CSU Online DSCI 369 Lab 2

Original lab written by: Emily J. King

Goals: Generate one-hot encoding vectors. Interpret networks and graphs into numeric data and back. Use heatmaps to visualize arrays.

Additional files needed: DSCI_369_Module_2_Lab.pdf

In this lab, in addition to NumPy, we also need MatPlotLib in order to create visualizations.

In [252…
```
import numpy as np
import matplotlib.pyplot as plt
```

One-hot encoding vectors (called standard basis vectors in math) are vectors with all entries equal to zero except for a single entry equal to one. Such vectors are the columns of very special matrices called identity matrices. We will learn more about identity matrices when we learn how multiply matrices. For now, they provide a useful short cut to make one-hot encoding vectors.

Let's look at a few identity matrices.

In [253…
```
np.eye(2)
```

Out[253…
```
array([[1., 0.],
       [0., 1.]])
```

In [254…
```
np.eye(3)
```

Out[254…
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [255…
```
np.eye(4)
```

Out[255…
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

So, if we wanted the one-hot encoding vectors for CIFAR-10, we would take the columns of the 10x10 identity matrix.

In [256…
```
E=np.eye(10)
E[:,0]
```

Out[256…
```
array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [257…
```
E[:,1]
```

Out[257…
```
array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [258…
```
E[:,2]
```

Out[258…
```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.])
```

... and so on.

Now let's practice entering adjacency matrice for different type of graph/networks. We begin with the undirected, unweighted graph we saw in lecture. Remind yourself what it looks like by opening Module2Lab.pdf. It has the following adjacency matrix.

In [259…
```
A = np.array([[0, 1, 0, 0, 1, 0], [1, 0, 1, 0, 1, 0], [0, 1, 0, 1, 0, 0], [0
print(A)
```

```
[[0 1 0 0 1 0]
 [1 0 1 0 1 0]
 [0 1 0 1 0 0]
 [0 0 1 0 1 1]
 [1 1 0 1 0 0]
 [0 0 0 1 0 0]]
```

Recall that the transpose of a matrix turns the rows into columns and the columns into rows. Let's transpose that above matrix. MATRIX.T is the shortest command to compute a matrix tranpose.

In [260…
```
A.T
```

Out[260…
```
array([[0, 1, 0, 0, 1, 0],
       [1, 0, 1, 0, 1, 0],
       [0, 1, 0, 1, 0, 0],
       [0, 0, 1, 0, 1, 1],
       [1, 1, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0]])
```

Note that these two matrices are equal.

In [261…
```
np.allclose(A,A.T)
```

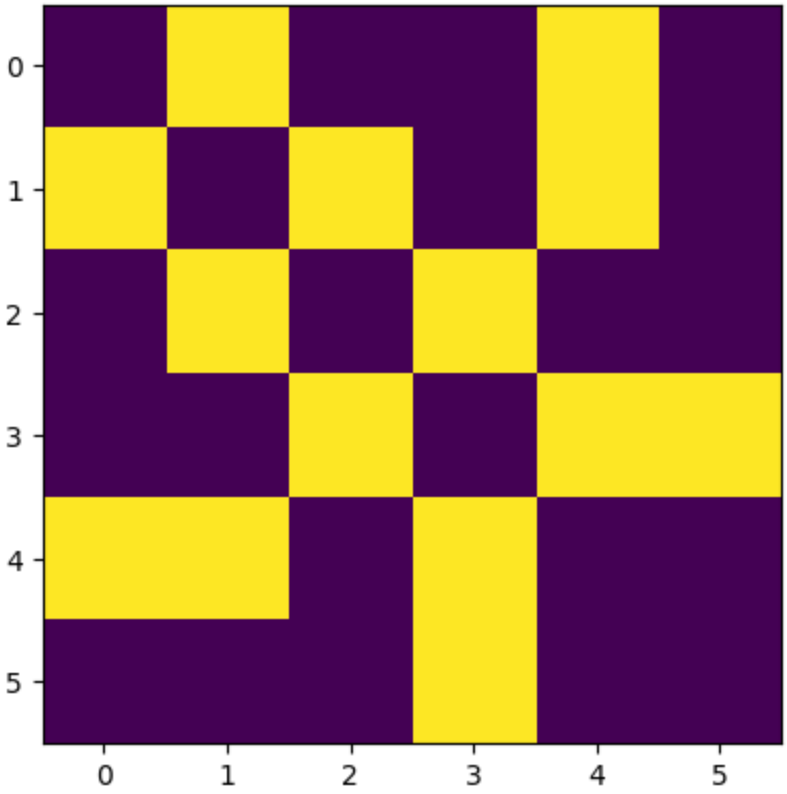
Out[261…
```
True
```

This makes sense. The graph is undirected, meaning that if vertex i is connected to vertex j, vertex j is connected to vertex i. That means that the (i,j) entry of the adjacency matrix is equal to the (j,i) entry and the adjacency matrix is equal to its own transpose. When this happens, we call it a symmetric matrix.

Now we would like to view the heatmap of the adjacency matrix. The simplest command to do this follows.

In [262…
```
plt.imshow(A)
```

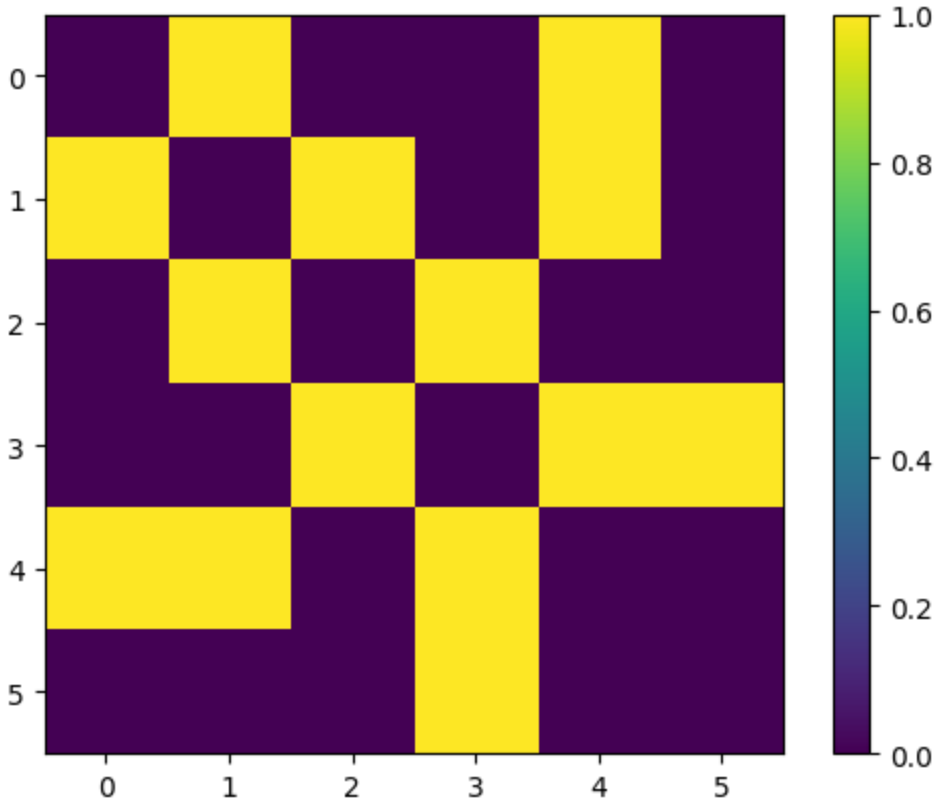
Out[262…
```
<matplotlib.image.AxesImage at 0x12ab2c470>
```



However, if we would like to add a colorbar so that we understand what numbers each of the colors stand for, we need more complicated code.

In [263…
```
fig, ax = plt.subplots()
pos = ax.imshow(A)
fig.colorbar(pos)
```

Out[263…
```
<matplotlib.colorbar.Colorbar at 0x12aee36e0>
```



Now let's practice entering the adjacency matrix of the directed, unweighted graph we saw in lecture. Again, look at the lab's PDF to remind yourself of what it looks like. We need to be careful about how we label the rows in columns. Here we will use the convention that rows stand for outgoing and columns for incoming. I.e.. if there is a edge from vertex i to vertex j, we put a 1 in (i,j).

In [264…
```
ADir = np.array([[0, 1, 0, 0],[1, 0, 1, 0], [0, 0, 0, 0], [0, 0, 0, 0]])
print(ADir)
```

```
[[0 1 0 0]
 [1 0 1 0]
 [0 0 0 0]
 [0 0 0 0]]
```

Note that vertex 2 is connected to vertex 3 but not vice versa. Thus, the adjacency matrix is not symmetric.

In [265…
```
np.allclose(ADir,ADir.T)
```

Out[265… `False`

Finally, we enter the adjacency matrix of the weighted graph from DSCI_369_Module_2_Lab.pdf.

In [266…
```
AWei = np.array([[0, -0.4, 0.3], [-0.4, 0, 1.1], [0.3, 1.1, 0]])
print(AWei)
```

```
[[ 0.  -0.4  0.3]
 [-0.4  0.   1.1]
 [ 0.3  1.1  0. ]]
```

Since the graph is undirected, the adjacency matrix is symmetric.

In [267…
```
np.allclose(AWei,AWei.T)
```

Out[267… True

Exercises

1. Enter the adjacency matrix for the unweighted, directed blue graph on 3 vertices from the lab sheet. Call the matrix A1.

In [268…
```
A1 = np.array([[0, 0, 0], [1, 0, 0], [0, 1, 0]])
print(A1)
```

```
[[0 0 0]
 [1 0 0]
 [0 1 0]]
```

2. Enter the adjacency matrix for the unweighted, undirected black graph on 4 vertices from the lab sheet. Call the matrix A2.

In [269…
```
A2 = np.array([[0, 1, 1, 1], [1, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 0]])
print(A2)
```

```
[[0 1 1 1]
 [1 0 1 1]
 [1 1 0 1]
 [1 1 1 0]]
```

3. Enter the adjacency matrix for the weighted, undirected green graph on 5 vertices from the lab sheet. Call the matrix A3.

In [270…
```
A3 = np.array([[0, -2, 0, 0, 3], [-2, 0, 5, 0, 0], [0, 5, 0, 1, 0], [0, 0, 1
print(A3)
```

```
[[  0  -2   0   0   3]
 [ -2   0   5   0   0]
 [  0   5   0   1   0]
 [  0   0   1   0 -11]
 [  3   0   0 -11   0]]
```