Numpy 특강 part 1

미래연구소 12기 2주차

1. Vectorization

○ Arrays are important because they enable you to express **batch operations** on data **without writing any for loops**. This is usually called vectorization. Any arithmetic operations between equal-size arrays applies the operation elementwise.

- 벡터화하여 계산

1> for문을 사용하면 1 코딩의 양도 늘고 2 속도도 느림

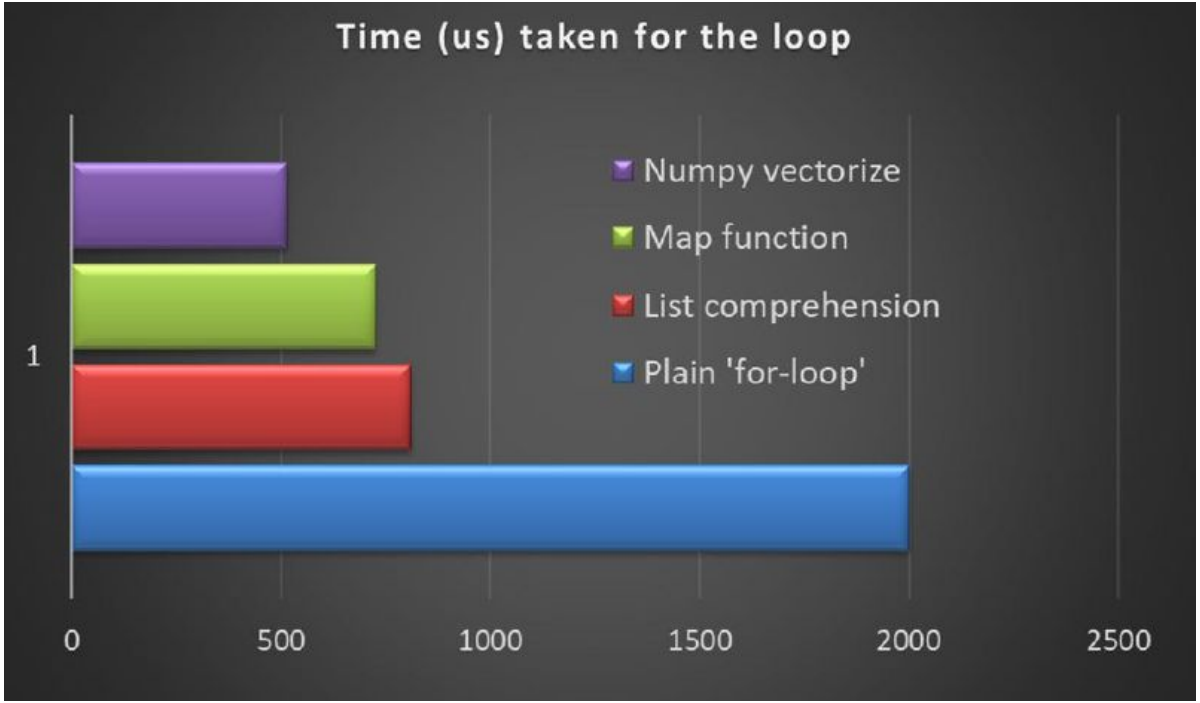
2> for문을 사용할 것을 행렬 연산으로 변환하여 한 번에 여러 개의 연산을 진행

-> 속도 향상 (연산 시간 단축)

3> (TMI) 컴퓨터 CPU에는 클럭이 있어서 한 클럭마다 어느 정도 연산량을 감당할 수 있는데, 이렇게 한 클럭 당 하는 연산, 병렬 처리 기능을 좀 더 활용하는 방법이라고 할 수 있습니다.

4> 그럼 vector? matrix? tensor?

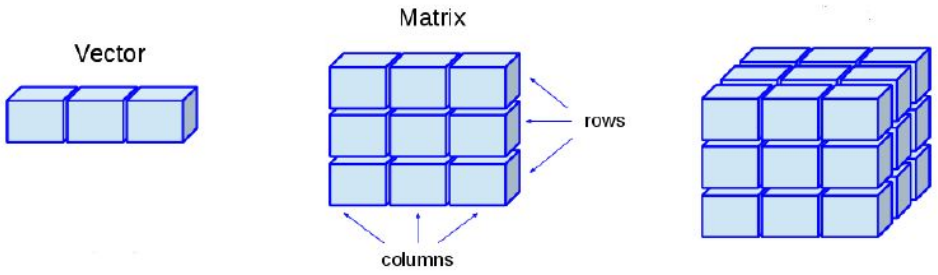
1. Vectorization



출처: https://www.codementor.io/tirthajyotisarkar/data-science-with-python-turn-your-conditional-loops-to-numpy-vectors-he1yo9265

2. Vector? Matrix? Array? Tensor?

	수학	Numpy	Tensorflow (TMI)	Pandas (TMI)
1차원	(scalar) Vector	1d array	1d tensor	Series
2차원	Matrix	2d array	2d tensor	DataFrame
3차원		3d array (nd array)	3d tensor (nd tensor)	Panel



3. Numpy

1) 정의: 벡터, 행렬 연산을 위한 수치해석용 python 라이브러리

2) 왜 사용하는가?

1> 빠르다. 2> 편하다. 3> vectorization을 위해 행렬을 다뤄야 한다.

3) 기타 특징

1> data science 라이브러리의 90%가 내부 구조 numpy로 되어있다.

2> 특히, tensorflow와 아주 유사함 -> numpy 제대로 배우면 tensorflow 정말 쉽게 배울 수 있음

3. Numpy

행렬을 다루는 라이브러리

(2D보다 저차원 혹은 고차원도 다룸)

1) 정의: 벡터, 행렬 연산을 위한 수치해석용 python 라이브러리

2) 왜 사용하는가?

1> 빠르다. 2> 편하다. 3> vectorization을 위해 행렬을 다뤄야 한다.

3) 기타 특징

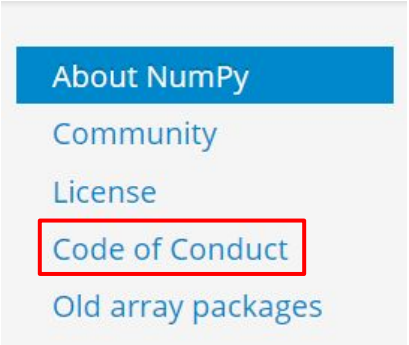
1> data science 라이브러리의 90%가 내부 구조 numpy로 되어있다.

2> 특히, tensorflow와 아주 유사함 -> numpy 제대로 배우면 tensorflow 정말 쉽게 배울 수 있음

4. Numpy 등 라이브러리 학습에서 명심할 것

1) 다 외울 수 없습니다.

2) 잘 찾는 능력이 중요합니다.



1> 함수를 처음 봤을 때: shift + tab == np.info('궁금한 함수').

2> 함수 일부 기억 날 듯 말 듯: np.lookfor('궁금한 함수 일부분')

3> 둘 다 하고 싶을 때: numpy 공식 사이트 -> 'code of conduct' 들어가서 검색

4> 정 안 되면 구글링

3) 처음 보는 함수를 능숙하게 쓰는 능력도 중요합니다.

1> 찾아서 읽어보고 스스로 사용해보는 습관 (처음이 힘들지 직접 해보면 금방 실력 향상됩니다.)

2> Class, Module, Package등 관련 문법을 잘 모르는 분은 초반에 어려울 수 있습니다. -> 튜터에게 질문

5. 11강에 나오는 numpy 함수 array

1) np.array: N-D array를 만드는 함수

```
[1]: import numpy as np
```

1> 1-D array 만들기

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

array([1, 2, 3])
```

2> 2-D array 만들기

```
np.array([[0,1],[2,3],[4,5]])

array([[0, 1],
       [2, 3],
       [4, 5]])
```

3> 3-D array 만들기

```
np.array([[[0,1],[2,3],[4,5]],[[0,-1],[-2,-3],[-4,-5]]])

array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 0, -1],
        [-2, -3],
        [-4, -5]]])
```

numpy.array

numpy.array(*object, dtype=None, copy=True, order='K', subok=False, ndmin=0*)

Create an array.

Parameters:

object : *array_like*

An array, any object exposing the array interface, an object whose __array__ method returns an array, or any (nested) sequence.

출처: https://numpy.org/devdocs/reference/generated/numpy.array.html?highlight=array#numpy.array

5. 11강에 나오는 numpy 함수 dot

numpy.dot

numpy.dot(*a*, *b*, *out=None*)

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using **matmul** or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to **multiply** and using `numpy.multiply(a, b)` or `a * b` is preferred.

	a	b	dot 연산	비고
1	1-D array	1-D array	inner product (내적)	
2	2-D array	2-D array	matrix multiply	대체: np.matmul(a,b) == a@b
3	0-D array	0-D array	곱셈	대체: np.multply(a,b) == a * b
4				
5				

```
a = np.array([1,2,3])
b = np.array([-1,0,2])
print(a)
print(b)
print(np.dot(a,b))
```

```
[1 2 3]
[-1  0  2]
5
```

4, 5번 case가 있으나 거의 쓰이지 않음

참고:
https://numpy.org/devdocs/reference/generated/numpy.dot.html?highlight=dot#numpy.dot

TMI 몰라도 됩니다.

5. 11강에 나오는 numpy 함수 random.Randomstate.rand

1) np.random.RandomState:

원하는 **shape**과 원하는 **distribution**에 맞는 **random array** 생성해주는 **class**

class numpy.random.RandomState(*seed=None*)

Container for the slow Mersenne Twister pseudo-random number generator. Consider using a different BitGenerator with the Generator container instead.

RandomState and Generator expose a number of methods for generating random numbers drawn from a variety of probability distributions. In addition to the distribution-specific arguments, each method takes a keyword argument *size* that defaults to `None`. If *size* is `None`, then a single value is generated and returned. If *size* is an integer, then a 1-D array filled with generated values is returned. If *size* is a tuple, then an array with that shape is filled and returned.

출처: https://numpy.org/devdocs/reference/random/legacy.html?highlight=random%20randomstate#numpy.random.RandomState

5. 11강에 나오는 numpy 함수 random.Randomstate.rand

2) np.random.RandomState.rand:

0이상 **1**미만의 구간에서 **uniform distribution**를 가지는 수 중 **random**하게 산출

=> 그냥 [0, 1) 구간에서 원하는 shape에 해당하는 N-D array를 random 산출했다고 생각하면 됩니다.

numpy.random.RandomState.rand

method

RandomState.rand(*d0*, *d1*, ..., *dn*)

Random values in a given shape.

Note:
This is a convenience function for users porting code from Matlab, and wraps **numpy.random.random_sample**. That function takes a tuple to specify the size of the output, which is consistent with other NumPy functions like numpy.zeros and numpy.ones.

Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).

Parameters:

d0, d1, ..., dn : *int, optional*

The dimensions of the returned array, must be non-negative. If no argument is given a single Python float is returned.

Returns:

out : *ndarray, shape (d0, d1, ..., dn)*

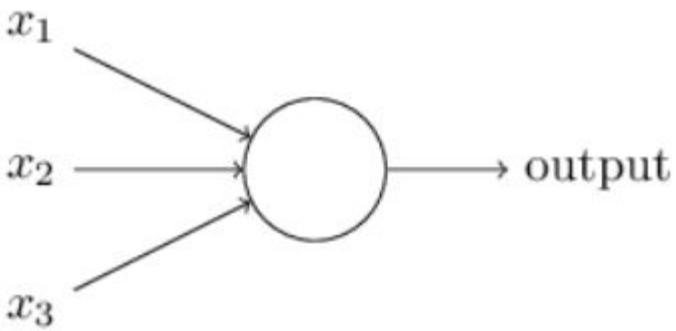
Random values.

```
np.random.rand(3,2)
```

```
array([[0.3442924 , 0.7994408 ],
       [0.31805549, 0.62630638],
       [0.29475843, 0.31488426]])
```

출처:
https://numpy.org/devdocs/reference/random/generated/numpy.random.RandomState.rand.html?highlight=rand#numpy.random.RandomState.rand

Course 1 Week 2의 전반적인 내용



1) parameter (w, b) intialization

2) forward propagation (wx+b = z -> δ(z) = y^)

3) cost function
(Binary_crossentropyloss(y, y^) = J(w,b))

4) backward propagation (dw, db)

5) gradient descent (w=w-αdw, b= b-αdb)