딥알못 탈출하기 #8

고려대학교 지능시스템 연구실
유민형 제작

Framework

- 신경망을 자유롭게 만들고 학습시키기 위한 툴을 Framework이라 한다. 여러 툴이 있지만 파이썬을 이용한 툴만 다루려고 한다.

- (문제1) TensorFlow, Pytorch, Keras, Jax, Theano, Caffe의 개발사와 각각의 역할 및 관계를 조사해서 설명하시오. (다른 것은 조사하지 말 것)

- 이번 회차에서는 TensorFlow와 Pytorch의 차이점을 배우고, Pytorch를 이용해 GPU를 다루는 방법을 알아볼 것이다.

12ymh12@gmail.com

Framework

- (문제2) TensorFlow와 Pytorch의 차이점 중 동적/정적 그래프에 대한 것을 설명하시오.

- (문제3) CPU 대신 GPU를 이용하면 연산 속도가 왜 빨라지는지 설명하시오.

12ymh12@gmail.com

Tensor

- (문제4) 실습을 해보기 전에 Tensor의 개념에 대해 찾아보아라.

12ymh12@gmail.com

Pytorch - Tensor 만들기

- torch.Tensor([]) 형태로 선언, []안에는 실제 값을 넣는다.
- 쉼표로 구분한다.

1차원

```
In [1]: import torch

In [2]: a = torch.Tensor([1,2,3])

In [3]: print(a)
tensor([1., 2., 3.])
```

2차원

```
In [4]: b = torch.Tensor([[1,2,3],[4,5,6]])

In [5]: print(b)
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

3차원

```
In [6]: c = torch.Tensor([[[1,2,3],[4,5,6]],[[7,8,9],[0,1,2]]])

In [7]: print(c)
tensor([[[1., 2., 3.],
         [4., 5., 6.]],

        [[7., 8., 9.],
         [0., 1., 2.]]])
```

4차원

```
In [8]: d = torch.Tensor([[[[1,2,3],[4,5,6]],[[7,8,9],[0,1,2]]],[[[3,4,5],[6,7,8]],[[9,0,1],[2,3,4]]]])

In [9]: print(d)
tensor([[[[1., 2., 3.],
          [4., 5., 6.]],

         [[7., 8., 9.],
          [0., 1., 2.]]],


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

         [[9., 0., 1.],
          [2., 3., 4.]]]])
```

12ymh12@gmail.com

Pytorch - 각 Tensor의 Dimension 확인하기

- torch.Tensor([]).dim(): 가장 상위 차원 리턴
- torch.Tensor([]).size(): 전체 차원을 텐서로 리턴
- torch.Tensor([]).shape: 전체 차원을 텐서로 리턴
- .size()는 함수고, .shape은 값이므로 ()유무에 차이

```
In [18]: a
Out[18]: tensor([1., 2., 3.])

In [19]: a.dim()
Out[19]: 1

In [20]: a.size()
Out[20]: torch.Size([3])

In [21]: a.shape
Out[21]: torch.Size([3])
```

```
In [22]: b
Out[22]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [23]: b.dim()
Out[23]: 2

In [24]: b.size()
Out[24]: torch.Size([2, 3])

In [25]: b.shape
Out[25]: torch.Size([2, 3])
```

```
In [26]: c
Out[26]:
tensor([[[1., 2., 3.],
         [4., 5., 6.]],

        [[7., 8., 9.],
         [0., 1., 2.]]])

In [27]: c.dim()
Out[27]: 3

In [28]: c.size()
Out[28]: torch.Size([2, 2, 3])

In [29]: c.shape
Out[29]: torch.Size([2, 2, 3])
```

```
In [30]: d
Out[30]:
tensor([[[[1., 2., 3.],
          [4., 5., 6.]],

         [[7., 8., 9.],
          [0., 1., 2.]]],


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

         [[9., 0., 1.],
          [2., 3., 4.]]]])

In [31]: d.dim()
Out[31]: 4

In [32]: d.size()
Out[32]: torch.Size([2, 2, 2, 3])

In [33]: d.shape
Out[33]: torch.Size([2, 2, 2, 3])
```

12ymh12@gmail.com

Pytorch – Tensor Indexing

- 텐서의 원하는 부분만 이용하고 싶을 때 인덱스를 이용해 호출
- 텐서 뒤에 []기호를 써서 원하는 인덱스를 넣는다.
- 텐서의 원하는 부분을 수정할 때에도 인덱싱을 활용한다.

```
In [41]: b
Out[41]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [42]: b[0]
Out[42]: tensor([1., 2., 3.])

In [43]: b[1]
Out[43]: tensor([4., 5., 6.])

In [44]: b[0] + b[1]
Out[44]: tensor([5., 7., 9.])

In [45]: b[0][0] + b[1][2]
Out[45]: tensor(7.)
```

```
In [62]: b
Out[62]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [63]: b[0] = torch.Tensor([7,8,9])

In [64]: b[1][2] = 0

In [65]: b
Out[65]:
tensor([[7., 8., 9.],
        [4., 5., 0.]])
```

12ymh12@gmail.com

Pytorch

- (문제5) RGB채널을 가진 넓이:20 높이:30 이미지를 배치 사이즈 128로 샘플링 했을 때, 한 미니배치 텐서의 (1) 최상위 차원은 얼마고, (2) 전체 차원은 어떻게 표시되는가?

- (문제6) 위의 미니배치 텐서의 전체 차원 중 3번째 차원의 값은 무엇이고, 호출하는 코드는 어떻게 되는가?

- (문제7) 위의 미니배치 텐서에서 5번째 이미지의 G채널의 텐서를 보고싶다면 인덱싱 코드는 어떻게 되는가?

12ymh12@gmail.com

Pytorch – Tensor type

- 텐서의 데이터 타입을 컨트롤 해보자.
- 딥러닝 학습시 Input 값은 Float이고, label은 Long인 경우가 많다. 타입 에러가 자주 발생하니 컨트롤 하는 방법을 알아놓자.
- .type()함수를 통해 텐서의 현재 타입을 출력하거나 바꿀 수 있다. 저장하려면 다시 선언해야한다.

```
In [50]: b
Out[50]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [51]: b.type()
Out[51]: 'torch.FloatTensor'

In [52]: b.type(torch.LongTensor)
Out[52]:
tensor([[1, 2, 3],
        [4, 5, 6]])

In [53]: b
Out[53]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

12ymh12@gmail.com

Pytorch – Tensor type

- .float()과 .long()함수를 이용해 곧바로 수정해줄 수도 있다. 저장하려면 다시 선언해야한다.
- 처음부터 float혹은 long타입을 지정해 텐서를 생성할 수도 있다. torch.LongTensor([])와 torch.FloatTensor([])이다. 대문자로 시작하는 것에 유의한다.

```
In [55]: b
Out[55]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [56]: b.long()
Out[56]:
tensor([[1, 2, 3],
        [4, 5, 6]])

In [57]: b
Out[57]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

```
In [58]: b = torch.LongTensor([[1,2,3],[4,5,6]])

In [59]: b
Out[59]:
tensor([[1, 2, 3],
        [4, 5, 6]])

In [60]: b = torch.FloatTensor([[1,2,3],[4,5,6]])

In [61]: b
Out[61]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

12ymh12@gmail.com

Pytorch – 원소 자동 생성 텐서

- 텐서 내부의 값을 하나하나 입력해주지 않고 바로 만들 수 있는 텐서들이 있다.
- 모든 원소가 1인 텐서, 모든 원소가 0인 텐서, Empty 텐서, Random 양수 텐서, random 실수 텐서가 있다.
- 각 텐서도 데이터 타입을 지정할 수 있다.
- 해당 텐서를 만들 때는 내부 값 대신 차원을 입력해준다.

```
In [66]: torch.ones(2,3)
Out[66]:
tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

```
In [67]: torch.zeros(2,3)
Out[67]:
tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

```
In [68]: torch.empty(2,3)
Out[68]:
tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

```
In [69]: torch.rand(2,3)
Out[69]:
tensor([[0.2879, 0.6676, 0.1459],
        [0.6222, 0.6755, 0.7974]])
```

```
In [70]: torch.randn(2,3)
Out[70]:
tensor([[-0.3549, -1.5333, -1.1704],
        [-2.3531,  1.6634,  0.1429]])
```

```
In [78]: torch.ones(2,3).long()
Out[78]:
tensor([[1, 1, 1],
        [1, 1, 1]])
```

12ymh12@gmail.com

Pytorch – 원소 자동 생성 텐서

- Empty 텐서의 용도: 큰 텐서를 미리 선언하고 인덱싱을 통해 값을 입력하는 것이, 생성된 여러 텐서를 합치는 것보다 메모리를 적게 차지한다. 메모리를 적게 사용하기 위해 필요하다. Empty 텐서 생성시 안에 구성되는 값은 현재 메모리에 남아있는 사용되지 않는 값들이 임의로 들어간다.
- torch.rand()와 torch.randn() 양수만 사용할 것인지 음수까지 사용할 것인지에 따라 용도가 다르다.

12ymh12@gmail.com

Pytorch - Tensor의 기본 연산

- 기본 사칙 연산은 어떻게 수행되는지 알아보자.
- 같은 차원을 가진 텐서는 파이썬 기본 연산 기호를 사용해도 되고, 원소끼리(element-wise)하게 연산이 수행된다.

```
In [93]: a1 = torch.Tensor([1,2,3])

In [94]: a2 = torch.Tensor([4,5,6])

In [95]: a1 + a2
Out[95]: tensor([5., 7., 9.])

In [96]: a1 - a2
Out[96]: tensor([-3., -3., -3.])

In [97]: a1 * a2
Out[97]: tensor([ 4., 10., 18.])

In [98]: a1 / a2
Out[98]: tensor([0.2500, 0.4000, 0.5000])
```

```
In [99]: b1 = torch.Tensor([[1,2,3],[4,5,6]])

In [100]: b2 = torch.Tensor([[7,8,9],[1,2,3]])

In [101]: b1 + b2
Out[101]:
tensor([[ 8., 10., 12.],
        [ 5.,  7.,  9.]])

In [102]: b1 - b2
Out[102]:
tensor([[-6., -6., -6.],
        [ 3.,  3.,  3.]])

In [103]: b1 * b2
Out[103]:
tensor([[ 7., 16., 27.],
        [ 4., 10., 18.]])

In [104]: b1 / b2
Out[104]:
tensor([[0.1429, 0.2500, 0.3333],
        [4.0000, 2.5000, 2.0000]])
```

```
In [105]: c1 = torch.Tensor([1,2])

In [106]: c2 = torch.Tensor([1,0])

In [107]: c1 / c2
Out[107]: tensor([1., inf])
```

12ymh12@gmail.com

Pytorch – Tensor 기본 연산

- 텐서의 차원이 일부만 같은 경우에는 가장 하위 차원이 동일한 경우에만 그 차원에 맞도록 연산이 수행된다.

```
In [110]: a1
Out[110]: tensor([1., 2., 3.])

In [111]: b1
Out[111]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [112]: a1 + b1
Out[112]:
tensor([[2., 4., 6.],
        [5., 7., 9.]])

In [113]: a1 - b1
Out[113]:
tensor([[ 0.,  0.,  0.],
        [-3., -3., -3.]])

In [114]: a1 * b1
Out[114]:
tensor([[ 1.,  4.,  9.],
        [ 4., 10., 18.]])

In [115]: a1 / b1
Out[115]:
tensor([[1.0000, 1.0000, 1.0000],
        [0.2500, 0.4000, 0.5000]])
```

```
In [119]: a1
Out[119]: tensor([1., 2., 3.])

In [120]: c1
Out[120]: tensor([1., 2.])

In [121]: a1 + c1
Traceback (most recent call last):

  File "<ipython-input-121-fe79aa3f0e07>", line 1, in <module>
    a1 + c1

RuntimeError: The size of tensor a (3) must match the size of tensor b (2) at non-singleton dimension 0
```

```
In [122]: b1
Out[122]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [123]: c1
Out[123]: tensor([1., 2.])

In [124]: b1 + c1
Traceback (most recent call last):

  File "<ipython-input-124-3be2a22683b1>", line 1, in <module>
    b1 + c1

RuntimeError: The size of tensor a (3) must match the size of tensor b (2) at non-singleton dimension 1
```

12ymh12@gmail.com

Pytorch –Tensor 기본 연산

- 스칼라 값과 텐서의 연산을 알아보자.
- Element-wise하게 연산이 수행된다.

```
In [125]: a = torch.Tensor([1,2,3])

In [126]: a + 5
Out[126]: tensor([6., 7., 8.])

In [127]: a - 5
Out[127]: tensor([-4., -3., -2.])

In [128]: a * 5
Out[128]: tensor([ 5., 10., 15.])

In [129]: a / 5
Out[129]: tensor([0.2000, 0.4000, 0.6000])
```

12ymh12@gmail.com

Pytorch – Tensor 기본 연산

- Pytorch에서 편의를 위해 제공하는 연산 함수도 있다.
- 내부 원소의 값을 더할 때, .sum()함수를 사용한다.
- 평균과 분산, 표준편차를 구할 수 있다. .mean(), .var(), .std()
- Minimum, maximum, absolute도 구할 수 있다.
- 위 모든 연산들은 특정 차원을 지정해서 수행할 수 있다.(dim=1)

```
In [137]: b
Out[137]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [138]: b.sum()
Out[138]: tensor(21.)

In [139]: b.sum(dim=0)
Out[139]: tensor([5., 7., 9.])

In [140]: b.sum(dim=1)
Out[140]: tensor([ 6., 15.])
```

```
In [150]: c
Out[150]:
tensor([[ 1.,  0., -1.],
        [-1.,  0.,  1.]])

In [151]: c.mean()
Out[151]: tensor(0.)

In [152]: c.var()
Out[152]: tensor(0.8000)

In [153]: c.std()
Out[153]: tensor(0.8944)
```

```
In [154]: b
Out[154]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [155]: b.min()
Out[155]: tensor(1.)

In [156]: b.max()
Out[156]: tensor(6.)

In [157]: b.min(dim=1)
Out[157]:
torch.return_types.min(
values=tensor([1., 4.]),
indices=tensor([0, 0]))
```

```
In [163]: c
Out[163]:
tensor([[ 1.,  0., -1.],
        [-1.,  0.,  1.]])

In [164]: c.abs()
Out[164]:
tensor([[1., 0., 1.],
        [1., 0., 1.]])
```

12ymh12@gmail.com

Pytorch – 특정 원소 찾기

- 텐서의 내부 원소 중 특정 조건을 만족하는 원소를 찾을 때 쓸 수 있는 테크닉을 알아보자.
- 우선 텐서에 대해 등호, 부등호를 사용하여 Boolean 텐서를 리턴할 수 있다. 0은 False, 1은 True를 의미한다.
- 등호, 부등호와 같은 조건을 인덱싱[]안에 넣어 이를 만족하는 원소를 얻어낼 수 있다.
- 등호, 부등호가 텐서를 리턴하기 때문에 여기에 다시 텐서에 대한 함수를 사용할 수 있다.
- .nonzero()나 .clamp()와 같은 함수를 활용해 해당 인덱스나 필요한 값을 찾아낼 수 있다.

```
In [173]: d
Out[173]:
tensor([[-2., -1.,  0.],
        [ 1.,  2.,  3.]])

In [174]: d >= 0
Out[174]:
tensor([[0, 0, 1],
        [1, 1, 1]], dtype=torch.uint8)
```

```
In [175]: d[d >= 0]
Out[175]: tensor([0., 1., 2., 3.])
```

```
In [176]: (d >= 0).nonzero()
Out[176]:
tensor([[0, 2],
        [1, 0],
        [1, 1],
        [1, 2]])
```

```
In [177]: d.clamp(min=0)
Out[177]:
tensor([[0., 0., 0.],
        [1., 2., 3.]])
```

12ymh12@gmail.com

Pytorch – 텐서의 행렬곱

- 텐서의 행렬곱(matrix product)을 하는 방법을 알아보자.
- 일반 곱하기 기호 * 로는 행렬곱이 되지 않는다.
- torch.mm()와 torch.matmul()을 이용하면 행렬곱을 수행한다.
- 생성된 텐서에 .mm()이나 .matmul()을 이용할 수도 있다.

```
In [180]: a1
Out[180]:
tensor([[1., 2.],
        [3., 4.]])

In [181]: a2
Out[181]:
tensor([[3., 4.],
        [1., 2.]])
```

```
In [182]: a1 * a2
Out[182]:
tensor([[3., 8.],
        [3., 8.]])
```

```
In [183]: torch.mm(a1,a2)
Out[183]:
tensor([[ 5.,  8.],
        [13., 20.]])

In [184]: torch.matmul(a1,a2)
Out[184]:
tensor([[ 5.,  8.],
        [13., 20.]])
```

```
In [185]: a1.mm(a2)
Out[185]:
tensor([[ 5.,  8.],
        [13., 20.]])

In [186]: a1.matmul(a2)
Out[186]:
tensor([[ 5.,  8.],
        [13., 20.]])
```

12ymh12@gmail.com

Pytorch – 텐서의 transpose, trace

- 행렬에 사용되는 연산인 transpose나 trace도 pytorch에서 제공한다.

```
In [187]: a1
Out[187]:
tensor([[1., 2.],
        [3., 4.]])

In [188]: a1.t()
Out[188]:
tensor([[1., 3.],
        [2., 4.]])

In [189]: a1.trace()
Out[189]: tensor(5.)
```

12ymh12@gmail.com

Pytorch – 텐서의 reshape

- 행렬의 차원을 조절하는 방법에 대해 알아보자.
- 행렬의 원소의 개수의 인수들을 이용하여 텐서의 차원을 자유자재로 조정할 수 있다. .reshape()함수나 .view()를 이용하고 () 안에 차원을 입력한다.

```
In [198]: b
Out[198]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [199]: b.reshape(6)
Out[199]: tensor([1., 2., 3., 4., 5., 6.])

In [200]: b.reshape(1,6)
Out[200]: tensor([[1., 2., 3., 4., 5., 6.]])

In [201]: b.reshape(1,1,6)
Out[201]: tensor([[[1., 2., 3., 4., 5., 6.]]])

In [202]: b.reshape(1,1,1,6)
Out[202]: tensor([[[[1., 2., 3., 4., 5., 6.]]]])

In [203]: b.reshape(3,2)
Out[203]:
tensor([[1., 2.],
        [3., 4.],
        [5., 6.]])

In [204]: b.reshape(1,3,2)
Out[204]:
tensor([[[1., 2.],
         [3., 4.],
         [5., 6.]]])
```

```
In [205]: b
Out[205]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [206]: b.view(6)
Out[206]: tensor([1., 2., 3., 4., 5., 6.])

In [207]: b.view(1,6)
Out[207]: tensor([[1., 2., 3., 4., 5., 6.]])

In [208]: b.view(1,1,6)
Out[208]: tensor([[[1., 2., 3., 4., 5., 6.]]])

In [209]: b.view(1,1,1,6)
Out[209]: tensor([[[[1., 2., 3., 4., 5., 6.]]]])

In [210]: b.view(3,2)
Out[210]:
tensor([[1., 2.],
        [3., 4.],
        [5., 6.]])

In [211]: b.view(1,3,2)
Out[211]:
tensor([[[1., 2.],
         [3., 4.],
         [5., 6.]]])
```

12ymh12@gmail.com

Pytorch – 텐서의 concatenate, stack

- 여러 이미지를 한 미니배치 안에 새롭게 집어넣는 경우가 있다.
- 차원에 맞게 미니배치 텐서에 넣는 방법을 알아보자.
- 텐서를 붙이는 것(concatenate)은 torch.cat()함수를 이용한다.
- 텐서를 쌓는 것(stack)은 torch.stack()함수를 이용한다.
- dim옵션을 통해 어떤 차원을 기준으로 붙이거나 쌓을지 결정할 수 있다.

```
In [228]: a1
Out[228]: tensor([1., 2., 3.])

In [229]: a2
Out[229]: tensor([4., 5., 6.])

In [230]: torch.cat((a1,a2))
Out[230]: tensor([1., 2., 3., 4., 5., 6.])

In [231]: torch.stack((a1,a2), dim=0)
Out[231]:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

In [232]: torch.stack((a1,a2), dim=1)
Out[232]:
tensor([[1., 4.],
        [2., 5.],
        [3., 6.]])
```

12ymh12@gmail.com

Pytorch – 텐서의 concatenate, stack

- (문제8) RGB채널을 가진 넓이:20 높이:30 이미지가 현재 127개로 미니배치가 구성되어 있다. 새롭게 얻은 같은 사이즈의 이미지 하나를 이 미니배치에 넣어 128개의 배치 사이즈를 맞추려면, 어떤 차원을 기준으로 concatenate 혹은 stack 해야하는가? 또 그 코드는 어떻게 되는가?

12ymh12@gmail.com

Pytorch – CPU/GPU 전환

- 텐서는 기본적으로 cpu에서 만들어진다. Gpu에서 이용할 수 있는 방법을 알아보자.
- 우선 gpu의 이용가능한 여부를 알아보는 코드를 보자.
- torch.cuda.is_available()함수를 통해 gpu 이용 가능 여부를 리턴할 수 있다. T/F값을 리턴한다.
- torch.cuda.device_count()함수를 통해 현재 컴퓨터가 이용할 수 있는 gpu의 개수가 나온다.

```
In [234]: print(torch.cuda.is_available())
True

In [235]: print(torch.cuda.device_count())
1
```

12ymh12@gmail.com

Pytorch – CPU/GPU 전환

- 이제 텐서를 cpu와 gpu로 전환하는 법을 알아보자.
- 기본적으로 텐서는 cpu 상에 생성된다.
- 두 가지 방법으로 gpu로 전환할 수 있다. .to()와 .cuda()이다.
- 앞서나온 torch.cuda.is_available() 함수를 이용하여 컴퓨터 사양에 맞게 자동적으로 cpu/gpu를 지정하는 방식이 general한 방법이다. 따라서 .to()를 주로 사용한다.
- Gpu 이름은 “cuda:0”가 default이다.

```
In [240]: a = torch.Tensor([1,2,3])

In [241]: a.device
Out[241]: device(type='cpu')
```

```
In [242]: device = "cuda:0" if torch.cuda.is_available() else "cpu"

In [243]: device
Out[243]: 'cuda:0'

In [244]: a.to(device)
Out[244]: tensor([1., 2., 3.], device='cuda:0')
```

```
In [245]: a.cuda()
Out[245]: tensor([1., 2., 3.], device='cuda:0')
```

12ymh12@gmail.com

Pytorch – CPU/GPU 전환

- 반대로 gpu에 있는 텐서를 cpu로 불러와야하는 경우도 있다. 예를 들어, GAN을 통해 생성된 이미지는 현재 숫자로 구성된 텐서인데 이를 matplotlib.pyplot 라이브러리를 이용해 plot 하려면 list 타입으로 변경해야한다. List로 전환하기 전에 gpu에서 cpu로 우선적으로 불러와야 한다.
- Cpu로 전환하는 방법은 역시 두 가지다.

```
In [247]: a
Out[247]: tensor([1., 2., 3.], device='cuda:0')

In [248]: a.to("cpu")
Out[248]: tensor([1., 2., 3.])

In [249]: a.cpu()
Out[249]: tensor([1., 2., 3.])
```

12ymh12@gmail.com

Pytorch – Tensor/numpy array 전환

- Pytorch tensor와 numpy array는 전환이 상당히 자유로운 편이다. Pytorch에서 제공하지 않는 기능이 numpy에 있다면 numpy array로 전환해서 기능을 수행한 후 다시 pytorch tensor로 전환하는 테크닉이 자주 사용된다.
- Pytorch tensor에서 Numpy array로: .numpy() 함수

```
In [254]: a
Out[254]: tensor([1., 2., 3.])

In [255]: a.numpy()
Out[255]: array([1., 2., 3.], dtype=float32)
```

- Numpy array에서 Pytorch tensor로: torch.from_numpy() 함수

```
In [257]: b
Out[257]: array([1, 2, 3])

In [258]: torch.from_numpy(b)
Out[258]: tensor([1, 2, 3], dtype=torch.int32)
```

12ymh12@gmail.com

Pytorch - Gradient가 흐르는 텐서

- 딥러닝 학습을 위해서 loss에서 계산된 gradient가 텐서에 흐르도록 할 수 있다.
- 두 가지 방법이 있다. torch.tensor()를 이용해 requires_grad 옵션에 True를 주는 방법이 있고, torch.Tensor()를 이용해 텐서를 생성한 후 .requires_grad에 True를 할당해주는 방법이 있다. 대소문자에 주의한다.

```
In [274]: a = torch.tensor([1,2,3], dtype=torch.float, requires_grad=True)

In [275]: a
Out[275]: tensor([1., 2., 3.], requires_grad=True)
```

```
In [276]: a = torch.Tensor([1,2,3])

In [277]: a.requires_grad = True

In [278]: a
Out[278]: tensor([1., 2., 3.], requires_grad=True)
```

12ymh12@gmail.com

Pytorch – Gradient가 흐르는 텐서

- 이제 backward함수를 통해 gradient를 구해보자.
- x = 3이고, y = x^2의 함수에서 y를 x에 대해서 미분하고 x=3일 때 gradient를 구해보자.
- x = 3을 gradient가 흐르는 텐서로 생성하고, y = x ** 2라는 수식을 정의해주고, y에 .backward()함수를 사용한다.
- 텐서 x에 .grad.data에 gradient값 6이 저장된 것을 확인할 수 있다.

```
In [286]: x = torch.tensor([3], dtype=torch.float, requires_grad=True)

In [287]: x
Out[287]: tensor([3.], requires_grad=True)

In [288]: y = x ** 2

In [289]: y.backward()

In [290]: x.grad.data
Out[290]: tensor([6.])
```

12ymh12@gmail.com

Pytorch – 텐서의 복사

- 텐서를 복사할 때는 .clone()함수를 이용한다. 그렇지 않으면 원래 텐서의 값을 변경하면 복사한 텐서의 값도 바뀐다.
- Gradient가 흐르는 텐서의 경우 gradient가 흐르는 것을 중지하지 않는 이상 numpy array로 옮길 수 없다. .detach() 함수를 이용해 gradient를 없애줄 수 있다.

```
In [299]: a1
Out[299]: tensor([1., 2., 3.])

In [300]: a2 = a1

In [301]: a3 = a1.clone()

In [302]: a1[0] = 4

In [303]: a1
Out[303]: tensor([4., 2., 3.])

In [304]: a2
Out[304]: tensor([4., 2., 3.])

In [305]: a3
Out[305]: tensor([1., 2., 3.])
```

```
In [291]: x
Out[291]: tensor([3.], requires_grad=True)

In [292]: x.detach()
Out[292]: tensor([3.])
```

12ymh12@gmail.com

Pytorch – 텐서의 복사

- (문제9)

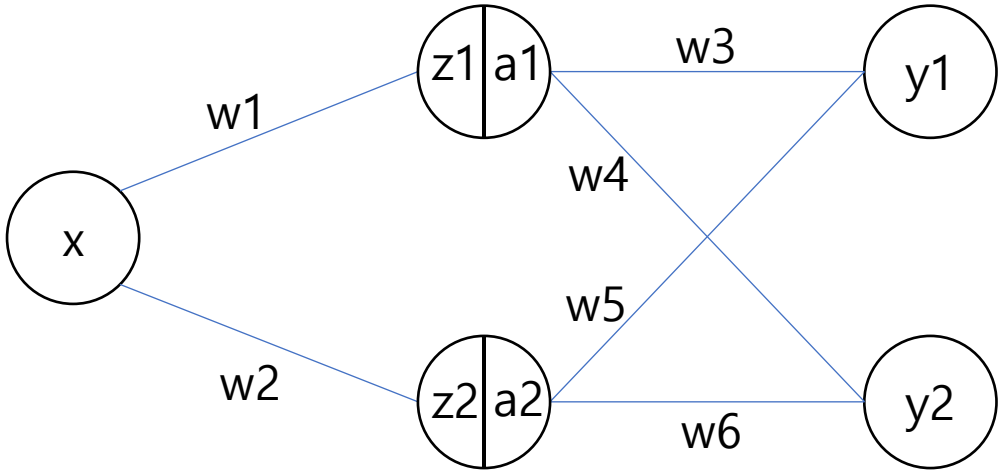
```
In [308]: x = torch.tensor([3], dtype=torch.float, requires_grad=True).to("cuda:0")

In [309]: x
Out[309]: tensor([3.], device='cuda:0', grad_fn=<CopyBackwards>)
```

- 위 텐서 x를 gradient를 없애고, 복사해서, cpu로 옮긴 후, numpy array로 옮겨서, list로 만드는 코드를 작성하여라.

12ymh12@gmail.com

Pytorch

- (문제10) 2장에서 나왔던 numpy로 구현했던 예제를 신경망을 Pytorch Tensor로 구현해보자. (1) Cpu텐서/Gpu텐서를 사용해 forward /backward를 행렬곱으로 구현하고, 각각을 timeit 라이브러리를 이용해 시간 차이를 구해라. (2) 행렬곱이 아닌 backward()함수를 이용해서 자동 미분 값으로 weight를 업데이트하는 코드를 작성하시오.



$$f(x) = \begin{cases} x \text{ if } x>0 \\ 0 \text{ else} \end{cases}$$

z1 = w1*x+b1
z2 = w2*x+b2
a1 = f(z1)
a2 = f(z2)
y1 = w3*a1+w5*a2
y2 = w4*a1+w6*a2

Weight 초깃값
w1 = 0.1
w2 = -0.2
w3 = 0.4
w4 = -0.3
w5 = -0.15
w6 = 0.12
b1 = 0.05
b2 = -0.03

데이터쌍 (x, y1, y2)
: (0.5, 1,-1)

12ymh12@gmail.com