.5,.75,.95])
s2q = np.quantile(scores[:,1],[.05,.25,.5,.75,.95])

idx = np.zeros([len(s1q),len(s2q)])

for i in range(len(s1q)):
    for j in range(len(s2q)):
        aux = ((scores[:,0] - s1q[i])**2 + (scores[:,1] - s2q[j])**2).
↪reshape(N,1)
        idx[i,j] = np.where(aux == min(aux))[0][0]

idx = idx.astype(int)
```

b) Now, add these points in red to your plot above in.

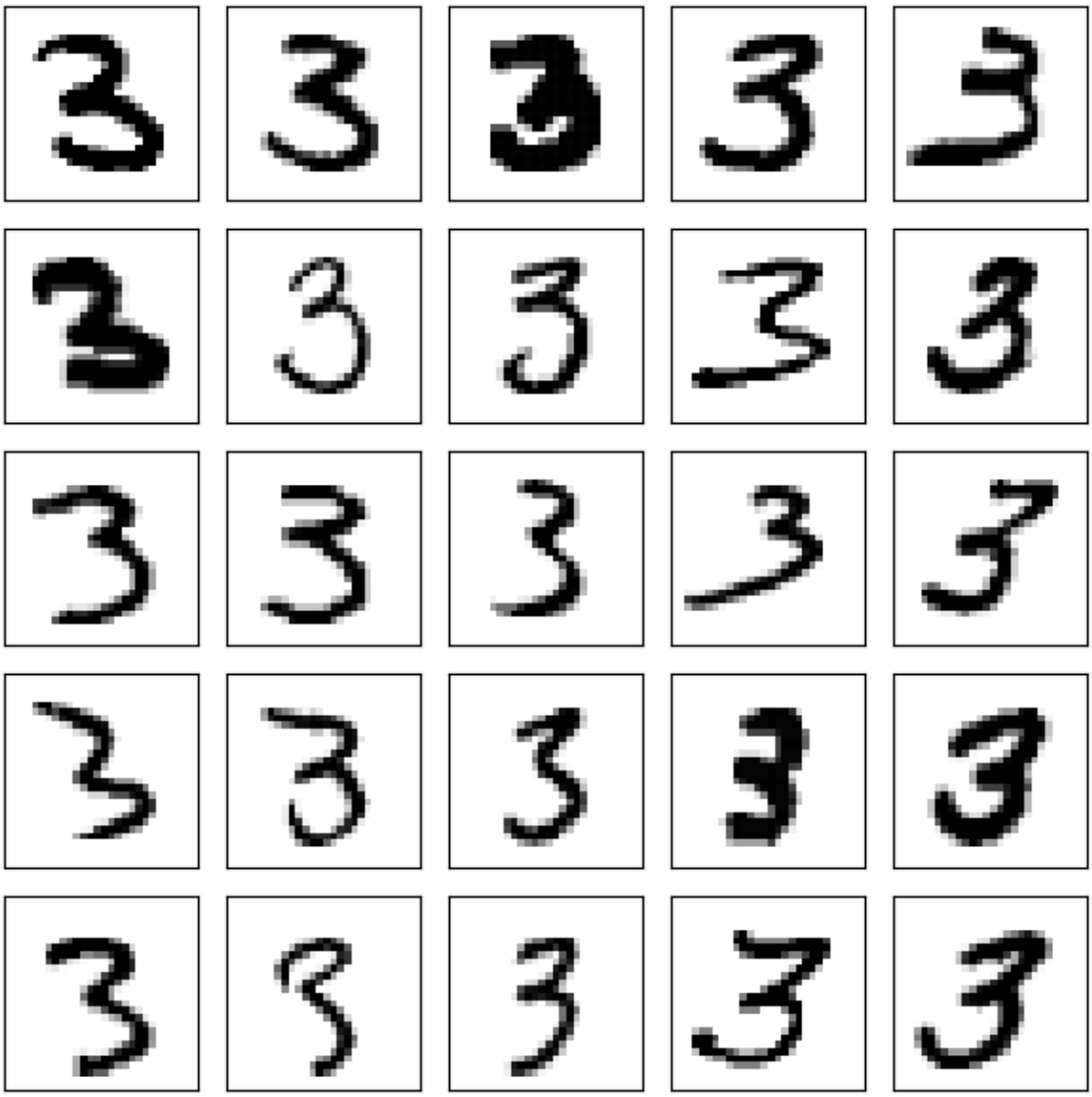
[33]:
```
plt.plot(scores[:,0], scores[:,1], 'o')
plt.plot(scores[idx[:,:],0], scores[idx[:,:],1], 'o', color = 'r')
```

```
plt.show()
```



c) Run the following code to plot the images corresponding to this grid of points. Describe the general pattern of the first (left to right) and second (down to up) principal component.

[29]:
```
fig, ax = plt.subplots(len(s1q),len(s2q),figsize=(6,6))
for i in range(len(s1q)):
    for j in range(len(s2q)):
        ax[len(s2q)-1-j,i].imshow(X_threes[idx[i,j],:].reshape((28,28)),␣
↪cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```

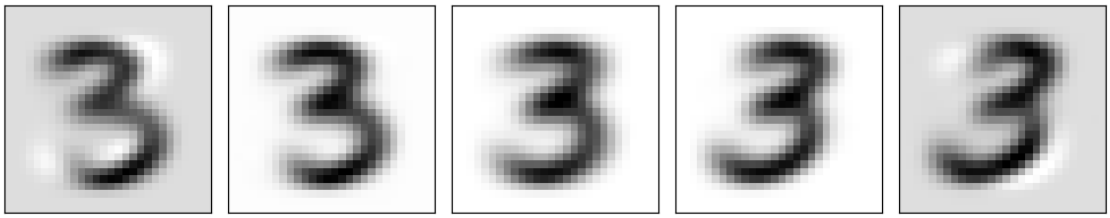


From left to right (first principal component), the pattern appears to be "how tilted the three is towards the left or the right", with the central column being central, the left and right columns being tilted in opposite directions. It makes sense why this would capture the most variance. Down to up (second principal component), the pattern appears to be more related to the "thickness" of the three.

You can also try to create some artificial images, by fixing different values of the weights. This can also help to interpret the latent dimensions.

[34]:
```
weight1 = [-1000,-500,0,500,1000]
weight2 = 0

images_pc1 = np.zeros([len(weight1),D])
```

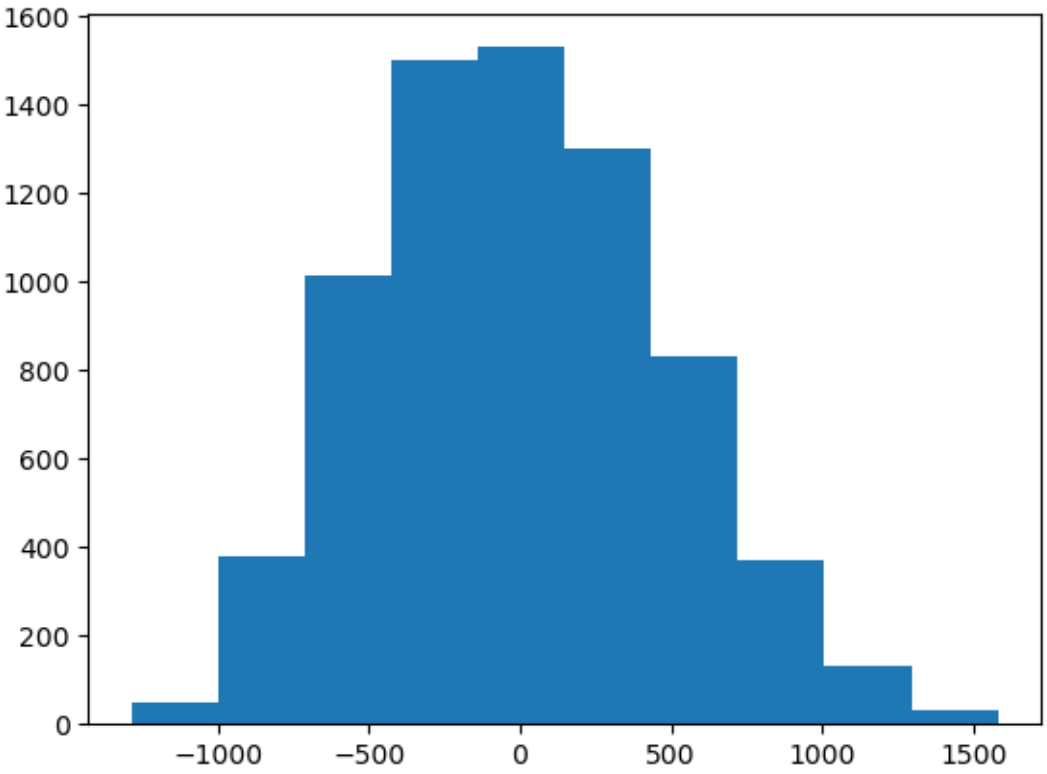
```
count = 0
for w in weight1:
    images_pc1[count,:] =(pca_threes.mean_ + pca_threes.components_[0,:
 ↪]*w+pca_threes.components_[1,:]*weight2)
    count += 1


fig, ax = plt.subplots(1,len(weight1),figsize=(10,6))
for i in range(len(weight1)):
    ax[i].imshow(images_pc1[i,:].reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```



3.1.4 Exercise 9 (CORE)

Repeat this to describe the third principal component. Look at the histogram of its scores to decide what values of weights to use.

[36]:
```
plt.hist(scores[:,2])
plt.show()
```



The histogram of scores for the third dimension is displayed above: it's much more mean-concentrated compared to the first two principal components, as expected.

3.1.5 Exercise 10 (EXTRA)

In lecture, we saw that we can also compute the basis vectors from an SVD decomposition of the data matrix. Use the `svd` function in `scipy.linalg` to compute the first three basis vectors and verify that they are the same (up to a change in sign – note that the signs may be flipped because each principal component specifies a direction in the D-dimensional space and flipping the sign has no effect as the direction does not change).

Does `PCA()` perform principal component analysis using an eigendecomposition of the empirical covariance matrix or using a SVD decomposition of the data matrix?

[ ]:

Now, is a good point to switch driver and navigator

3.2 Selecting the Number of Components

3.2.1 Exercise 11 (CORE)

Next, let's investigate how many components are needed by considering how much variance is explained by each component.

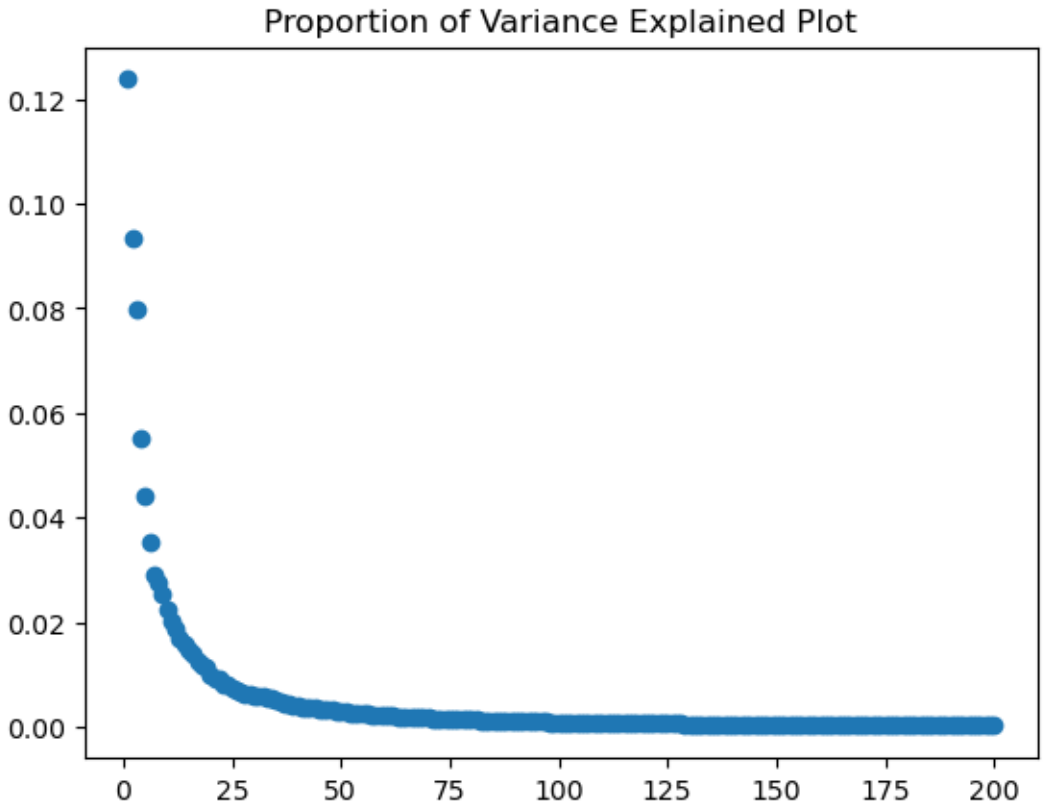
Note that the `pca_threes` object has an attribute `explained_variance_` (variance of each component) and `explained_variance_ratio_` (proportion of variance explained by each component).

Plot both the proportion of variance explained and the cummulative proportion of variance explained. Provide a suggestion of how many components to use. How much variance is explained by the suggest number of components? Comment on why we may be able to use this number of components in relation to the total number of features.

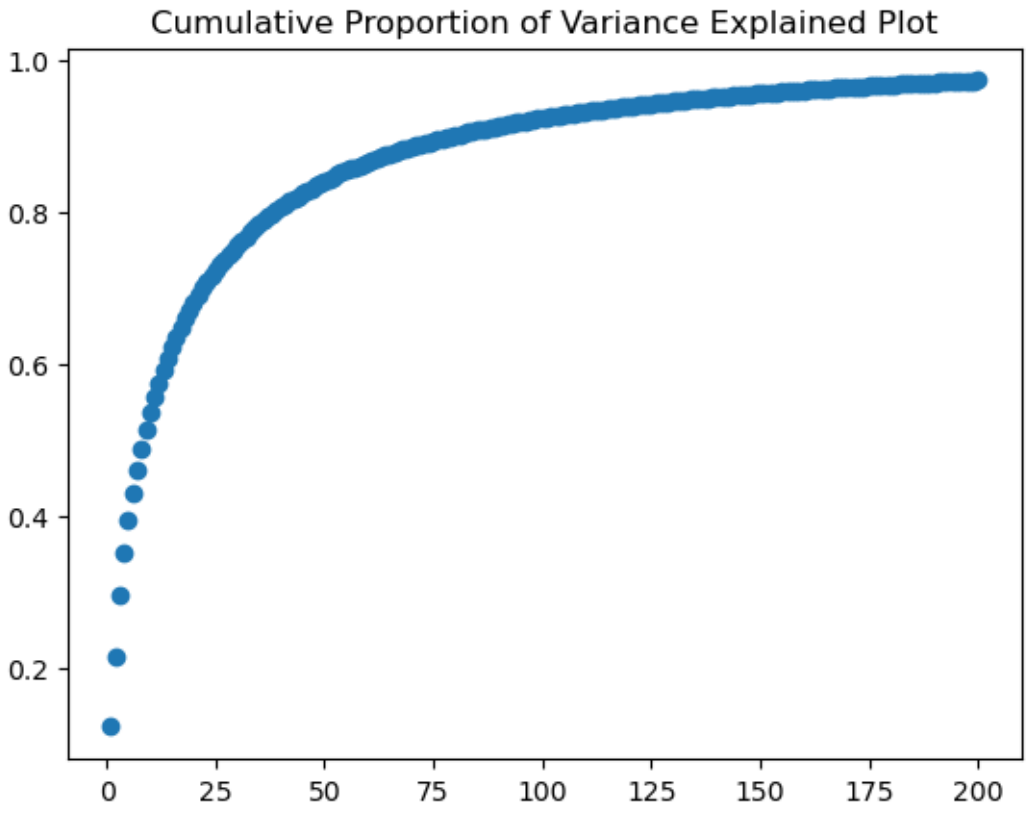
Hint

You can use `cumsum()` to compute the cummulative sum of the elements in a vector.

[41]:
```
plt.plot(range(1, 201), pca_threes.explained_variance_ratio_, 'o')
plt.title("Proportion of Variance Explained Plot")
plt.show()
```



[43]:
```
plt.plot(range(1, 201), pca_threes.explained_variance_ratio_.cumsum(), 'o')
plt.title("Cumulative Proportion of Variance Explained Plot")
plt.show()
```



As we can see from the plot above, including 75 components should be good enough to capture about 90% of the variance. 25 components would only explain about 65-70% of the variance, and all 200 components explain more than 95%. The total number of features is 784 (the number of pixels), so 75 (around 10%) is an appropriate number of components to use.

3.2.2 Exercise 12 (CORE)

For your selected number of components, compute the reconstruted images. Plot the reconstruction for a few images and compare with the original images. Comment on the results.

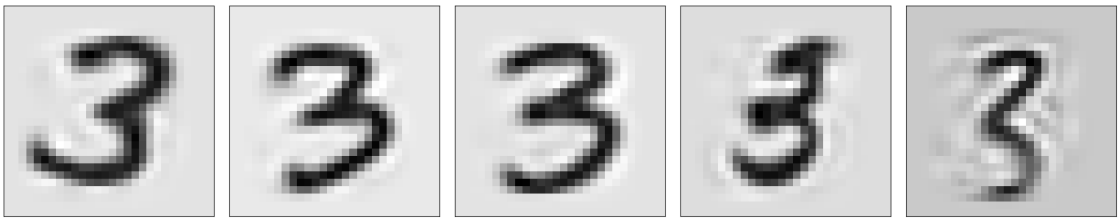
Hint

You can use `inverse_transform()` to decode the scores.

[45]:
```
pca = PCA(75)
X_inverse_threes = pca.inverse_transform(pca.fit_transform(X_threes))
```

[46]:
```
n_images_per_label = 5

fig = plt.figure(figsize=(4*n_images_per_label, 4))
for j in range(n_images_per_label):
    ax_number = 1 + j
    ax = fig.add_subplot(1, n_images_per_label, ax_number)
```

```
    ax.imshow(X_inverse_threes[j].reshape((28,28)), cmap='gray_r')
    ax.set_xticks([])
    ax.set_yticks([])
fig.tight_layout()
```



We note that with 75 components (aprrox. 90% of the variance), in the reconstruction we capture a fair variety of possible images for a "Three", althouth they appear a bit blurrier than the originals.

Now, is a good point to switch driver and navigator

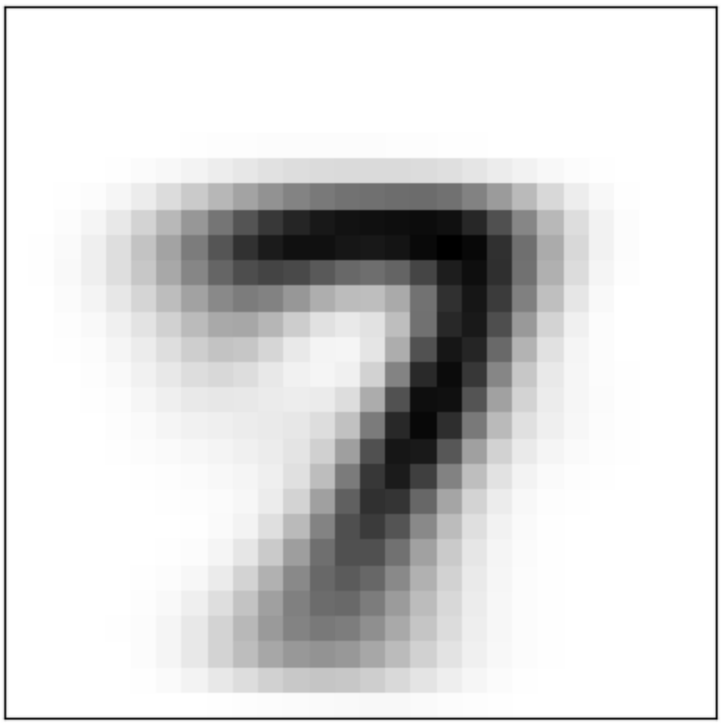
3.3 Other Digits

Now, let's consider another digit.

3.3.1 Exercise 13 (CORE)

Perform PCA for another choice of digit. What do the first two components describe? Do some digits have better approximations than others? Comment on why this may be.

[47]:
```
X_sevens = np.asarray(digits_dict[7])

X_sevens_mean = np.mean(X_sevens, axis=0)

X_sevens_centred = X_sevens - X_sevens_mean

pca_sevens = PCA(n_components = 200)
pca_sevens.fit(X_sevens)

fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(1,1,1)
ax.imshow(pca_sevens.mean_.reshape((28,28)), cmap='gray_r')
ax.set_xticks([])
ax.set_yticks([])
fig.tight_layout()
```



[48]:
```
n_images = 10

fig = plt.figure(figsize=(4*n_images, 4))
for j in range(n_images):
  ax = fig.add_subplot(1, n_images, j+1)
  ax.imshow(pca_sevens.components_[j].reshape((28,28)), cmap='gray_r')
  ax.set_xticks([])
  ax.set_yticks([])

fig.tight_layout()
```

When looking at data for the digit 7, it appears broadly that the first principal component focuses on the 'lower part of the Seven', whereas the second principal component is more focused on the 'curvature between the lines forming the 7'. The differences between quality in approximations among the digits arise as a result of the different geometries of the digits.

3.3.2 Exercise 14 (EXTRA)

Finally, consider now two digits of your choice (edit the code below if you wish to pick different digits).

[90]:
```
# Extract data
X_twodigits = np.concatenate((digits_dict['3'], digits_dict['8']))
N, D = X_twodigits.shape
```

Run the following code to compute and plot the mean and some of the principle components for this dataset.

[ ]:
```
# Fit PCA
pca_digits = PCA(n_components = 50)
pca_digits.fit(X_twodigits)
```

[ ]:
```
# Plot the mean image
fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(111)
ax.imshow(pca_digits.mean_.reshape(28, 28), cmap='gray_r')
ax.set_xticks([])
ax.set_yticks([])
fig.tight_layout()
```

[ ]:
```
# Plot basis vectors
n_plot = 5
fig, ax = plt.subplots(1,5,figsize=(10,4))
for n in range(n_plot):
  ax[n].imshow(pca_digits.components_[n,:].reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```

Plot the projection of the data in the latent space and color the data by the labels. What do you observe?

[ ]:
```

```

Try also to generate artificial images and decsribe how images change along the PCs.

[ ]:
```

```

4 Competing the Worksheet

At this point you have hopefully been able to complete all the CORE exercises and attempted the EXTRA ones. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Before generating the PDF, please go to Edit -> Edit Notebook Metadata and change 'Student 1' and 'Student 2' in the **name** attribute to include your name.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF.

[ ]:
```
!jupyter nbconvert --to pdf mlp_week02.ipynb
```

[ ]:
```

```