COMPSCI 590N
Lecture 6: NumPy 2

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

Outline

1 Shape Manipulation and Broadcasting

2 Miscellaneous NumPy

Shape Manipulation

NumPy provides various functions for changing the shape/size of an array. The most basic is `reshape` which returns an array with a new shape.

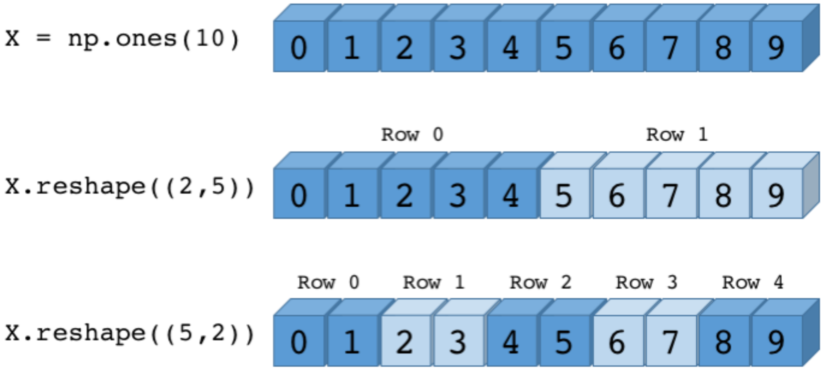
Shape Manipulation

NumPy provides various functions for changing the shape/size of an array. The most basic is `reshape` which returns an array with a new shape.

```
>>> A = np.arange(9)
>>> A
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> A.reshape((3,3))
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

>>> A.shape = (3,3)
>>> A
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Shape Manipulation



```
X = np.ones(10)
X.reshape((2,5))
X.reshape((5,2))
```

Shape Manipulation

Python allows you to specify a **single** unknown dimension using -1.

Shape Manipulation

Python allows you to specify a **single** unknown dimension using -1.

```
>>> A = np.arange(6)
>>> A.
>>> A.reshape((2,-1))
array([[0, 1, 2],
       [3, 4, 5]])
>>> A.reshape((-1,2))
array([[0, 1],
       [2, 3],
       [4, 5]])
```

Shape Manipulation

Other useful functions return specific shapes:

Shape Manipulation

Other useful functions return specific shapes:

```
>>> A.ravel()
array([0, 1, 2, 3, 4, 5])

>>> A.transpose() # Also use a.T
array([[0, 3],
       [1, 4],
       [2, 5]])
```

Fancy Indexing

NumPy arrays can be indexed with arrays of booleans. This is sometimes called *masking*.

Fancy Indexing

NumPy arrays can be indexed with arrays of booleans. This is sometimes called *masking*.

```
>>> A = np.arange(10)
>>> B = (A%2) == 0
>>> A[B] # Select all even numbers
array([0, 2, 4, 6, 8])

>>> A = np.array([[0,-1],[2,3],[-3,2]])
>>> A
array([[ 0, -1],
       [ 2,  3],
       [-3,  2]])
>>> B = A.sum(1) < 0
>>> A[B,:] # Select all rows whose sum is < 0
array([[ 0, -1],
       [-3,  2]])
```

Fancy Indexing

Arrays can also be indexed using other **integer** arrays.

Fancy Indexing

Arrays can also be indexed using other **integer** arrays.

```
>>> A = 2*np.arange(10)
>>> B = np.array([1,4,5,7])
>>> A[B]
array([ 2,  8, 10, 14])

>>> A = np.arange(6).reshape((2,3))
>>> A
array([[0, 1, 2],
       [3, 4, 5]])
>>> B = np.array([0,0,1])
>>> C = np.array([1,2,2])
>>> A[B,C] # One array for each dimension
array([1, 2, 5])
```

Broadcasting

In most cases, when the shapes of two arrays do not match, NumPy will not let you perform elementwise operation between them.

Broadcasting

In most cases, when the shapes of two arrays do not match, NumPy will not let you perform elementwise operation between them.

```
>>> A = np.arange(6).reshape((2,3))
>>> A
array([[0, 1, 2],
       [3, 4, 5]])
>>> B = np.arange(5)
>>> A * B
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together
    with shapes (2,3) (5,)
```

Broadcasting

In some special cases, NumPy will replicate one or more of the inputs so that it can perform the desired operation. This is called **broadcasting**.

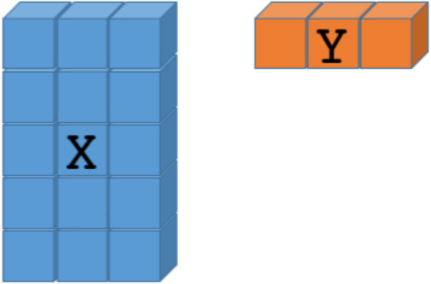
Broadcasting

In some special cases, NumPy will replicate one or more of the inputs so that it can perform the desired operation. This is called **broadcasting**.

```
>>> A = np.arange(6).reshape((2,3))
>>> A
array([[0, 1, 2],
       [3, 4, 5]])
>>> B = np.arange(3)
>>> A * B # B is copied for each row of A
array([[ 0,  1,  4],
       [ 0,  4, 10]])
```

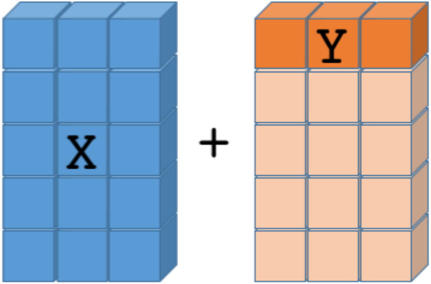
Broadcasting

```
X = np.ones((5,3))
Y = np.ones(3)
```



Broadcasting

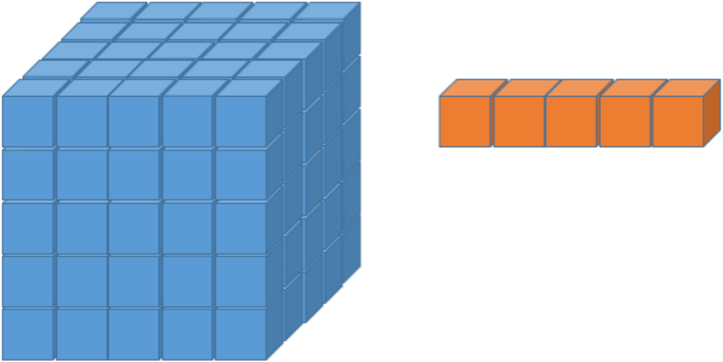
```
X = np.ones((5,3))
Y = np.ones(3)
Z = X + Y
```



Broadcasting

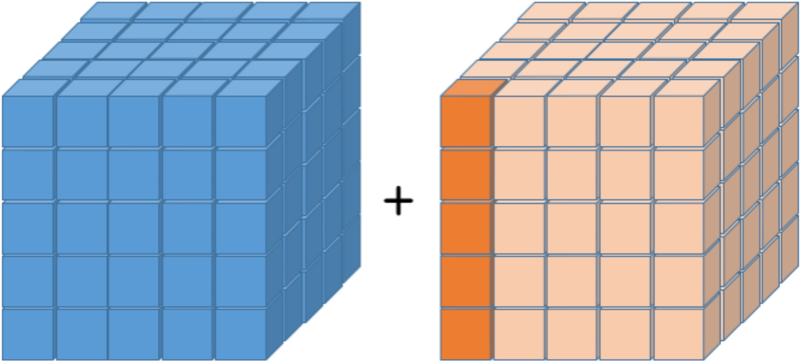
```
X = np.ones((5,5,5))
```

```
Y = np.ones(5)
```



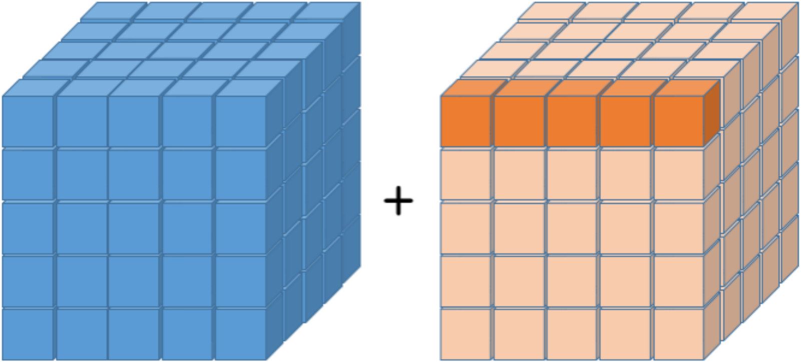
Broadcasting

```
Z = X + Y.reshape((5,1,1))
```



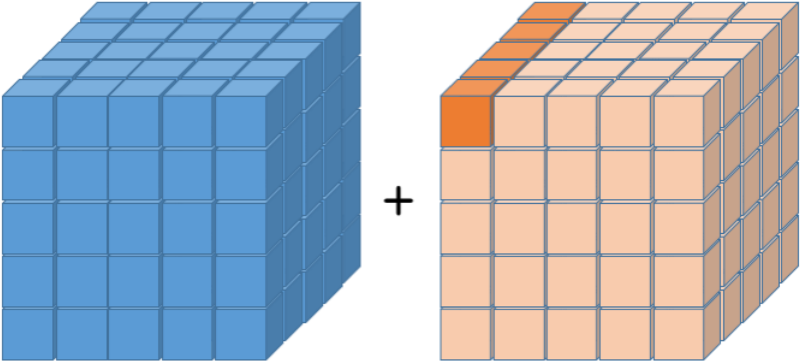
Broadcasting

```
Z = X + Y.reshape((1,5,1))
```



Broadcasting

```
Z = X + Y.reshape((1,1,5))
```



Interactive Demo

Interactive Demo

1. How do you set all of the negative elements in an array to zero?
2. Recall that you can set an entire slice of an array by using indexed assignment. What happens when assign to a slice indexed by an array and there are duplicates in the **index** array?
   - For example:
     `a[np.array([1,1,2])] = np.array([1,2,3])`
3. Use broadcasting to calculate the outer product of two one dimensional arrays (i.e. $A \otimes B = AB^T$).

Outline

1 Shape Manipulation and Broadcasting

2 Miscellaneous NumPy

Reductions

Reductions compute aggregate statistics on the array. For example:

Reductions

Reductions compute aggregate statistics on the array. For example:

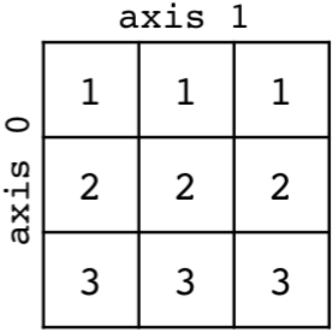
```
>>> A = np.arange(6).reshape((3,2))
>>> A.sum()
15
```

Reductions

Most reductions take an argument `axis` which allow you to specify the dimension along which the aggregation takes place.

Reductions

Most reductions take an argument `axis` which allow you to specify the dimension along which the aggregation takes place.

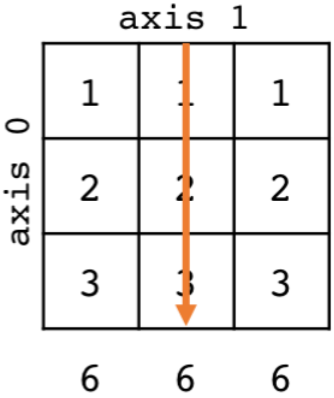


1	1	1
2	2	2
3	3	3

Reductions

Most reductions take an argument `axis` which allow you to specify the dimension along which the aggregation takes place.

```
X.sum(axis = 0)
```



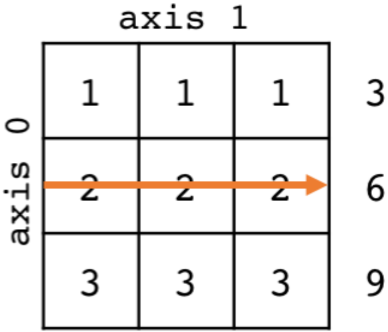
1	1	1
2	2	2
3	3	3

6 6 6

Reductions

Most reductions take an argument `axis` which allow you to specify the dimension along which the aggregation takes place.

```
X.sum(axis = 1)
```



1	1	1	3
2	2	2	6
3	3	3	9

Reductions

Other common reductions are:

Reductions

Other common reductions are:

```
>>> A = np.arange(6).reshape((3,2))
>>> A.prod(axis=0)
array([ 0,  6, 20])
>>> A.max(axis=1) # Also min
array([1, 3, 5])
>>> A.argmax(axis=1) # Also argmin
array([1, 1, 1])
>>> A.mean(axis=1)
array([ 0.5,  2.5,  4.5])
>>> A.std(axis=-1) # -1 corresponds to the last
    dimension
array([ 0.5,  0.5,  0.5])
>>> np.median(A,axis=1) # Not a member function
array([ 0.5,  2.5,  4.5])
```

Stacking

NumPy provides a number of functions for combining arrays.

Stacking

NumPy provides a number of functions for combining arrays.

```
>>> A = np.ones((3,4))
>>> B = np.ones((3,4))
>>> np.vstack((A,B)).shape # first dimension
(6, 4)
>>> np.hstack((A,B)).shape # second dimension
(3, 8)
>>> np.stack((A,B)).shape # new dimension
(2, 3, 4)
>>> np.concatenate((A,B),axis=0).shape # any dimension
(6, 4)
>>> np.concatenate((A,B),axis=1).shape # any dimension
(3, 8)
```

Stacking

NumPy provides a number of functions for combining arrays.

```
>>> A = np.ones((3,4))
>>> B = np.ones((3,4))
>>> np.vstack((A,B)).shape # first dimension
(6, 4)
>>> np.hstack((A,B)).shape # second dimension
(3, 8)
>>> np.stack((A,B)).shape # new dimension
(2, 3, 4)
>>> np.concatenate((A,B),axis=0).shape # any dimension
(6, 4)
>>> np.concatenate((A,B),axis=1).shape # any dimension
(3, 8)
```

Analogues to each of these functions exist for splitting arrays.

Copy

Because the contents of an array are mutable, we often need to copy the contents of an array to avoid overwriting the original.

Copy

Because the contents of an array are mutable, we often need to copy the contents of an array to avoid overwriting the original.

```
>>> A = np.ones((3,4))
>>> B = A
>>> B is A
True
>>> B = A.copy()  # Create a new copy of 'A' in memory
>>> B is A
False
```

Basic Linear Algebra

NumPy implements many of the standard linear algebra functions:

- `np.dot(A,B)`: $A^T B$

Basic Linear Algebra

NumPy implements many of the standard linear algebra functions:

- `np.dot(A,B)`: $A^T B$
- `np.outer(A,B)`: AB^T

Basic Linear Algebra

NumPy implements many of the standard linear algebra functions:

- `np.dot(A,B)`: $A^T B$
- `np.outer(A,B)`: AB^T
- `np.trace(A)`: $tr(A)$

Basic Linear Algebra

NumPy implements many of the standard linear algebra functions:

- `np.dot(A,B)`: $A^T B$
- `np.outer(A,B)`: AB^T
- `np.trace(A)`: $tr(A)$
- `np.inv(A)`: A^{-1}

Basic Linear Algebra

NumPy implements many of the standard linear algebra functions:

- `np.dot(A,B)`: A^TB
- `np.outer(A,B)`: AB^T
- `np.trace(A)`: $tr(A)$
- `np.inv(A)`: A^{-1}
- `np.svd(A)`: Singular value decomposition.

Basic Linear Algebra

NumPy implements many of the standard linear algebra functions:

- `np.dot(A,B)`: $A^T B$
- `np.outer(A,B)`: AB^T
- `np.trace(A)`: $tr(A)$
- `np.inv(A)`: A^{-1}
- `np.svd(A)`: Singular value decomposition.
- And many more...

File I/O

NumPy has two main sets of File I/O functions. `load` and `save` read and write .npy files.

File I/O

NumPy has two main sets of File I/O functions. `load` and `save` read and write .npy files.

```
>>> np.save("my_array.npy",my_array)
>>> my_array = np.load("my_array.npy")
```

File I/O

NumPy has two main sets of File I/O functions. `load` and `save` read and write .npy files.

```
>>> np.save("my_array.npy",my_array)
>>> my_array = np.load("my_array.npy")
```

`loadtxt` and `savetxt` read and write text files. The `delimiter` argument specifies what character will be used to separate entries and defaults to tab.

File I/O

NumPy has two main sets of File I/O functions. `load` and `save` read and write .npy files.

```
>>> np.save("my_array.npy",my_array)
>>> my_array = np.load("my_array.npy")
```

`loadtxt` and `savetxt` read and write text files. The `delimiter` argument specifies what character will be used to separate entries and defaults to tab.

```
>>> np.savetxt("my_array.csv",my_array,delimiter=’,’)
>>> my_array = np.loadtxt("my_array.npy",delimiter=’,’)
```

’Object’ Arrays

In general arrays store only a single type, however, you can create mixed type arrays by setting the data type to `object`. In python, everything an `object` so these arrays can store anything.

’Object’ Arrays

In general arrays store only a single type, however, you can create mixed type arrays by setting the data type to `object`. In python, everything an `object` so these arrays can store anything.

```
>>> A = np.array([1,2.0,"string"],dtype=object)
>>> A + A
array([2, 4.0, ’stringstring’], dtype=object)
>>> A / 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: ’str’ and
    ’int’
```

Interactive Demo

Interactive Demo

1. What is the difference between the reductions `sum` and `cumsum`?
2. `view()` can be used in a similar way as copy, but has a slightly different effect. Try to figure out what view does.