

파이토치로 배우는
자연어 처리

1장. PyTorch

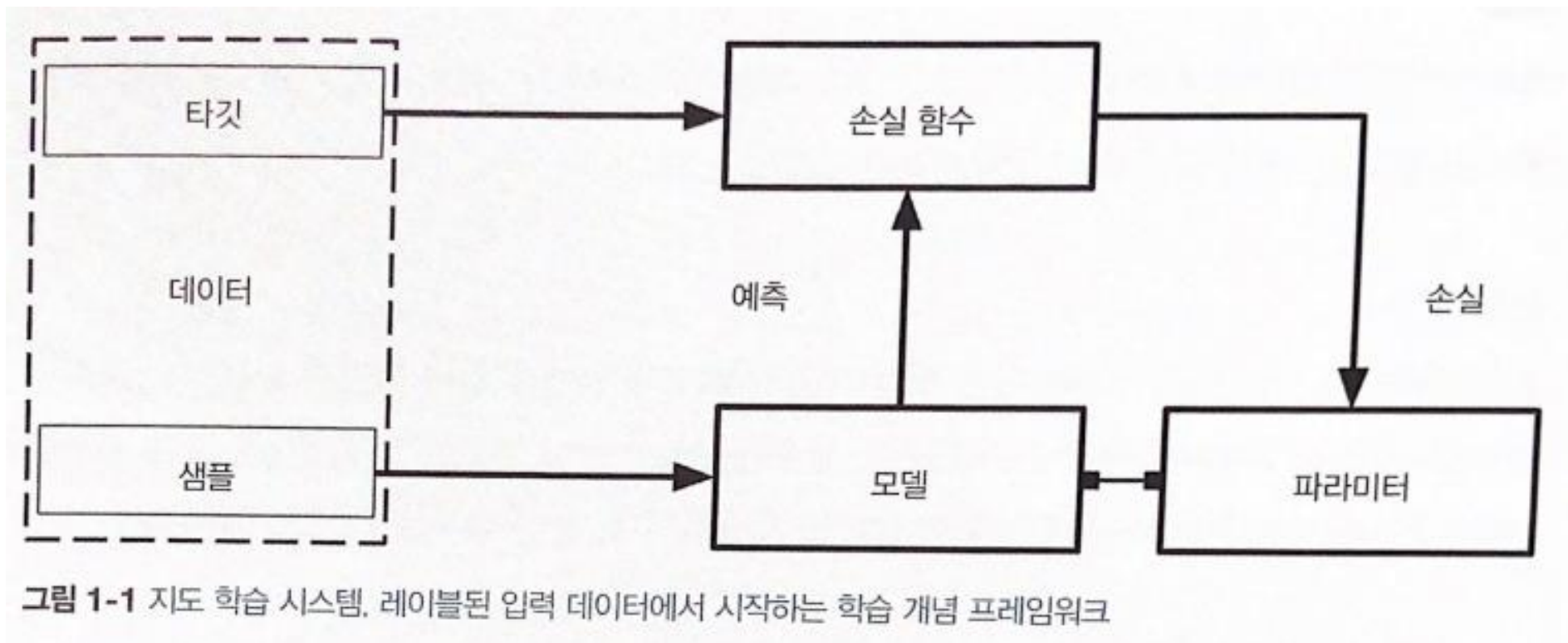
송선영

CONTENTS

1. 지도 학습
2. 샘플과 타겟의 인코딩
3. 계산 그래프
4. pytorch
5. 연습문제 풀이

지도 학습이란?

- 샘플에 대응하는 타겟(예측 값)의 정답을 제공하는 방식
- 목적 : 주어진 데이터셋에서 손실 함수를 최소화하는 파라미터 값 고르기 -> 경사 하강법 사용



02 샘플과 타겟의 인코딩

- 원-핫 표현

Time flies like an arrow.
Fruit flies like a banana.

토큰화 (구두점 무시)

{ time, fruit, flies, like, a, an, arrow, banana }

	time	fruit	flies	like	a	an	arrow	banana
1 _{time}	1	0	0	0	0	0	0	0
1 _{fruit}	0	1	0	0	0	0	0	0
1 _{flies}	0	0	1	0	0	0	0	0
1 _{like}	0	0	0	1	0	0	0	0
1 _a	0	0	0	0	1	0	0	0
1 _{an}	0	0	0	0	0	1	0	0
1 _{arrow}	0	0	0	0	0	0	1	0
1 _{banana}	0	0	0	0	0	0	0	1

- 'like a banana' 의 원-핫 표현은?

	time	fruit	flies	like	a	an	arrow	banana
1 _{like}	0	0	0	1	0	0	0	0
1 _a	0	0	0	0	1	0	0	0
1 _{banana}	0	0	0	0	0	0	0	1

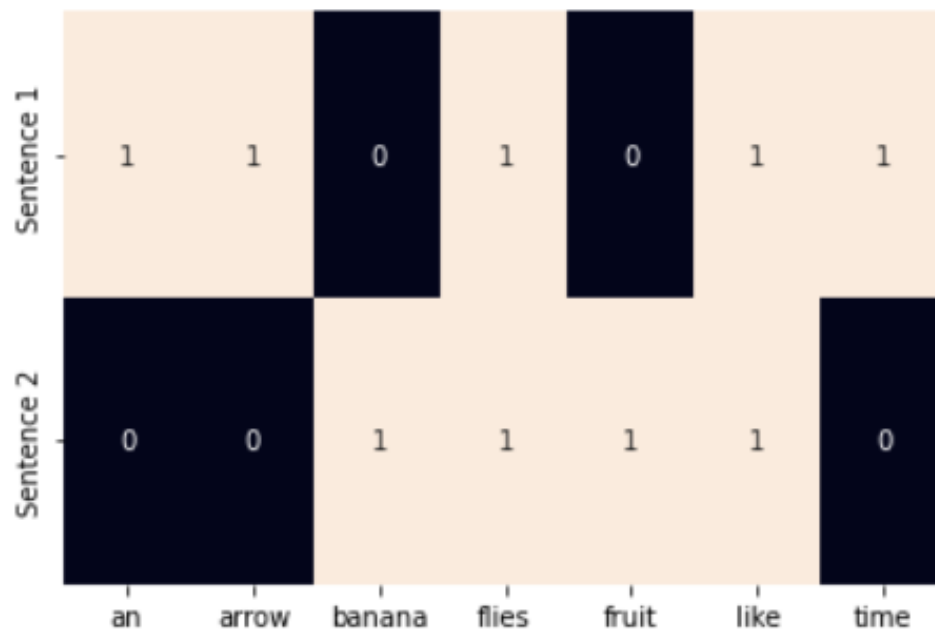
→ 3X8 크기의 행렬,
→ 이진 인코딩은 [0,0,0,1,1,0,0,1]

02 샘플과 타깃의 인코딩

- 원-핫 벡터 또는 이진 표현 만들기

```
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns

corpus = ['Time flies like an arrow.',
          'Fruit flies like a banana.']
one_hot_vectorizer = CountVectorizer(binary=True)
one_hot = one_hot_vectorizer.fit_transform(corpus).toarray()
vocab = one_hot_vectorizer.get_feature_names()
sns.heatmap(one_hot, annot=True,
            cbar=False, xticklabels=vocab,
            yticklabels=['Sentence 1', 'Sentence 2'])
```



- TF (Term-Frequency : 문서 빈도)
- 'Fruit flies like time flies a fruit'의 TF 표현

time	fruit	flies	like	a	an	arrow	banana
1	2	2	1	1	0	0	0

→ 단어 fruit의 TF : $TF(\text{fruit}) = 2$

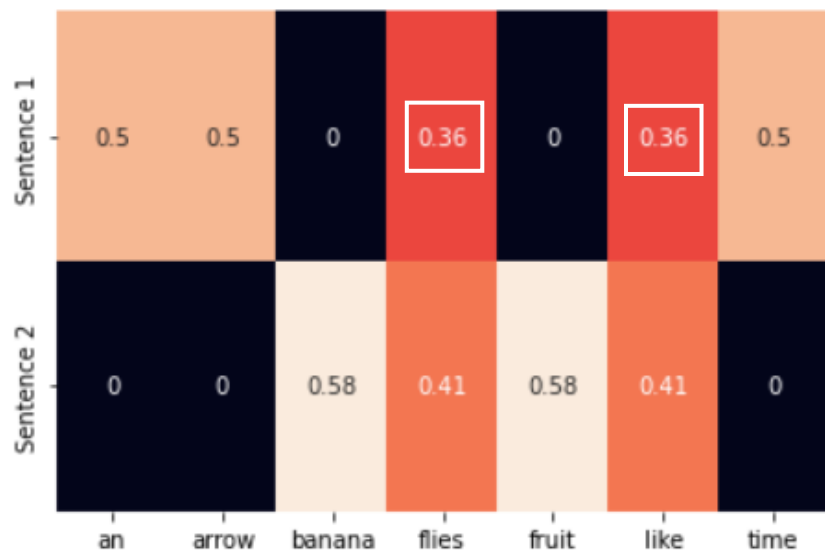
- TF-IDF (Term-Frequency-Inverse-Document-Frequency : 문서 빈도 - 역문서 빈도)

$$IDF(w) = \log \frac{N}{n_w} \quad \rightarrow \quad \text{TF-IDF 점수는 } TF(w) * IDF(w)$$

- TF-IDF 표현 만들기

```
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns

tfidf_vectorizer = TfidfVectorizer()
tfidf = tfidf_vectorizer.fit_transform(corpus).toarray()
sns.heatmap(tfidf, annot=True, cbar=False, xticklabels=vocab,
            yticklabels = ['Sentence 1', 'Sentence 2'])
```



- Sentence 1

'Time flies like an arrow.'
'Fruit flies like a banana.'

* tfidfVectorizer 에서는 $IDF(w) = \log\left(\frac{N+1}{n_w+1}\right) + 1$ 로 사용

- flies, like의 TF-IDF 값은?

$$TF = 1, IDF = \log\left(\frac{2+1}{2+1}\right) + 1 = 1$$

$$TF-IDF = TF * IDF = 1 * 1 = 1$$

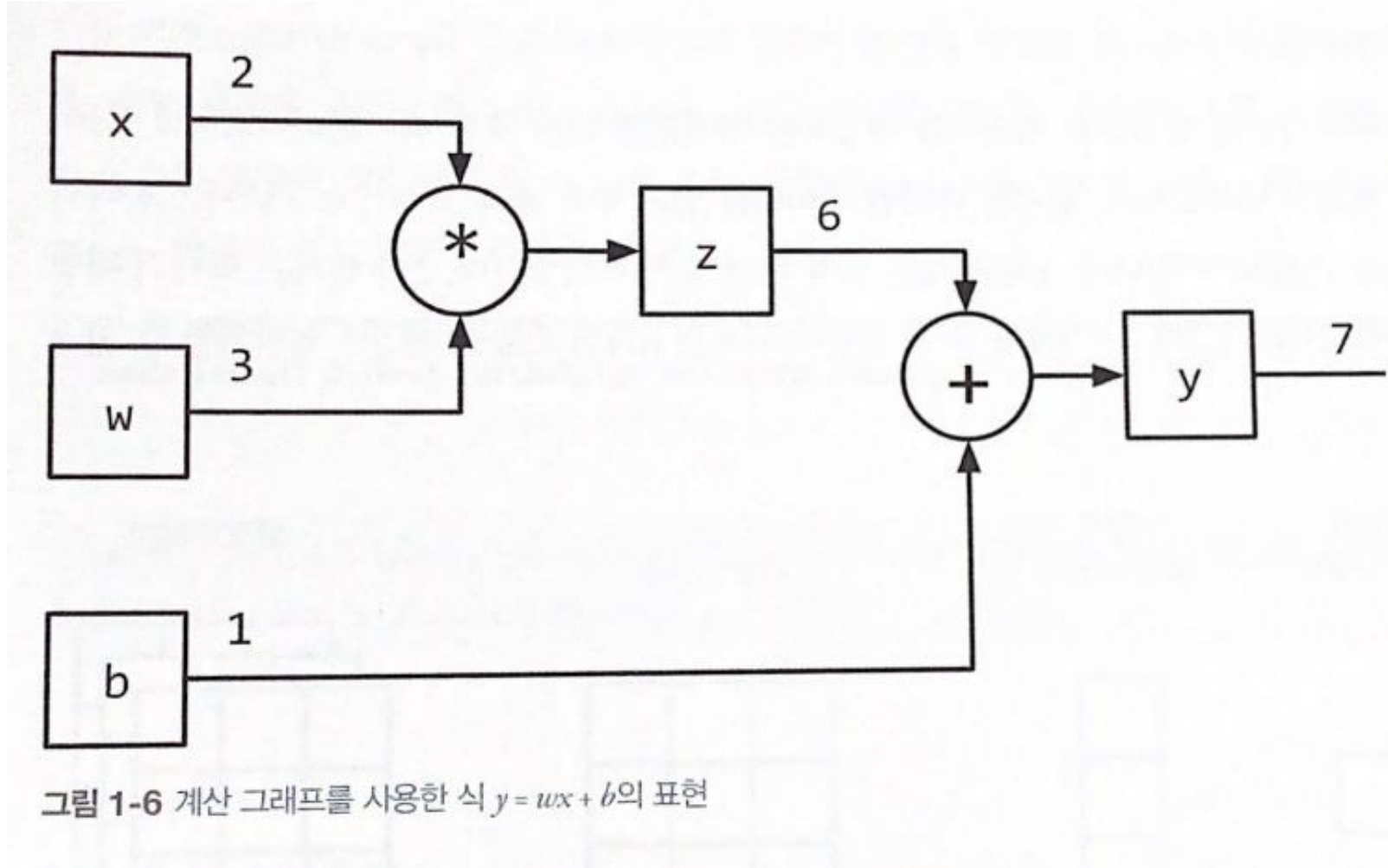
- 단어 an, arrow, time 의 TF-IDF 값은?

$$TF = 1, IDF = \log\left(\frac{2+1}{1+1}\right) + 1 = 1.405$$

$$TF-IDF = TF * IDF = 1 * 1.405 = 1.405$$

- flies, like의 L2 Norm 정규화 값은?

$$\frac{1}{\sqrt{2 * 1^2 + 3 * 1.405^2}} = 0.3552$$



- 텐서 만들기

```
# 텐서의 속성 출력함수
def describe(x):
    print("타입: {}".format(x.type()))
    print("크기: {}".format(x.shape))
    print("값: \n{}".format(x))
```

1) `import torch`

```
# 차원을 2x3으로 초기화, 값은 랜덤하게
describe(torch.Tensor(2,3))
```

```
타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[ -3.2454e-33,  3.0684e-41,  1.5695e-43],
        [ 1.5554e-43,  1.5975e-43,  1.6255e-43]])
```

2) `describe(torch.rand(2,3))` # [0,1) 범위에서 랜덤하게
`describe(torch.randn(2,3))` # 표준 정규 분포값으로 랜덤하게

```
타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[0.6511, 0.6530, 0.9024],
        [0.8751, 0.5086, 0.0415]])
타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[ -1.4694,  0.1454, -0.1947],
        [ 0.5183, -0.6032, -0.2123]])
```


3) `describe(torch.zeros(2,3))` # 0으로 채운 텐서

```
x = torch.ones(2,3) # 1로 채운 텐서
describe(x)
```

```
x.fill_(5) # 5로 채운 텐서
describe(x)
```

타입: torch.FloatTensor
크기: torch.Size([2, 3])

값:
tensor([[0., 0., 0.],
 [0., 0., 0.]])

타입: torch.FloatTensor
크기: torch.Size([2, 3])

값:
tensor([[1., 1., 1.],
 [1., 1., 1.]])

타입: torch.FloatTensor
크기: torch.Size([2, 3])

값:
tensor([[5., 5., 5.],
 [5., 5., 5.]])

4) `x = torch.Tensor([[1,2,3],
 [4,5,6]])`
`describe(x)`

타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[1., 2., 3.],
 [4., 5., 6.]])

5) `import numpy as np`

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

```
# numpy로 텐서 만들기
# numpy 배열은 float64 타입이기 때문에 tensor type은 DoubleTensor
describe(torch.from_numpy(np))
```

타입: torch.DoubleTensor
크기: torch.Size([2, 3])
값:
tensor([[0.7036, 0.6288, 0.0352],
 [0.3571, 0.3298, 0.9034]], dtype=torch.float64)

- 텐서 타입

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

타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[1., 2., 3.],
[4., 5., 6.]])

2) `x = torch.LongTensor([[1,2,3],
[4,5,6]])`
`describe(x)`

타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
tensor([[1, 2, 3],
[4, 5, 6]])

3) `x = torch.tensor([[1,2,3],
[4,5,6]], dtype=torch.int64)`
`describe(x)`

타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
tensor([[1, 2, 3],
[4, 5, 6]])

4) `x = x.long()`
`describe(x)`

타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
tensor([[1, 2, 3],
[4, 5, 6]])

5) `x = x.float()`
`describe(x)`

타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[1., 2., 3.],
[4., 5., 6.]])

- 텐서 연산

```
x = torch.randn(2,3)
describe(x)
```

타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[0.3957, -1.3422, -1.7639],
 [0.4326, -1.9294, 0.6513]])

1) describe(torch.add(x,x))

타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[0.7914, -2.6844, -3.5277],
 [0.8653, -3.8588, 1.3026]])

2) describe(x + x)

타입: torch.FloatTensor
크기: torch.Size([2, 3])
값:
tensor([[0.7914, -2.6844, -3.5277],
 [0.8653, -3.8588, 1.3026]])

- 차원별 텐서 연산

```
x = torch.arange(6) # 0~5까지 1씩 증가하는 텐서 생성
describe(x)
```

타입: torch.LongTensor
크기: torch.Size([6])
값:
tensor([0, 1, 2, 3, 4, 5])

1) `# 동일한 데이터를 공유하는 텐서를 2x3 크기로 생성`
`x = x.view(2,3)`
`describe(x)`

타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
tensor([[0, 1, 2],
 [3, 4, 5]])

2) `describe(torch.sum(x, dim=0)) # 행`

타입: torch.LongTensor
크기: torch.Size([3])
값:
tensor([3, 5, 7])

3) `describe(torch.sum(x, dim=1)) # 열`

타입: torch.LongTensor
크기: torch.Size([2])
값:
tensor([3, 12])

4) `describe(torch.transpose(x,0,1))`

타입: torch.LongTensor
크기: torch.Size([3, 2])
값:
tensor([[0, 3],
 [1, 4],
 [2, 5]])

- 텐서 인덱싱

```
타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
x: tensor([[0, 1, 2],
          [3, 4, 5]])
```

1) `describe(x[:, 1, :2])`

```
타입: torch.LongTensor
크기: torch.Size([1, 2])
값:
tensor([[0, 1]])
```

2) `describe(x[0, 1])`

```
타입: torch.LongTensor
크기: torch.Size([1])
값:
1
```

3) `indices = torch.LongTensor([0, 2])`
`describe(torch.index_select(x, dim=1, index=indices))`

```
타입: torch.LongTensor
크기: torch.Size([2, 2])
값:
tensor([[0, 2],
        [3, 5]])
```

→ `x[:, [0, 2]]` 로 표현할 수 있음
 (텐서는 배열 인덱싱 가능)

4) `indices = torch.LongTensor([0, 0])`
`describe(torch.index_select(x, dim=0, index=indices))`

```
타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
tensor([[0, 1, 2],
        [0, 1, 2]])
```

→ `x[[0, 0], :]` 로 표현할 수 있음

5) `row_indices = torch.arange(2).long()`
`col_indices = torch.LongTensor([0, 1])`
`describe(x[row_indices, col_indices])`

```
타입: torch.LongTensor
크기: torch.Size([2])
값:
tensor([0, 4])
```

- 텐서 연결

```
타입: torch.LongTensor  
크기: torch.Size([2, 3])  
값:  
tensor([[0, 1, 2],  
        [3, 4, 5]])
```

1) `describe(torch.cat([x, x], dim=0))`

```
타입: torch.LongTensor  
크기: torch.Size([4, 3])  
값:  
tensor([[0, 1, 2],  
        [3, 4, 5],  
        [0, 1, 2],  
        [3, 4, 5]])
```

2) `describe(torch.cat([x, x], dim=1))`

```
타입: torch.LongTensor  
크기: torch.Size([2, 6])  
값:  
tensor([[0, 1, 2, 0, 1, 2],  
        [3, 4, 5, 3, 4, 5]])
```

3) `describe(torch.stack([x, x]))`

```
타입: torch.LongTensor  
크기: torch.Size([2, 2, 3])  
값:  
tensor([[[0, 1, 2],  
         [3, 4, 5]],  
        [[0, 1, 2],  
         [3, 4, 5]])
```

- 텐서 선형 대수 계산: 행렬 곱셈

1)

```
x1 = torch.arange(6).view(2, 3)
describe(x1)
```

타입: torch.LongTensor
크기: torch.Size([2, 3])
값:
tensor([[0, 1, 2],
 [3, 4, 5]])

2)

```
x2 = torch.ones(3, 2)
x2[:, 1] += 1
describe(x2)
```

타입: torch.FloatTensor
크기: torch.Size([3, 2])
값:
tensor([[1., 2.],
 [1., 2.],
 [1., 2.]])

3)

```
# 행렬 곱
describe(torch.mm(x1, x2))
```

타입: torch.FloatTensor
크기: torch.Size([2, 2])
값:
tensor([[3., 6.],
 [12., 24.]])

- 그래디언트 연산을 할 수 있는 텐서 만들기

1)

```
# requires_grad=True : gradient기반 학습에 필요한 손실함수와
#           텐서의 gradient를 기록하는 부가 연산을 활성화시킴
x = torch.ones(2,2,requires_grad=True)
describe(x)
print(x.grad is None)
```

타입: torch.FloatTensor
크기: torch.Size([2, 2])
값:
tensor([[1., 1.],
 [1., 1.]], requires_grad=True)
True

2)

```
y = (x + 2) * (x + 5) + 3
describe(y)
print(x.grad is None)
```

타입: torch.FloatTensor
크기: torch.Size([2, 2])
값:
tensor([[21., 21.],
 [21., 21.]], grad_fn=<AddBackward0>)
True

3)

```
z = y.mean()
describe(z)
z.backward() # 역방향 계산
print(x.grad is None) # 파라미터 값 업데이트
```

타입: torch.FloatTensor
크기: torch.Size([1])
값:
21.0
False

- CUDA 텐서 만들기

1) `import torch`

```
# GPU를 사용할 수 있는지 확인  
print(torch.cuda.is_available())
```

True

2) `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`
`print(device)`

cuda

3) `# .to(device) : 초기화되는 모든 텐서를 device로 이동`
`x = torch.rand(3,3).to(device)`
`describe(x)`

타입: `torch.cuda.FloatTensor`

크기: `torch.Size([3, 3])`

값:

```
tensor([[0.2583, 0.7546, 0.5726],  
        [0.6367, 0.9254, 0.2455],  
        [0.6666, 0.2728, 0.5424]], device='cuda:0')
```

1. 2D 텐서를 만들고 차원0 위치에 크기가 1인 차원을 추가하세요.

```
x = torch.rand(3,3)
x.unsqueeze(0) # unsqueeze : 차원 추가
```

```
tensor([[[[0.8014, 0.0514, 0.9397],
          [0.9315, 0.2283, 0.2760],
          [0.4364, 0.6279, 0.5328]]]])
```

2. 이전 텐서에 추가한 차원을 삭제하세요.

```
x.squeeze(0) # squeeze : 차원 삭제
```

```
tensor([[0.8014, 0.0514, 0.9397],
        [0.9315, 0.2283, 0.2760],
        [0.4364, 0.6279, 0.5328]])
```

3. 범위가 [3,7) 이고 크기가 5x3인 랜덤한 텐서를 만드세요.

```
3 + torch.rand(5,3) * (7 - 3)
```

```
tensor([[6.9068, 4.8757, 4.5152],
        [4.6034, 3.7457, 3.3121],
        [6.3022, 6.9982, 4.1004],
        [3.4428, 3.0758, 3.8916],
        [4.4835, 4.2571, 4.9507]])
```

4. 정규 분포(평균=0, 표준편차=1)를 사용해 텐서를 만드세요.

```
y = torch.rand(3,3)
y.normal_()

tensor([[ 0.8542, -0.3467,  0.1965],
        [ 0.3784, -1.5019,  0.9928],
        [ 1.4040, -1.5264, -1.1240]])
```

5. 텐서 `torch.Tensor([1,1,1,0,1])`에서 0이 아닌 원소의 인덱스를 추출하세요.

```
z = torch.Tensor([1,1,1,0,1])
torch.nonzero(z) # 0이 아닌 원소의 인덱스를 각 행에 담은 2차원 텐서 반환

tensor([[0],
        [1],
        [2],
        [4]])
```

6. 크기가 (3,1)인 랜덤한 텐서를 만들고 네 번을 복사해 쌓으세요.

```
a = torch.rand(3,1)
a.expand(3,4) # expand : 차원을 지정한 크기로 늘림

tensor([[ 0.5165,  0.5165,  0.5165,  0.5165],
        [ 0.6917,  0.6917,  0.6917,  0.6917],
        [ 0.6468,  0.6468,  0.6468,  0.6468]])
```

7. 2차원 행렬 두 개 ($a=\text{torch.rand}(3,4,5)$, $b=\text{torch.rand}(3,5,4)$)의 배치 행렬 곱셈을 계산하세요.

```
a = torch.rand(3,4,5)
b = torch.rand(3,5,4)
torch.bmm(a,b)

tensor([[[[1.0386, 1.4453, 1.7456, 1.3421],
          [0.4723, 0.9527, 0.8589, 0.9381],
          [1.0421, 1.6625, 1.3155, 1.0276],
          [0.9465, 1.5191, 1.2271, 1.1003]],

        [[1.1134, 2.3441, 1.6213, 1.3923],
          [1.1273, 2.0009, 1.0427, 0.9958],
          [1.0874, 2.1094, 1.1745, 1.0444],
          [0.5968, 1.1984, 0.9247, 0.7648]],

        [[1.1146, 1.2656, 0.7895, 1.1093],
          [0.9750, 1.5759, 0.5496, 1.0222],
          [1.2607, 2.2045, 1.2941, 1.6045],
          [1.2394, 1.6743, 0.8356, 1.2604]]]])
```

8. 3차원 행렬 ($a=\text{torch.rand}(3,4,5)$)과 2차원 행렬 ($b=\text{torch.rand}(5,4)$)의 배치 행렬 곱셈을 계산하세요.

```
a = torch.rand(3,4,5)
b = torch.rand(5,4)
torch.bmm(a,b.unsqueeze(0).expand(a.size(0), *b.size()))

tensor([[[[2.1304, 1.9087, 1.4814, 2.4756],
          [1.5502, 1.1930, 1.7015, 1.7419],
          [0.9293, 1.0629, 1.3755, 1.0421],
          [1.7158, 1.6989, 1.4377, 1.8749]],

        [[1.4000, 1.3929, 1.6851, 1.7808],
          [1.6698, 1.4921, 1.8138, 2.0628],
          [1.3281, 0.9839, 1.0700, 1.2057],
          [1.2654, 1.5894, 1.1394, 1.5462]],

        [[1.4998, 1.8576, 1.3409, 1.7861],
          [1.1226, 1.3150, 1.5093, 1.6007],
          [1.8970, 1.6597, 1.6978, 1.8861],
          [1.0390, 1.0869, 0.9939, 1.2537]]]])
```

```
torch.matmul(a,b)

tensor([[[[2.1304, 1.9087, 1.4814, 2.4756],
          [1.5502, 1.1930, 1.7015, 1.7419],
          [0.9293, 1.0629, 1.3755, 1.0421],
          [1.7158, 1.6989, 1.4377, 1.8749]],

        [[1.4000, 1.3929, 1.6851, 1.7808],
          [1.6698, 1.4921, 1.8138, 2.0628],
          [1.3281, 0.9839, 1.0700, 1.2057],
          [1.2654, 1.5894, 1.1394, 1.5462]],

        [[1.4998, 1.8576, 1.3409, 1.7861],
          [1.1226, 1.3150, 1.5093, 1.6007],
          [1.8970, 1.6597, 1.6978, 1.8861],
          [1.0390, 1.0869, 0.9939, 1.2537]]]])
```

끝