NUMPY:

BROADCASTING

9.28.2018

RECAP: ASSIGNMENT

* you can assign values to specific elements in a numpy ndarray

* assignment can either be done with a single value (i.e. “set all of these elements equal to this one value”)

* or with an array of values (i.e. “set these 34 elements to these 34 values”)

RECAP: ASSIGNMENT

* when you SLICE an array, you create a VIEW on the same data

* when you INDEX an array, you create a COPY with new data

* you can force a slice to be a copy using the .copy() method

BROADCASTING

* suppose we have an array called arr with arr.shape = (50, 10), which we can think of as 50 length-10 vectors (each one is a row)

* suppose we have another array called to_add with to_add.shape = (10,) (i.e. to_add is a length-10 vector)

* now suppose we want to add to_add to each of the 50 rows in arr

BROADCASTING

* one way to do this would be using a for loop

* this stinks

BROADCASTING

* in *sigh* MATLAB, in addition to for loops you have the option of using the always-inscrutable “cellfun”, “arrayfun”, and “bsxfun” functions

* good luck

BROADCASTING

* fortunately, numpy provides a solution!

* it’s called “**broadcasting**”, in case you hadn’t guessed that

BROADCASTING

* with broadcasting, all you need to do is:

  >>> arr + to_add

* holy crap that’s simple

BROADCASTING

* broadcasting (effectively) stretches your arrays so that they match in dimensionality and can be added/multiplied/whatever!

BROADCASTING

* fyi: you've already seen broadcasting in action!

* >>> np.arange(10) + 5
  array([5,6,7,8,9,10,11,12,13,14])

* here the 5 is "broadcasted" into a length-10 array so that it can be added to np.arange(10)! (neat!)

BROADCASTING

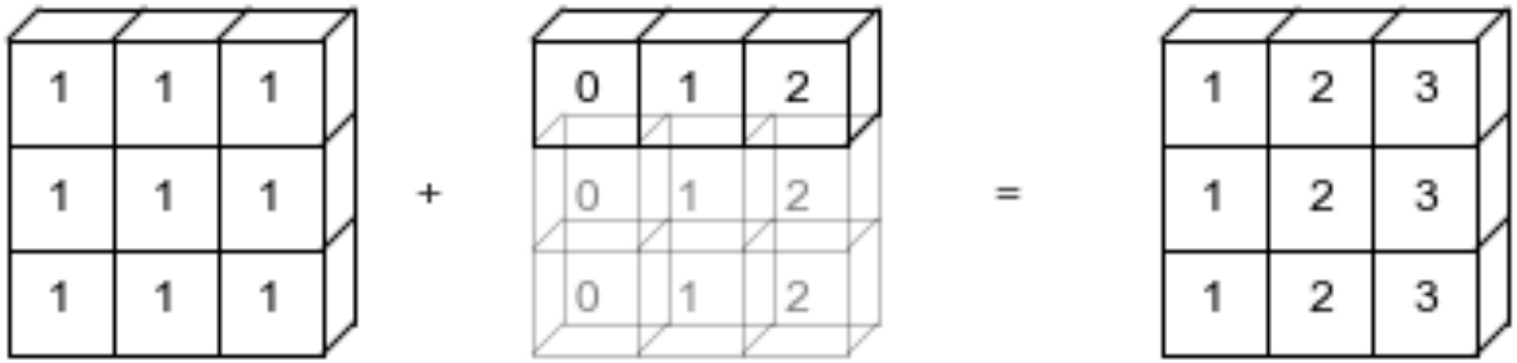
* broadcasting can also do cool things like this:

*
```
>>> a = np.arange(3)
>>> a.shape
(3,) # this is the same as (1,3)
>>> b = np.arange(3)[:,np.newaxis]
>>> b.shape
(3, 1)
>>> a + b
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

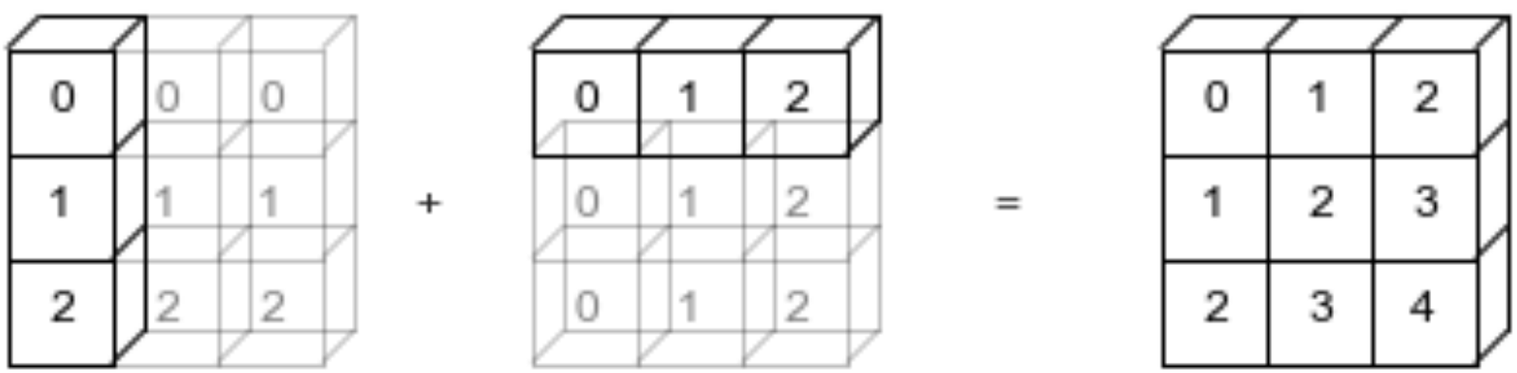
BROADCASTING

$\texttt{np.arange}(3)+5$

$\texttt{np.ones}((3,3))+\texttt{np.arange}(3)$



$\texttt{np.arange}(3).\texttt{reshape}((3,1))+\texttt{np.arange}(3)$



from: https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html

RULE 1: PAD

* Rule 1 of broadcasting:
  * if the two arrays have different numbers of dimensions, the one with fewer dimensions is **padded** with new dimensions (on the left side!)

* e.g. in a+b, if a.shape = (7,3,5) and b.shape = (5,), b gets padded to (1,1,5)

RULE 2: STRETCH

* Rule 2 of broadcasting (applied after rule 1!):
  * if, for some dimension, the shapes don’t match, then the array with shape 1 gets **stretched** to have the same shape as the other array

* e.g. with a.shape = (3,5), b.shape = (1,5), b gets stretched to (3,5)

RULE 3: FAIL

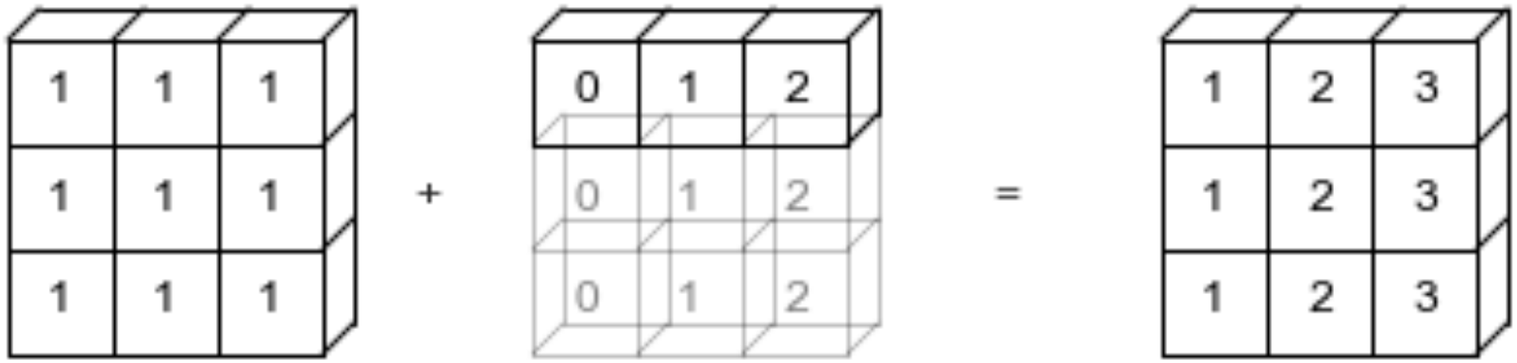
* Rule 3 of broadcasting (applied pretty much at the same time as rule 2):
  * if, for some dimension, the shapes don’t match, **and neither shape is 1**, then an error is raised

  * e.g. if a.shape = (7,3,5) and b.shape = (1,2,5), then 2≠3 so an error is raised

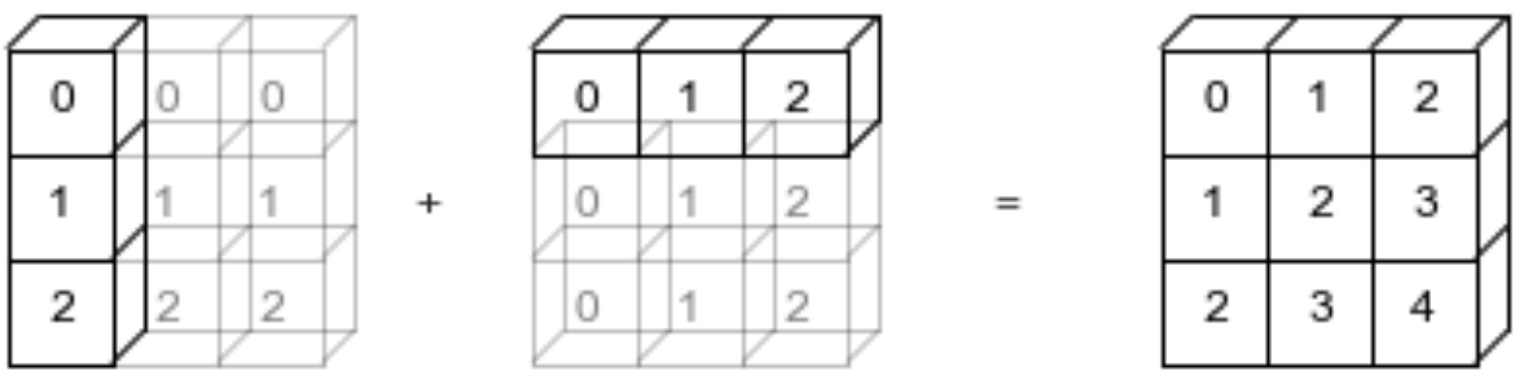
BROADCASTING

$\texttt{np.arange}(3)+5$

$\texttt{np.ones}((3,3))+\texttt{np.arange}(3)$



$\texttt{np.arange}(3).\texttt{reshape}((3,1))+\texttt{np.arange}(3)$



from: https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html

END