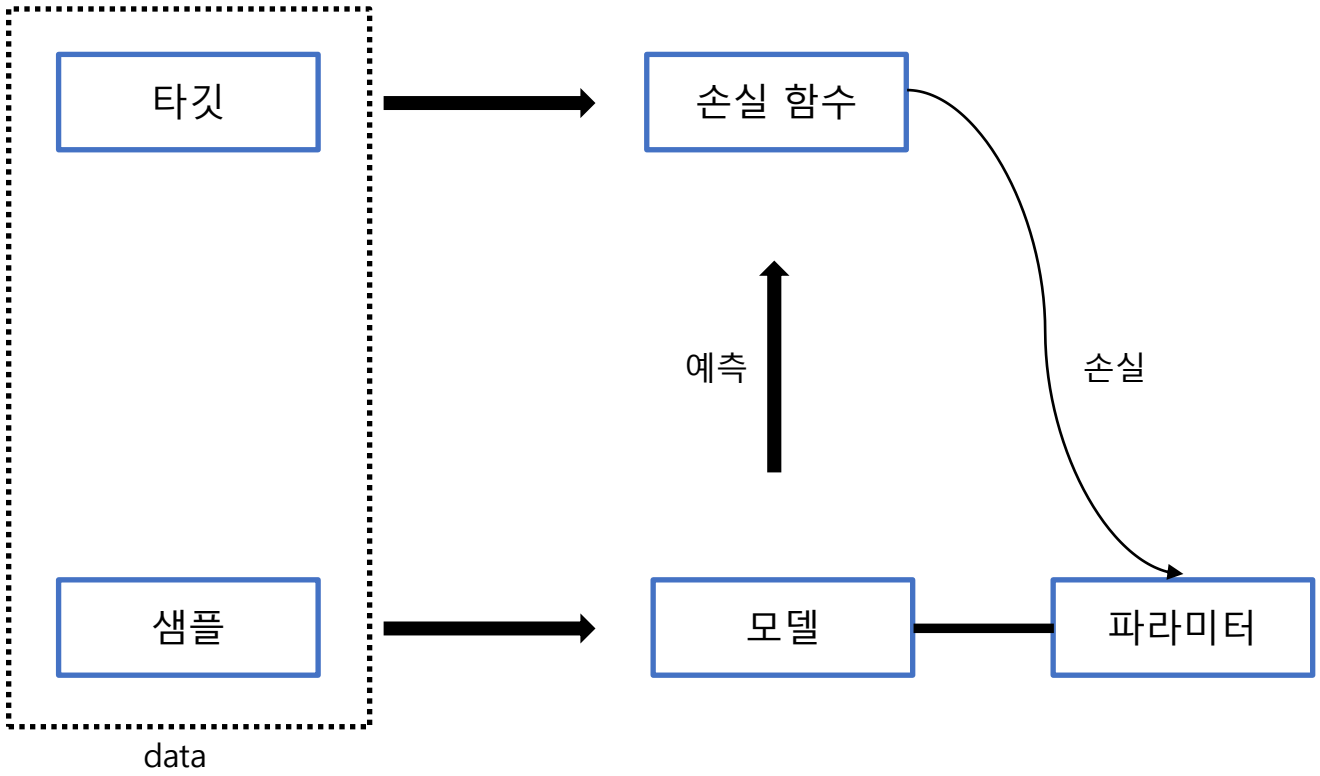
Natural Language Processing with PyTorch
-1장-

정 시 열

목차

지도학습



샘플과 타깃의 인코딩

- One-hot Representation
  - Zero Vector에서 시작하여 문장이나 문서에 등장하는 단어에 상응하는 원소를 1로 설정하는 기법

Time flies like an arrow.

Fruit flies like a banana.

→ vocab: {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' one-hot 표현 (3x8 Matrix)

	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

'like a banana' 이진 인코딩 표현 → [0,0,0,1,1,0,0,1]

샘플과 타깃의 인코딩

- ## TF(Term-Frequency) Representation

특정 문서에서 특정 단어의 등장 횟수. 소속 단어의 One-Hot representation을 합해서 만든다.
(각 원소는 해당 단어가 문장에 등장하는 횟수)

'Fruit flies like time flies a fruit' 에 대한 TF 표현

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

- ## TF – IDF Representation

출현 빈도가 낮은 단어가 해당 문서의 특징을 잘 표현하는 경우도 존재함 (ex 특허 문서)
단어의 빈도와 역 문서 빈도를 사용하여 각 단어의 중요한 정도를 가중치로 주는 방법

TF- IDF 점수: $TF(w) \times IDF(w)$

모든 문서에 등장하는 단어는 IDF(w)의 값이 0이다.
따라서 이런 단어는 완전히 제외한다

$$IDF(w) = \log\frac{N}{n_w}$$

n_w: 단어 w를 포함하는 문서의 개수, N: 전체 문서 개수 ← 흔한 단어의 점수를 낮추고 드문 단어의 점수를 높이겠다.

계산 그래프

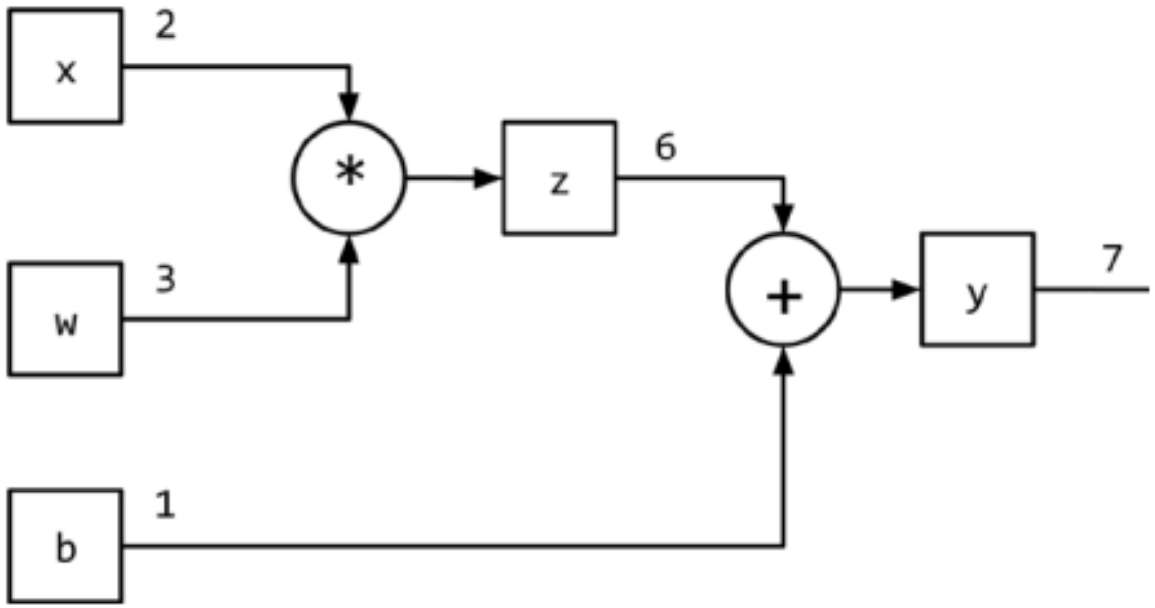


그림 1-6 계산 그래프를 사용한 식 $y = wx + b$의 표현

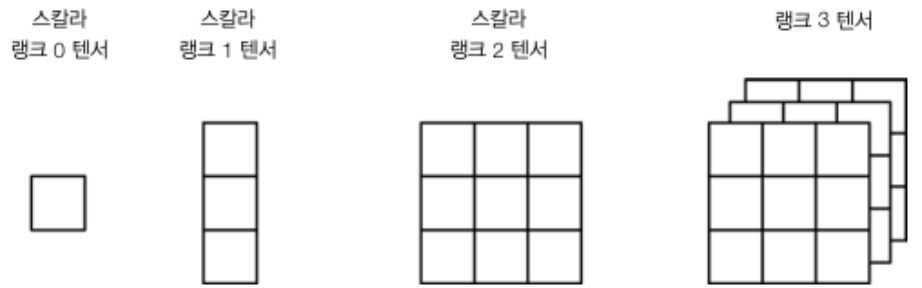
계산 그래프는 수학식을 추상적으로 모델링한 것

$$y = wx + b$$

$$z = wx$$
$$y = z + b$$

파이토치 기초

파이토치의 핵심은 '텐서'



- 텐서 만들기

```
In[0]  def describe(x):
           print("타입: {}".format(x.type()))
           print("크기: {}".format(x.shape))
           print("값: \n{}".format(x))
```

```
In[0]  import torch
       describe(torch.Tensor(2, 3))

Out[0] 타입: torch.FloatTensor
       크기: torch.Size([2, 3])
       값:
       tensor([[ 3.2018e-05,  4.5747e-41,  2.5058e+25],
               [ 3.0813e-41,  4.4842e-44,  0.0000e+00]])
```

랜덤 초기화 생성

```
In[0]  import torch
       describe(torch.rand(2, 3))   # 균등 분포
       describe(torch.randn(2, 3))  # 표준 정규 분포

Out[0] 타입:  torch.FloatTensor
       크기:  torch.Size([2, 3])
       값:
       tensor([[ 0.0242,  0.6630,  0.9787],
               [ 0.1037,  0.3920,  0.6084]])

       타입: torch.FloatTensor
       크기: torch.Size([2, 3])
       값:
       tensor([[ -0.1330, -2.9222, -1.3649],
               [  2.3648,  1.1561,  1.5042]])
```

균등 분포/표준 정규분포 초기화 생성

파이토치

- 텐서 만들기

코드 1-5 filled() 메서드 사용하기

```
In[0]  import torch
       describe(torch.zeros(2, 3))
       x = torch.ones(2, 3)
       describe(x)
       x.fill_(5)
       describe(x)

Out[0] 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.]])
```

특정값으로 채워진 텐서 생성

```
In[0]  x = torch.Tensor([[1, 2, 3],
                         [4, 5, 6]])
       describe(x)

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

```
In[0]  import torch
       import numpy as np
       npy = np.random.rand(2, 3)
       describe(torch.from_numpy(npy))

Out[0] 타입: torch.DoubleTensor
       크기: torch.Size([2, 3])
       값:
       tensor([[ 0.8360,  0.8836,  0.0545],
               [ 0.6928,  0.2333,  0.7984]], dtype=torch.float64)
```

리스트 혹은 넘파이를 통한 텐서 생성

파이토치 기초

- 텐서 타입과 크기

```
In[0]  x = torch.FloatTensor([[1, 2, 3],
                              [4, 5, 6]])
       describe(x)

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

In[1]  x = x.long()
       describe(x)

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

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

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

- 텐서 연산

```
In[2]  describe(x + x)
```

```
In[1]  describe(torch.add(x, x))
```

```
In[0]21 import torch
        x = torch.arange(6)
        describe(x)

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

In[1]22 x = x.view(2, 3)
        describe(x)

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

```
In[2]  describe(torch.sum(x, dim=0))

Out[2] 타입: torch.FloatTensor
       크기: torch.Size([3])
       값:
       tensor([ 3.,  5.,  7.])

In[3]  describe(torch.sum(x, dim=1))

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

In[4]23 describe(torch.transpose(x, 0, 1))

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

파이토치 기초

- 인덱싱. 슬라이싱, 연결

```
In[0]  import torch
       x = torch.arange(6).view(2, 3)
       describe(x)

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

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

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

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

Out[2] 타입: torch.FloatTensor
       크기: torch.Size([])
       값:
       1.0
```

```
In[0]  import torch
       x = torch.arange(6).view(2,3)
       describe(x)

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

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

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

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

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

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

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

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

파이토치 기초

- 학습 준비하기

```
In[0]  import torch
       x = torch.ones(2, 2, requires_grad=True)
       describe(x)
       print(x.grad is None)
```

requires_grad = True 지정을 통해

손실함수와 그레디언트를 기록하는 부가 연산을 활성화

```
In[0]  import torch
       print (torch.cuda.is_available())

Out[0] True

In[1]  # 바람직한 방법: 장치에 무관한 텐서 초기화
       device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
       print (device)

Out[1] cuda

In[2]  x = torch.rand(3, 3).to(device)
       describe(x)

Out[2] 타입: torch.cuda.FloatTensor
       크기: torch.Size([3, 3])
       값:
       tensor([[ 0.9149,  0.3993,  0.1100],
               [ 0.2541,  0.4333,  0.4451],
               [ 0.4966,  0.7865,  0.6604]], device='cuda:0')
```

1. torch.cuda.is_available()로 GPU 사용여부 판단
2. torch.device()로 장치 이름 가지고 오기
3. .to(device)메서드 사용

END