Machine Learning in Practice
#1-3: numpy Library

Sang-Hyun Yoon

Summer 2019

Outline

numpy Arrays

- numpy array는 다차원 배열로 TensorFlow의 Tensor type과 호환되어 머신러닝 프로그램에 많이 사용
  - numpy array 형태의 데이터를 TensorFlow에 바로 입력으로
  - list보다 더 많이 사용됨
- Python list와 달리 ragged array를 허용하지 않음

```
import numpy as np

a = np.array([1,2,3])     # rank(차원)가 1인 array 생성
print(type(a))            # <type 'numpy.ndarray'>
print(a.shape)            # (3,)
print(a[0],a[1],a[2])     # 1 2 3
a[0] = 5
print(a)                  # [5,2,3]

b = np.array([[1,2,3],[4,5,6]])   # rank가 2인 배열 생성
print(b.shape)                    # (2, 3)
print(b[0,0],b[0,1], b[1,0])      # 1 2 4
```

Creating Arrays with Initialization

생성하려는 array의 shape를 지정(2,3)과 같이 tuple로 지정

```
a = np.zeros((2,3))    # [[0. 0. 0.]
print(a)               #  [0. 0. 0.]]

b = np.ones((1,3))     # [[1. 1. 1.]]
c = np.ones((3,))      #  [1. 1. 1.]
d = np.ones(3)         #  [1. 1. 1.]

e = np.full((2,3), 7) # [[7. 7. 7.]
                       #  [7. 7. 7.]]

f = np.eye(2)          # [[1. 0.]   2x2 identity matrix
                       #  [0. 1.]]

g = np.random.random((2,2)) # [[0.91940167  0.08143941]
                            #  [0.41231213  0.21231230]]
                            #    2x2 random matrix

h = np.arange(10,5,-1) # [10 9 8 7 6]
```

Indexing

다음 3가지 방식 모두 가능

- Python list와 마찬가지로 [·][·][·] 형태
- Tuple [(·, ·, ·)] 형태
- Comma로만 나열한 [·, ·, ·] 형태

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

print(a[1][2])   # 7
print(a[1,2])    # 7
print(a[(1,2)])  # 7

print(a[1])      # [5 6 7 8]
```

Slicing (1/2)

- List에서는 첫번째 차원에 대해서만 slicing 가능했었는데
- numpy array에서는 한번에 여러 차원 slicing 가능
- Slicing을 indexing과 섞어서 써도 됨

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

b = a[:2, 1:3]  # [[2 3]
                #  [6 7]]

c = a[1:2, :]   # [[5 6 7 8]]
d = a[1, :]     # [5 6 7 8]

e = a[:, 1:2]   # [[ 2],
                #  [ 6]
                   [10]]
f = a[:, 1]     # [2 6 10]    (column을 추출하는 효과)
```

Slicing (2/2)

- List slicing은 새로운 sublist를 생성하는데 (deep copy)
- Array slicing은 sublist의 alias므로 주의
  - Sliced array를 copy.deepcopy(·) 해주면 새로운 array

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

b = a[:2, 1:3]  # [[2 3]
                #  [6 7]]
b[0][0] = 0     # a[0][1] changed from 2 to 0
print(a)        # [[ 1  0  3  4]
                #  [ 5  6  7  8]
                #  [ 9 10 11 12]]

c = copy.deepcopy(a[:2, 1:3])  # [[0 3]
                               #  [6 7]]
c[0][0] = 100   # a[0][1] unchanged
```

Indexing with Integer List/Array

- Index로서 사용될 list/array는 1차원 정수 list/array
- Array slicing에서와 달리 deep copy된 새로운 array

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]
#  [13 14 15 16]]

b = a[[1,0,3,0]]  # [[ 5  6  7  8]
                  #  [ 1  2  3  4]
                  #  [ 9 10 11 12]
                  #  [ 1  2  3  4]

b[0][0] = 100     # a unchanged
```

Indexing with Boolean List/Array

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

idx = (a > 7)     # [[False False False False]
                  #  [False False False  True]
                  #  [ True  True  True  True]]

print(a[idx])     # [8 9 10 11 12]

print(a[a > 7])   # [8 9 10 11 12]
```

Datatype

Array의 각 원소들은 동일한 type을 가져야만 함

```
x = np.array([1, 2])
print(x.dtype)   # int64
print(type(x))   # <class 'numpy.ndarray'>

x = np.array([1.0, 2.0])
print(x.dtype)   # float64

x = np.array([1, 2], dtype=np.int32)   # force datatype
print(x.dtype)   # int32

x = np.array([1, 2], dtype=np.float32)
print(x.dtype)   # float32

x = np.zeros(10)
print(x.dtype)   # float64

x = np.ones(10)
print(x.dtype)   # float64
```

Math Operations/Functions on Arrays

```
x = np.array([[1,2],[3,4]]);  y = np.array([[5,6],[7,8]])

print(x + y)              # [[ 6  8]  (elementwise addition)
print(np.add(x,y))        #  [10 12]]

print(x - y)              # [[-4 -4]
print(np.subtract(x,y))   #  [-4 -4]]

print(x * y)              # [[ 5 12]
print(np.multiply(x,y))   #  [21 32]]

print(x / y)              # [[0.2        0.33333333]
print(np.divide(x,y))     #  [0.42857143 0.5       ]]

print(np.sqrt(x))         # [[1.         1.41421356]
                          #  [1.73205081 2.        ]]

print(x + 10)             # [[11 12]   (broadcast에서 다룸)
                          #  [13 14]]
```

Math Operations/Functions with Dimension Reduction

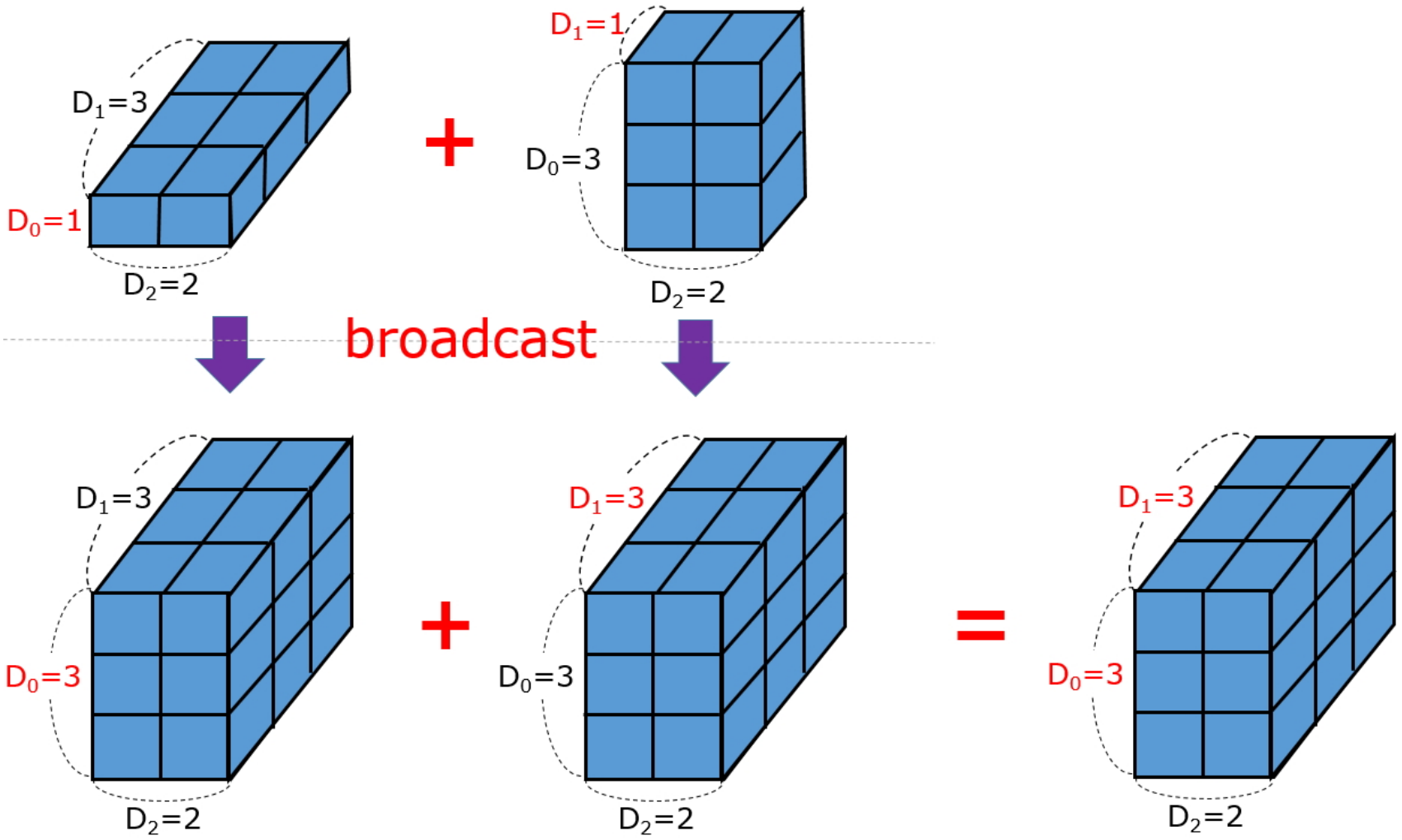
TensorFlow에서 Tensor 연산을 다룰 때 자세히

```
x = np.array([[1,2],
              [3,4]])

np.sum(x)          # 10 (모두 더한 값)
np.sum(x, axis=0)  # [4 6]  (첫번째 차원을 없애면서 더함)
np.sum(x, axis=1)  # [3 7]  (두번째 차원을 없애면서 더함)
```

Broadcasting

차원이 다른 array 간에 연산할 때 복사를 하면서 차원을 자동으로 늘려줌 (TensorFlow 다룰 때 자세히)



Reshape

- reshape 함수를 array의 shape를 바꿀 수 있음
- 파라미터에 -1이 있는 자리는 다른 나머지 차원 크기를 맞추고 남은 크기를 해당 차원에 할당

```
a = np.arange(12)     # [0,1,2,3,4,5,6,7,8,9,10,11,12]

b = a.reshape(3, 4)   # [[ 0, 1, 2, 3],
                      #  [ 4, 5, 6, 7],
                      #  [ 8, 9,10,11]]

a.reshape(2, -1, 2)   # [[[ 0, 1],    (-1 => 3)
                      #     [ 2, 3],
                      #     [ 4, 5]],
                      #    [[ 6, 7],
                      #     [ 8, 9],
                      #     [10,11]]]

b.flatten()           # [0,1,2,3,4,5,6,7,8,9,10,11,12]
```

np.rollaxis(a, axis,start=0)

- axis: The axis to roll backwards. The positions of the other axes do not change relative to one another.
- start: The axis is rolled until it lies before this position.

```
a = np.ones((3, 4, 5, 6))
np.rollaxis(a, 3, 1).shape  # (3, 6, 4, 5)
np.rollaxis(a, 2).shape     # (5, 3, 4, 6)
np.rollaxis(a, 1, 4).shape  # (3, 5, 6, 4)

b = np.arange(8).reshape(2,2,2) # [[[ 0, 1],
                                  [ 2, 3]],
                                 [[ 4, 5],
                                  [ 6, 7]]]
np.rollaxis(b, 2, 1)            # [[[ 0, 2],
                                  [ 1, 3]],
                                 [[ 4, 6],
                                  [ 5, 7]]]
np.rollaxis(b, 2, 0)            # [[[ 0, 2],
                                  [ 4, 6]],
                                 [[ 1, 3],
```

Misc.

```
np.ones(5).dtype                    # float64 (default dtype)
np.ones(5).astype(np.int32).dtype # int32


np.random.randint(2,10)  # random int in [2, 10)
np.random.random()       # random float in [0.0, 1.0)
np.random.random(3)      # [0.157322  0.5030706 0.1886311]


a = np.random.random((3,3)) # [[0.188708 0.642642 0.588565]
                             [0.037587 0.118505 0.260982]
                             [0.294038 0.962710 0.704648]]
a.max()     # 0.962710
a.argmax()  # 7


np.random.normal(5,1)  # samples from a normal distribution
                        # with (mean, std) = (5, 1)
np.random.normal(0.5,0.1, 3)  # [0.45257 0.592248 0.480700]
```