MLPy Workshop 2

Student 1, Student 2

January 26, 2024

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 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.

[2]:
```
import matplotlib.pyplot as plt
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:

[3]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', parser = 'auto')
X = mnist.data
y = mnist.target
```

2.2.1 Exercise 1 (CORE)

What object is returned by the command above? What shape/datatype etc if an array?

[4]:
```
X.shape
```

[4]:
```
(70000, 784)
```

[5]:
```
y.shape
```

[5]:
```
(70000,)
```

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.

[6]:
```
digits_dict = {}
X_= X.values
count = 0

for label in y:
  if label in digits_dict:
    digits_dict[label] += [X_[count]]
```

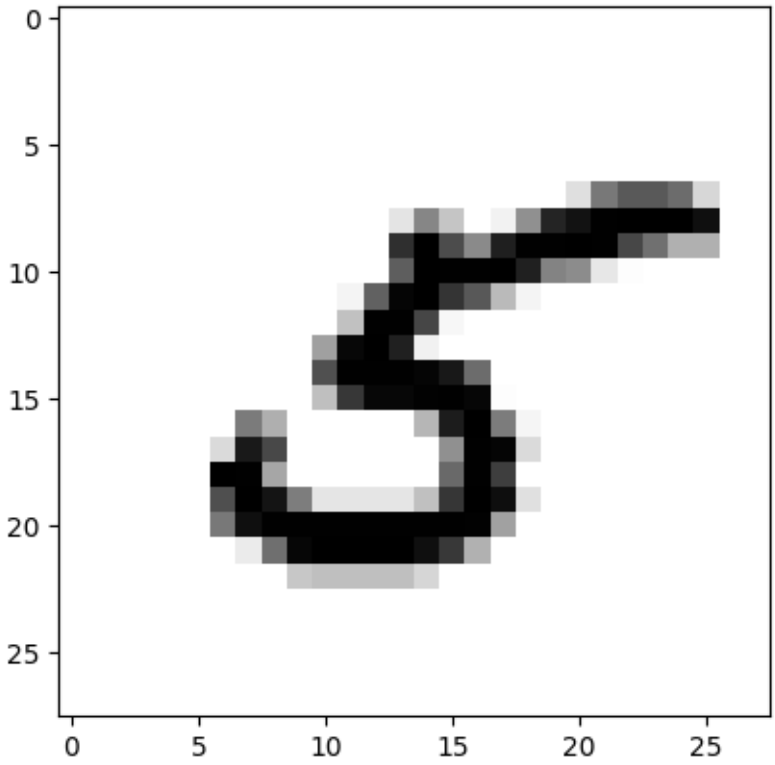
```
  else:
    digits_dict[label] = [X_[count]]
  count += 1
```

2.2.2 Exercise 2 (CORE)

Picking some of labels and plot a few images from within the dictionary. Note that each image contains a total of 784 pixels (28 by 28) and you will need to `reshape` the image to plot with `imshow(...,cmap='gray_r')`.

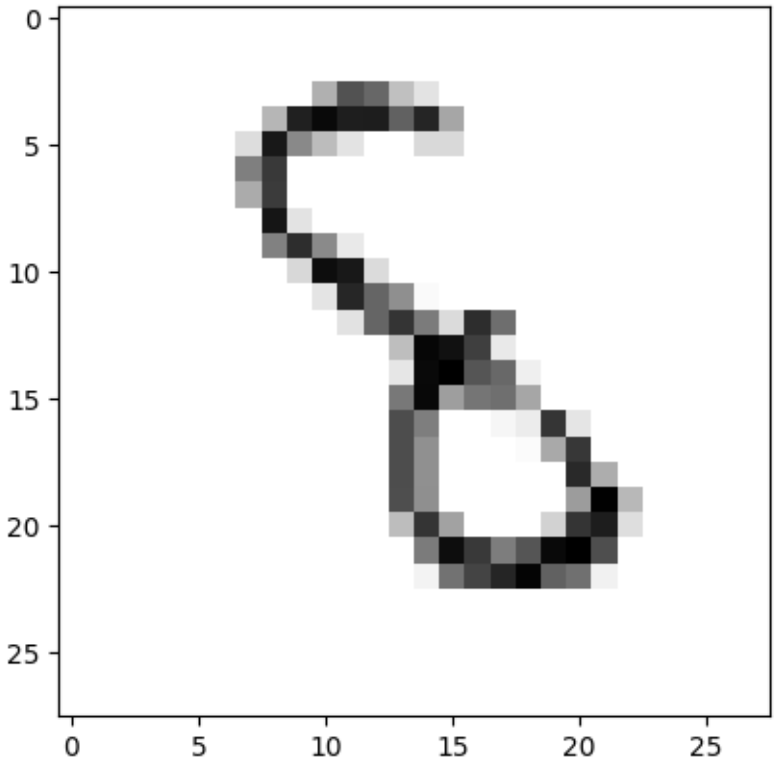
[7]:
```
plt.imshow(digits_dict['5'][10].reshape(28, 28), cmap='gray_r')
```

[7]:
```
<matplotlib.image.AxesImage at 0x16dceef30>
```



[8]:
```
plt.imshow(digits_dict['8'][100].reshape(28, 28), cmap='gray_r')
```

[8]:
```
<matplotlib.image.AxesImage at 0x17784fd10>
```



2.2.3 Exercise 3 (CORE)

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

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

[9]:
```
X_threes = digits_dict['3']
```

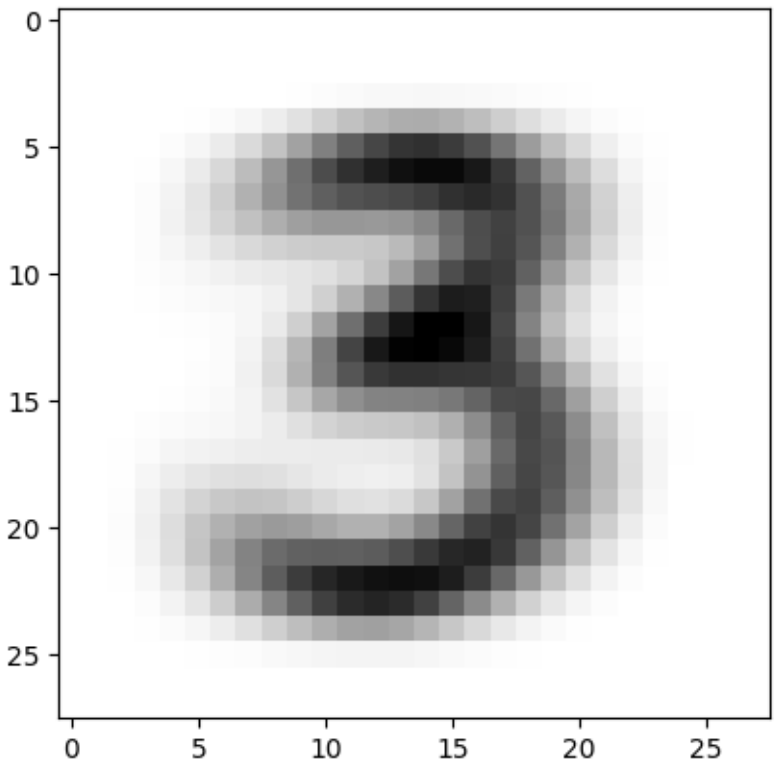
This dataset has 784 features with 7141 images. Each feature represents the respective pixel intensity in grayscale.

2.2.4 Exercise 4 (CORE)

Now compute and plot the mean image of three.

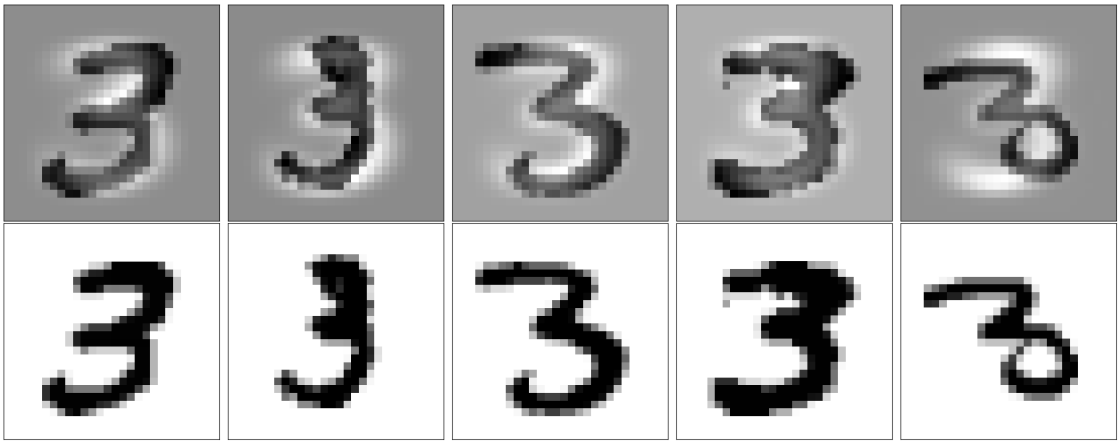
[10]:
```
X_three_mean = np.mean(np.stack(X_threes), axis=0)
plt.imshow(X_three_mean.reshape(28, 28), cmap='gray_r')
```

[10]:
```
<matplotlib.image.AxesImage at 0x1774649b0>
```



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.

[11]:
```
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(np.array(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 imaging data does not need to be scaled since the pixel intensities used are on the same scale of [0, 255].

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_`
- 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()`:

[12]:
```
pca_threes = PCA(n_components = 200)
pca_threes.fit(X_threes)
```

[12]:
```
PCA(n_components=200)
```

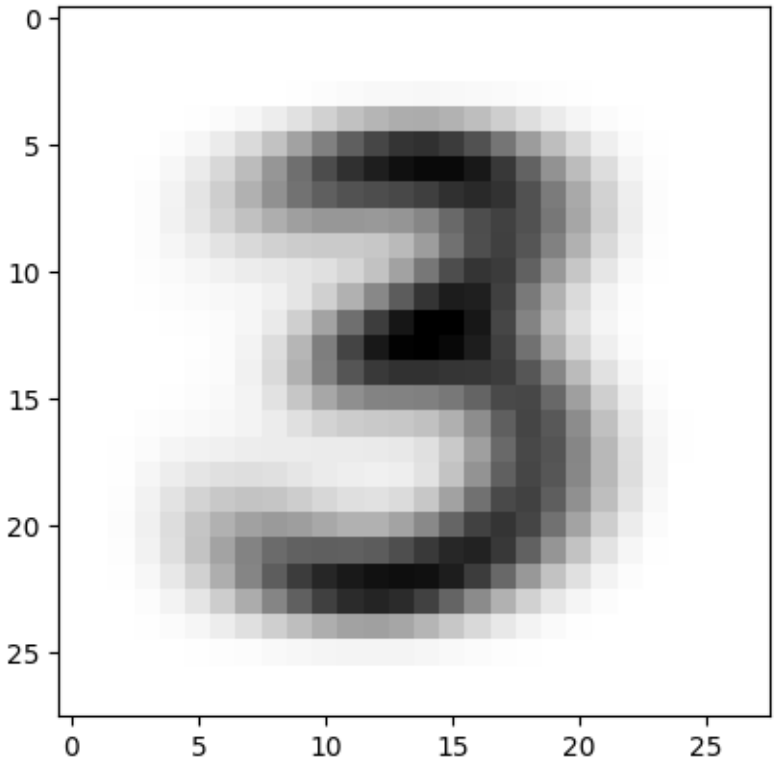
3.1 Examining the Basis Vectors and Scores

3.1.1 Exercise 6 (CORE)

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

[13]:
```
plt.imshow(pca_threes.mean_.reshape(28, 28), cmap='gray_r')
```

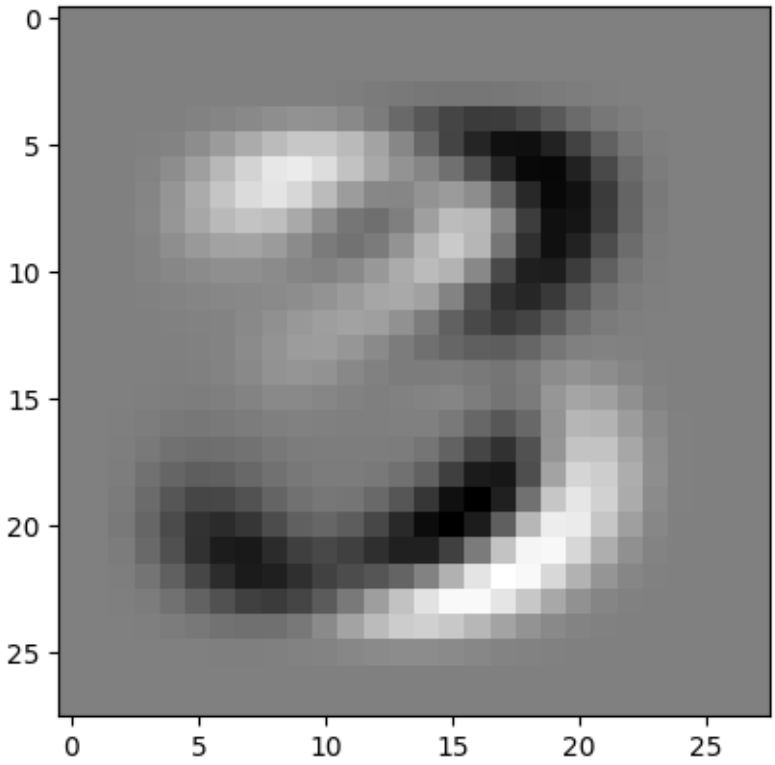
[13]:
```
<matplotlib.image.AxesImage at 0x1778cc920>
```

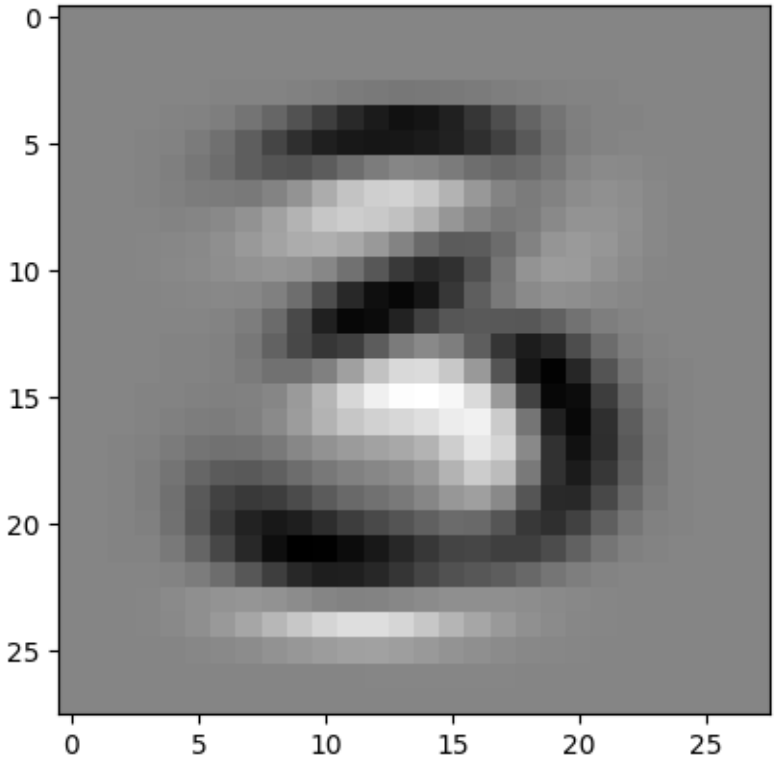


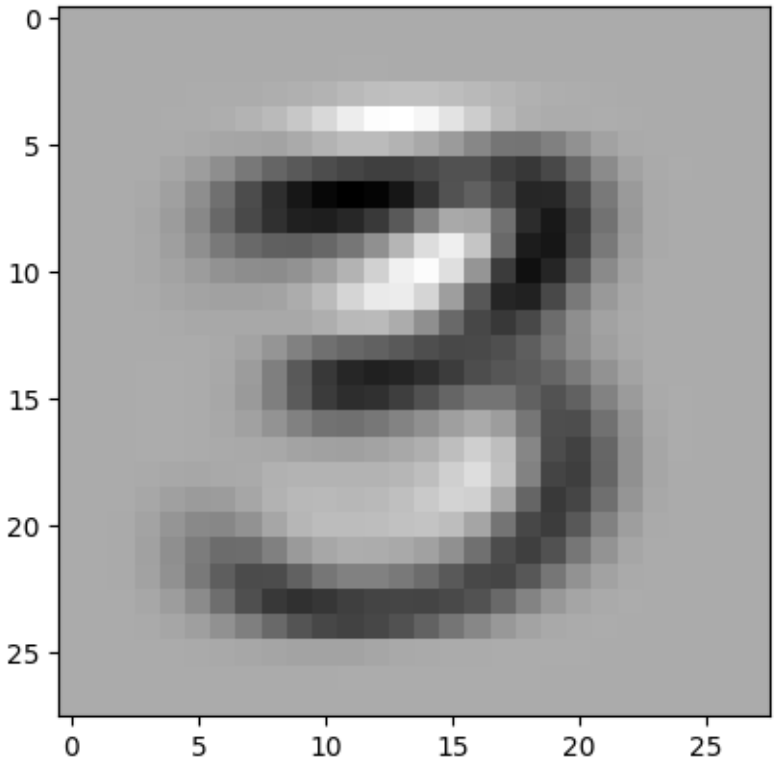
3.1.2 Exercise 7 (CORE)

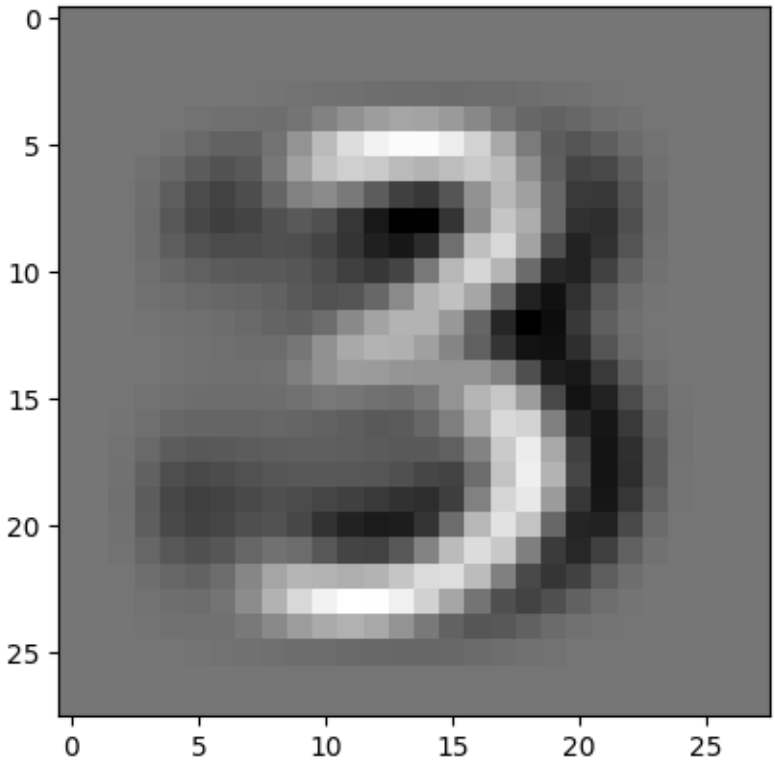
Plot the the first ten basis vectors as images by accessing the `components_` attribute. What patterns do they describe? From an exploratory perspective, do they all describe interesting patterns?

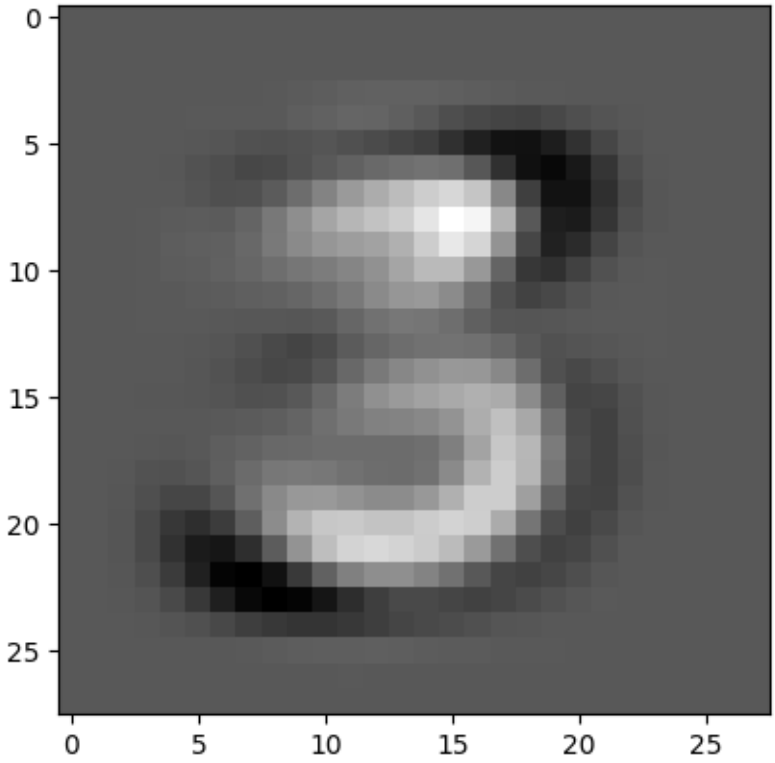
[14]:
```
for i in range(10):
    plt.imshow(pca_threes.components_[i].reshape(28, 28), cmap='gray_r')
    plt.show()
```

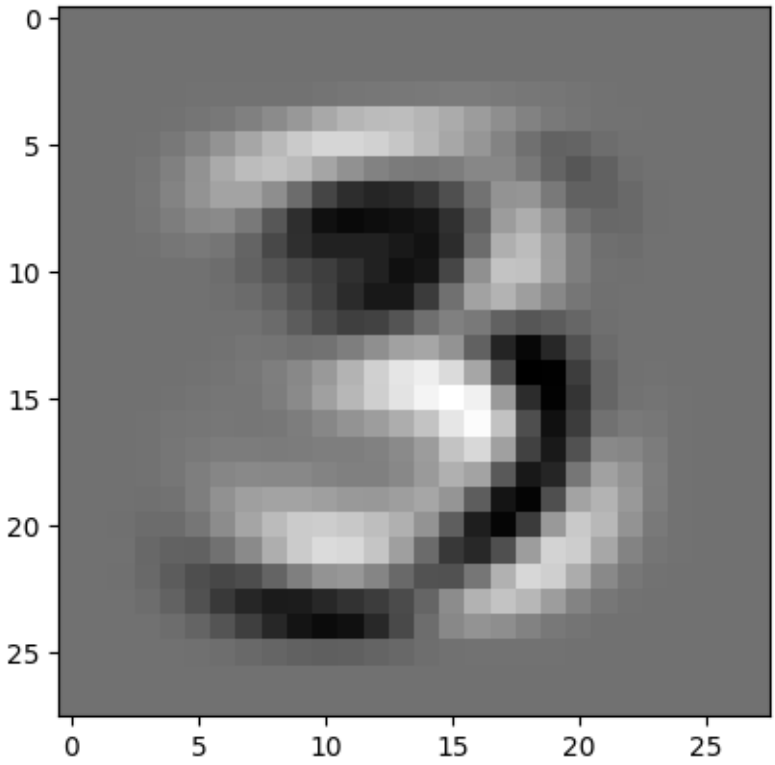

0
5
10
15
20
25
0
5
10
15
20
25

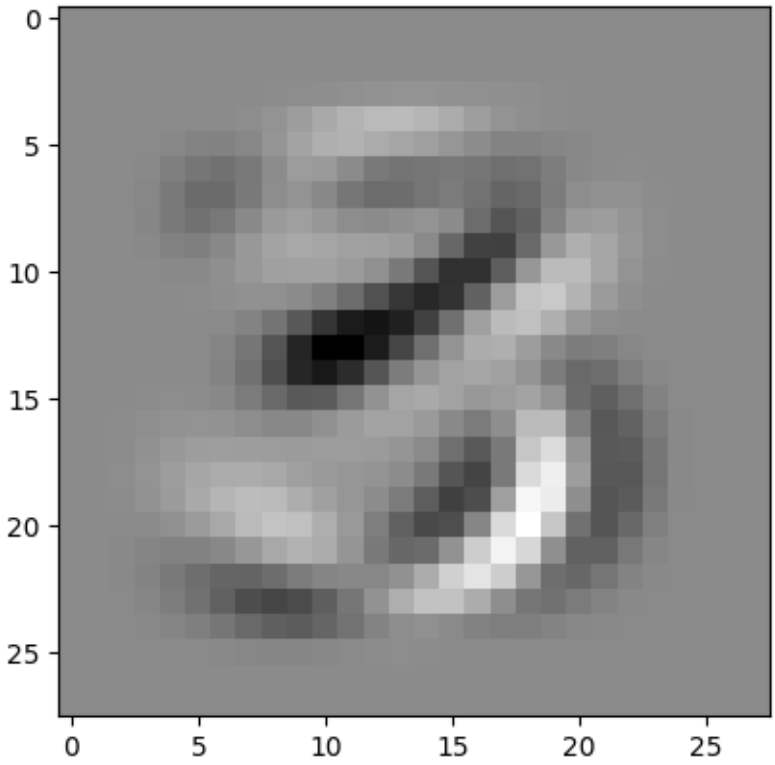

0
5
10
15
20
25
0
5
10
15
20
25

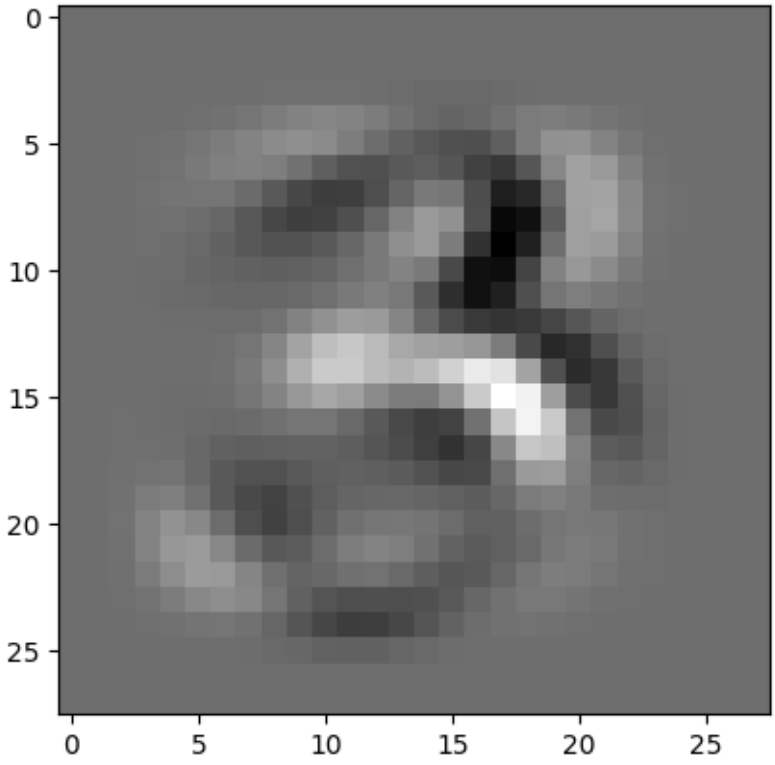

0
5
10
15
20
25
0
5
10
15
20
25

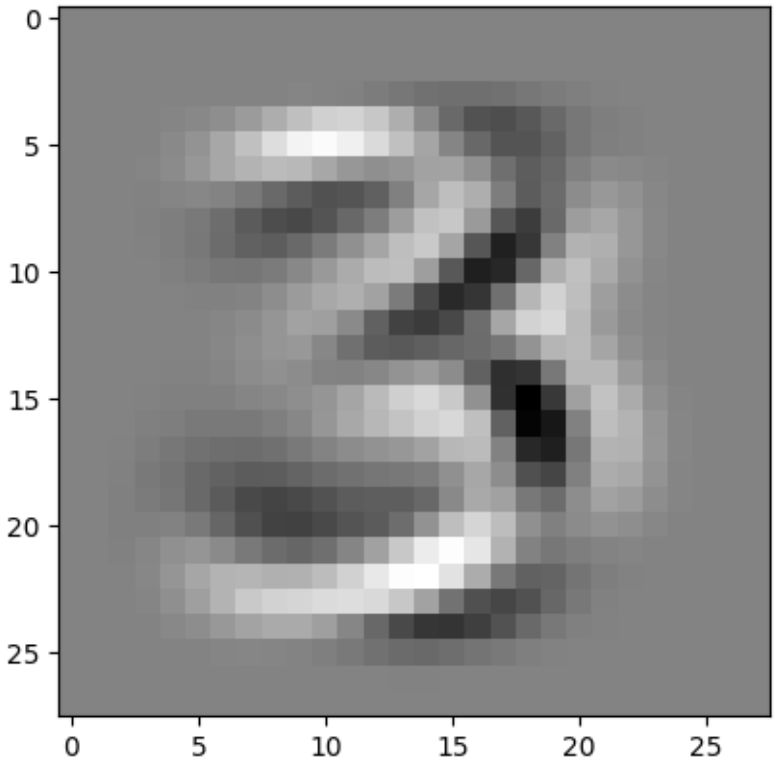

0
5
10
15
20
25
0
5
10
15
20
25

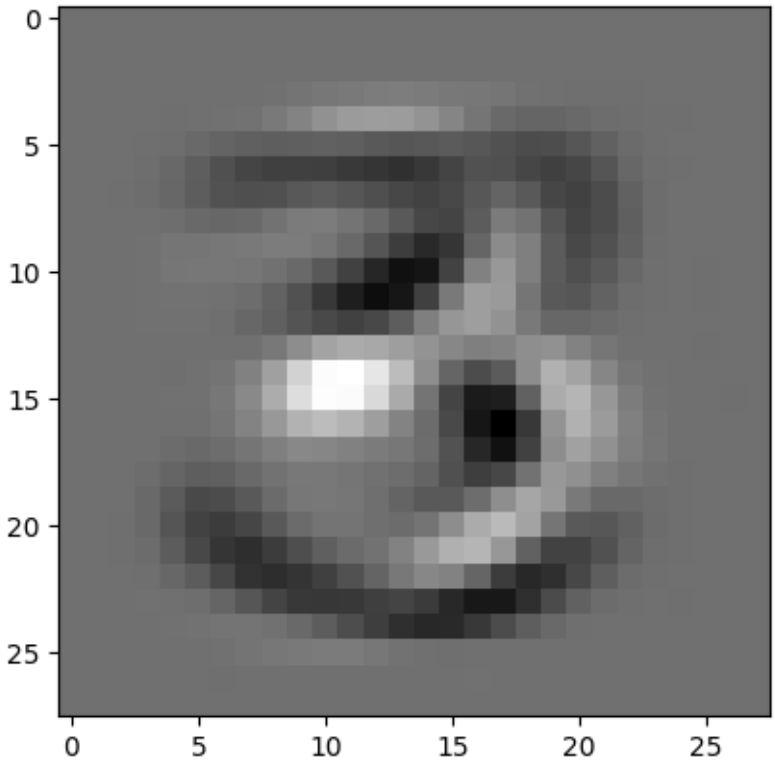

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



They all somehow display figures resembling the number however the colormap seems to be inverted might be a result of the dimensionality reduction and the loss of some variance in the dataset.

3.1.3 Exercise 8 (CORE)

a) Use the `transform()` method to compute the 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.

[15]:
```
scores = pca_threes.transform(X_threes)
```

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.

[16]:
```
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(7141,1)
        idx[i,j] = np.where(aux == min(aux))[0][0]
```

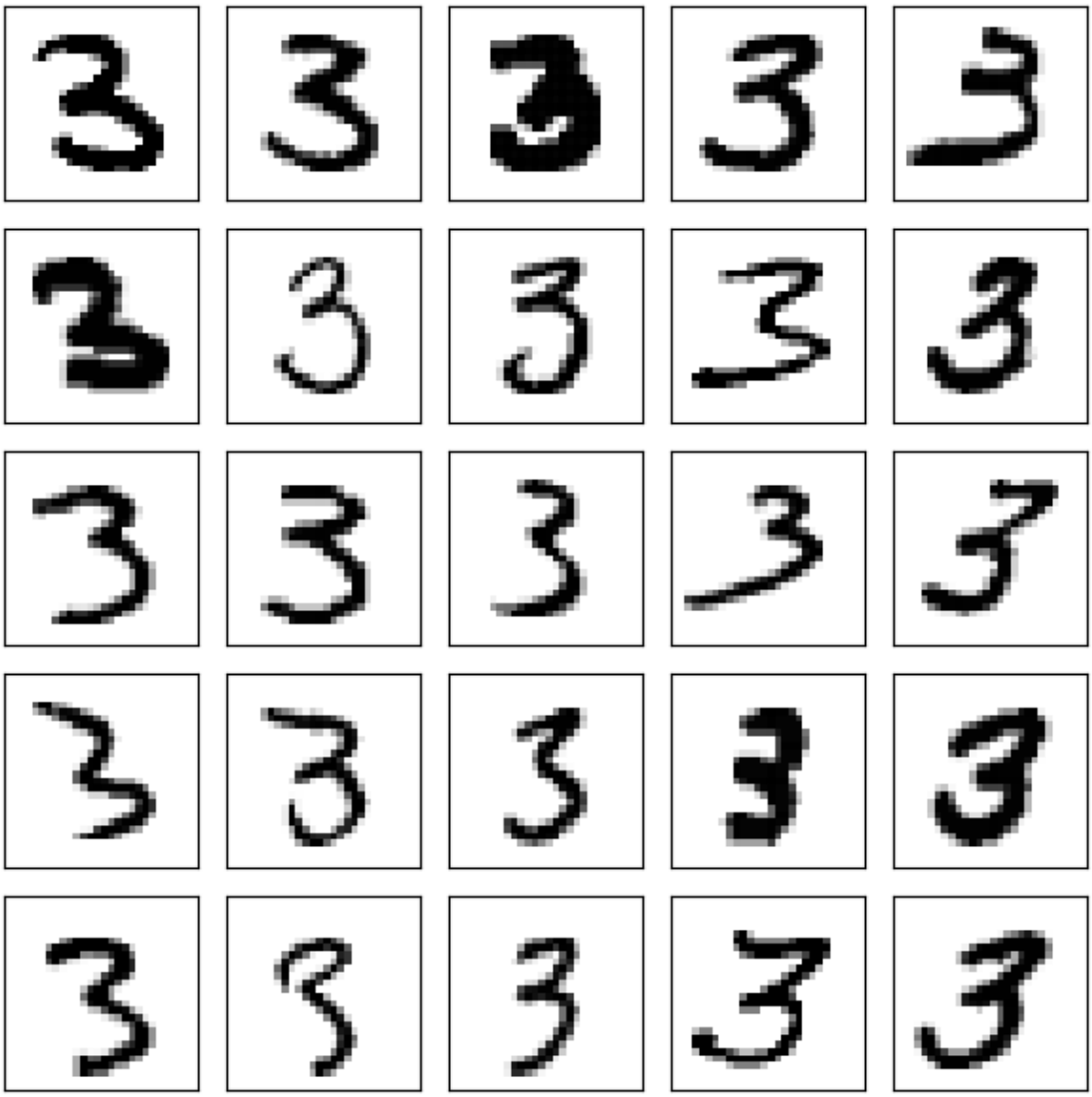
b) Now, add these points in red to your plot above in.

[17]:
```
idx
```

[17]:
```
array([[6396., 4562., 6506., 6077.,  452.],
       [ 584., 4811., 5928.,  612., 1001.],
       [1620., 6261.,  589., 5318.,  861.],
       [2480.,  896., 4300., 4990., 7020.],
       [5834., 3379., 5868., 6891., 5516.]])
```

c) Run the following code to plot the images corresponding to this grid of points. Describe the patterns of the first and second principal components.

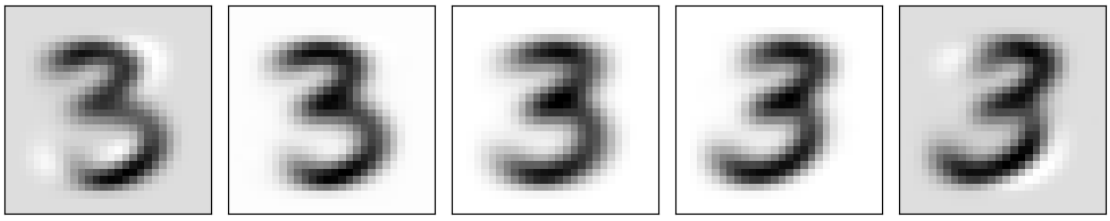
[18]:
```
fig, ax = plt.subplots(len(s1q),len(s2q),figsize=(6,6))
for i in range(len(s1q)):
    for j in range(len(s2q)):
        ax[len(s2q)-1-j,i].imshow(np.array(X_threes)[int(idx[i,j]),:].
↪reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```



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.

[19]:
```
weight1 = [-1000,-500,0,500,1000]
weight2 = 0

images_pc1 = np.zeros([len(weight1),784])

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

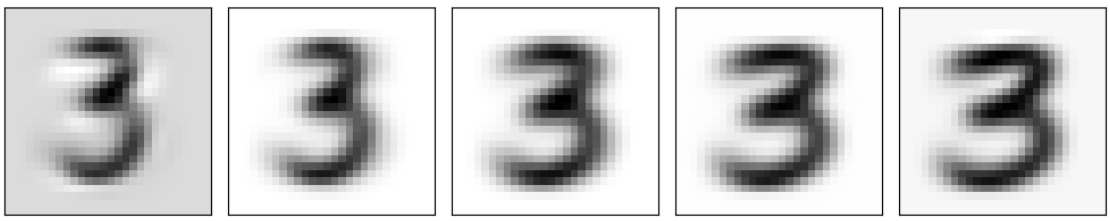
```
fig, ax = plt.subplots(1,len(weight1),figsize=(10,6))
for i in range(len(weight1)):
    ax[i].imshow(images_pc1[i,:].reshape((28,28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```



3.1.4 Exercise 9 (CORE)

Repeat this to describe the third principal component. What values of weights should you use?

[20]:
```
weight1 = 0
weight2 = 0
weight3 = [-1000, -500, 0, 500, 1000]

images_pc3 = np.zeros([len(weight3), 784])

for count, w in enumerate(weight3):
    images_pc3[count, :] = (pca_threes.mean_
                            + pca_threes.components_[0, :] * weight1
                            + pca_threes.components_[1, :] * weight2
                            + pca_threes.components_[2, :] * w)

fig, ax = plt.subplots(1, len(weight3), figsize=(10, 6))
for i in range(len(weight3)):
    ax[i].imshow(images_pc3[i, :].reshape((28, 28)), cmap='gray_r')
plt.setp(ax, xticks=[], yticks=[])
fig.tight_layout()
```



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?

[23]:
```
from scipy.linalg import svd

U, S, Vh = svd(X_threes, full_matrices=False)

svd_threes = Vh[:3, :]

for i in range(3):
    svd_vector = svd_threes[i, :]
    pca_vector = pca_threes.components_[i, :]

    same_direction = np.allclose(svd_vector, pca_vector) or np.
 ↪allclose(svd_vector, -pca_vector)
    print(f"Basis vector {i+1}: {'Same' if same_direction else 'Different'}")
```

```
Basis vector 1: Different
Basis vector 2: Different
Basis vector 3: Different
```

The PCA is done using the emprical covariance matrix 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 varaince is explained by each component.

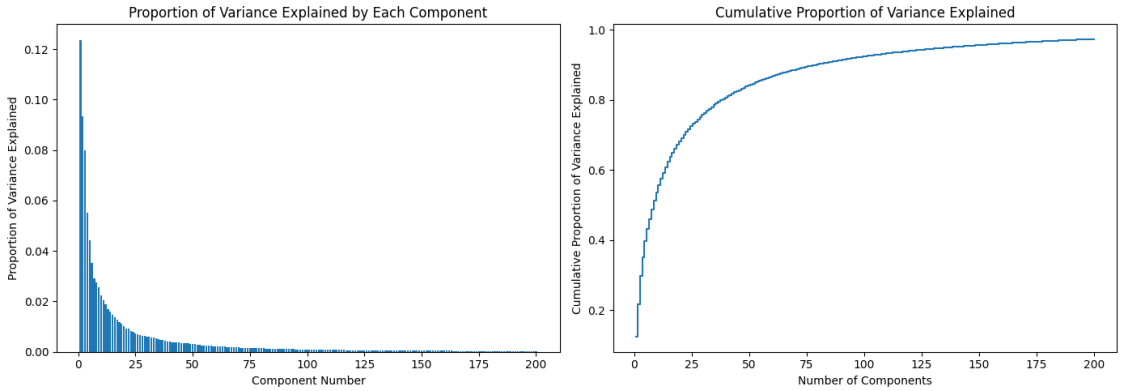
Note that the `pca_threes` object has an attribute `explained_variance_` (variance of each component) and `explained_variance_ratio_` (proportion of variance explained by each component).

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.

[25]:
```
explained_variance = pca_threes.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance)

plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.bar(range(1, len(explained_variance) + 1), explained_variance)
plt.xlabel('Component Number')
plt.ylabel('Proportion of Variance Explained')
plt.title('Proportion of Variance Explained by Each Component')

plt.subplot(1, 2, 2)
plt.step(range(1, len(cumulative_explained_variance) + 1),␣
 ↪cumulative_explained_variance, where='mid')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Proportion of Variance Explained')
plt.title('Cumulative Proportion of Variance Explained')

plt.tight_layout()
plt.show()
```



Depending on the plots seen above there seems to be an "elbow" at around 50 components therefore one could suggest the number of components from an interval close to 50.

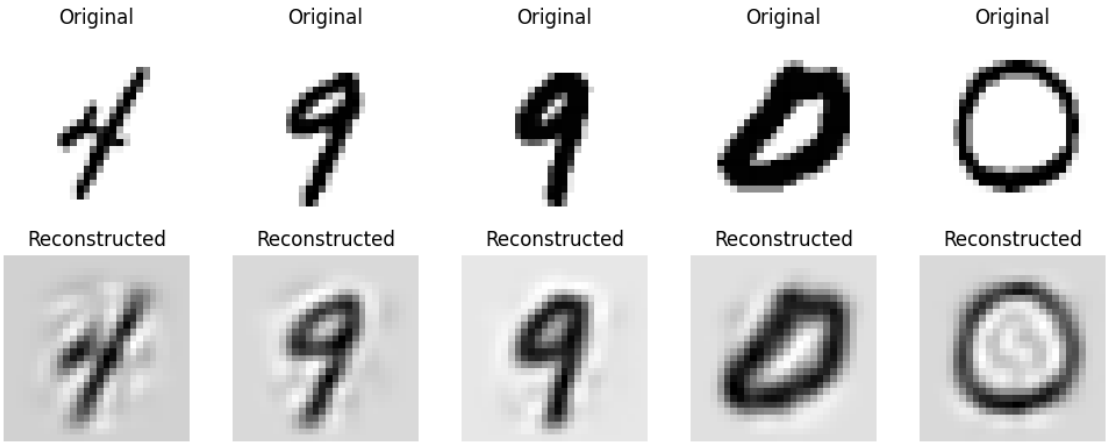
3.2.2 Exercise 12 (CORE)

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.

[34]:
```
import matplotlib.pyplot as plt

X = X.to_numpy()
pca = PCA(n_components=50)
pca.fit(X)
scores = pca.transform(X)

# Reconstruct the images from the PCA scores
X_reconstructed = pca.inverse_transform(scores)

# Select a few images for demonstration
num_images = 5
selected_images = np.random.choice(X.shape[0], num_images, replace=False)

# Plotting the original and reconstructed images
plt.figure(figsize=(10, 4))

for i, image_idx in enumerate(selected_images):
    # Original Image
    plt.subplot(2, num_images, i + 1)
    plt.imshow(X[image_idx].reshape(28, 28), cmap='gray_r')
    plt.title("Original")
    plt.axis('off')

    # Reconstructed Image
    plt.subplot(2, num_images, i + 1 + num_images)
    plt.imshow(X_reconstructed[image_idx].reshape(28, 28), cmap='gray_r')
    plt.title("Reconstructed")
    plt.axis('off')

plt.tight_layout()
plt.show()
```



The white and black contrast seems to be lost a bit and there is some blur to the reconstructed images.

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 digits. What do the first two components describe? Do some digits have better approximations than others? Comment on why this may be.

3.4 Did this for Exercise 12 accidentally.

3.4.1 Exercise 14 (EXTRA)

Finally, consider now all of your images (for all digits). Compute and plot some of the principle components for this dataset.

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

[ ]:

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.

[92]:
```
!jupyter nbconvert --to pdf mlp_week02.ipynb
```

```
[NbConvertApp] Converting notebook mlp_week02.ipynb to pdf
[NbConvertApp] Support files will be in mlp_week02_files/
[NbConvertApp] Making directory ./mlp_week02_files
[NbConvertApp] Writing 52940 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
```

```
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 227182 bytes to mlp_week02.pdf
```

[ ]: