

딥러닝 프레임워크 : PyTorch

이건명

충북대학교 대학원 산업인공지능학과

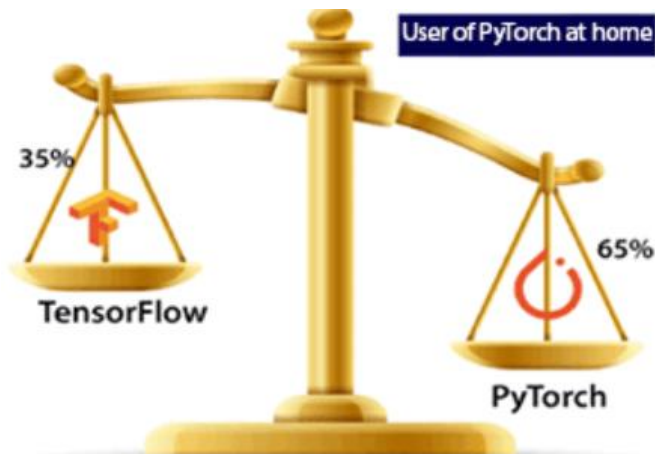
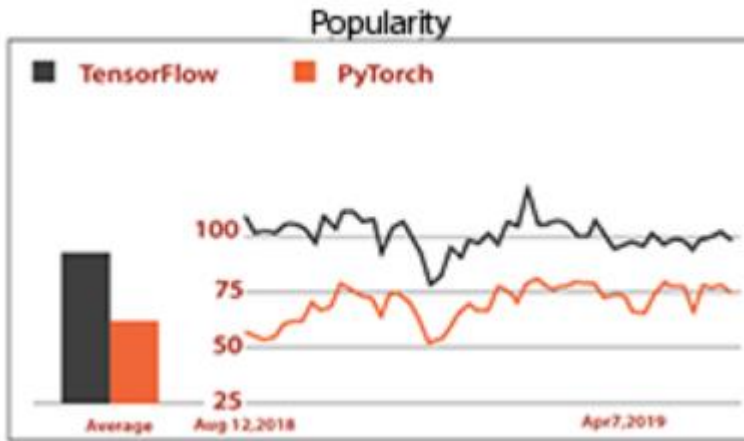
학습 내용

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

1. 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

2. PyTorch 설치

❖ PyTorch 설치

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



```
선택 명령 프롬프트 - deactivate
C:\Users\kml>conda install pytorch torchvision cpuonly -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

```
선택 명령 프롬프트 - deactivate
C:\Users\kml>conda install pytorch torchvision cudatoolkit=10.2 -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done
C:\Users\kml>
```

3. PyTorch 기초

❖ 텐서(Tensor)

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

rank
(dimension)

랭크 0
스칼라



랭크 1

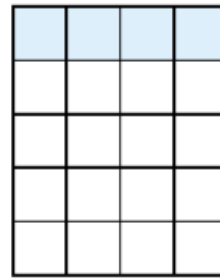


배열

랭크 4

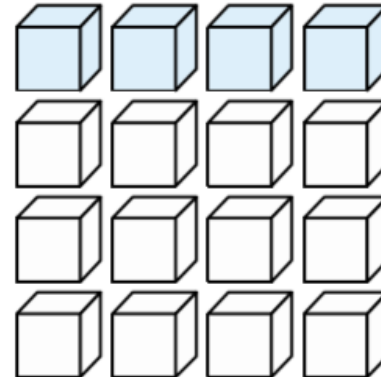


랭크 2

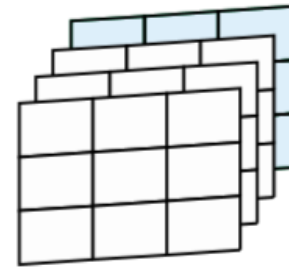


행렬

랭크 5



랭크 3



<pre> 1 import torch 2 3 x = torch.FloatTensor([4 [1, 2], 5 [3, 4], 6 [5, 6], 7 [7, 8] 8]) </pre>	<pre> 1 import numpy as np 2 3 t = np.array([[0, 1, 2], 4 [3, 4, 5]], 5 [[6, 7, 8], 6 [9, 10, 11]], 7 [[12, 13, 14], 8 [15, 16, 17]], 9 [[18, 19, 20], 10 [21, 22, 23]] 11]) 12 ft = torch.FloatTensor(t) </pre>	<pre> 1 ft.sum(axis=1) </pre> <pre> tensor([[3., 5., 7.], [15., 17., 19.], [27., 29., 31.], [39., 41., 43.]]) </pre>
<pre> 1 x.size() </pre>		<pre> 1 ft.sum(axis=2) </pre> <pre> tensor([[3., 12.], [21., 30.], [39., 48.], [57., 66.]]) </pre>
<pre>torch.Size([4, 2])</pre>	<pre> 1 ft.shape </pre>	<pre> 1 ft.sum(axis=-1) </pre>
<pre> 1 x.shape </pre>	<pre>torch.Size([4, 2, 3])</pre>	<pre> tensor([[3., 12.], [21., 30.], [39., 48.], [57., 66.]]) </pre>
<pre>torch.Size([4, 2])</pre>	<pre> 1 ft[0] </pre>	
<pre> 1 x.shape[0] </pre>	<pre> tensor([[0., 1., 2.], [3., 4., 5.]]) </pre>	<pre> 1 ft.view([-1,3]) # 크기가 (? ,3)이 되도록 재배치 (res </pre>
<pre>4</pre>	<pre> 1 ft[1] </pre>	<pre> 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.]]) </pre>
<pre> 1 x.dim() </pre>	<pre> tensor([[6., 7., 8.], [9., 10., 11.]]) </pre>	
<pre>2</pre>	<pre> 1 ft[0][0] </pre>	
<pre> 1 x.sum(axis=0) </pre>	<pre> tensor([0., 1., 2.]) </pre>	<pre> 1 ft.view([-1,4,2]) # 크기가 (? ,4,2)이 되도록 재배치 </pre>
<pre> tensor([16., 20.]) </pre>	<pre> 1 ft[2][1] </pre>	<pre> 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.]])]) </pre>
<pre> 1 x.sum(axis=1) </pre>	<pre> tensor([15., 16., 17.]) </pre>	
<pre> tensor([3., 7., 11., 15.]) </pre>	<pre> 1 ft[1][0][2] </pre>	
<pre> 1 x.sum(axis=-1) </pre>	<pre> tensor(8.) </pre>	
<pre> tensor([3., 7., 11., 15.]) </pre>	<pre> 1 ft.sum(axis=0) # 해당 axis가 없어지도록 연산 </pre>	
	<pre> tensor([[36., 40., 44.], [48., 52., 56.]]) </pre>	

```
1 x = np.array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
2             17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31])
3 tx = torch.FloatTensor(x)
```

```
1 ty = tx.view([4,2,4])
```

```
1 ty
```

```
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.]],

        [[24., 25., 26., 27.],
          [28., 29., 30., 31.]])
```

```
1 t = torch.Tensor([0, 1, 2])
```

```
1 t.shape
```

```
torch.Size([3])
```

```
1 s = t.unsqueeze(0) # 0은 첫번째 차원에 차원 추가
```

```
1 s.shape
```

```
torch.Size([1, 3])
```

```
1 u = s.view([3,1])
```

```
1 u
```

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

```
1 v = u.squeeze(-1)
```

```
1 v
```

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

```
1 x = torch.FloatTensor([1, 4])
2 y = torch.FloatTensor([2, 5])
3 z = torch.FloatTensor([3, 6])
```

```
1 s1 = torch.stack([x, y, z])
```

```
1 s1
```

```
tensor([[1., 4.],
        [2., 5.],
        [3., 6.]])
```

```
1 s2 = torch.stack([x, y, z], dim=1)
```

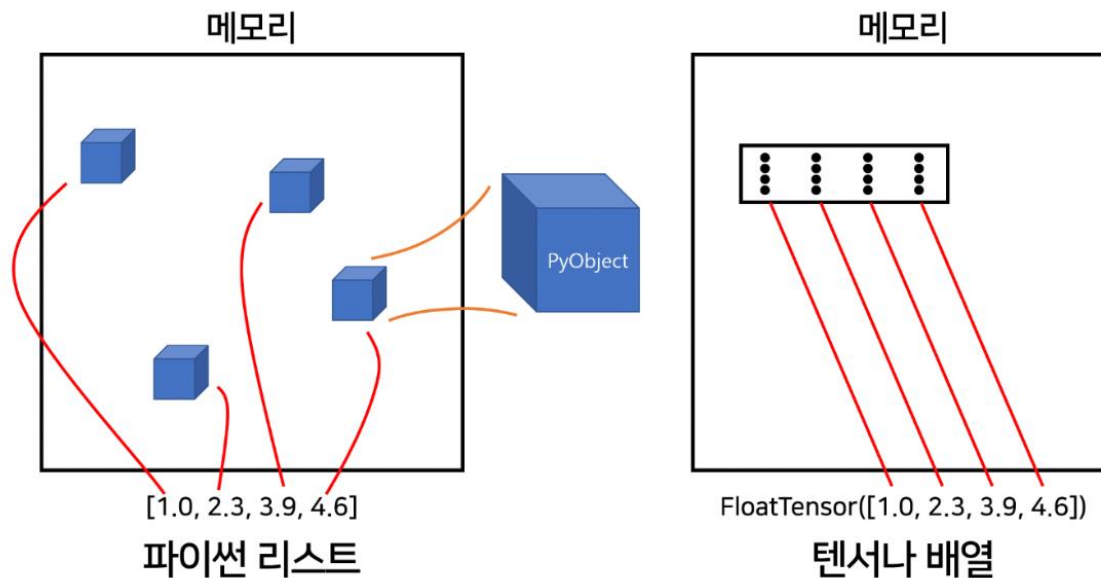
```
1 s2
```

```
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

PyTorch 기초

❖ Tensor

- NumPy의 (**n차원**) **배열 ndarray** 객체와 유사하지만 GPU등 HW가속기에서 실행 할 수 있는 자료구조
- Python **list**는 object 포인터들을 모아놓은 집합
 - 메모리 할당에 있어 비효율적이고, 느림
- **numpy array**와 **tensor**는 메모리상 인접한 곳에 공간이 할당



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 등)
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])
```

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 \Rightarrow 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 \Rightarrow Numpy Array)

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

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

PyTorch 기초

❖ 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]])
```

```
shape = (2,3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)
```

```
print(rand_tensor)
print(ones_tensor)
print(zeros_tensor)
```

```
tensor([[0.8398, 0.8787, 0.4099],
        [0.6517, 0.2316, 0.1294]])
tensor([[1., 1., 1.],
        [1., 1., 1.]])
tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

PyTorch 기초

❖ Array 연산

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

```
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], [0, 0, 0]])
print(zeros > ones)
```

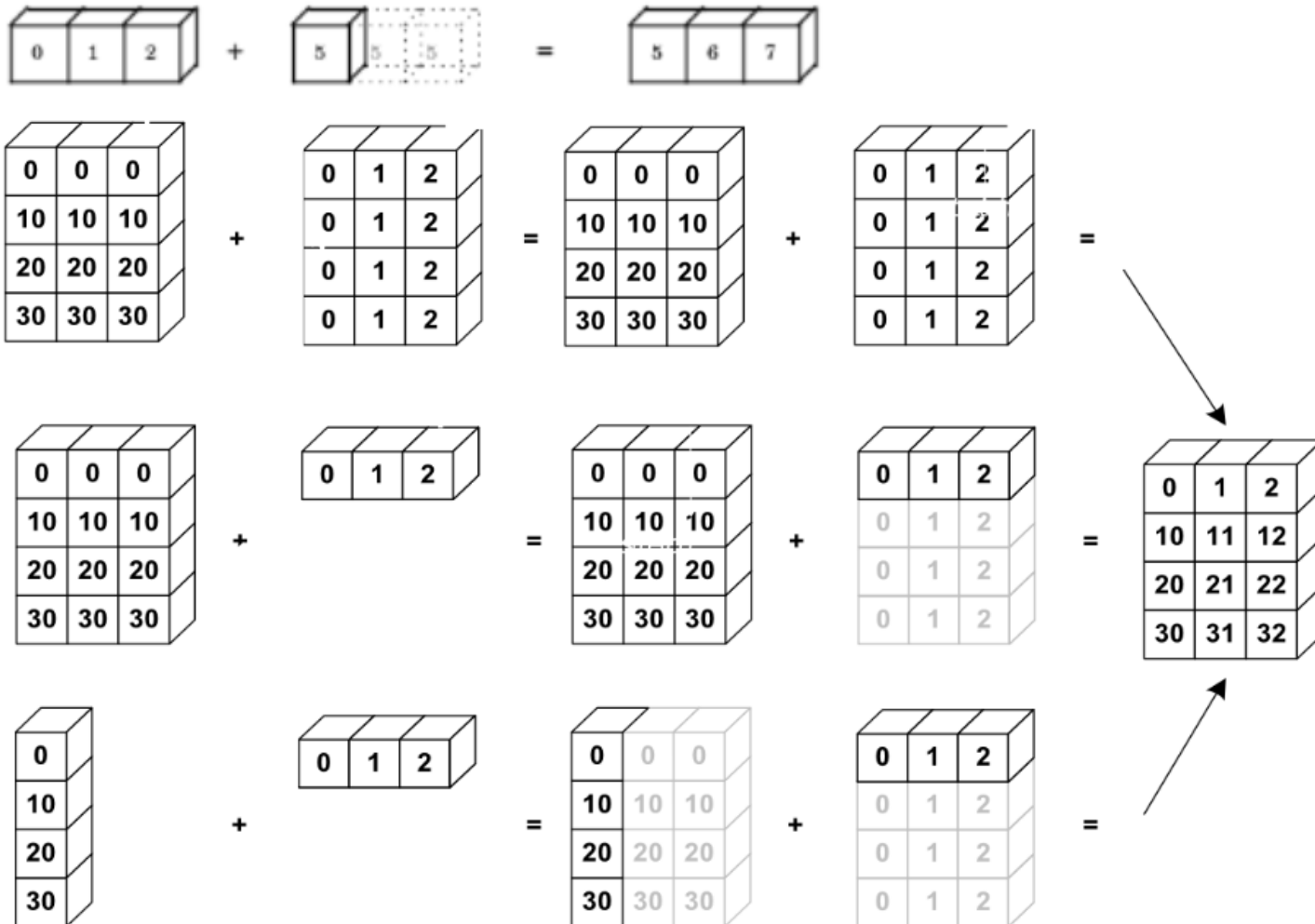
```
[[1.  1.  1.]
 [1.  1.  1.]
 [1.  1.  1.]]
[[0.5  0.5  0.5]
 [0.5  0.5  0.5]
 [0.5  0.5  0.5]]
[[2.  2.  2.]
 [2.  2.  2.]
 [2.  2.  2.]]

[[ True True True]
 [ True True True]
 [ True True True]]
[[False False False]
 [False False False]
 [False False False]]
```

PyTorch 기초

❖ Broadcasting

- shape이 다른 연산 지원을 위한 **복사**를 통해 **shape 일치**



PyTorch 기초

```
1 from PIL import Image
2 import requests
3 from io import BytesIO
4 from matplotlib.pyplot import imshow
5
6 url = 'https://i.imgur.com/BBcy6Wc.jpg'
7 response = requests.get(url)
8 img = Image.open(BytesIO(response.content))
9 a = np.asarray(img)
10 print('a.shape : ', a.shape)
11 imshow(a)
```

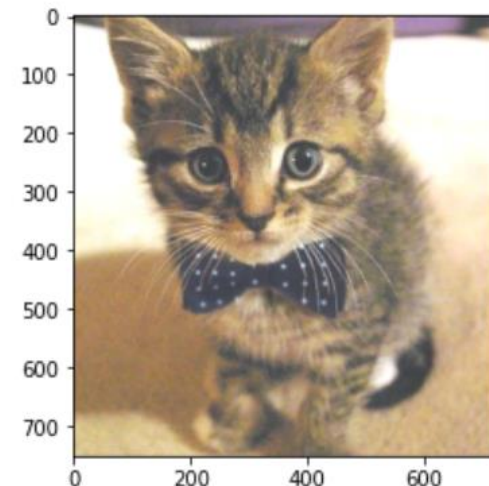
```
1 b = [64, 64, 64]
2 print('(a+b).shape : ', (a+b).shape)
3 imshow(a+b)
```

❖ **PIL** (Python Image Manipulation Library,
Python Imaging Library)

```
a.shape : (750, 722, 3)
<matplotlib.image.AxesImage at 0x7f72530c5d10>
```



```
(a+b).shape : (750, 722, 3)
<matplotlib.image.AxesImage at 0x7f725302f790>
```



PyTorch 기초

❖ Array **indexing**과 **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])           [1 2 3]
```

```
print(a[1][-1])       6
```

```
print(a[0:2])          [[1 2 3]  
                       [4 5 6]]
```

```
print(a[:, 1])          [2 5 8]
```

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

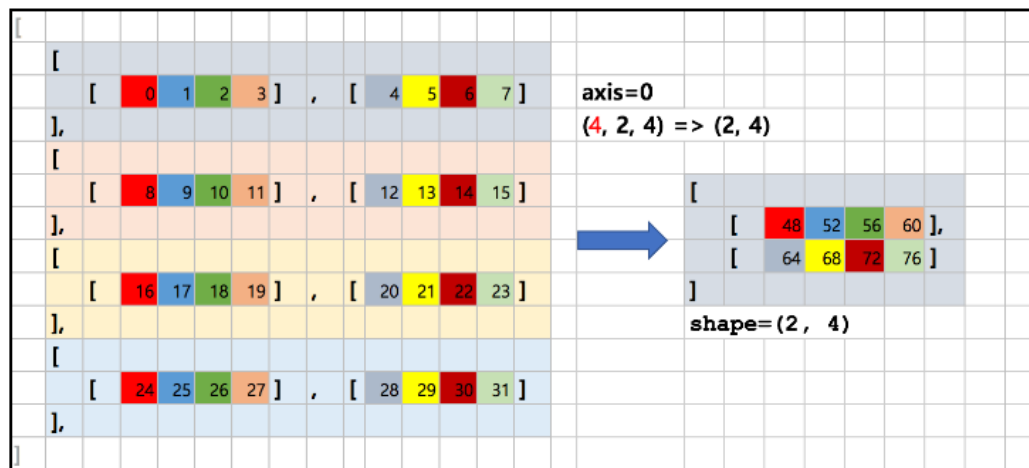
PyTorch 기초

❖ `np.sum()`/`torch.sum()`

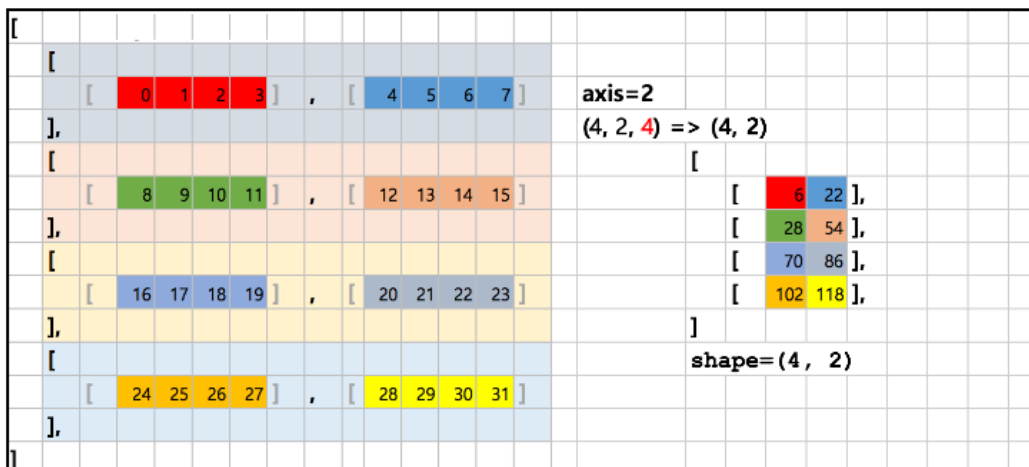
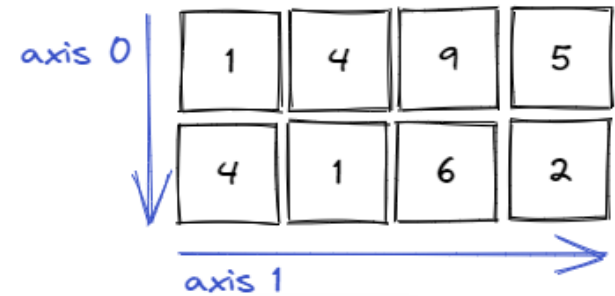
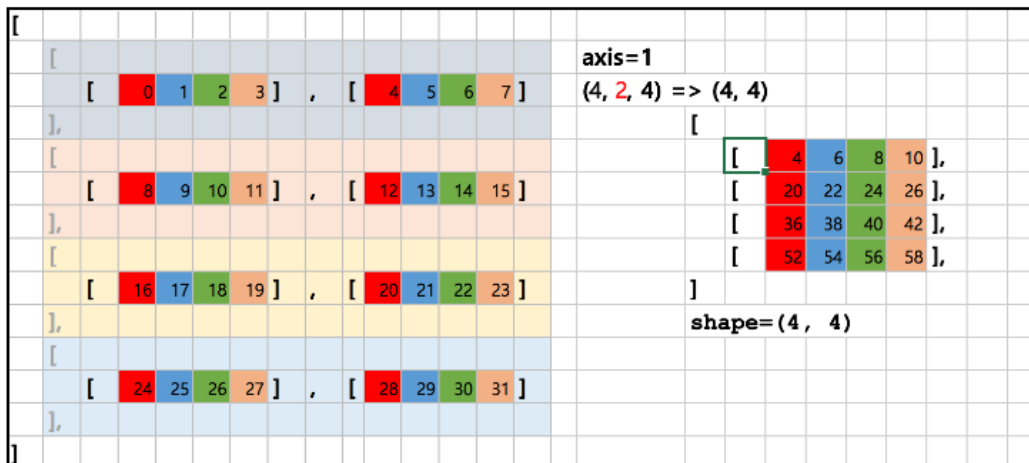
- 특정 축(axis)을 따라 모든 원소를 더함
- `np.sum(a, axis=None, keepdims=False)`
 - `a` : 대상 array
 - `axis` : element를 따라 더할 축의 index
 - `keepdims` : True일 경우 해당 축에 1차원 벡터로 남음.
False일 경우 해당 축에 (0차원) element로 남음.

```
a = np.array([[1, 2, 3], [3, 5, 7]])  
print(np.sum(a))  
print(np.sum(a, axis=0, keepdims=True))  
print(np.sum(a, axis=1, keepdims=True))  
print(np.sum(a, axis=1, keepdims=False))
```

21
[[4 7 10]]
[[6]
[15]]
[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

```
a1 = np.arange(1, 13)
```

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

→

1	2	3	4
5	6	7	8
9	10	11	12

```
a1.reshape(3, 4)  
a1.reshape(-1, 4)  
a1.reshape(3, -1)  
.ravel() # back to 1D
```

↓

1	4	7	10
2	5	8	11
3	6	9	12

```
a1.reshape(3, -1, order='F')  
.ravel(order='F') # back to 1D
```

PyTorch 기초

❖ torch.view()

```
import torch
ft = torch.FloatTensor([[[0, 1, 2],
                        [3, 4, 5]],
                       [[6, 7, 8],
                        [9, 10, 11]],
                       [[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.]])
```

```
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]))
```

```
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.]])
```

```
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.]])
```

```
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)`

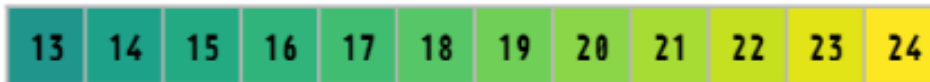
```
a1 = np.arange(1, 13)
```



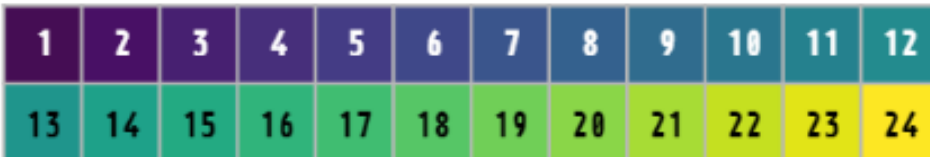
```
np.stack((a1, a2), axis=1)
```



```
a2 = np.arange(13, 25)
```



```
np.stack((a1, a2))
```



```
np.hstack((a1, a2))
```



PyTorch 기초

❖ stack() – cont.

```
a1 = np.arange(1, 13).reshape(3, 4)
a2 = np.arange(13, 25).reshape(3, -1)
```

1	2	3	4
5	6	7	8
9	10	11	12

13	14	15	16
17	18	19	20
21	22	23	24

```
# stack along axis 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape: (3, 4, 2)
```

```
# retrieve a1
a3_2[:, :, 0]
```

		9	21
		10	22
		11	23
		12	24
	5	17	
	6	18	
	7	19	
	8	20	
1	13		
2	14		
3	15		
4	16		

```
# stack along axis 0
a3_0 = np.stack((a1, a2))
a3_0.shape: (2, 3, 4)
```

		13	14	15	16
		17	18	19	20
		21	22	23	24
1	2	3	4		
5	6	7	8		
9	10	11	12		

```
# retrieve a1
a3_0[0]
```

```
a3_0[0, :, :]
```

```
# stack along axis 1
a3_1 = np.stack((a1, a2), axis=1)
a3_1.shape: (3, 2, 4)
```

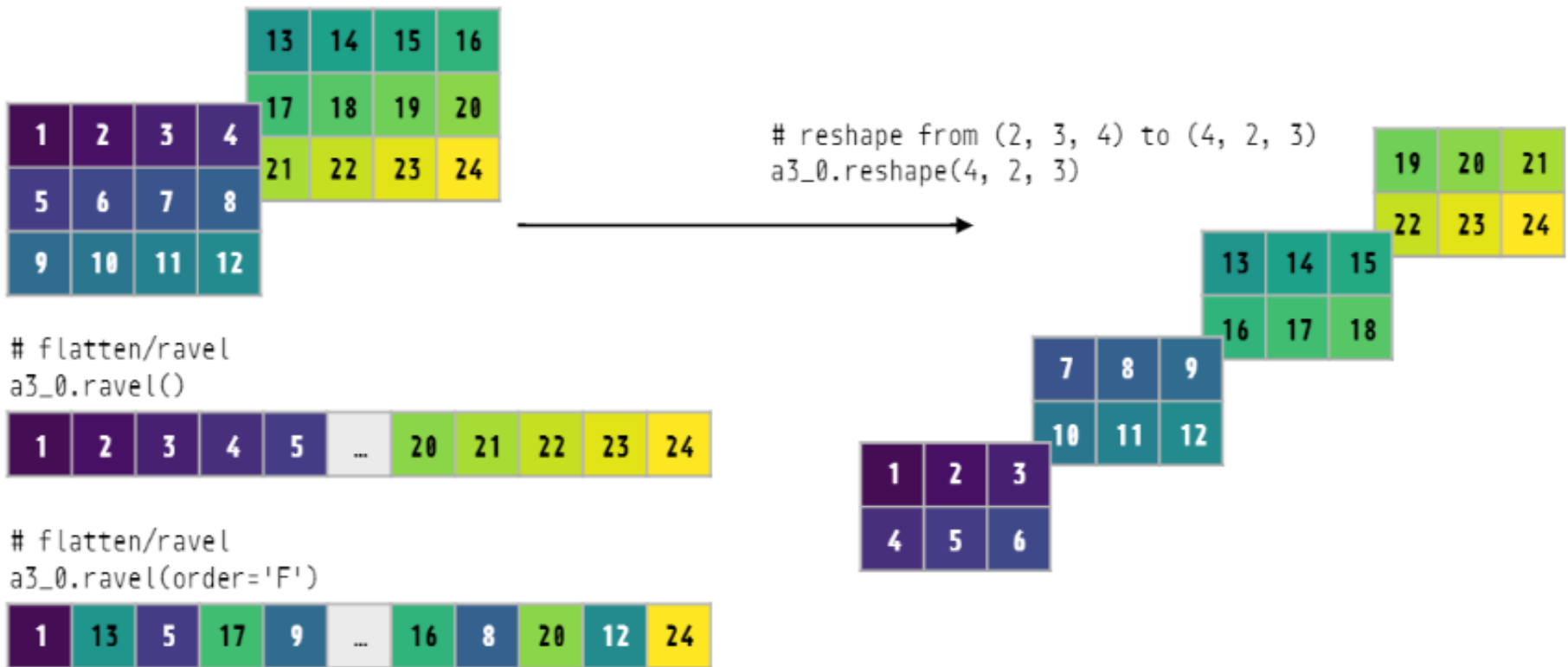
				9	10	11	12
				21	22	23	24
	5	6	7	8			
	17	18	19	20			
1	2	3	4				
13	14	15	16				

```
# retrieve a1
a3_1[:, 0, :]
```

PyTorch 기초

❖ flatten and reshape

- `ravel()`



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]]
```

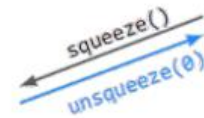
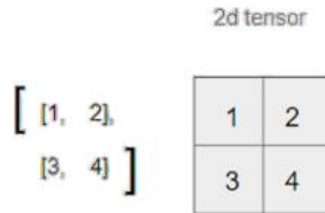
```
[[ 1  2  7  8  9]  
 [ 3  4 10 11 12]]
```

- `torch.cat()`

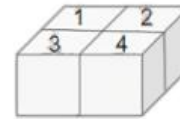
PyTorch 기초

❖ Tensor의 squeeze와 unsqueeze

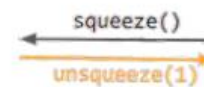
- **squeeze** : 차원(dimension)의 추가
- **unsqueeze** : 크기가 1인 차원의 삭제



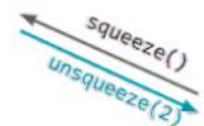
3d tensor



`[[[1, 2],
[3, 4]]]`



`[[1, 2],
[3, 4]]`



`[[1], [2],
[3], [4]]`

```
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])
```

<https://bit.ly/3CgkVWK>

PyTorch 기초

❖ `np.random.random(size)`

- 0에서 1사이의 무작위 값들로 `size` 크기의 array 생성

❖ `np.random.randint(low, high=None)`

- `high`가 `None`일 때 : 0 이상 `low` 이하의 무작위 정수인, `size` 크기의 array 생성
- `high`가 주어졌을 때 : `low` 이상 `high` 이하의 무작위 정수인, `size` 크기의 array 생성

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

- 평균이 `loc`이고 표준편차가 `scale`인 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]
 [0.65416614 0.36848698 0.43153996]]
[[2 2 2]
 [2 2 0]]

[[6 5 5]
 [3 3 6]]
[[3.25859848 1.92304744 1.42427906]
 [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, ...**

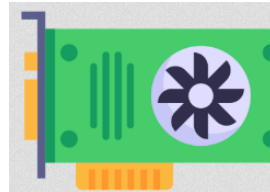
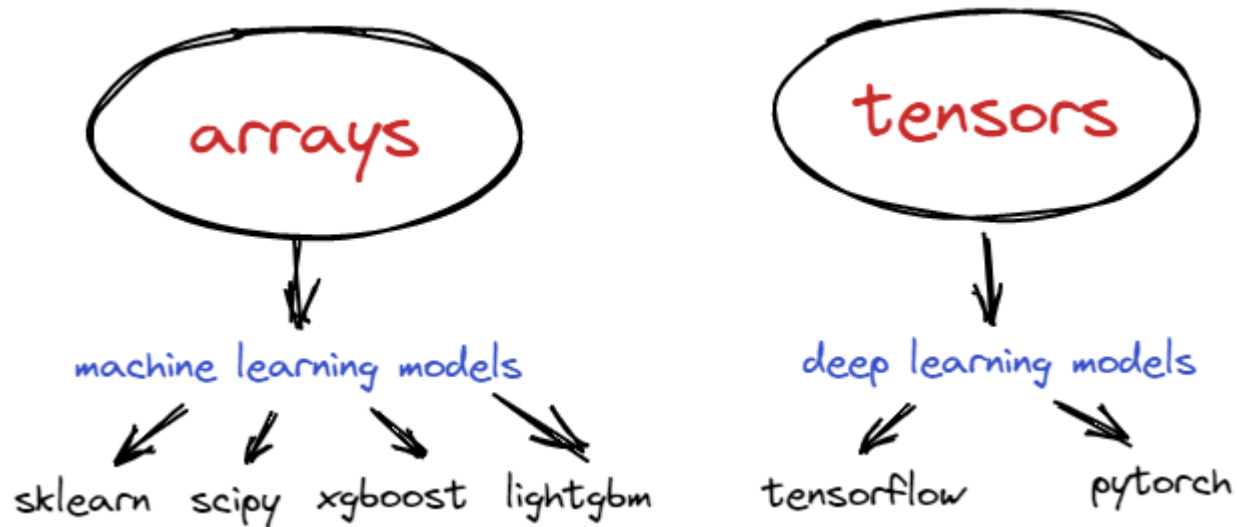
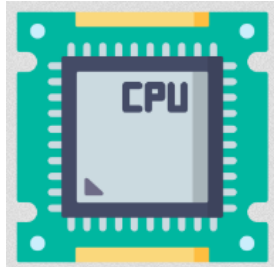
- element-wise 함수들

❖ **np.matmul**

- 행렬 곱

PyTorch 기초

❖ Numpy array vs tensor



Arrays	Numpy	mutable	CPU	no automatic differentiation	numerical
Tensors	pytorch Tensorflow	immutable	CPU/GPU/TPU	automatic differentiation	numerical/strings

[실습] PyTorch의 텐서

```
import numpy as np
Import torch
```

```
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( ))
```

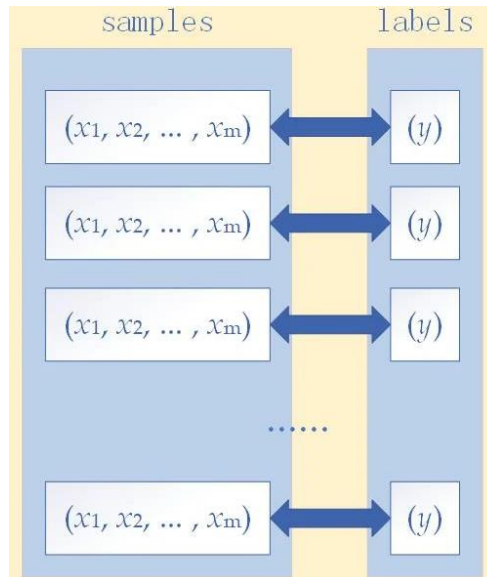
```
A = tensor([[ 1., -1.],
             [ 1., -1.]])
B = tensor([[1, 2, 3],
            [4, 5, 6]], dtype=torch.int32)

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.)
```

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)



PyTorch 기초

❖ PyTorch 기초 – cont.

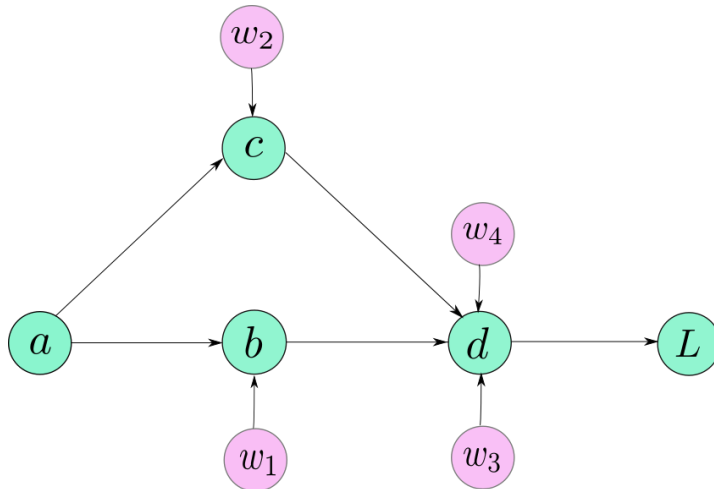
- **TensorLoader(tensorDataset, batch_size=64, shuffle=True)**
 - TensorDataSet 객체를 학습 및 추론에 사용하기 편리한 객체로 변환
 - 배치 단위로 데이터 제공
 - batch_size : 신경망 가중치를 한번 수정할 때 사용하는 데이터 개수
 - shuffle : 데이터 순서를 무작위로 섞을지 여부

PyTorch 기초

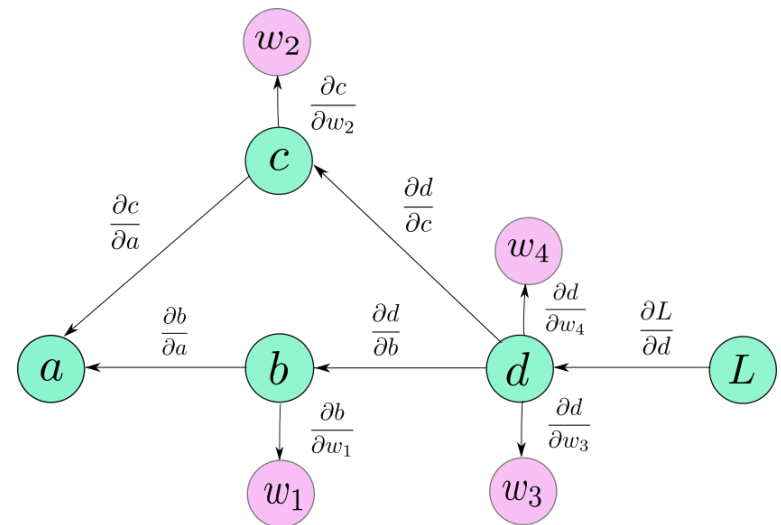
❖ 계산 그래프(Computation graph)

- 연산 과정을 data flow로 나타낸 그래프 구조

$$\begin{aligned}b &= w_1 * a \\c &= w_2 * a \\d &= (w_3 * b) + (w_4 * c) \\L &= f(d)\end{aligned}$$



forward()

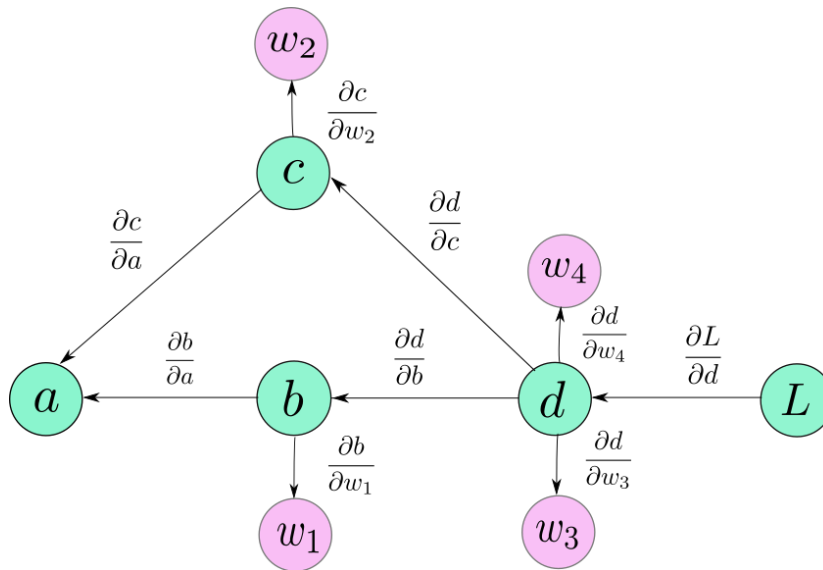


backward()

PyTorch 기초

❖ 계산 그래프(Computation graph) – cont.

- gradient 계산
 - Computation graph를 이용한 chain rule 적용



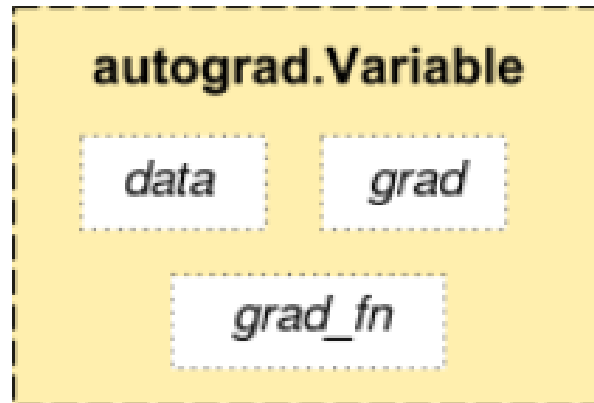
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial d} \frac{\partial d}{\partial b} \frac{\partial b}{\partial a} + \frac{\partial L}{\partial d} \frac{\partial d}{\partial c} \frac{\partial c}{\partial a}$$

- autograd (automatic gradient)

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('y = ', y)
print('y.data = ', y.data)
print('y.grad = ', y.grad)
print('y.grad_fn() = ', y.grad_fn())
```

```
z = y**2
print('z = ', z)
print('z.data = ', z.data)
print('z.grad = ', z.grad)
```

```
z.backward()
print('After invocation of backward()')
print('x = ', x)
print('x.data = ', x.data)
print('x.grad = ', x.grad)
print('x.grad_fn() = ', x.grad_fn())
print('y = ', y)
print('y.data = ', y.data)
print('y.grad = ', y.grad)
print('y.grad_fn() = ', y.grad_fn())
print('z = ', 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
```

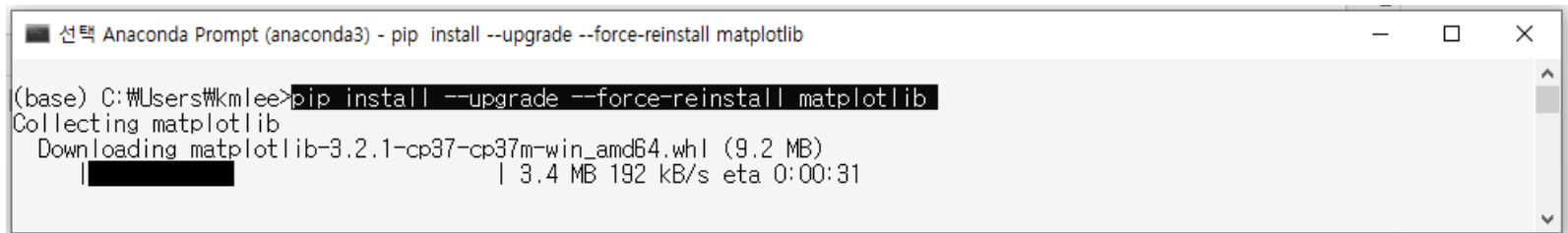

PyTorch

❖ 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



```
선택 Anaconda Prompt (anaconda3) - pip install --upgrade --force-reinstall matplotlib  
  
(base) C:\Users\kml\ee>pip install --upgrade --force-reinstall matplotlib  
Collecting matplotlib  
  Downloading matplotlib-3.2.1-cp37-cp37m-win_amd64.whl (9.2 MB)  
    |██████████| 3.4 MB 192 kB/s eta 0:00:31
```

[실습] PyTorch의 MLP 프로그래밍

```
#-*- coding: utf-8 -*-
```

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)
```

```
X = mnist.data/255
y = mnist.target
```

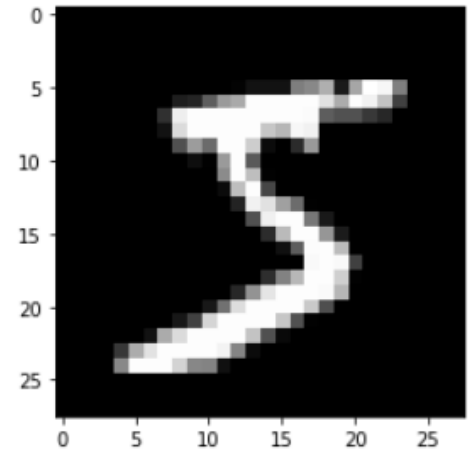
```
import matplotlib.pyplot as plt
plt.imshow(X[0].reshape(28,28), cmap='gray')
plt.show( )
print("이미지 레이블 : {}".format(y[0]))
```

```
import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=1/7, random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))
```

```
ds_train = TensorDataset(X_train, y_train)
ds_test = TensorDataset(X_test, y_test)
```

```
loader_train = DataLoader(ds_train, batch_size=64, shuffle=True)
loader_test = DataLoader(ds_test, batch_size=64, shuffle=False)
```



이미지 레이블 : 5

```
from torch import nn
model = nn.Sequential( )
model.add_module('fc1', nn.Linear(28*28*1, 100))    # 모델 구성
model.add_module('relu1', nn.ReLU())
model.add_module('fc2', nn.Linear(100,100))
model.add_module('relu2', nn.ReLU())
model.add_module('fc3', nn.Linear(100,10))
```

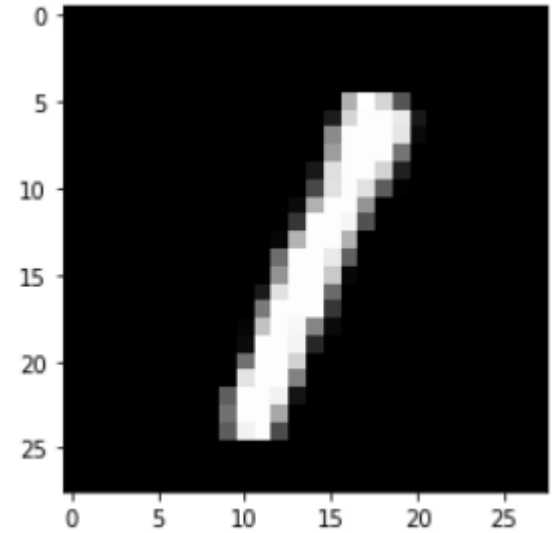
```
from torch import optim
loss_fn = nn.CrossEntropyLoss( ) # 손실 함수
optimizer = optim.Adam(model.parameters( ), lr=0.01)
```

```
def train(epoch):
    model.train( ) # 학습 모드로 변환
    for data, targets in loader_train:
        optimizer.zero_grad( ) # 그레디언트 초기화
        outputs = model(data)
        loss = loss_fn(outputs, targets)
        loss.backward( )
        optimizer.step( )
    print('에포크 {}: 완료'.format(epoch))
```

```
def test(head):
    model.eval( ) # 테스트 모드로 변환
    correct = 0
    with torch.no_grad( ):
        for data, targets in loader_test:
            outputs = model(data)
            _, predicted = torch.max(outputs.data, 1)
            correct += predicted.eq(targets.data.view_as(predicted)).sum()
    data_num = len(loader_test.dataset)
    print('{ } 정확도: { }/{ }({:.0f}%)'.format(head, correct, data_num, 100.*correct/data_num))
```

```
test('시작')
for epoch in range(3):
    train(epoch)
    test('학습중')
test('학습 후')
```

```
index = 10 # 테스트 데이터 중에서 확인해볼 데이터의 인덱스
model.eval() # 모델 테스트 모드로 전환
data = X_test[index]
output = model(data) # 모델 적용
print('{ } 번째 학습데이터의 테스트 결과 : {}'.format(index,output))
_, predicted = torch.max(output.data, 0)
print('{ }번째 데이터의 예측 : {}'.format(index, predicted))
X_test_show = (X_test[index]).numpy()
plt.imshow(X_test_show.reshape(28,28), cmap='gray')
print(' 실제 레이블: {}'.format(y_test[index]))
```



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

Spyder (Python 3.7)

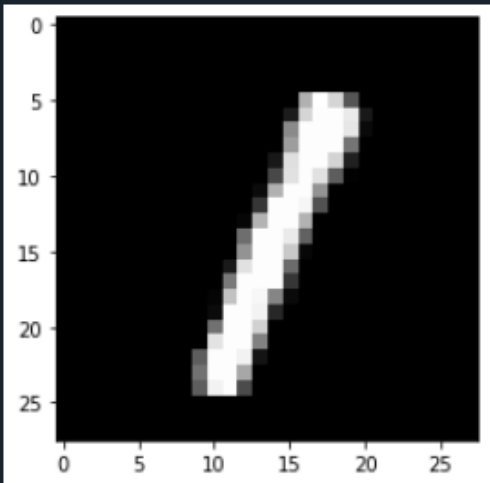
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Wkmlee\Google 드라이브\2020 Machine Learning\20-1.py

temp.py x 19-1.py x 19-2.py x 20-1.py x 20-2.py x

```
1  -*- coding: utf-8 -*-
2
3  from sklearn.datasets import fetch_openml
4  mnist = fetch_openml('mnist_784', version=1, cache=True)
5
6  X = mnist.data/255
7  y = mnist.target
8
9  import matplotlib.pyplot as plt
10 plt.imshow(X[0].reshape(28,28), cmap='gray')
11 plt.show()
12 print("이미지 레이블 : {}".format(y[0]))
13
14 import torch
15 from torch.utils.data import TensorDataset, DataLoader
16 from sklearn.model_selection import train_test_split
17
18 X_train, X_test, y_train, y_test = train_test_split(X,y, t
19 X_train = torch.Tensor(X_train)
20 X_test = torch.Tensor(X_test)
21 y_train = torch.LongTensor(list(map(int, y_train)))
22 y_test = torch.LongTensor(list(map(int, y_test)))
23
24 ds_train = TensorDataset(X_train, y_train)
25 ds_test = TensorDataset(X_test, y_test)
26
27 loader_train = DataLoader(ds_train, batch_size=64, shuffle
28 loader_test = DataLoader(ds_test, batch_size=64, shuffle=F
29
30 from torch import nn
31 model = nn.Sequential()
32 model.add_module('fc1', nn.Linear(28*28*1, 100))
```

107 %



Variable explorer Help Plots Files

Console 1/A x

```
에포크 0: 완료
학습중 정확도: 9470/10000(95%)
에포크 1: 완료
학습중 정확도: 9559/10000(96%)
에포크 2: 완료
학습중 정확도: 9540/10000(95%)
학습 후 정확도: 9540/10000(95%)
10 번째 학습데이터의 테스트 결과 : tensor([ -6.2667,  11.5346,
        -5.6269,  -5.8054,  -1.7754, -11.0771,  -3.6514,
```

History IPython console

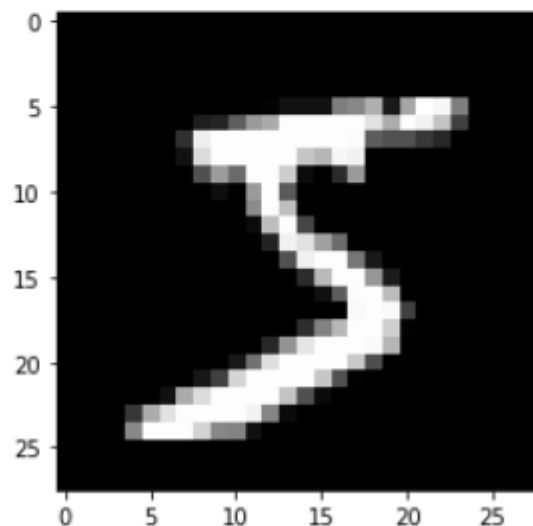
LSP Python: ready conda: PT (Python 3.7.7) Line 79, Col 1 UTF-8 CRLF RW Mem 44%

[실습] Colab에서 PyTorch 실행

```
[ ] 1 #-*- coding: utf-8 -*-
```

```
[ ] 1 from sklearn.datasets import fetch_openml  
2 mnist = fetch_openml('mnist_784', version=1, cache=True)  
3 X = mnist.data/255.0  
4 y = mnist.target
```

```
[ ] 1 import matplotlib.pyplot as plt  
2 plt.imshow(X[0].reshape(28,28), cmap='gray')  
3 plt.show()  
4 print('이미지 레이블 : {}'.format(y[0]))
```



이미지 레이블 : 5

```
[ ] 1 import torch
    2 from torch.utils.data import TensorDataset, DataLoader
    3 from sklearn.model_selection import train_test_split
    4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/7., random_state=0)
    5 X_train = torch.Tensor(X_train)
    6 X_test = torch.Tensor(X_test)
    7 y_train = torch.LongTensor(list(map(int, y_train)))
    8 y_test = torch.LongTensor(list(map(int, y_test)))
    9 ds_train = TensorDataset(X_train, y_train)
   10 ds_test = TensorDataset(X_test, y_test)
   11 loader_train = DataLoader(ds_train, batch_size=64, shuffle=True)
   12 loader_test = DataLoader(ds_test, batch_size=64, shuffle=False)
```

```
[7] 1 from torch import nn
    2 model = nn.Sequential()
    3 model.add_module('fc1', nn.Linear(28*28+1, 100))
    4 model.add_module('relu1', nn.ReLU())
    5 model.add_module('fc2', nn.Linear(100, 100))
    6 model.add_module('relu2', nn.ReLU())
    7 model.add_module('fc3', nn.Linear(100, 10))
```

```
[8] 1 from torch import optim
    2 loss_fn = nn.CrossEntropyLoss()
    3 optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```
[9] 1 def train(epoch):
    2     model.train()
    3     for data, targets in loader_train:
    4         optimizer.zero_grad()
    5         outputs = model(data)
    6         loss = loss_fn(outputs, targets)
    7         loss.backward()
    8         optimizer.step()
    9     print('epoch {}: 완료'.format(epoch))
```

```
[15] 1 def test(head):
    2     model.eval()
    3     correct = 0
    4     with torch.no_grad():
    5         for data, targets in loader_test:
    6             outputs = model(data)
    7             _, predicted = torch.max(outputs.data, 1)
    8             correct += predicted.eq(targets.data.view_as(predicted)).sum()
    9     data_num = len(loader_test.dataset)
   10     print('accuracy = ', 100.*correct/data_num)
```



```
1 for epoch in range(3):
    2     train(epoch)
    3     test('학습중')
```



```
epoch 0: 완료
accuracy = tensor(96.8700)
epoch 1: 완료
accuracy = tensor(96.6000)
epoch 2: 완료
accuracy = tensor(96.8600)
```


[실습] CNN 모델을 이용한 MNIST 데이터 분류

```
#-*- coding: utf-8 -*-
```

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, cache=True)
X = mnist.data
y = mnist.target
```

```
import torch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/7, random_state=0)
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.LongTensor(list(map(int, y_train)))
y_test = torch.LongTensor(list(map(int, y_test)))
```

```
import torch.nn as nn
import torch.nn.functional as F
from torch import optim
from torch.autograd import Variable
```

```
X_train = X_train.view(-1, 1,28,28).float()
X_test = X_test.view(-1,1,28,28).float()
print(X_train.shape)
print(X_test.shape)
```

```
train = TensorDataset(X_train, y_train)
test = TensorDataset(X_test, y_test)
BATCH_SIZE = 32
loader_train = DataLoader(train, batch_size = BATCH_SIZE, shuffle = False)
loader_test = DataLoader(test, batch_size = BATCH_SIZE, shuffle = False)
```

```

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=5)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=5)
        self.fc1 = nn.Linear(3*3*64, 256)
        self.fc2 = nn.Linear(256, 10)

        self.loss_fn = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.parameters(), lr=0.01)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv3(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = x.view(-1, 3*3*64)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

```

```

torch.nn.Conv2d(in_channels, out_channels, kernel_size,
                 stride=1, padding=0, dilation=1, groups=1,
                 bias=True, padding_mode='zeros')

```

```

def fit(model, loader_train):
    optimizer = torch.optim.Adam(model.parameters( ))
    error = nn.CrossEntropyLoss( )
    EPOCHS = 1
    model.train( )
    for epoch in range(EPOCHS):
        correct = 0
        for batch_idx, (X_batch, y_batch) in enumerate(loader_train):
            var_X_batch = Variable(X_batch).float( )
            var_y_batch = Variable(y_batch)
            optimizer.zero_grad( )
            output = model(var_X_batch)
            loss = error(output, var_y_batch)
            loss.backward( )
            optimizer.step( )
            predicted = torch.max(output.data, 1)[1]
            correct += (predicted == var_y_batch).sum( )
            if batch_idx % 50 == 0:
                print('에포크 : {} [{} / {}] ( {:.0f}%) Wt 손실함수 : {:.6f} Wt Accuracy: {:.3f}%'.format(
                    epoch, batch_idx*len(X_batch), len(loader_train),
                    100.*batch_idx / len(loader_train),
                    loss.data,
                    correct*100./ (BATCH_SIZE*(batch_idx+1))))

```

```

def evaluate(model):
    correct = 0
    for test_imgs, test_labels in loader_test:
        test_imgs = Variable(test_imgs).float()
        output = model(test_imgs)
        predicted = torch.max(output,1)[1]
        correct += (predicted == test_labels).sum()
    print("테스트 데이터 정확도: {:.3f}% ".format( float(correct) /
    (len(loader_test)*BATCH_SIZE)))

cnn = CNN( )
evaluate(cnn)
fit(cnn, loader_train)
cnn.eval( ) # 모델 테스트 모드로 전환
evaluate(cnn)

index = 10 # 테스트 데이터 중에서 확인해볼 데이터의 인덱스
data = X_test[index].view(-1, 1,28,28).float( )
output = cnn(data) # 모델 적용
print('{} 번째 학습데이터의 테스트 결과 : {}'.format(index, output))
_, predicted = torch.max(output, 1)
print('{}번째 데이터의 예측 : {}'.format(index, predicted.numpy()))
print('실제 레이블 : {}'.format(y_test[index]))

```

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.**