week2

October 30, 2019

0.1 Distances and Angles between Images

We are going to compute distances and angles between images.

0.2 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 first use the functions you wrote to compare MNIST digits. Furthermore, we will use these concepts for implementing the K Nearest Neighbors algorithm, which is a useful algorithm for classifying object according to distance.

```
In [2]: # PACKAGE: DO NOT EDIT THIS LINE
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import numpy as np
        import scipy

        import sklearn
        from ipywidgets import interact
        from load_data import load_mnist
```

The next cell loads the MNIST digits dataset.

```
In [6]: MNIST = load_mnist()
        images = MNIST['data'].astype(np.double)
        labels = MNIST['target'].astype(np.int)
```

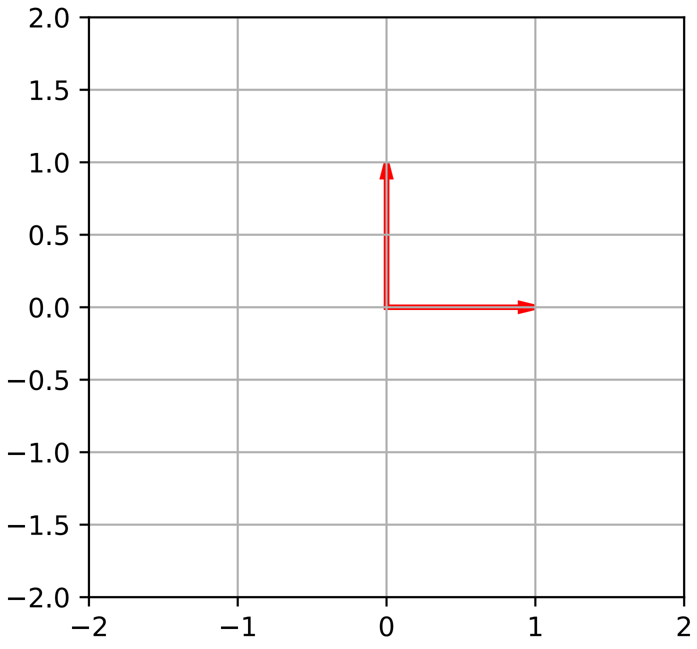
```
In [7]: # Plot figures so that they can be shown in the notebook
        %matplotlib inline
        %config InlineBackend.figure_format = 'svg'
```

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.

```
In [33]: # GRADED FUNCTION: DO NOT EDIT THIS LINE

         def distance(x0, x1):
             """Compute distance between two vectors x0, x1 using the dot product"""
             xdotx = np.dot(x1-x0,x1-x0)
             distance = np.sqrt(xdotx)
             return distance

         def angle(x0, x1):
             """Compute the angle between two vectors x0, x1 using the dot product"""
             lenx0 = np.sqrt(np.dot(x0,x0))
             lenx1 = np.sqrt(np.dot(x1,x1))
             x0dotx1 = np.dot(x0,x1)
             angle = np.arccos((x0dotx1/(lenx0*lenx1))) # <-- EDIT THIS to compute angle betwe
             return angle
```

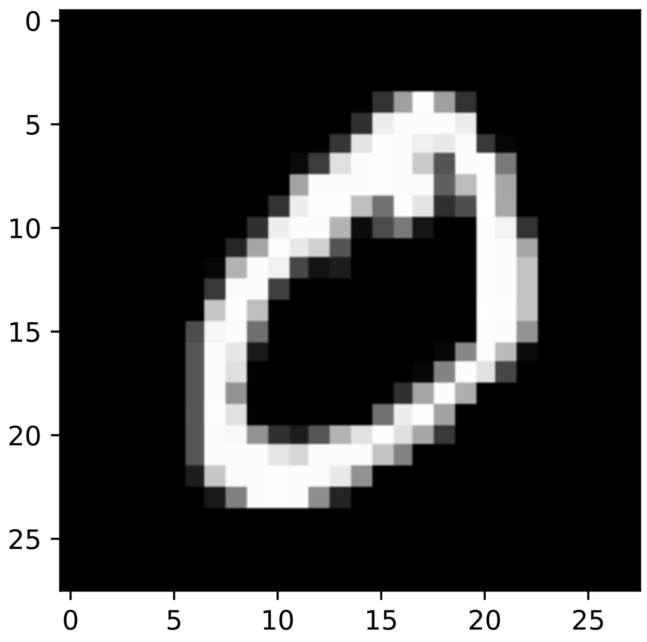
We have created some helper functions for you to visualize vectors in the cells below. You do not need to modify them.

```
In [34]: def plot_vector(v, w):
             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 [35]: # Some sanity checks, you may want to have more interesting test cases to test your i
         a = np.array([1,0])
         b = np.array([0,1])
         np.testing.assert_almost_equal(distance(a, b), np.sqrt(2))
         assert((angle(a,b) / (np.pi * 2) * 360.) == 90)

In [36]: plot_vector(b, a)
```



The next cell shows some digits from the dataset.

```
In [37]: 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 [38]: distances = []
         for i in range(len(images[:500])):
             for j in range(len(images[:500])):
                 distances.append(distance(images[i], images[j]))
```

```
In [39]: @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, 46000, "Distance is {:.2f}".format(d), size=12)
             ax2.set(xlabel='distance', ylabel='number of images')
             plt.show()
```

```
interactive(children=(IntSlider(value=249, description='first', max=499), IntSlider(value=249,
```

```
In [40]: distOfFirst = distances[0]
         mindist = abs(distances[1]-distOfFirst)
         retVal = 1
         for i in range(len(distances)):
             if i>0:
                 diff = (distances[i]-distOfFirst)
                 if diff > 0 and diff < mindist:
                     mindist = diff
                     retVal = i
                     print(diff)
                     print(i)
         print(retVal)
```

```
1020.6473436
61
1013.00098717
6019
1011.97381389
9170
976.93551476
15540
615.939932136
36576
36576
```

```
In [41]: # GRADED FUNCTION: DO NOT EDIT THIS LINE
         def most_similar_image():
             """Find the index of the digit, among all MNIST digits
                that is the second-closest to the first image in the dataset (the first image
                Your answer should be a single integer.
             """
             index = 61 #<-- Change the -1 to the index of the most similar image.
             # You should do your computation outside this function and update this number
             # once you have computed the result
             return index
```

```
In [42]: result = most_similar_image()
```

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

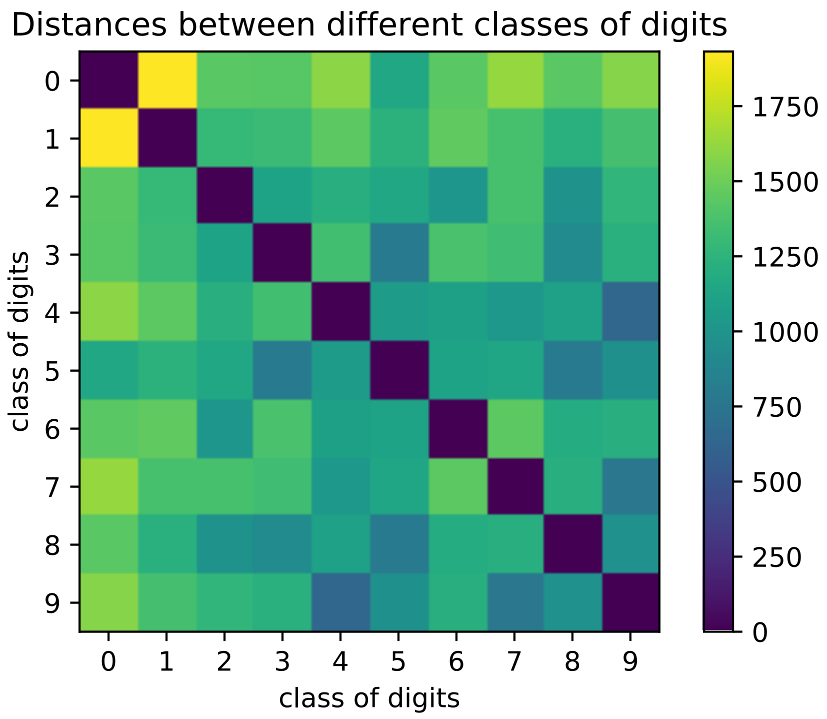
For each pair of classes, we compute the pairwise distance and store them into MD (mean distances). We store the angles between the mean digits in AG

```
In [44]: MD = np.zeros((10, 10))
         AG = np.zeros((10, 10))
         for i in means.keys():
             for j in means.keys():
                 MD[i, j] = distance(means[i], means[j])
                 AG[i, j] = angle(means[i].ravel(), means[j].ravel())
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:14: RuntimeWarning: invalid value
```

Now we can visualize the distances! Here we put the pairwise distances. The colorbar shows how the distances map to color intensity.

```
In [45]: 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 [46]: fig, ax = plt.subplots()
         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();
```