배열 연산: 브로드케스팅¶

앞에서 느린 파이썬 루프를 제거하기 위해 연산을 벡터화하는 NumPy 의 유니버설 함수 사용법을 알아봤다. 벡터화 연산의 또 다른 방법은 NumPy 의 브로드캐스팅 기능을 사용하는 것이다. 브로드캐스팅은 단지 다른 크기의 배열에 이항 유니버설 함수(덧셈, 뺄셈, 곱셈 등)를 적용하기 위한 규칙의 집합일 뿐이다.

브로드캐스팅 소개¶

값은 크기의 배열에서 이항 연산은 배열의 요소 단위로 수행된다는 점을 기억하자.

```
In [1]:import numpy as np
In [2]:a = np.array([0, 1, 2])
b = np.array([5, 5, 5])
a + b
Out[2]:
array([5, 6, 7])
```

브로드캐스팅을 사용하면 이러한 유형의 이항 연산을 서로 다른 크기의 배열에서 수행할 수 있다. 예를 들어 배열에 스칼라(0 차원 배열이라고 생각하면 된다)를 쉽게 더할 수 있다.

```
In [3]:a + 5
Out[3]:array([5, 6, 7])
```

이것은 값 5 를 배열 [5, 5, 5]로 확장하거나 복제하거나 복제하고 그 결과를 더하는 연산으로 생각하면 된다. NumPy 브로드캐스팅의 입점은 이 값 복제가 실제로 발생하지는 않는다는 것이다. 하지만 브로드캐스팅을 이러한 방식으로 생각하면 이해하기 쉽다.

이것을 더 높은 차원의 배열로 확장할 수도 있다. 1 차원 배열을 2 차원 배열에서 더할 때 어떤 결과가 나오는지 살펴보자.

```
In [4]:M = np.ones((3, 3))
        M
Out[4]:array([[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])
In [5]:M + a
Out[5]:array([[ 1.,  2.,  3.],
        [ 1.,  2.,  3.],
        [ 1.,  2.,  3.]])
```

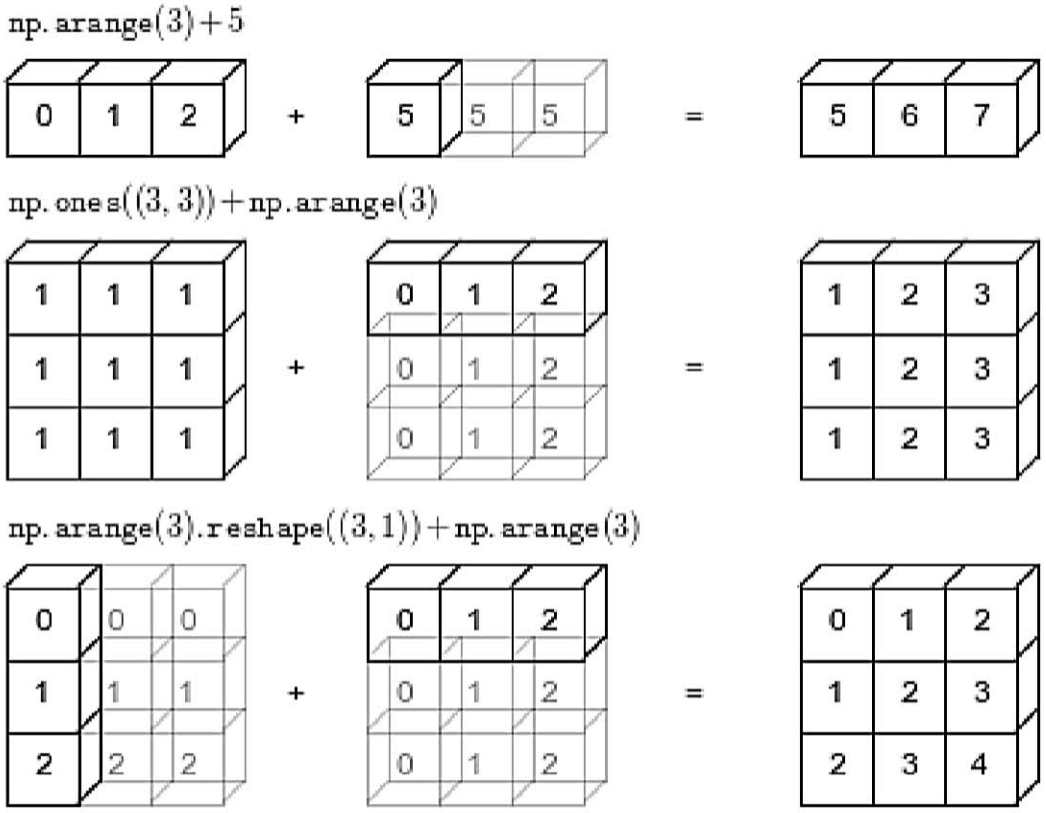
여기서 1 차원 배열 a 는 M 의 형상에 맞추기 위해 두번째 차원까지 확장 또는 브로드캐스팅된다.

이 예제들은 비교적 이해하기 쉽지만. 두 배열 모두 브로드캐스팅해야 하는 경우에는 더 복잡해진다. 다음 예제를 생각해보자.

```
In [6]:a = np.arange(3)
        b = np.arange(3)[:, np.newaxis]
        print(a)
        print(b)
[0 1 2]
[[0]
 [1]
 [2]]
```

```
In [7]:a + b
Out[7]:array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

이전 예제와 마찬가지로 하나의 값을 늘리거나 브로드캐스팅해서 다른 형상에 일치시켰다. 이 예제 에서는 공통 형상에 맞기 위해 a 와 b 모두 확장했고, 그결과 2 차원 배열을 얻었다.



투명하게 보이는 박스는 브로드캐스팅된 값을 나타낸다. 다시 말하자면 이 부가적 메모리는 그 연산과정에서 실제로 할당되지 않지만 개념적으로 이렇게 상상하는 것이 이해하는데 도움이 된다..

브로드캐스팅 규칙¶

NumPy 의 브로드케스팅은 두 배열 사이의 상호작용을 결정하기 위해 엄격한 규칙을 따른다.

- 규칙 1: 두 배열의 차원 수가 다르면 더 작은 수의 차원을 가진 배열 형상의 앞쪽(왼쪽)을 1 채운다.
- 규칙 2: 두 배열의 형상이 어떤 차원에서도 일치하지 않는다면 해당 차원의 형상이 1 인 배열이 다른 형상과 일치하도록 늘어난다..
- 규칙 3: 임의의 차원에서 크기가 일치하지 않고 1 도 아니라면 오류가 발생한다..

브로드캐스팅 예제 1¶

1 차원 배열에 2 차원 배열을 더하는 것을 보자:

```
In [8]:M = np.ones((2, 3))
        a = np.arange(3)
```

이 두 배열간의 연산을 생각해보자.

- M.shape = (2, 3)
- a.shape = (3,)

규칙 1 에 따라 배열 a 가 더 작은 차원을 가지므로 왼쪽을 1 로 채운다.:
- M.shape -> (2, 3)
- a.shape -> (1, 3)

규칙 2 에 따른 첫번째 차원이 일치하지 않으므로 이 차원이 일치하도록 늘린다.
- M.shape -> (2, 3)
- a.shape -> (2, 3)

모양이 일치하면 최종 형상이 (2, 3)이 된다는 것을 알 수 있다.

```
In [9]:M + a
Out[9]:array([[ 1.,  2.,  3.],
              [ 1.,  2.,  3.]])
```

브로드캐스팅 예제 2¶

이번에는 두 배열 모두 브로드캐스팅이 필요한 예제를 보자

```
In [10]:a = np.arange(3).reshape((3, 1))
         b = np.arange(3)
```

다시 배열의 형상부터 확인하자.
- a.shape = (3, 1)
- b.shape = (3,)

규칙 1 에 따라 b 의 형상에 1 을 덧붙여야 한다.
- a.shape -> (3, 1)
- b.shape -> (1, 3)

그리고 규칙 2 에 따라 각 차원을 그에 대응하는 다른 배열의 크기에 일치하도록 늘린다.
- a.shape -> (3, 3)
- b.shape -> (3, 3)

결과가 일치하기 때문에 이 형상들은 다음과 같이 서로 호환된다.

```
In [11]:a + b
Out[11]:array([[0, 1, 2],
               [1, 2, 3],
               [2, 3, 4]])
```

브로드캐스팅 예제 3¶

두 개의 배열이 호환되지 않는 경우의 예제를 살펴보자:

```
In [12]:M = np.ones((3, 2))
a = np.arange(3)
```

이 예제는 첫번째 예제와 약간 다른 경우다. 행렬 M 의 형상이 뒤바뀌었다. 이것이 계산에는 어떤 영향을 끼칠까? 배열의 형상은 다음과 같다.
- M.shape = (3, 2)
- a.shape = (3,)

다시, 규칙 1 에 따라 a 의 형상에 1 을 채운다.
- M.shape -> (3, 2)
- a.shape -> (1, 3)

규칙 2 에 따라 a 의 첫번째 차원을 M 의 첫번째 차원과 일치하도록 늘린다.
- M.shape -> (3, 2)
- a.shape -> (3, 3)

이제 규칙 3 에서 최종 형상이 서로 일치하지 않으므로 이 두 배열은 호환되지 않는다. 다음 연산을 시도해 보면 그 사실을 확인 할 수 있다.

```
In [13]:M + a
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-9e16e9f98da6> in <module>()
----> 1 M + a

ValueError: operands could not be broadcast together with shapes (3,2) (3,)
```

여기서 a 의 형상에 왼쪽이 아니라 오른쪽에 1 을 덧붙이면 a 와 M 이 서로 호환 될 수 있지 않을까하는 생각에 혼란스러울 수도 있다. 그러나 브로드캐스팅은 그런 방식으로 동작하지 않는다. 그런 종류의 유연성이 유용한 경우도 있겠지만 모호함이 생길 여지가 있다. 오른쪽 덧붙이기를 원한다면 면시적으로 배열의 형상을 변경하면 된다.

```
In [14]: a[:, np.newaxis].shape
Out[14]:(3, 1)
In [15]:M + a[:, np.newaxis]
Out[15]:
array([[ 1.,  1.],
       [ 2.,  2.],
       [ 3.,  3.]])
```

이 예제에서는 + 연산자를 중심적으로 사용했지만, 이 브로드캐스팅 규칙은 모든 이항 ufunc 에 적용된다. 예를 들면, 다음과 같이 log(exp(a) + exp(b))를 기본 방식보다 더 정확하게 계산하는 logaddexp(a, b) 함수가 있다.

```
In [16]:np.logaddexp(M, a[:, np.newaxis])
Out[16]:array([[ 1.31326169,  1.31326169],
               [ 1.69314718,  1.69314718],
               [ 2.31326169,  2.31326169]])
```

실전 브로드캐스팅¶

배열을 중앙 정렬하기¶

앞에서 ufunc 을 사용하려면 NumPy 사용자가 느린 파이썬 루프를 명시적으로 작성하지 않아도 된다는 사실을 알아야 한다. 브로드캐스팅은 이능력을 확장한다. 흔히 볼수 있는 예는 데이터 배열을 중앙 정렬하는 것이다, 10 개의 관측치로 이뤄진 배열이 있고 각 관측치는 3 개의 값으로 구성된다고 생각해 보자.

```
In [17]:X = np.random.random((10, 3))
```

Mean 집계 함수를 사용해 첫번째 차원의 특성값을 계산할 수 있다.

```
In [18]:Xmean = X.mean(0)
         Xmean

Out[18]: array([ 0.53514715,  0.66567217,  0.44385899])
```

이제 평균값을 뺌으로써 X 배열을 중앙 정렬할 수 있다 .(이것이 브로드캐스팅 연산이다.):

```
In [19]:X_centered = X - Xmean
```

제대로 됐는지 확인하려면 중앙 정렬된 배열의 평균이 거의 0 에 가까운지 확인하면 된다.

```
In [20]:X_centered.mean(0)
Out[20]:array([  2.22044605e-17,  -7.77156117e-17,  -1.66533454e-17])
```

기계 정밀도 내에서 평균 값이 0 이다..

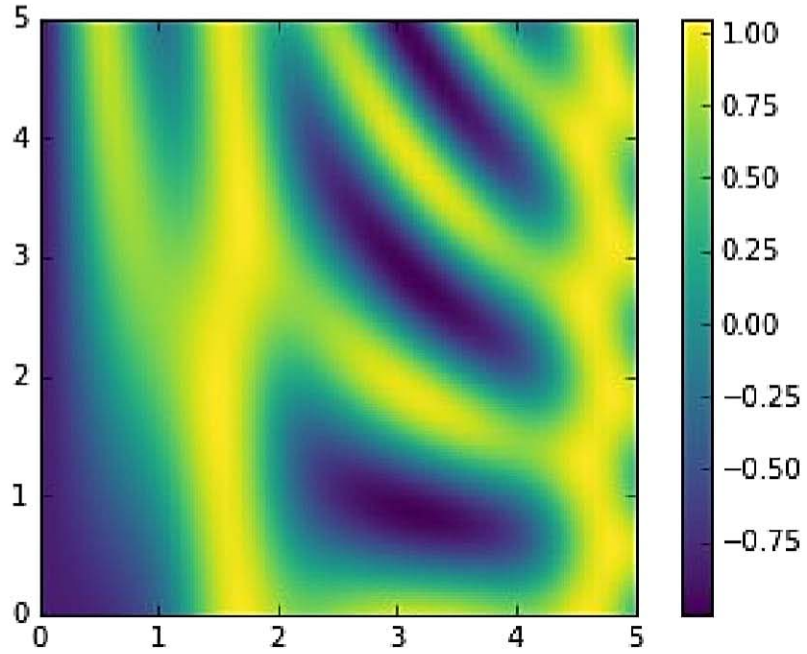
2 차원 함수 플로팅¶

브로드캐스팅은 2 차원 함수를 기반으로 이미지를 그릴 때도 매우 유용하다. 함수를 정의하고 싶다면 브로드캐스팅을 사용해 그리드에 이함수를 계산할 수 있다.

```
In [21]:# x 와 y 는 0 에서 5 까지 50 단계로 나눈 배열임
         x = np.linspace(0, 5, 50)
         y = np.linspace(0, 5, 50)[:, np.newaxis]
z = np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

2 차원 배열을 플로팅하는데 Matplotlib 을 사용할 것이다.

```
In [22]:%matplotlib inline
         import matplotlib.pyplot as plt
In [23]:plt.imshow(z, origin='lower', extent=[0, 5, 0, 5],cmap='viridis')
         plt.colorbar();
```



2 차원 함수를 강렬한 색상으로 시각화해서 보여주 ㄴ다.