이건명

**딥러닝 프레임워크 : PyTorch**

충북대학교 대학원 산업인공지능학과

**학습 내용**

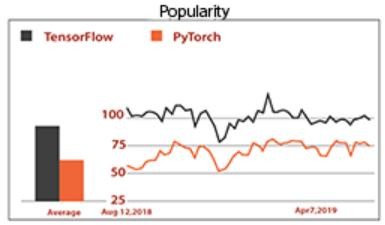
* PyTorch 기본 이론에 대해서 알아본다.
* PyTorch를 이용한 딥러닝 프로그래밍 사례에 대해서 살펴본다.

기계학습, 이건명

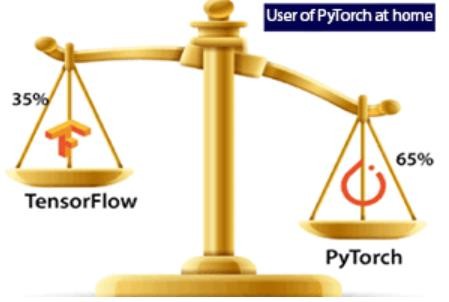
- 2 -

# PyTorch

### PyTorch

* + - Python으로 기존 딥러닝 프레임워크 Torch를 구현한 것

|  |  |  |  |
| --- | --- | --- | --- |
|  | Comparison Factors | Pass | Fail |
| **1.** | Features | TensorFlow | PyTorch |
| **2.** | Community | TensorFlow | PyTorch |
| **3.** | Level of API | TensorFlow | PyTorch |
| **4.** | Speed | PyTorch | TensorFlow |
| **5.** | Popularity | TensorFlow | PyTorch |
| **6.** | Ramp-Up Time | PyTorch | TensorFlow |
| **7.** | Coverage | TensorFlow | PyTorch |
| **8.** | Deployment | TensorFlow | PyTorch |
| **9.** | Serialization | TensorFlow | PyTorch |
| **10.** | Graph constructing and  Debugging | PyTorch | TensorFlow |
| **11.** | Visualization | TensorFlow | PyTorch |
| **12.** | Architecture | PyTorch | TensorFlow |
| **13.** | Dataset | TensorFlow | PyTorch |
| **14.** | Documentation | PyTorch, TensorFlow |  |
| **15.** | Device Management | TensorFlow | PyTorch |
| **16.** | Custom Extension | PyTorch | TensorFlow |

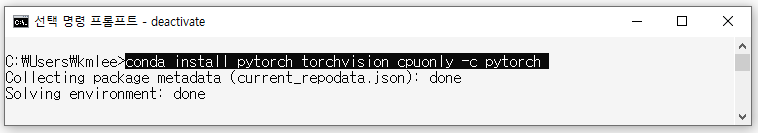
기계학습, 이건명

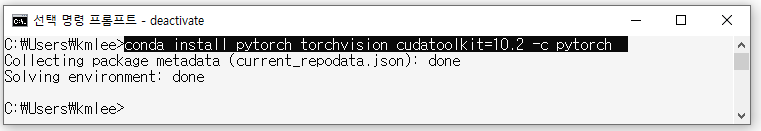
<https://www.javatpoint.com/pytorch-vs-tensorflow>

# PyTorch 설치

### PyTorch 설치

* + - <https://pytorch.org/get-started/locally/>





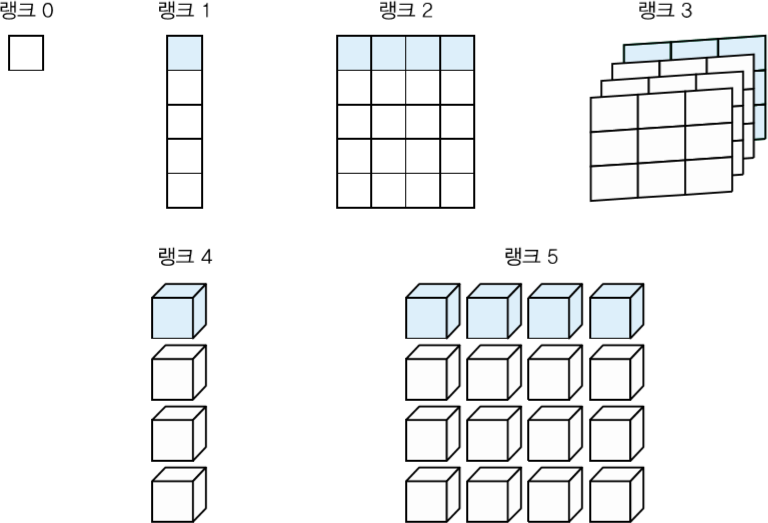
기계학습, 이건명

# PyTorch 기초

### 텐서(Tensor)

* + - 벡터와 행렬의 개념을 확장한 것
    - 다차원 배열(multidimensional array)

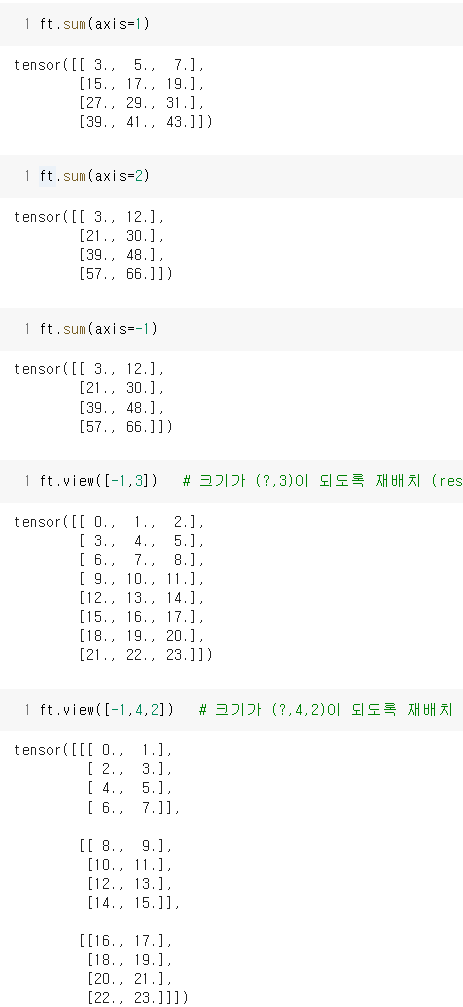
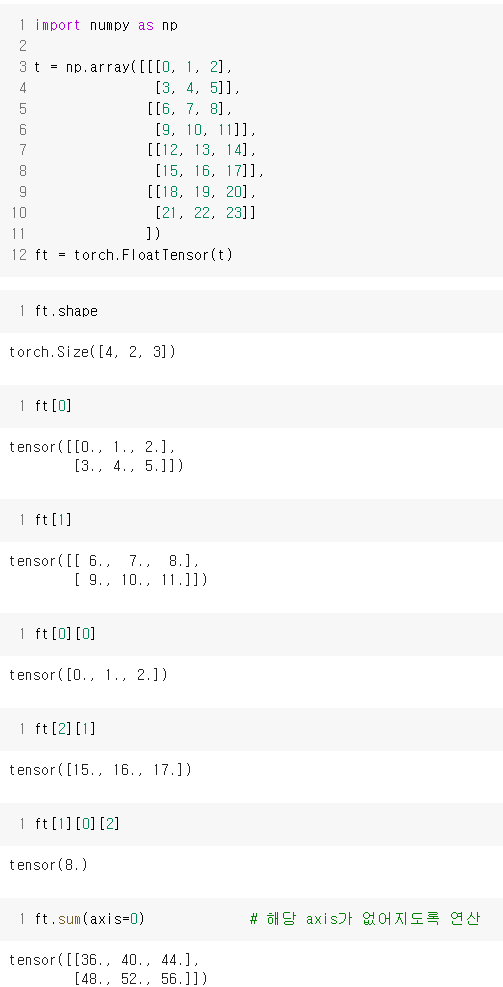
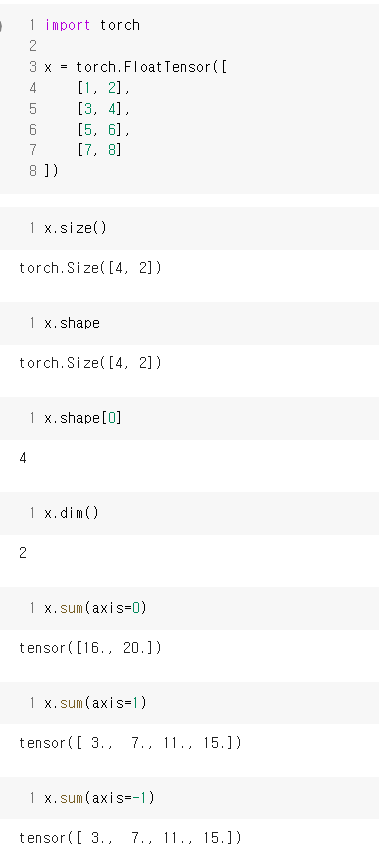
rank (dimension)



스칼라

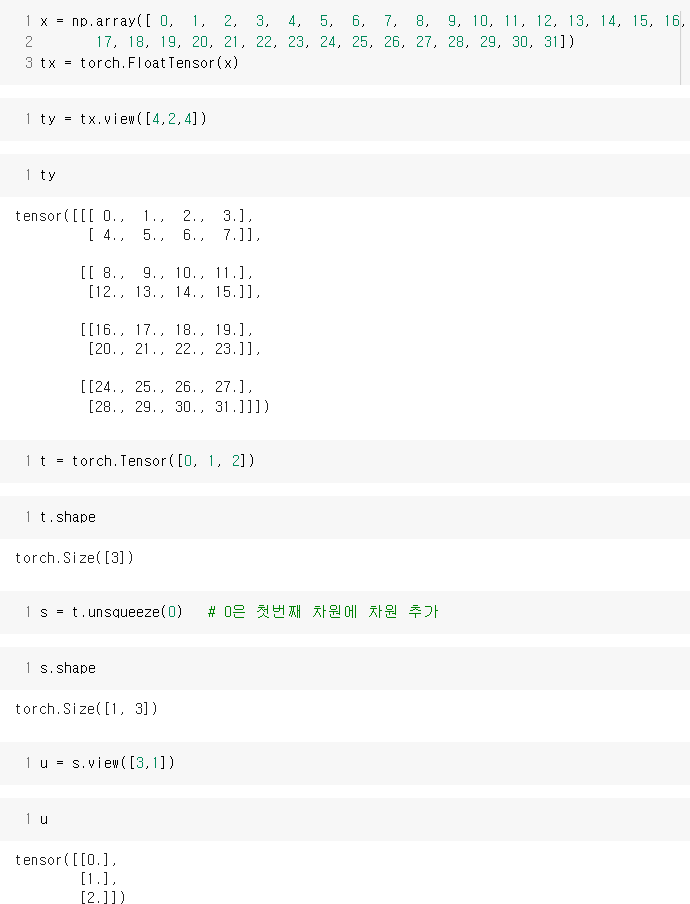
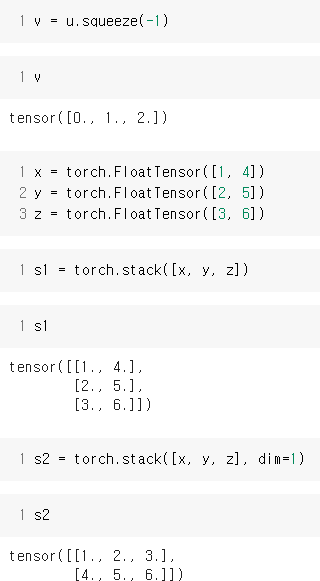
배열

행렬



* + **텐서의 shape**

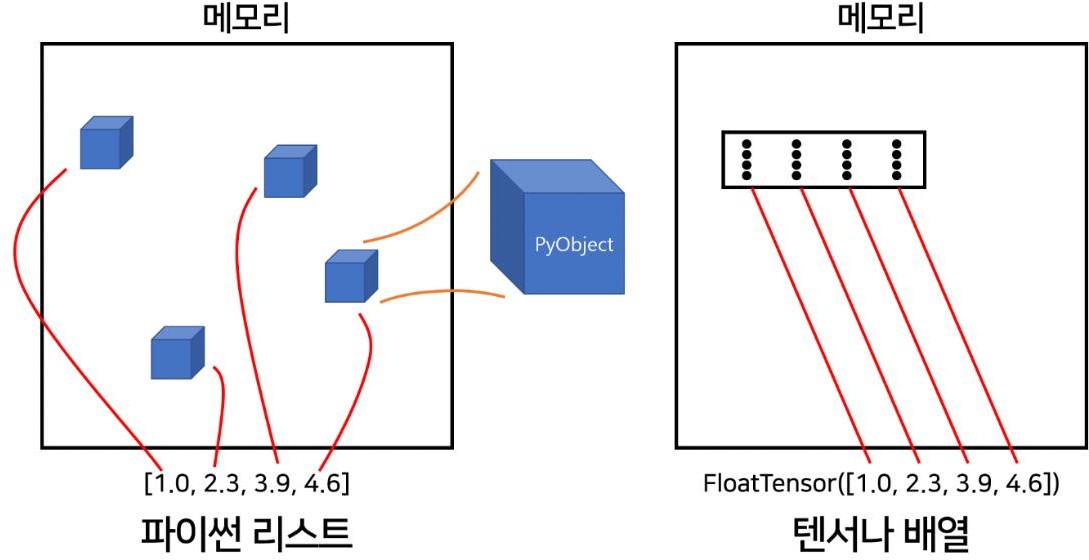
# PyTorch 기초

# PyTorch 기초

### Tensor

* + - NumPy의 (n차원) 배열 ndarray 객체와 유사하지만 GPU⑨ HW가속기에서 실행 할 수 있는 자료구조
    - Python list는 object 포인터들을 모아놓은 집합
      * 메모리 할당에 있어 비효율적이고, 느림
    - numpy array와 tensor는 메모리상 인접한 곳에 공간이 할당



https://hiddenbeginner.github.io/deeplearning/2020/01/21/pytorch\_tensor.html)

# PyTorch 기초

### Numpy

* + - Array 및 Tensor들과 관련된 연산을 편리하게 하기 위한 라이브러리
    - np.array(object, dtype=None)
      * numpy ndarray (numpy array)를 만드는 함수
        + object : 값을 가져올 python array
        + dtype : 각 원소의 자료형(data type) (int, float, bool ⑨) 입력하지 않을 시 objec에 맞춰서 자동으로 결정

a = np.array([1, 2, 3, 4, 5])

b = np.array([[1, 2], [3, 4]], dtype=float)

c = np.array([1, 0, 0, 1], dtype=bool)

[1 2 3 4 5]

[[1. 2.]

[3. 4.]]

[ True False False True]

# PyTorch 기초

### shape

* + - Numpy array 또는 tensor의 모양(shape)을 나타내는 속성(property)

**a = np.array([1, 2, 3, 4, 5])**

**b = np.array([[1, 2], [3, 4]], dtype=float)**

**c = np.array([1, 0, 0, 1], dtype=bool)**

print(a.shape) print(b.shape) print(c.shape)

(5,)

(2, 2)

(4,)

x = torch.FloatTensor([

[1, 2],[3, 4],[5, 6],[7, 8]

])

print(x.shape) print(x.shape[0]

torch.Size([4, 2])

4

# PyTorch 기초

### 주어진 형태(shape)의 numpy array 생성 함수

* + - np.zeros(shape, dtype=None)
    - np.ones(shape, dtype=None)
    - np.full(shape, fill\_value, dtype=None)

zeros = np.zeros([3, 3]) ones = np.ones([3, 3])

twos = np.full([3, 3], 2)

print(zeros, zeros.shape) print(ones, ones.shape) print(twos, twos.shape)

**[[0. 0. 0.]**

**[0. 0. 0.]**

**[0. 0. 0.]] (3, 3)**

|  |  |  |  |
| --- | --- | --- | --- |
| **[[1.** | **1.** | | **1.]** |
| **[1.** | **1.** | | **1.]** |
| **[1.** | **1.** | | **1.]] (3, 3)** |
| **[[2** | **2** | **2]** | |
| **[2** | **2** | **2]** | |
| **[2** | **2** | **2]] (3, 3)** | |

# PyTorch 기초

### Numpy 배열로부터 tensor 생성 (Numpy Array  tensor)

np\_array = np.array(data)

x\_np = torch.**from\_numpy**(np\_array) print(x\_np)

tensor([[1, 2],

[3, 4]])

### tensor를 Numpy 배열로 변환하기 (tensor  Numpy Array)

t = torch.ones(5) n = t.**numpy**() print(n)

[1. 1. 1. 1. 1.]

# PyTorch 기초

|  |  |  |
| --- | --- | --- |
| shape = (2,3,) |  | |
| rand\_tensor = torch.**rand**(shape) | tensor([[0.8398, | 0.8787, 0.4099], |
| ones\_tensor = torch.**ones**(shape) | [0.6517, | 0.2316, 0.1294]]) |
| zeros\_tensor = torch.**zeros**(shape) | tensor([[1., 1., | 1.], |
|  | [1., 1., | 1.]]) |
| print(rand\_tensor) | tensor([[0., 0., | 0.], |
| print(ones\_tensor) | [0., 0., | 0.]]) |
| print(zeros\_tensor) |  |  |

### 다른 tensor로부터 tensor 생성하기 (tensor  tensor)

x\_ones = torch.**ones\_like**(x\_data) # x\_data의 속성 유지

x\_rand = torch.**rand\_like**(x\_data, dtype=torch.float)

# x\_data의 속성 변경

|  |  |  |  |
| --- | --- | --- | --- |
| print(x\_ones) print(x\_rand) |  |  | |
| tensor([[1, 1],  [1, 1]])  tensor([[0.7839, | 0.3701], |
| [0.8131, | 0.3344]]) |  |  |

# PyTorch 기초

### Array 연산

* + - 기본적으로 같은 크기의 array 간의 사칙연산 및 비교연산 가능
    - 같은 위치의 성분간의 연산(element-wise operations)

|  |  |  |
| --- | --- | --- |
| [[1. | 1. | 1.] |
| [1. | 1. | 1.] |
| [1. | 1. | 1.]] |
| [[0.5 | 0.5 0.5] | |
| [0.5 | 0.5 0.5] | |
| [0.5 | 0.5 0.5]] | |
| [[2. | 2. 2.] | |
| [2. | 2. 2.] | |
| [2. | 2. 2.]] | |

zeros = np.zeros([3, 3]) ones = np.ones([3, 3])

twos = np.full([3, 3], 2)

print(zeros + ones) print(ones / twos) print(zeros + ones \* twos)

print(zeros == [[0, 0, 0], [0, 0, 0],

|  |  |  |  |
| --- | --- | --- | --- |
| [[ | True | True | True] |
| [ | True | True | True] |
| [ | True | True | True]] |
| [[False | | False | False] |
| [False | | False | False] |
| [False | | False | False]] |

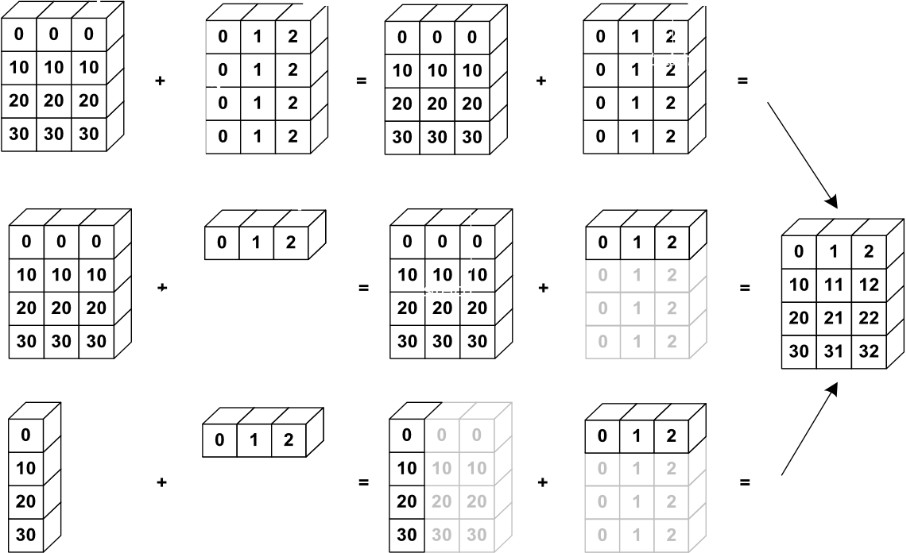
[0, 0, 0]])

print(zeros > ones)

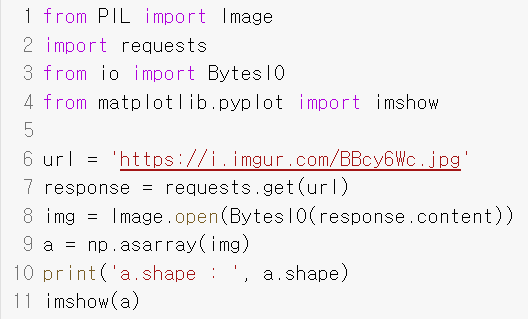
# PyTorch 기초

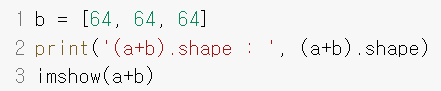
### Broadcasting

* + - shape이 다른 연산 지원을 위한 복사를 통해 shape 일치

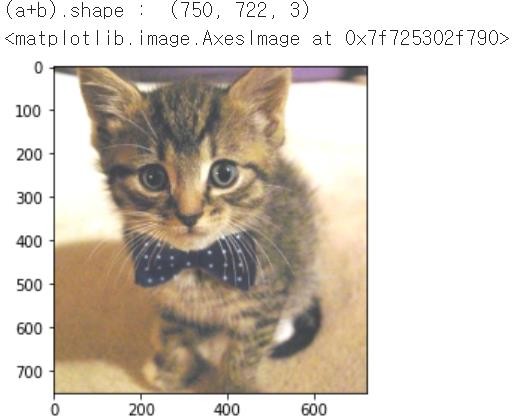


# PyTorch 기초



* + **PIL** (Python Image Manipulation Library,

Python Imaging Library)

# PyTorch 기초

### Array indexing과 slicing

* + - index는 0에서 시작
    - -1 맨 끝 원소
    - n:m n번째 원소부터 m-1번째 원소까지
    - : 해당 차원에 있는 원소 전부

|  |  |  |  |
| --- | --- | --- | --- |
| a = | np.array([[1, | 2, | 3], |
|  | [4, | 5, | 6], |
|  | [7, | 8, | 9]]) |

print(a[0]) print(a[1][-1])

print(a[0:2])

print(a[:, 1])

print(a[1:-1, :])

[1 2 3]

6

[[1 2 3]

[4 5 6]]

[2 5 8]

[[4 5 6]]

# PyTorch 기초

### Tensor indexing과 slicing

tensor = torch.ones(4, 4) print(tensor[**0**]) print(tensor[**:**, 0])

print(tensor[**...**, **-1**])

tensor[:,1] = 0 print(tensor)

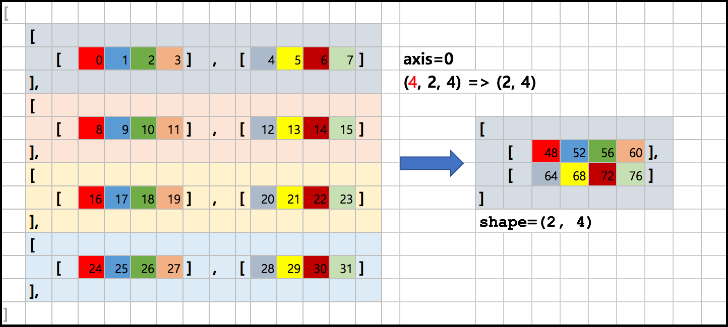
|  |  |  |  |
| --- | --- | --- | --- |
| tensor([1., | 1., | 1., | 1.]) |
| tensor([1., | 1., | 1., | 1.]) |
| tensor([1., | 1., | 1., | 1.]) |
| tensor([[1., | 0., | 1., | 1.], |
| [1., | 0., | 1., | 1.], |
| [1., | 0., | 1., | 1.], |
| [1., | 0., | 1., | 1.]]) |

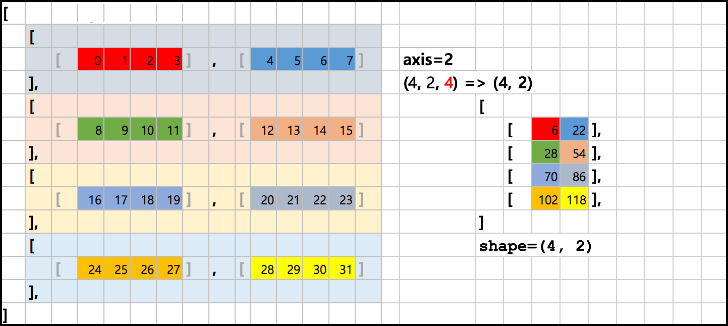
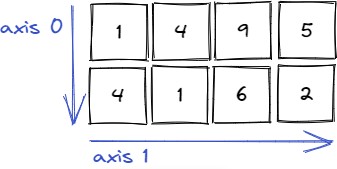
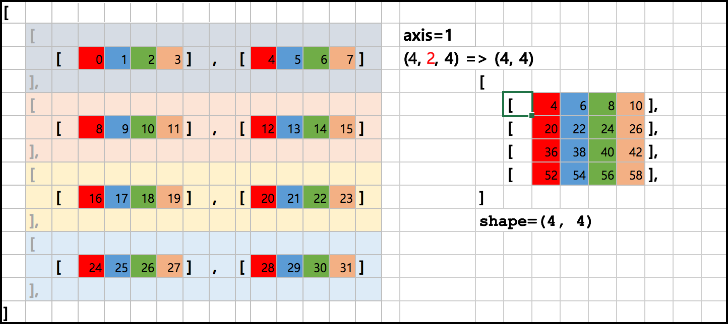
# PyTorch 기초

### np.sum( )/torch.sum( )

* + - 특정 축(axis)을 따라 모든 원소를 더함
    - np.sum(a, axis=None, keepdims=False)
      * a : 대상 array
      * axis : element를 따라 더할 축의 index
      * keepdims : True일 경우 해당 축에 1차원 벡터로 남음. False일 경우 해당 축에 (0차원) element로 남음.

|  |  |  |
| --- | --- | --- |
| a = np.array([[1, 2, 3], [3, 5, 7]]) | 21 |  |
| print(np.sum(a)) | [[ 4 7 | 10]] |
| print(np.sum(a, axis=0, keepdims=True)) | [[ 6] |  |
| print(np.sum(a, axis=1, keepdims=True)) | [15]] |  |
| print(np.sum(a, axis=1, keepdims=False)) | [ 6 15] |  |

np.sum( )/torch.sum( ) 지정된 축(axis)을 따라 모든 원소를 더함



# PyTorch 기초

### np.mean, np.std, np.var

* + - 평균, 표준편차, 분산 계산
    - np.mean(a, axis=None, keepdims=False)
    - np.std(a, axis=None, keepdims=False)
    - np.var(a, axis=None, keepdims=False)

a = np.array([[1, 2, 3],

[3, 5, 7]])

print(np.mean(a)) print(np.std(a, axis=0)) print(np.var(a, axis=1))

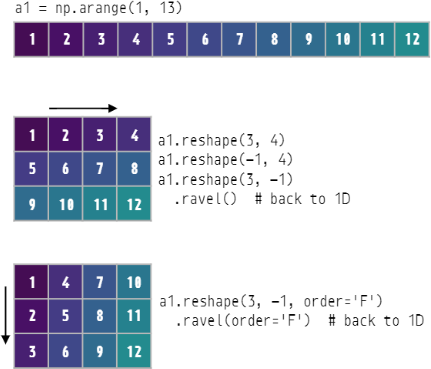
3.5

[1. 1.5 2. ]

[0.66666667 2.66666667]

# PyTorch 기초

### reshape / ravel



https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b

# PyTorch 기초

### torch.view( )

import torch

ft = torch.FloatTensor([[[0, 1, 2],

|  |  |
| --- | --- |
| [3, | 4, 5]], |
| [[6, | 7, 8], |
| [9, | 10, 11]], |

tensor([[[ 0., 1., 2.],

|  |  |  |
| --- | --- | --- |
| [ | 3., | 4., 5.]], |
| [[ | 6., | 7., 8.], |

[ 9., 10., 11.]],

|  |  |  |
| --- | --- | --- |
| [[12, | 13, | 14], |
| [15, | 16, | 17]], |
| [[18, | 19, | 20], |
| [21, | 22, | 23]] |

|  |  |  |
| --- | --- | --- |
| [[12., | 13., | 14.], |
| [15., | 16., | 17.]], |
| [[18., | 19., | 20.], |
| [21., | 22., | 23.]]]) |

])

print(ft) print(ft.view([-1,3]))

tensor([[ 0., 1., 2.],

[ 3., 4., 5.],

|  |  |  |  |
| --- | --- | --- | --- |
| [ | 6., | 7., 8.], | |
| [ | 9., | 10., 11.], | |
| [12., | | 13., | 14.], |
| [15., | | 16., | 17.], |
| [18., | | 19., | 20.], |
| [21., | | 22., | 23.]]) |

# PyTorch 기초

### torch.view( ) – cont.

print(ft.view([-1,4,2]))

print(ft.view([4,-1]))

|  |  |  |
| --- | --- | --- |
|  | [10., | 11.], |
| [12., | 13.], |
| **tensor([[[ 0., 1., 2.],** | [14., | 15.]], |
| **[ 3., 4., 5.]],** | [[16., | 17.], |
| **[[ 6., 7., 8.],** | [18., | 19.], |
| **[ 9., 10., 11.]],** | [20., | 21.], |
| **[[12., 13., 14.],** | [22., | 23.]]]) |
| **[15., 16., 17.]],** |  |  |

**[[18., 19., 20.],**

**[21., 22., 23.]]])**

tensor([[[ 0., 1.],

|  |  |  |
| --- | --- | --- |
| [ | 2., | 3.], |
| [ | 4., | 5.], |
| [ | 6., | 7.]], |
| [[ | 8., | 9.], |

tensor([[ 0., 1., 2., 3., 4., 5.],

[ 6., 7., 8., 9., 10., 11.],

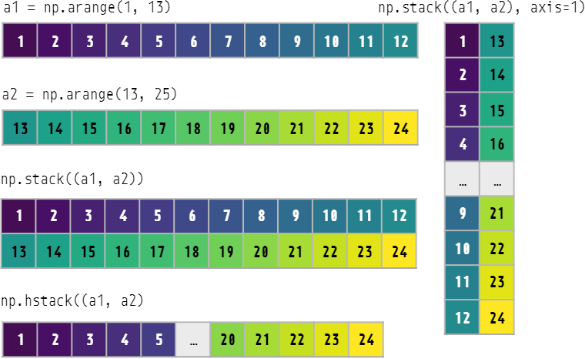
[12., 13., 14., 15., 16., 17.],

[18., 19., 20., 21., 22., 23.]])

# PyTorch 기초

### stack( )

* + - 새로운 축(axis)을 만든 뒤 array들을 해당 축 방향으로 쌓음
    - np.stack(arrays, axis=0)



https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b

# PyTorch 기초

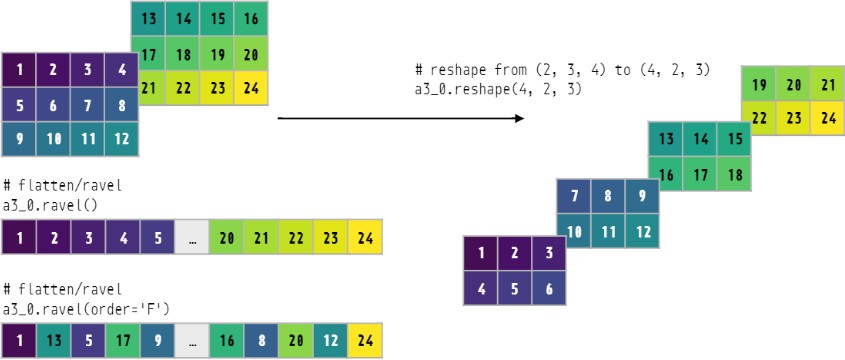
### stack( ) – cont.

https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b

# PyTorch 기초

### flatten and reshape

* + - **ravel()**



https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b

# PyTorch 기초

* + **np.concatenate**
    - array들을 기존의 주어진 축 방향으로 연결
    - array들의 shape는 연결할 축을 제외하고 동일해야 함
    - np.concatenate(arrays, axis=0)

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

b = np.array([[5, 6]]) print(np.concatenate((a,b), axis=0))

c = np.array([[7, 8, 9], [10, 11, 12]])

print(np.concatenate((a, c), axis=1))

|  |  |
| --- | --- |
| [[1 | 2] |
| [3 | 4] |
| [5 | 6]] |

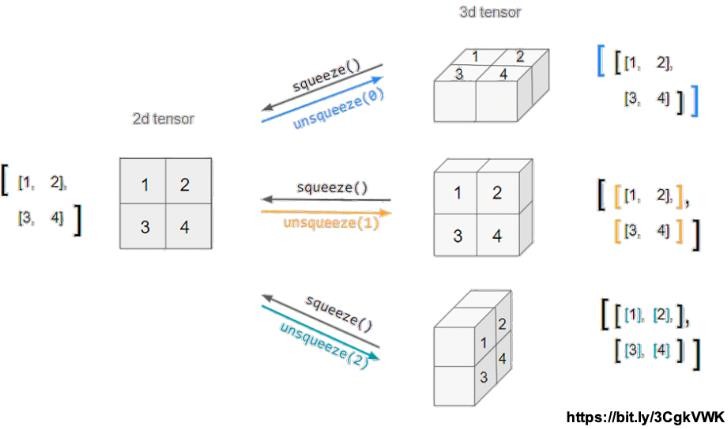
[[ 1 2 7 8 9]

[ 3 4 10 11 12]]

* + - torch.cat()

# PyTorch 기초

### Tensor의 sequeeze와 unsqueeze

* + - sequeeze : 크기가 1인 차원의 삭제
    - unsequeeze : 차원(dimension)의 추가

tensor\_ex = **torch.rand**(size=(2, 1, 2)) tensor\_ex.squeeze()

# tensor([[0.8510, 0.8263],

# [0.7602, 0.1309]])

tensor\_ex = **torch.rand**(size=(2, 2))

tensor\_ex.**unsqueeze**(0).shape # torch.Size([1, 2, 2])

tensor\_ex.**unsqueeze**(1).shape # torch.Size([2, 1, 2])

tensor\_ex.**unsqueeze**(2).shape # torch.Size([2, 2, 1])

# PyTorch 기초

### np.random.random(size)

* + - 0에서 1사이의 무작위 값들로 size 크기의 array 생성

### np.random.randint(low, high=None)

* + - high가 None일 때 : 0 이상 low 이하의 무작위 정수인, size 크기의 array 생성
    - high가 주어졌을 때 : low 이상 high 이하의 무작위 정수인, size 크기의 array 생성

### np.random.normal(loc=0.0, scale=1.0)

* + - 평균이 loc이고 표준편차가 scale인 정규분포에서 추출된, size 크기의 array 생성

print(np.random.random(size=(2, 3)))

print(np.random.randint(3, size=(2, 3)))

print(np.random.randint(3, 8, size=(2, 3)))

print(np.random.normal(2, 1, size=(2, 3)))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [[0.30388459 | 0.40514482 | 0.77824637] | [[6 | 5 5] |
| [0.65416614 | 0.36848698 | 0.43153996]] | [3 | 3 6]] |

[[2 2 2] [[3.25859848 1.92304744 1.42427906]

[2 2 0]] [2.41969025 1.9417514 2.48017538]]

# PyTorch 기초

### np.max, np.min

* + - 최대, 최소값 반환

### np.argmax, np.argmin

* + - 최대, 최소로 만드는 index(들) 반환

### np.where

* + - 특정 조건을 만족하는 곳에서 x의 값을, 아닌 곳에서 y의 값을 가지는 array 반환

### np.argwhere

* + - 특정 조건을 만족하는 index(들) 반환

### np.take

* + - array에서 해당 index들을 뽑아 만든 array 반환

### np.sort

* + - array를 주어진 기준에 대하여 정렬

# PyTorch 기초

### np.transpose / np.ndarray.T

* + - 해당 array의 transpose

### np.expand\_dims

* + - 해당 array에 축 추가

### np.squeeze

* + - 차원 크기가 1인 축 제거

### np.exp, np.log, np.sqrt, np.sin, np.cos, np.tan, ...

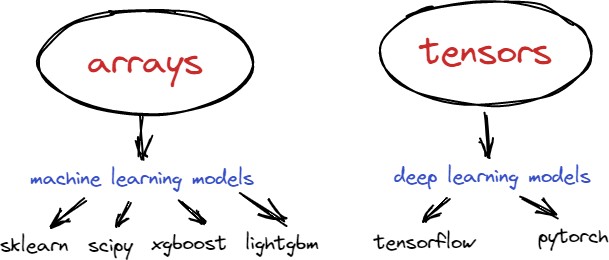
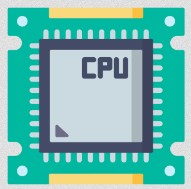
* + - element-wise 함수들

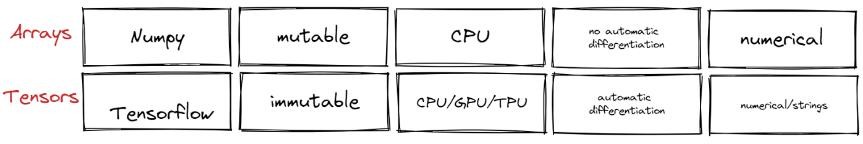
### np.matmul

* + - 행렬 곱

# PyTorch 기초

### Numpy array vs tensor





https://python.plainenglish.io/numpy-arrays-vs-tensorflow-tensors-95a9c39e1c17

## [실습] PyTorch의 텐서

**import** numpy **as** np

**Import** torch

A = tensor([[ 1., -1.],

[ 1., -1.]])

B = tensor([[1, 2, 3],

[4, 5, 6]], dtype=torch.int32)

C = tensor([[0.4306, 0.4923, 0.6163],

[0.8168, 0.6739, 0.3506],

[0.0116, 0.2050, 0.6086]])

D = [[0.43059546 0.49226767

0.6162876 ]

[0.8168291 0.6738524 0.3505581 ]

[0.01155788 0.20499885 0.60861003]]

E = tensor([[[[1, 2, 3],

[4, 5, 6]]]], dtype=torch.int32) sum of A = tensor(0.)

mean of A = tensor(0.)

A = **torch.tensor**([[1., -1.], [1., -1.]]) print('A = ', A)

B = **torch.tensor**(np.array([[1, 2, 3], [4, 5, 6]]))

print('B = ', B)

C = **torch.rand**(3,3) print('C = ', C)

D = C.**numpy**( )

print('D = ', D)

E = B.**view**(1,1,2,3)

print('E = ', E)

print('sum of A = ', A.**sum**( ))

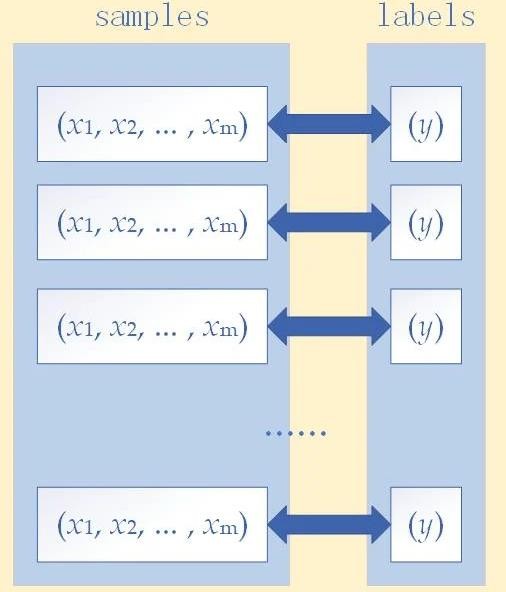
print('mean of A = ', A.**mean**( ))

기계학습, 이건명

# PyTorch 기초

### PyTorch 기초 – cont.

* + - **torch.LongTensor( )**
      * LongTensor로 변환
    - **TensorDataSet(X, Y)**
      * 배열 쌍을 대응되는 원소끼리 결합하여 하나의 데이터 집합 생성
        + 예. (입력 데이터, 출력 레이블)



기계학습, 이건명

# PyTorch 기초

### PyTorch 기초 – cont.

* + - **TensorLoader(tensorDataset, batch\_size=64, shuffle=True)**
      * TensorDataSet 객체를 학습 및 추론에 사용하기 편리한 객체로 변환
      * 배치 단위로 데이터 제공
        + batch\_size : 신경망 가중치를 한번 수정할 때 사용하는 데이터 개수
        + suffle : 데이터 순서를 무작위로 섞을지 여부

기계학습, 이건명

# PyTorch 기초

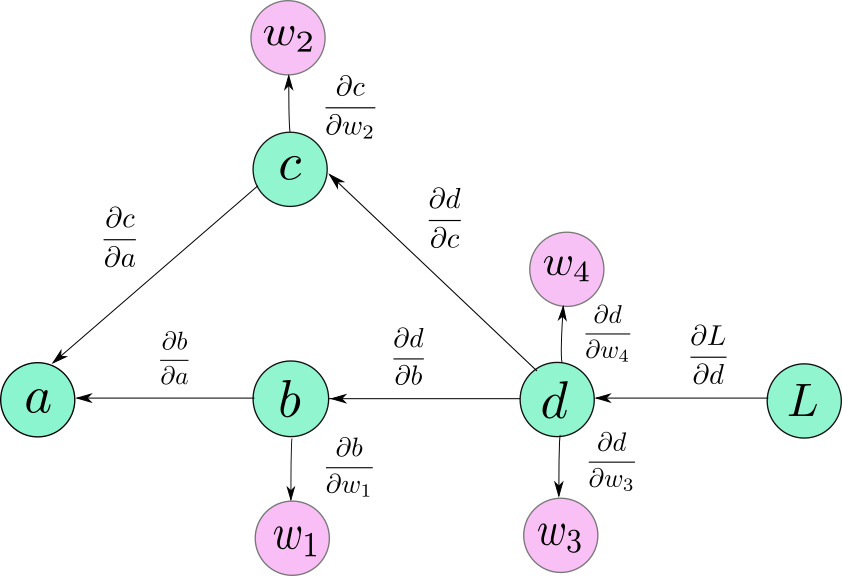
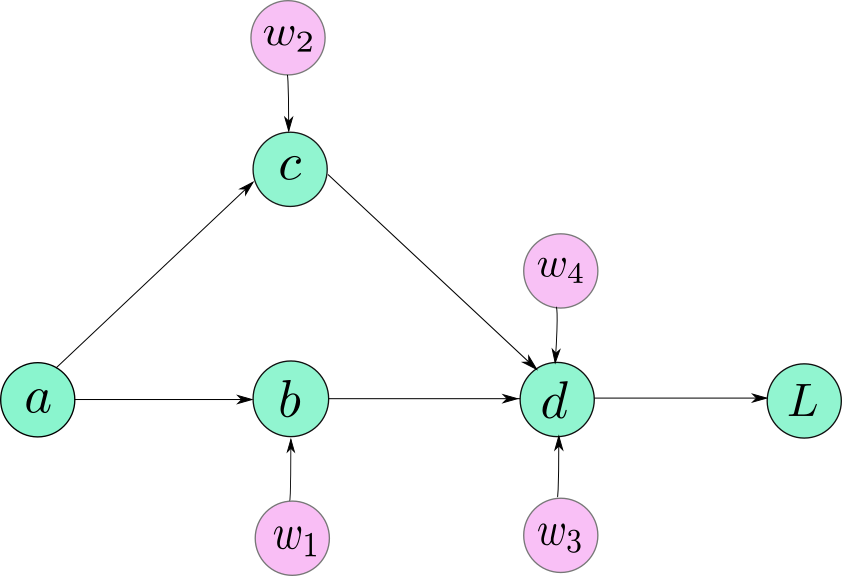
### 계산 그래프(Computation graph)

* + - 연산 과정을 data flow로 나타낸 그래프 구조

b = w1 \* a

c = w2 \* a

d = (w3 \* b) + (w4 \* c) L = f(d)



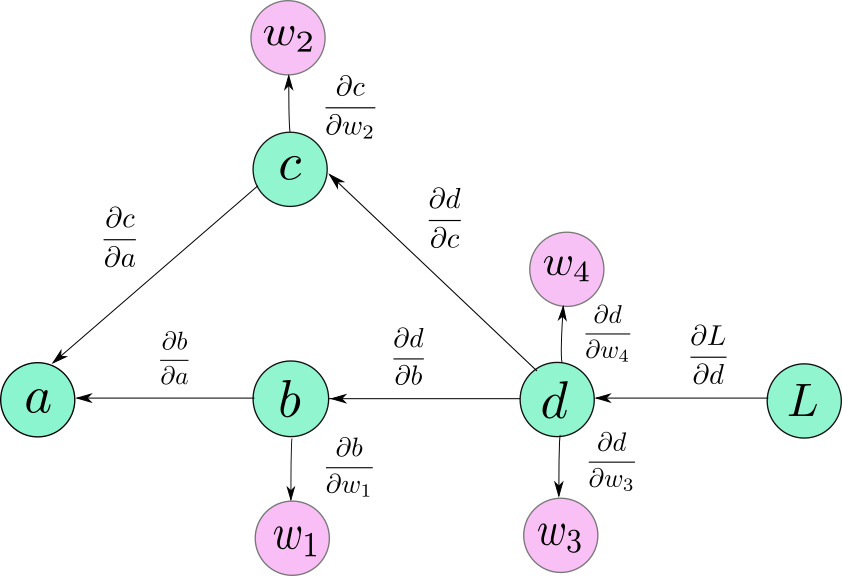
기계학습, 이건명

forward( ) backward( )

https://towardsdatascience.com/getting-started-with-pytorch-part-1-understanding-how-automatic-differentiation-works-5008282073ec

# PyTorch 기초

### 계산 그래프(Computation graph) – cont.

* + - gradient 계산
      * Computation graph를 이용한 chain rule 적용
    - autograd (automatic gradient)

𝛛𝐿

𝛛𝑎

= 𝛛𝐿 𝛛𝑑 𝛛𝑏

𝛛𝑑 𝛛𝑏 𝛛𝑎

+ 𝛛𝐿 𝛛𝑑 𝛛𝑐

𝛛𝑑 𝛛𝑐 𝛛𝑎

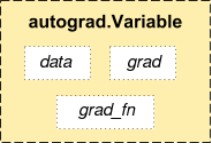
기계학습, 이건명

https://towardsdatascience.com/getting-started-with-pytorch-part-1-understanding-how-automatic-differentiation-works-5008282073ec

# PyTorch 기초

### Variable

* + - autograd.Variable 클래스의 객체
    - Tensor를 감싸고(wrap) 있으며, Tensor 기반으로 정의된 거의 대부분의 연산 지원
    - 계산이 완료된 후 .backward() 를 호출하여 모든 gradient를 계산
    - .data 속성 : tensor 자체(raw tensor)에 접근 가능
    - .grad 속성: 이 변수와 관련된 gradient 누적
    - .grad\_fn : Variable을 생성한 Function 을 참조



## [실습] PyTorch의 텐서

**import** torch

**from** torch.autograd **import** Variable

**x = Variable(torch.tensor([[2.]]), requires\_grad = True)**

print('x = ', x)

print('x.data = ', x.data)

print('x.grad = ', x.grad) print('x.grad\_fn() = ', x.grad\_fn)

**y = x \* x \* 3**

print('\ny = ', y)

print('y.data = ', y.data)

print('y.grad = ', y.grad) print('y.grad\_fn() = ', y.grad\_fn) **z = y\*\*2**

print('\nz = ', z)

print('z.data = ', z.data)

print('z.grad = ', z.grad)

**z.backward( )**

print('\nAfter invocation of backward()') print('\nx = ', x)

print('x.data = ', x.data)

print('x.grad = ', x.grad) print('x.grad\_fn( ) = ', x.grad\_fn) print('\ny = ', y)

print('y.data = ', y.data)

print('y.grad = ', y.grad) print('y.grad\_fn( ) = ', y.grad\_fn) print('\nz = ', z)

print('z.data = ', z.data)

print('z.grad = ', z.grad)

x = tensor([[2.]], requires\_grad=True) x.data = tensor([[2.]])

x.grad = None x.grad\_fn() = None

y = tensor([[12.]], grad\_fn=<MulBackward0>) y.data = tensor([[12.]])

y.grad = None

y.grad\_fn() = <MulBackward0 object at 0x0000022A669C3508>

z = tensor([[144.]], grad\_fn=<PowBackward0>)

z.data = tensor([[144.]]) z.grad = None

After invocation of backward()

x = tensor([[2.]], requires\_grad=True)

x.data = tensor([[2.]])

x.grad = tensor([[288.]]) x.grad\_fn() = None

y = tensor([[12.]], grad\_fn=<MulBackward0>) y.data = tensor([[12.]])

y.grad = None

y.grad\_fn() = <MulBackward0 object at 0x0000022A669BB188>

z = tensor([[144.]], grad\_fn=<PowBackward0>) z.data = tensor([[144.]])

z.grad = None

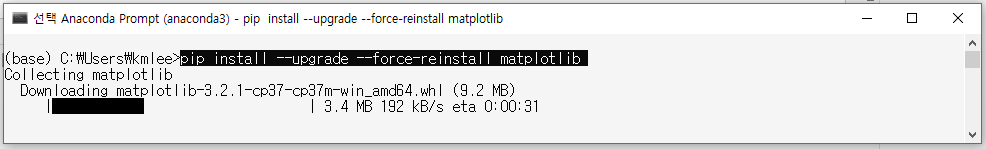
# PyTorch

### PyTorch 기초 – cont.

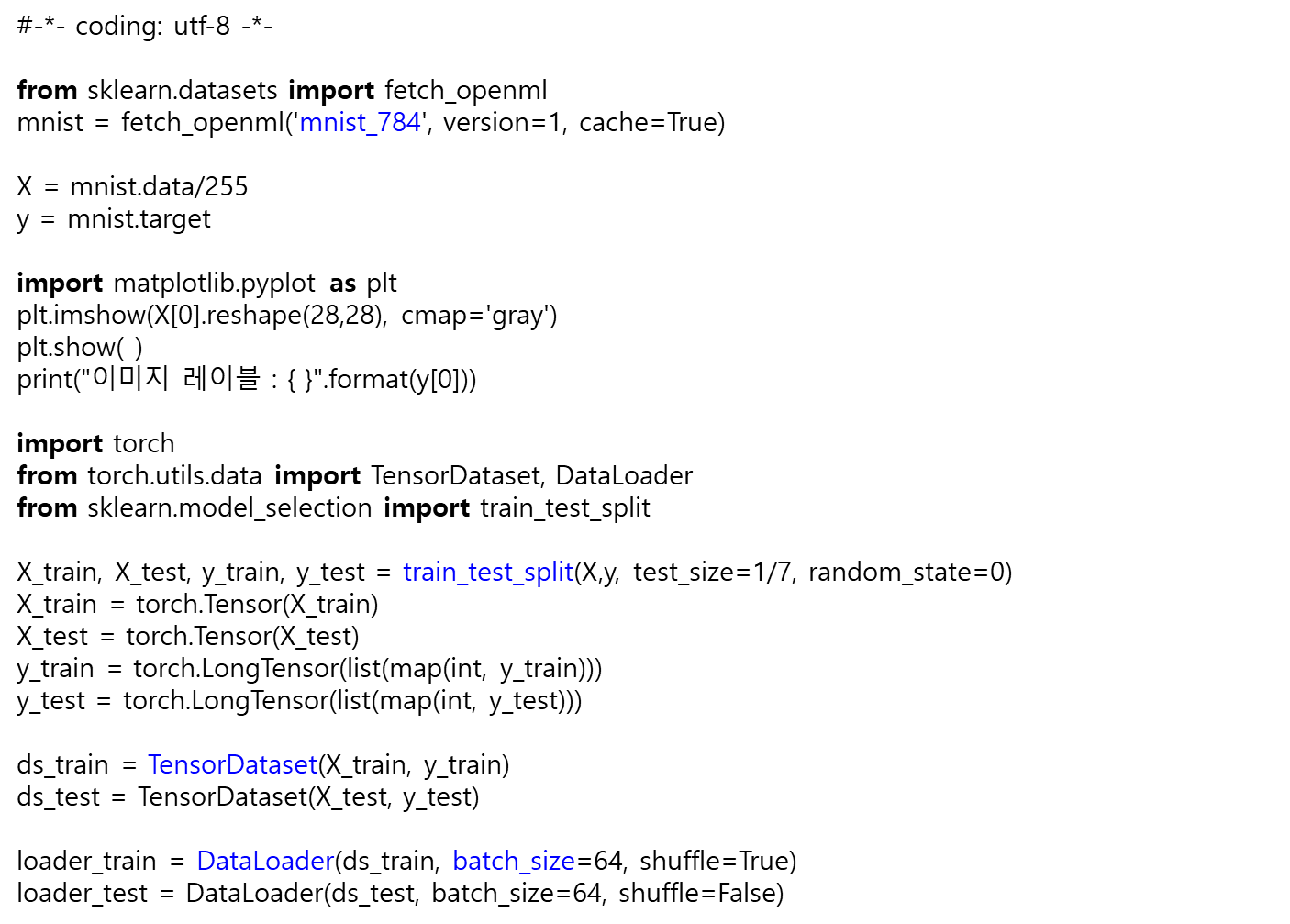
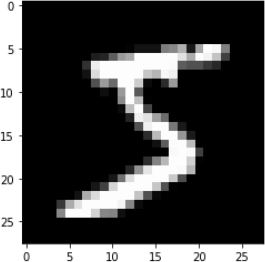
* + model.train( )
    - 신경망 모델을 학습 모드로 전환
  + model.eval( )
    - 신경망 모델을 추론 모델로 전환
  + optimizer.zero\_grad( )
    - 역전파 오차(그레디언트) 계산의 초기화
  + with.torch.no\_grad( )
    - 추론 과정에서는 그레디언트 계산 불필요

### Anaconda에 설치된 패키지와 윈도우 설치 패키지 충돌시

* + Anaconda 환경에서 재설치

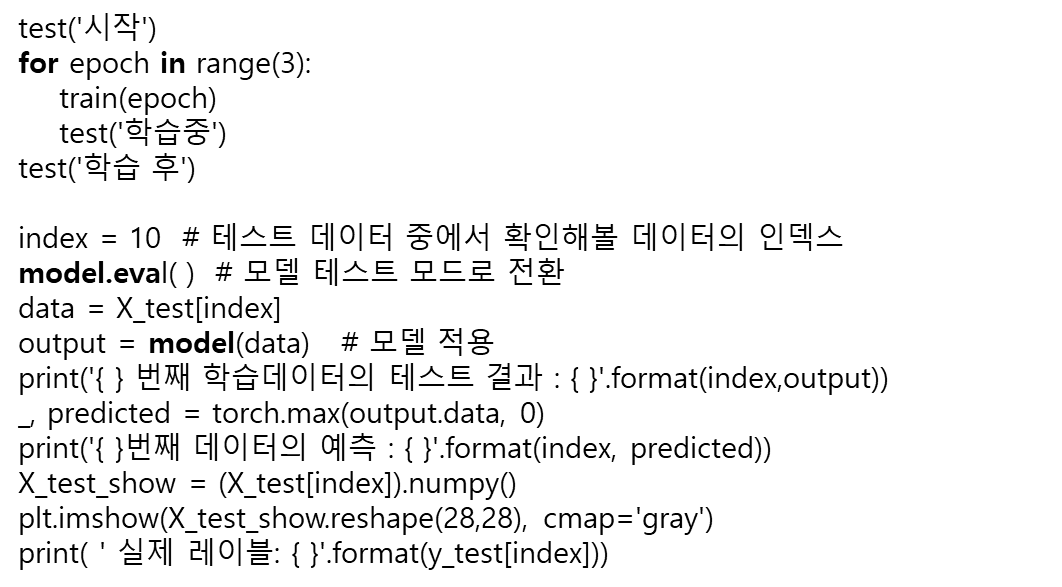
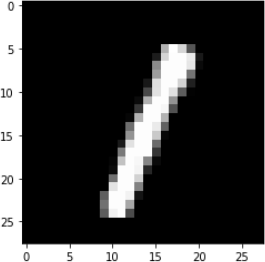


기계학습, 이건명

**[실습] PyTorch의 MLP 프로그래밍**

**이미지 레이블 : 5**





시작 정확도: 796/10000(8%) 에포크 0: 완료

학습중 정확도: 9429/10000(94%) 에포크 1: 완료

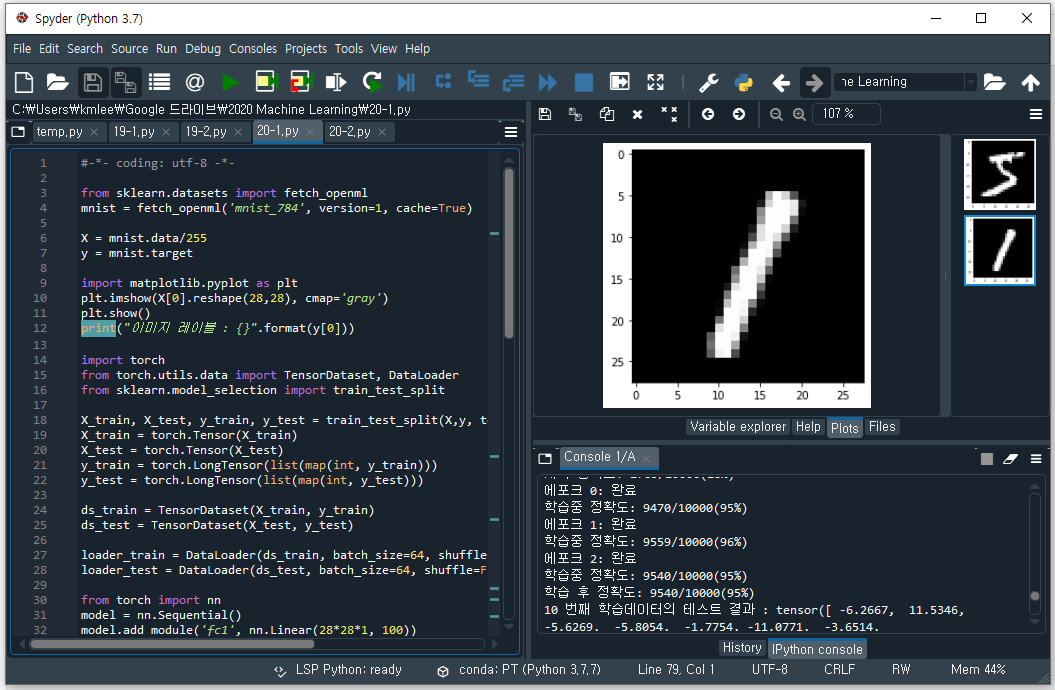
학습중 정확도: 9514/10000(95%) 에포크 2: 완료

학습중 정확도: 9589/10000(96%) 학습 후 정확도: 9589/10000(96%)

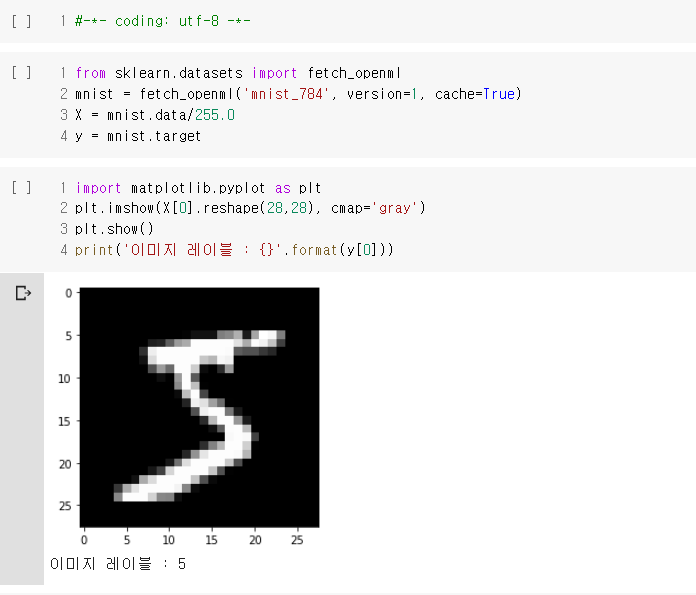
10 번째 학습데이터의 테스트 결과 : tensor([-18.3571, 22.7998, -12.3894, -21.2029, -4.9429, -20.4559, -11.2541,

6.2497, -1.3856, -11.9634], grad\_fn=<AddBackward0>) 10번째 데이터의 예측 : 1

실제 레이블 : 1

**[실습] Spyder에서 PyTorch 실행**

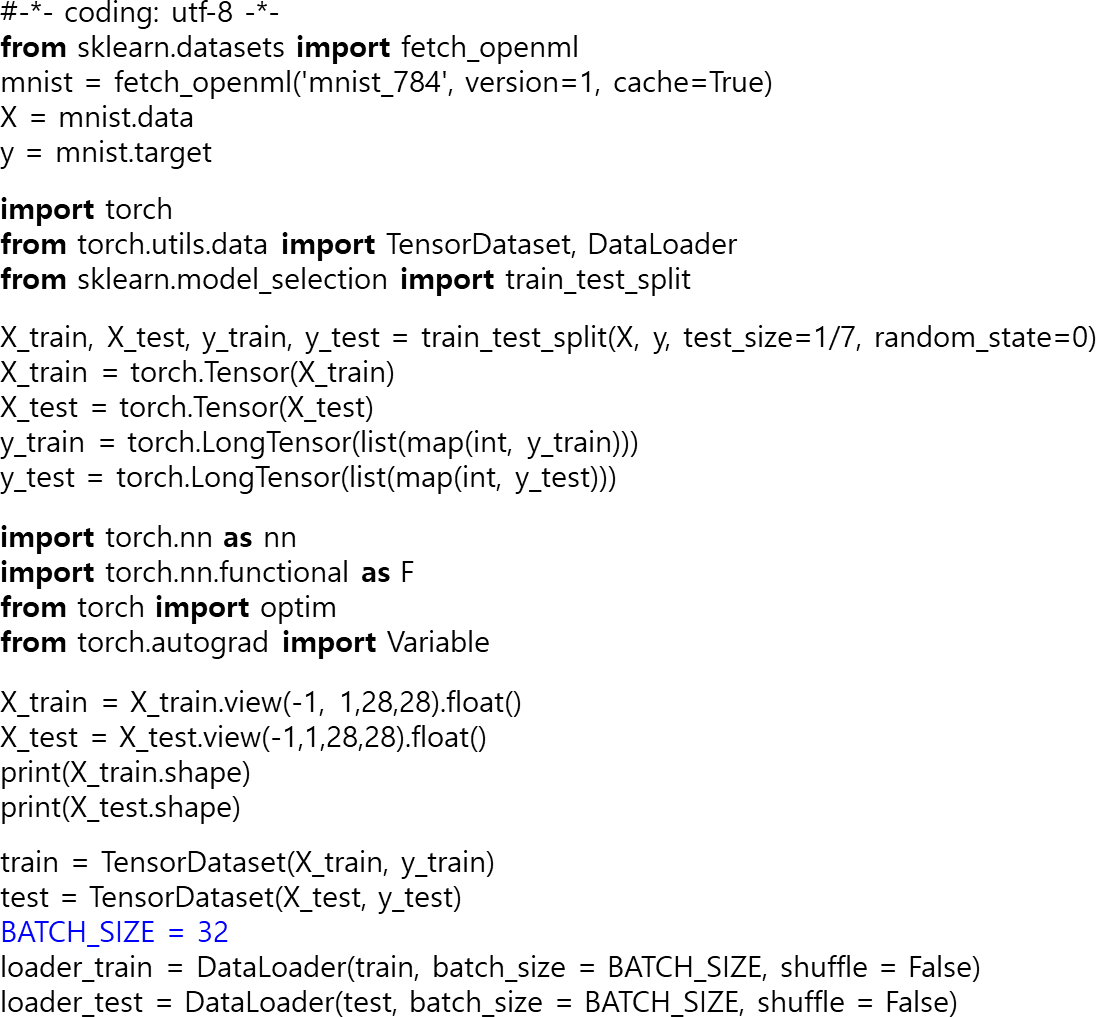
**[실습] Colab에서 PyTorch 실행**

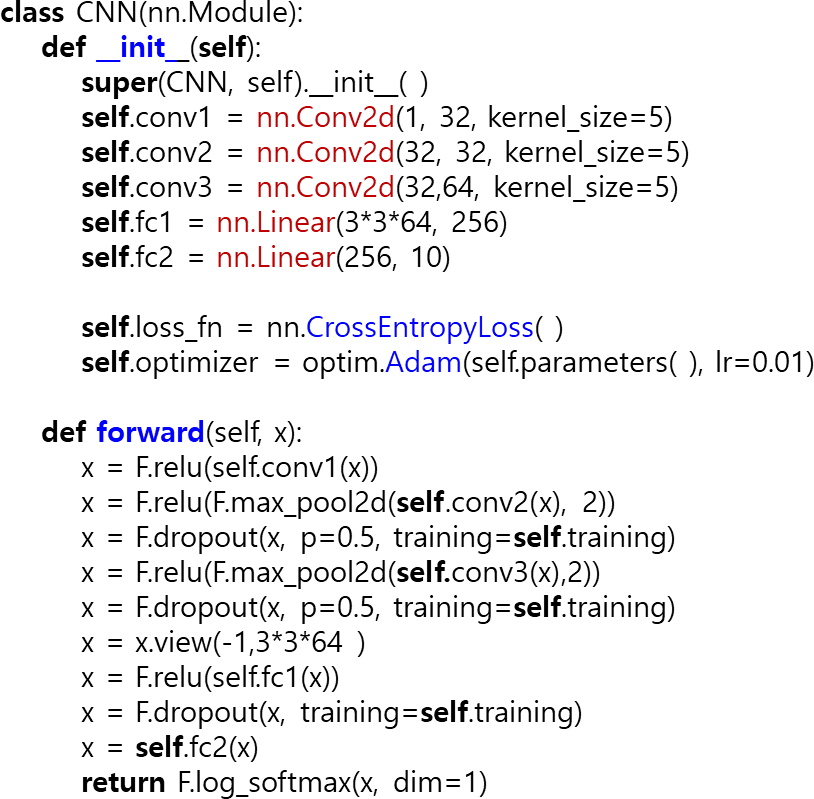




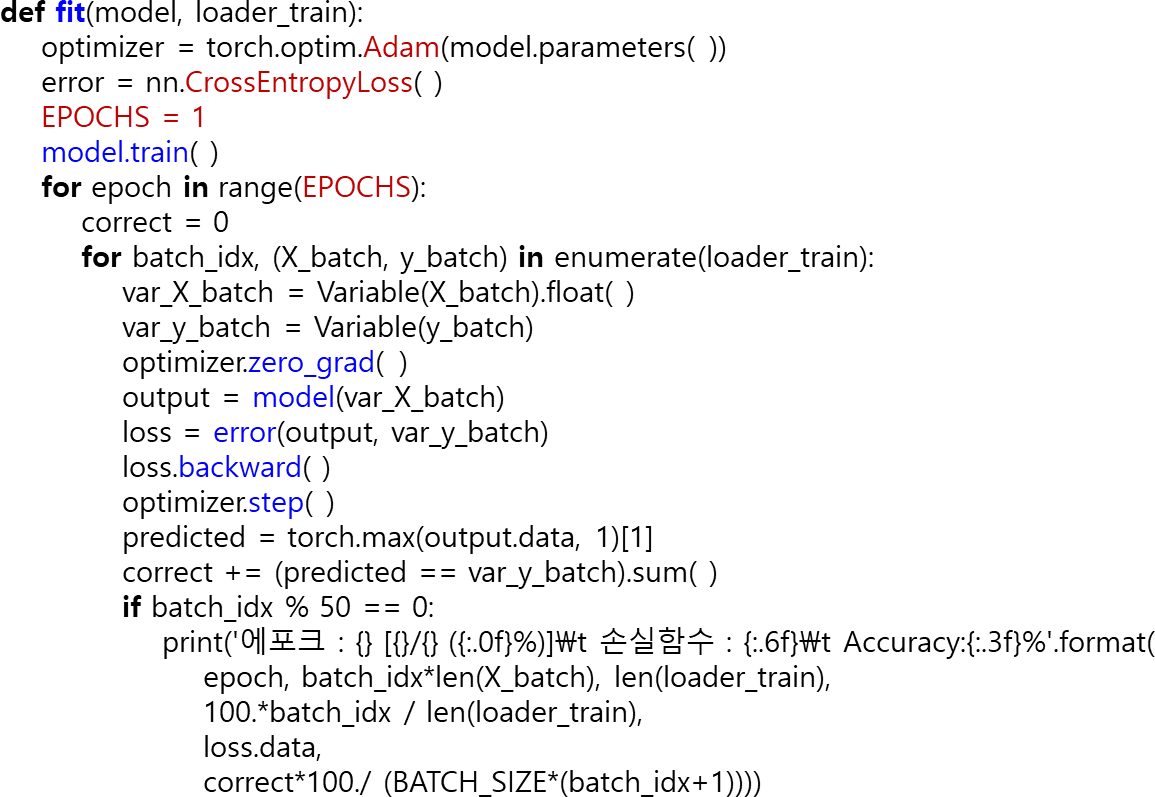


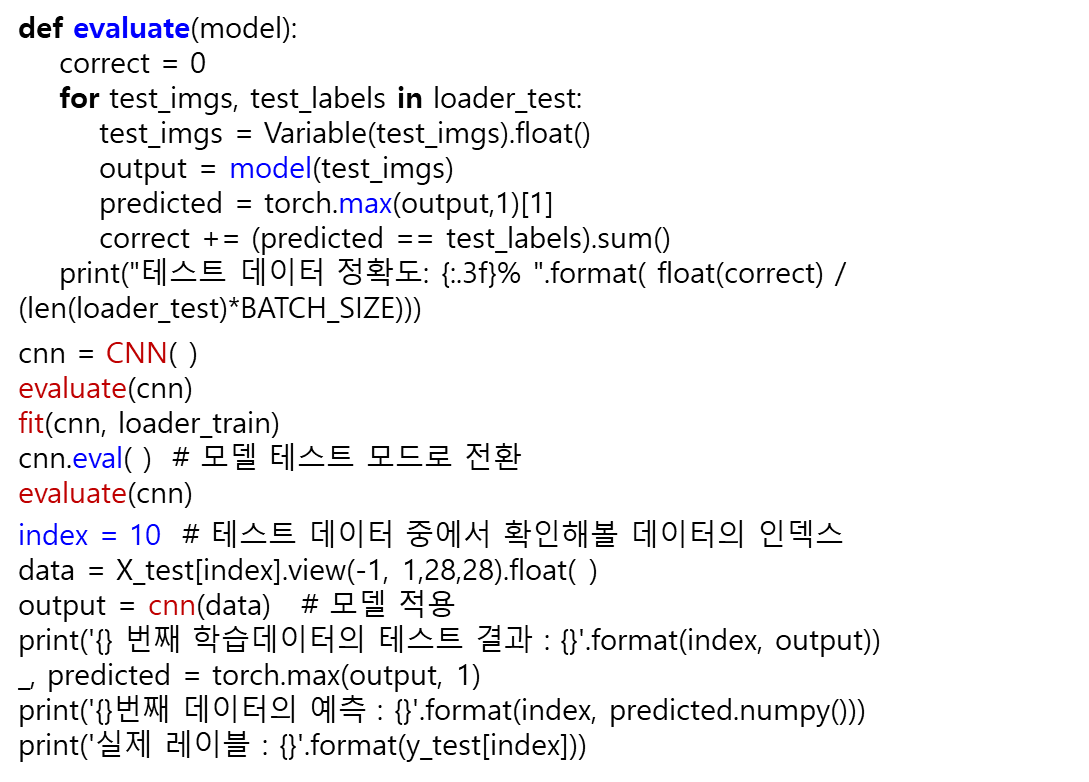
**[실습] CNN 모델을 이용한 MNIST 데이터 분류**





**torch.nn.Conv2d**(in\_channels, out\_channels, kernel\_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding\_mode='zeros')





테스트 데이터 정확도: 0.101%

에포크 : 0 [0/1875 (0%)] 손실함수 : 16.765696 Accuracy:6.250%

에포크 : 0 [1600/1875 (3%)] 손실함수 : 1.837372 Accuracy:21.691%

**:**

에포크 : 0 [59200/1875 (99%)] 손실함수 : 0.256999 Accuracy:86.894%

테스트 데이터 정확도: 0.930%

10 번째 학습데이터의 테스트 결과 : tensor([[-9.7553e+00, -1.5448e-03, -9.4535e+00, -9.9060e+00, -8.7322e+00,

-8.8163e+00, -1.0269e+01, -7.7631e+00, -7.7663e+00, -8.7147e+00]], grad\_fn=<LogSoftmaxBackward>)

10번째 데이터의 예측 : [1] 실제 레이블 : 1

# 실습

1. PyTorch 환경을 구성한다.
2. 실습 프로그래밍 예제를 PyTorch와 Colab 환경에서 직접 실행해 본다.

기계학습, 이건명