Distances and Angles between Images

We are going to compute distances and angles between images.

Learning objectives

By the end of this notebook, you will learn to

1. Write programs to compute distance.
2. Write programs to compute angle.

"distance" and "angle" are useful beyond their usual interpretation. They are useful for describing **similarity** between objects. You will use the functions you wrote to compare MNIST digits.

In [3]:
```
# PACKAGE: DO NOT EDIT THIS CELL
import numpy as np
import scipy
```

In [4]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import sklearn
from ipywidgets import interact
from load_data import load_mnist

# Plot figures so that they can be shown in the notebook
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

The next cell loads the MNIST digits dataset.

In [5]:
```
from load_data import load_mnist
MNIST = load_mnist('./')
images = MNIST['data'].astype(np.double)
labels = MNIST['target'].astype(np.int)
```

For this assignment, you need to implement the two functions (`distance` and `angle`) in the cell below which compute the distance and angle between two vectors.

Distances

In [6]:
```
# GRADED FUNCTION: DO NOT EDIT THIS LINE

def distance(x0, x1):
    """Compute distance between two vectors x0, x1 using the dot product.

    Args:
    x0, x1: ndarray of shape (D,) to compute distance between.

    Returns:
    the distance between the x0 and x1.
    """
    # YOUR CODE HERE
    ### Uncomment and modify the code below
    distance = np.dot(x1-x0,x1-x0)**(0.5) # <-- EDIT THIS to compute the distance between x0 and x1
    return distance
```

In [7]:
```
# Some sanity checks, you may want to have more interesting test cases to test your implementation
a = np.array([1, 0])
b = np.array([0, 1])
np.testing.assert_allclose(distance(a, b), np.sqrt(2), rtol=1e-7)

a = np.array([1, 0])
b = np.array([1., np.sqrt(3)])
np.testing.assert_allclose(distance(a, b), np.sqrt(3), rtol=1e-7)
```

In [8]:
```
# Some hidden tests below
### ...
```

Angles

In [9]:
```
# GRADED FUNCTION: DO NOT EDIT THIS LINE

def angle(x0, x1):
    """Compute the angle between two vectors x0, x1 using the dot product.

    Args:
    x0, x1: ndarray of shape (D,) to compute the angle between.

    Returns:
    the angle between the x0 and x1.
    """
    # YOUR CODE HERE
    ### Uncomment and modify the code below
    angle = np.arccos((np.dot(x0,x1)/(np.dot(x0,x0)*np.dot(x1,x1))**(0.5))) # <-- EDIT THIS to compute angle between x0 and x1
    return angle
```

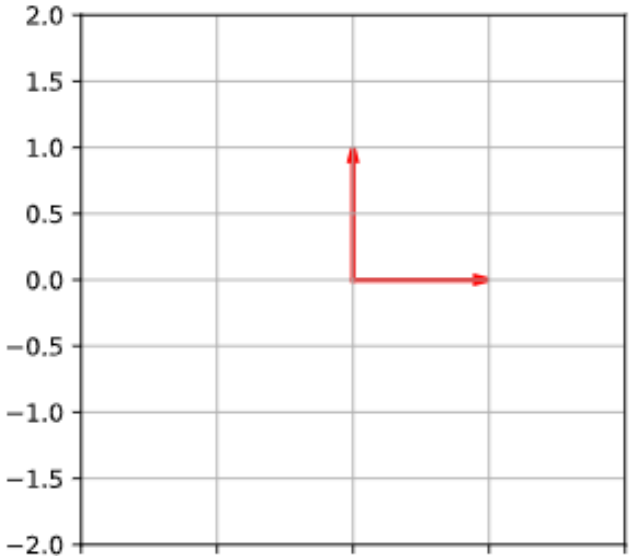
In [10]:
```
# Some sanity checks, you may want to have more interesting test cases to test your implementation
a = np.array([1, 0])
b = np.array([0, 1])
np.testing.assert_allclose(angle(a,b) / (np.pi * 2) * 360., 90)

a = np.array([1, 0])
b = np.array([1., np.sqrt(3)])
np.testing.assert_allclose(angle(a,b) / (np.pi * 2) * 360., 60., rtol=1e-4)
```

In [11]:
```
# Some hidden tests below
```

We have created some helper functions for you to visualize vectors in the cells below. You do not need to modify them.

In [12]:
```
def plot_vector(v, w):
    """Plot two 2D vectors."""
    fig = plt.figure(figsize=(4,4))
    ax = fig.gca()
    plt.xlim([-2, 2])
    plt.ylim([-2, 2])
    plt.grid()
    ax.arrow(0, 0, v[0], v[1], head_width=0.05, head_length=0.1,
             length_includes_head=True, linewidth=2, color='r');
    ax.arrow(0, 0, w[0], w[1], head_width=0.05, head_length=0.1,
             length_includes_head=True, linewidth=2, color='r');
```

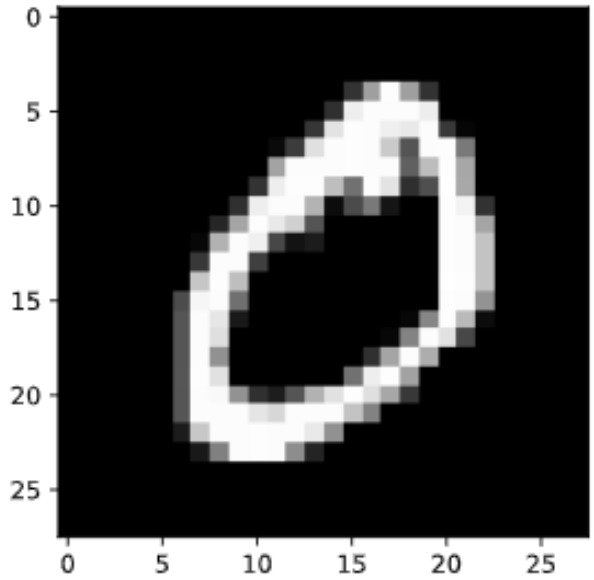
In [13]:
```
a = np.array([1, 0])
b = np.array([0, 1])
plot_vector(b, a)
```



In [14]:
```
# Tests symmetry
random = np.random.RandomState(42)
x = random.randn(3)
y = random.randn(3)
for _ in range(10):
    np.testing.assert_allclose(distance(x,y), distance(y,x))
    np.testing.assert_allclose(angle(x,y), angle(y,x), rtol=1e-4)
```

The next cell shows some digits from the dataset.

In [15]:
```
plt.imshow(images[labels==0].reshape(-1, 28, 28)[0], cmap='gray');
```



But we have the following questions:

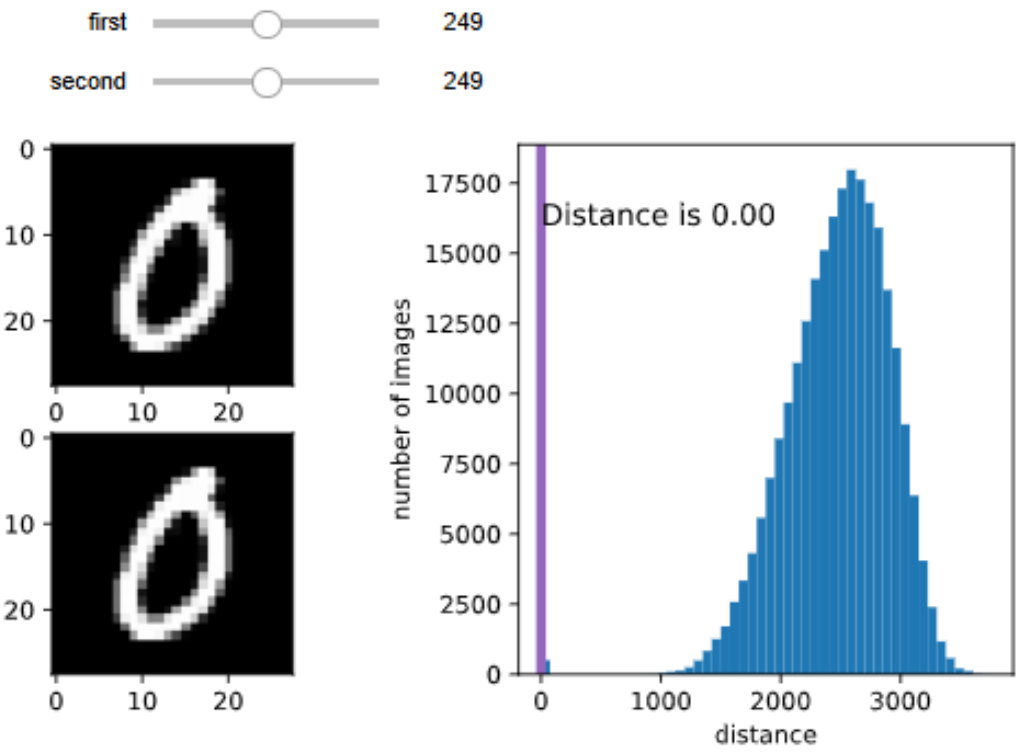
1. What does it mean for two digits in the MNIST dataset to be *different* by our distance function?
2. Furthermore, how are different classes of digits different for MNIST digits? Let's find out!

For the first question, we can see just how the distance between digits compare among all distances for the first 500 digits. The next cell computes pairwise distances between images.

In [16]:
```
distances = []
for i in range(len(images[:500])):
    for j in range(len(images[:500])):
        distances.append(distance(images[i], images[j]))
```

In [17]:
```
@interact(first=(0, 499), second=(0, 499), continuous_update=False)
```

In [17]:
```
@interact(first=(0, 499), second=(0, 499), continuous_update=False)
def show_img(first, second):
    plt.figure(figsize=(8,4))
    f = images[first].reshape(28, 28)
    s = images[second].reshape(28, 28)

    ax0 = plt.subplot2grid((2, 2), (0, 0))
    ax1 = plt.subplot2grid((2, 2), (1, 0))
    ax2 = plt.subplot2grid((2, 2), (0, 1), rowspan=2)

    #plt.imshow(np.hstack([f,s]), cmap='gray')
    ax0.imshow(f, cmap='gray')
    ax1.imshow(s, cmap='gray')
    ax2.hist(np.array(distances), bins=50)
    d = distance(f.ravel(), s.ravel())
    ax2.axvline(x=d, ymin=0, ymax=40000, color='C4', linewidth=4)
    ax2.text(0, 16000, "Distance is {:.2f}".format(d), size=12)
    ax2.set(xlabel='distance', ylabel='number of images')
    plt.show()
```



Next we will find the index of the most similar image to the image at index 0. We will do this by writing some code in another cell.

Write some code in this scratch cell below to find out the most similar image

In [18]:
```
### Scratch cell for you to compute the index of the most similar image
distances = np.zeros((500))
for i in range(500):
    # Write some code to compute the distance between 0th and ith image.
    pass
print(np.argmin(np.array(distances)[1:]) + 1)  # Add one since we excluded the 0th image.
```

```
1
```

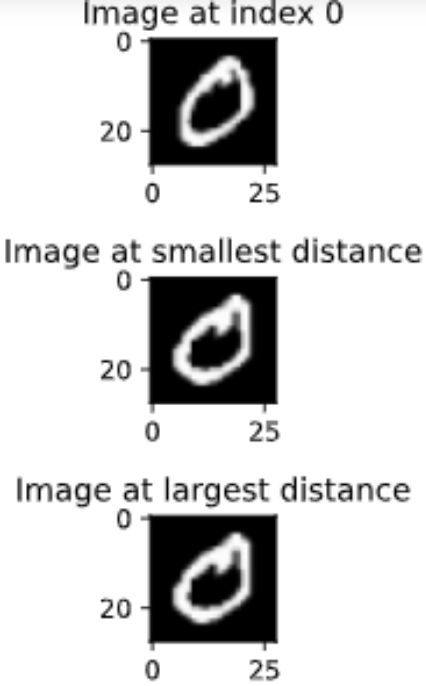
Then copy the solution you found (an index value) and replace the -1 in the function `most_similar_image` with this value. Don't perform any computation in the next cell that accesses the dataset as the autograder will not have access to the dataset and will raise an error.

In [19]:
```
# GRADED FUNCTION: DO NOT EDIT THIS LINE

def most_similar_image():
    """Find the index of the digit, among all MNIST digits (excluding the first),
       that is the closest to the first image in the dataset, your answer should be a single integer
    """
    # YOUR CODE HERE
    idx = 50
    return idx
```

In [20]:
```
### Some hidden tests below
```

Let us similarly find the image which is at the farthest distance from the image at index 0. We shall then plot the images and visualize what it is for an image to be at a smaller distance or a larger distance from another image. Remember that distance in this case is a measure of pixel-wise similarity of two images. Two images which are at a small distance from one another are expected to have similar pixel intensity values.

In [21]:
```
idx_min = np.argmin(np.array(distances)[1:]) + 1
idx_max = np.argmax(np.array(distances)[1:]) + 1

f, ax = plt.subplots(3, 1)
ax[0].imshow(images[0].reshape(28, 28), cmap='gray')
ax[0].set(title='Image at index 0')
ax[1].imshow(images[idx_min].reshape(28, 28), cmap='gray')
ax[1].set(title='Image at smallest distance')
ax[2].imshow(images[idx_max].reshape(28, 28), cmap='gray')
ax[2].set(title='Image at largest distance')
plt.tight_layout()
plt.show()
```



Clearly, the first two images overlap more than the first and third image do.

For the second question, we can compute a `mean` image for each class of image, i.e. we compute mean image for digits of `1`, `2`, `3`,..., `9`, then we compute pairwise distance between them. We can organize the pairwise distances in a 2D plot, which would allow us to visualize the dissimilarity between images of different classes.

First we compute the mean for digits of each class.

In [22]:
```
mean_images = {}
for n in np.unique(labels):
    mean_images[n] = np.mean(images[labels==n], axis=0)
```

For each pair of classes, we compute the pairwise distance and store them into MD (mean distances). We store the angles between the mean digits in AG

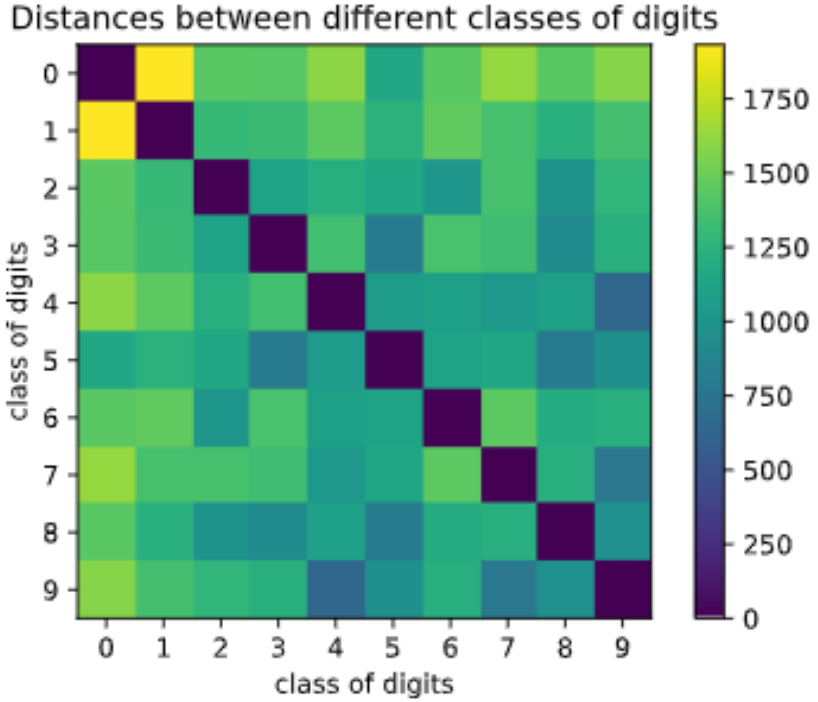
In [23]:
```
MD = np.zeros((10, 10))
AG = np.zeros((10, 10))
for i in mean_images.keys():
    for j in mean_images.keys():
        MD[i, j] = distance(mean_images[i], mean_images[j])
        AG[i, j] = angle(mean_images[i].ravel(), mean_images[j].ravel())
```

```
    for j in mean_images.keys():
        MD[i, j] = distance(mean_images[i], mean_images[j])
        AG[i, j] = angle(mean_images[i].ravel(), mean_images[j].ravel())
```

Now we can visualize the distances! Here we put the pairwise distances. The colorbar shows how the distances map to color intensity. Which digits do you think are the most similar to one another and would have the least distance between them? Try to overlay a few handwritten digits on top of one another to answer this! Consider 4 and 9 or 1 and 0. Are there any other examples which make sense to you?

Once you are ready, execute the code cell below to test your hypothesis.

In [24]:
```
fig, ax = plt.subplots()
grid = ax.imshow(MD, interpolation='nearest')
ax.set(title='Distances between different classes of digits',
       xticks=range(10),
       xlabel='class of digits',
       ylabel='class of digits',
       yticks=range(10))
fig.colorbar(grid)
plt.show()
```



Similarly for the angles.

In [25]:
```
fig, ax = plt.subplots()
AG = np.nan_to_num(AG)
grid = ax.imshow(AG, interpolation='nearest')
ax.set(title='Angles between different classes of digits',
       xticks=range(10),
       xlabel='class of digits',
       ylabel='class of digits',
       yticks=range(10))
fig.colorbar(grid)
plt.show();
```