Numpy Tutorial Supplement

CS114B Lab 1

Kenneth Lai

January 20, 2023

Numpy Tutorial

- https://cs231n.github.io/python-numpy-tutorial/

Numpy Arrays, Row Vectors, and Column Vectors

- ```
  >>> import numpy as np

  >>> a = np.array([1, 2, 3])
  >>> b = np.array([[1, 2, 3]])
  >>> c = np.array([[1], [2], [3]])
  ```
  ```

Numpy Arrays, Row Vectors, and Column Vectors

- ```
  >>> import numpy as np

  >>> a = np.array([1, 2, 3])
  >>> b = np.array([[1, 2, 3]])
  >>> c = np.array([[1], [2], [3]])
  ```
- ```
  >>> b
  array([[1, 2, 3]])
  >>> b.shape
  (1, 3)
  ```

  $$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

  - "Row vector" (really a row matrix)

Numpy Arrays, Row Vectors, and Column Vectors

-
```
>>> import numpy as np

>>> a = np.array([1, 2, 3])
>>> b = np.array([[1, 2, 3]])
>>> c = np.array([[1], [2], [3]])
```
-
```
>>> c
array([[1],
       [2],
       [3]])
>>> c.shape
(3, 1)
```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

  - "Column vector" (really a column matrix)

Numpy Arrays, Row Vectors, and Column Vectors

- ```
  >>> import numpy as np

  >>> a = np.array([1, 2, 3])
  >>> b = np.array([[1, 2, 3]])
  >>> c = np.array([[1], [2], [3]])
  ```
- ```
  >>> a
  array([1, 2, 3])
  >>> a.shape
  (3,)
  ```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \text{ or } \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

  - Vector (either row or column)
    - Be sure to know when your vector is acting as a row or acting as a column!

Views and (Deep) Copies

- Slicing creates a view (shallow copy) of the data; modifying it will modify the original array
- ```
  >>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
  >>> b = a[:2, 1:3]
  >>> b += 1
  >>> a
  array([[ 1,  3,  4,  4],
         [ 5,  7,  8,  8],
         [ 9, 10, 11, 12]])
  ```
  ```

Views and (Deep) Copies

- Integer and boolean indexing create (deep) copies of the data; modifying them will not modify the original array
-
```
>>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> c = a[[0, 1, 2], [0, 1, 0]]
>>> d = a[a > 6]
>>> c += 1
>>> d += 1
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Views and (Deep) Copies

- To create a (deep) copy of a slice, use `copy()`
-
```
>>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> e = a[:2, 1:3].copy()
>>> e += 1
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Preserving Rank

- Mixing integer indexing with slices yields an array of lower rank, while using only slices yields an array of the same rank as the original array
- ```
  >>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
  >>> row_r1 = a[1, :]
  >>> row_r2 = a[1:2, :]
  >>> print(row_r1, row_r1.shape)
  [5 6 7 8] (4,)
  >>> print(row_r2, row_r2.shape)
  [[5 6 7 8]] (1, 4)
  ```
  ```

Preserving Rank

- (At least) three other ways to get a new axis:

1. Use `numpy.expand_dims`:

- ```
  >>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
  >>> row_r2a = np.expand_dims(a[1, :], 0)
  >>> print(row_r2a, row_r2a.shape)
  [[5 6 7 8]] (1, 4)
  >>> row_r2b = np.expand_dims(a[1, :], 1)
  >>> print(row_r2b, row_r2b.shape)
  [[5]
   [6]
   [7]
   [8]] (4, 1)
  ```
  ```

Preserving Rank

2. Use `reshape()`:

```
>>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> row_r2c = a[1, :].reshape((1, 4))
>>> print(row_r2c, row_r2c.shape)
[[5 6 7 8]] (1, 4)
>>> row_r2d = a[1, :].reshape((4, 1))
>>> print(row_r2d, row_r2d.shape)
[[5]
 [6]
 [7]
 [8]] (4, 1)
```

Preserving Rank

3. Use `None` (or `numpy.newaxis`):

-
```
>>> a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> row_r2e = a[None, 1, :]
>>> print(row_r2e, row_r2e.shape)
[[5 6 7 8]] (1, 4)
```
- Note that the new axis is inserted in the position of `None`:
-
```
>>> row_r2f = a[1, :, None]
>>> print(row_r2f, row_r2f.shape)
[[5]
 [6]
 [7]
 [8]] (4, 1)
```

numpy.dot

- Product that depends on the shapes of its arguments:
- ```
  >>> x = np.array([[1, 2], [3, 4]])
  >>> y = np.array([[5, 6], [7, 8]])
  >>> v = np.array([9, 10])
  >>> w = np.array([11, 12])
  ```

1. Two vectors: inner product ("true" dot product)

- ```
  >>> np.dot(v, w)
  219
  ```

$$\begin{bmatrix} 9 & 10 \end{bmatrix} \cdot \begin{bmatrix} 11 & 12 \end{bmatrix} = 9 \times 11 + 10 \times 12 = 219$$

numpy.dot

2. Two matrices: matrix product

- `>>> np.dot(x, y)`

```
array([[19, 22],
       [43, 50]])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

- Alternative: use `@` (or `numpy.matmul`)
- `>>> x @ y`

```
array([[19, 22],
       [43, 50]])
```

numpy.dot

3. Matrix and vector (in either order): sum product

- `>>> np.dot(x, v)`
  `array([29, 67])`

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 9 \\ 10 \end{bmatrix} = \begin{bmatrix} 1 \times 9 + 2 \times 10 \\ 3 \times 9 + 4 \times 10 \end{bmatrix} = \begin{bmatrix} 29 \\ 67 \end{bmatrix}$$

- Note that `v` is acting as a column!

numpy.dot

3. Matrix and vector (in either order): sum product

- `>>> np.dot(v, x)`
  `array([39, 58])`

$$\begin{bmatrix} 9 & 10 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 9 \times 1 + 10 \times 3 & 9 \times 2 + 10 \times 4 \end{bmatrix}$$

$$= \begin{bmatrix} 39 & 58 \end{bmatrix}$$

- Note that `v` is acting as a row!

Axes

- When giving an axis to a function like `numpy.sum` (or `numpy.argmax`, etc.), the axis is the axis you want to "squash"
  - This is why `sum(x, axis=0)` computes the sum of each column, and `sum(x, axis=1)` computes the sum of each row
- ```
  >>> x = np.array([[1, 2], [3, 4]])
  >>> x
  array([[1, 2],
         [3, 4]])
  >>> np.sum(x, axis=0)
  array([4, 6])
  >>> np.sum(x, axis=1)
  array([3, 7])
  ```
  ```

Broadcasting

- (this is also in the tutorial, but it's important enough to see again!)
- Rules:
  1. If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length
  2. In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension

Broadcasting

- ```
  >>> x = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
  >>> v = np.array([1, 0, 1])
  ```

	x	v	x+v
	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$	
Shape	(4, 3)	(3,)	
  ```

Broadcasting

- ```
  >>> x = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
  >>> v = np.array([1, 0, 1])
  ```

	x	v	x+v
	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$	
Shape	(4, 3)	(1, 3)	

- Prepend 1's
  - Note that v is acting as a row!
  ```

Broadcasting

- ```
  >>> x = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
  >>> v = np.array([1, 0, 1])
  ```

	x	v	x+v
	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$	
Shape	(4, 3)	(4, 3)	

- Make copies
  ```

Broadcasting

- ```
  >>> x = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
  >>> v = np.array([1, 0, 1])
  ```

	x	v	x+v
	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 & 4 \\ 5 & 5 & 7 \\ 8 & 8 & 10 \\ 11 & 11 & 13 \end{bmatrix}$
Shape	(4, 3)	(4, 3)	(4, 3)

- Add element-wise
  ```