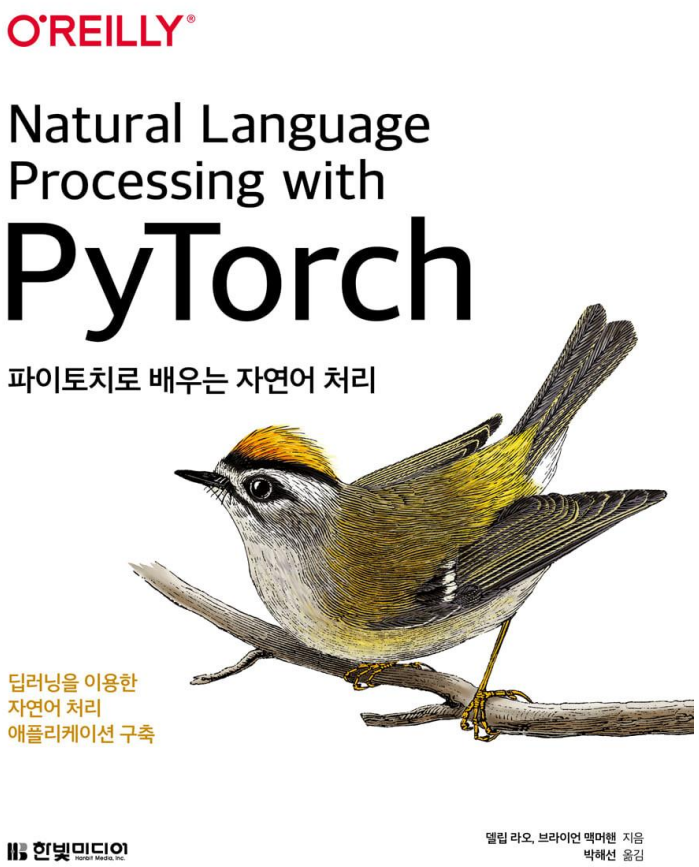
Data Intelligence Lab

2022.6.29 ~ 7.6

Progress

인천대학교 컴퓨터공학부 강병하

| textbook



원서 2019년 1월 출판

번역서 2021년 6월 출판

학습한 내용

Concept

- 텍스트 데이터
- 신경망의 구성요소

Practice

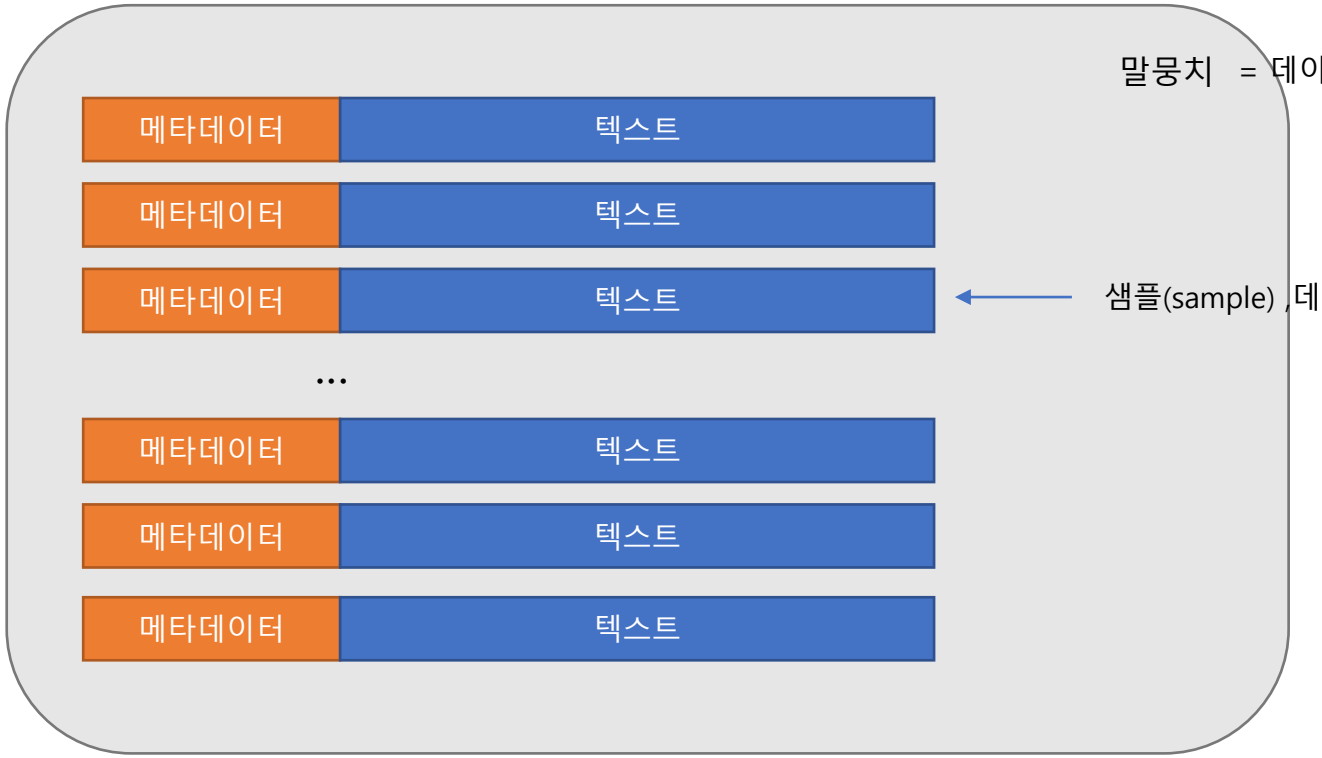
옐프 레스토랑 리뷰 감성 분류하기(긍정,부정)

말뭉치(corpus)

원시 텍스트(ASCII,UTF-8)와 이 텍스트에 연관된 메타데이터의 모음

식별자, 레이블, 타임스태프 등의 **부가 정보**

말뭉치 = 데이터셋(dataset)



샘플(sample) ,데이터 포인트(data point)

토큰화(tokenization)

텍스트를 토큰으로 나누는 과정

공백과 구두점으로 나누기 → **교착어가 아닌 경우** 가능(영어)

```
[13]  # spacy를 사용한 토큰화
      import spacy
      nlp = spacy.load('en_core_web_sm')
      text = "A simple idea that i knew would change everything"
      print([str(token) for token in nlp(text.lower())])

      ['a', 'simple', 'idea', 'that', 'i', 'knew', 'would', 'change', 'everything']
```

*교착어

하나의 어절이 하나의 어근(or어간)과

하나의 이상의 접사로 이루어진 언어

Ex) 잡히셨겠더라

어근

어휘사전, 단어집합(vocabulary)

중복을 제거한 말뭉치의 단어 집합

=말뭉치에 등장하는 모든 **타입**의 집합

타입(type) : 말뭉치에 등장하는 고유한 토큰

```
[26]  import spacy
      nlp = spacy.load('en_core_web_sm')
      vocabulary = []
      corpus = ["A simple idea that i knew would change everything",
                "Simple is the best"]
      for text in corpus:
        for token in nlp(text.lower()):
          if str(token) not in vocabulary:
            vocabulary.append(str(token))
      print(vocabulary)
```

```
['a', 'simple', 'idea', 'that', 'i', 'knew', 'would', 'change', 'everything', 'is', 'the', 'best']
```

표제어(lemma)

단어의 **기본형**(사전에 등재된 단어)

표제어 추출(lemmatization)

: 토큰을 표제어로 바꾸어 벡터 표현의 차원을 줄이는 것

```
# 표제어 추출
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(u"he was running rate")
for token in doc:
  print('{} --> {}'.format(token, token.lemma_))
```

```
he --> he
was --> be
running --> run
rate --> rate
```

어간(stem)

단어에서 개념적 의미를 가지고 변하지 않는 부분

어간 추출(stemming)

: 정해진 규칙으로 단어의 끝을 잘라 어간으로 축소하는 기법.

```
from nltk.stem import PorterStemmer
s=PorterStemmer()
words=['perfection', 'is', 'achieved', ',', 'not', 'when', 'there', 'is', 'nothing', 'more', 'to', 'add']
print([s.stem(w) for w in words])
```

```
['perfect', 'is', 'achiev', ',', 'not', 'when', 'there', 'is', 'noth', 'more', 'to', 'add']
```

단어 분류 : 품사 태깅

단어에 품사(POS, part of speech) 레이블 할당하기

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(u"A simple idea that i knew would change everything")
for token in doc:
  print('{} - {}'.format(token, token.pos_)) # part of speech, 품사
```

```
A - DET
simple - ADJ
idea - NOUN
that - PRON
i - PRON
knew - VERB
would - AUX
change - VERB
everything - PRON
```

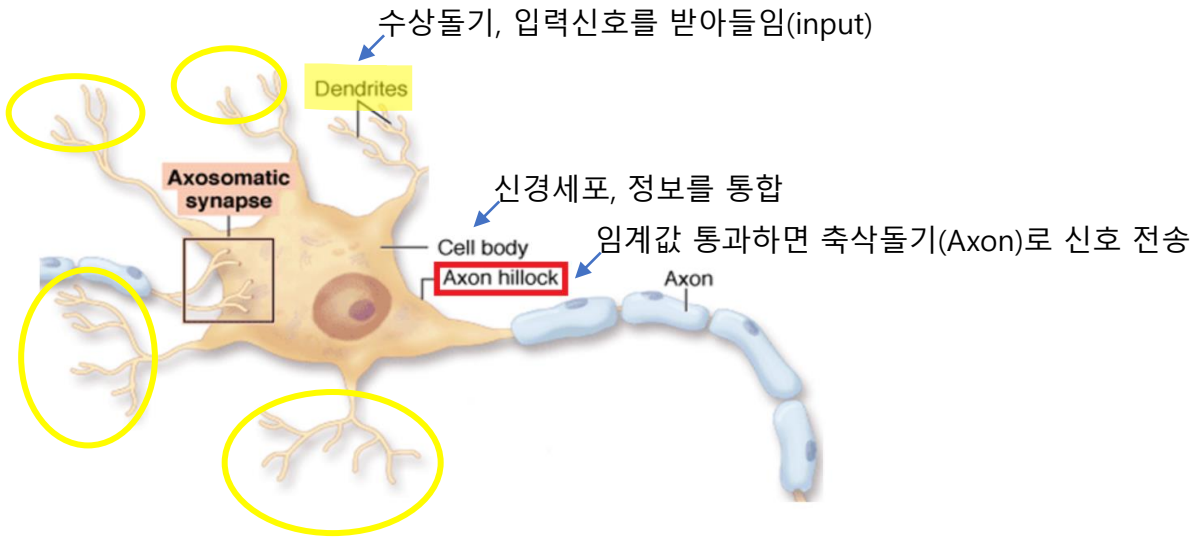
청크나누기(chunking)

연속된 토큰으로 구분되는 '구'에 레이블 할당 -> **부분 구문 분석(shallow parsing)**이라고도 함

```
[32] import spacy
     nlp = spacy.load('en_core_web_sm')
     doc = nlp(u"Simplicity is the ultimate sophistication.")
     for chunk in doc.noun_chunks:
       print('{} - {}'.format(chunk, chunk.label_))
```

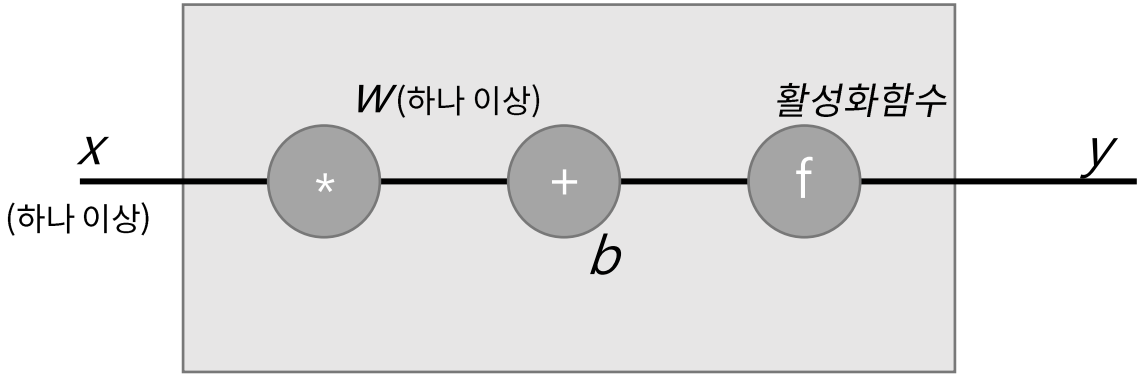
```
Simplicity - NP
the ultimate sophistication - NP
```

퍼셉트론 : 신경망의 구성요소



-> **다수의 입력 신호**를 받아 **하나**의 신호 출력(**임계점을 넘은 경우**)

퍼셉트론 : 신경망의 구성요소



$$Y = f(w * x + b)$$

활성화 함수

설계자의 직관에 따라 결정 -> ***하이퍼 파라미터***

[x1, x2, … , x99] [w1,
w2,
… ,
w99]

스칼라곱(dot product)

퍼셉트론 in 파이토치

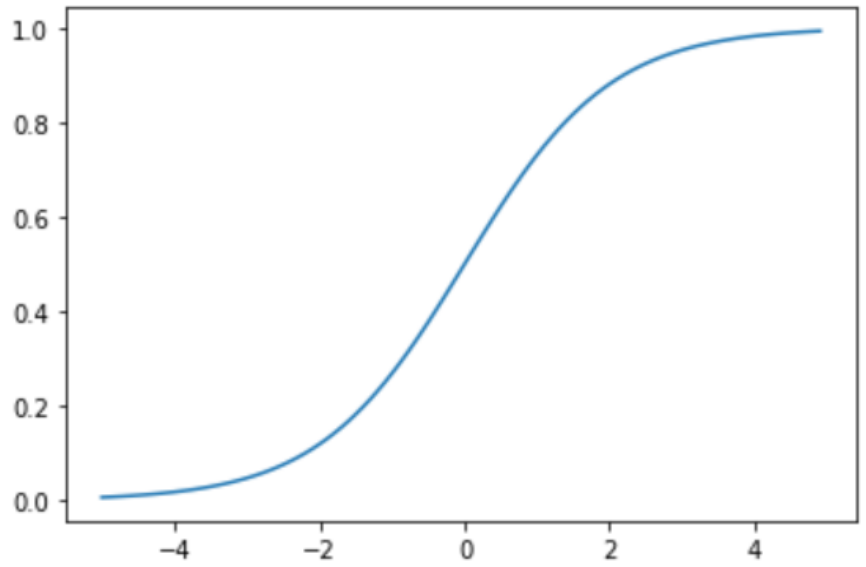
```
[1]  # 파이토치로 구현한 퍼셉트론
     import torch
     import torch.nn as nn

     class Perceptron(nn.Module):
       # 퍼셉트론은 하나의 선형 층이다.
       def __init__(self, input_dim): # input_dim(int) 입력 특성의 크기.
         super(Perceptron, self).__init__()
         self.fc1 = nn.Linear(input_dim, 1) # torcg.nn 모듈 아래 가중치와 절편에 필요한 부가 작업과 아핀 변환을 수행해주는 클래스.

       # 퍼셉트론의 정방향 계산
       def forward(self, x_in): # x_in(torch.Tensor): 입력 데이터 텐서, x_in.shape는 (batch, num_features)
         return torch.sigmoid(self.fc1(x_in)).squeeze() # 결과 : 텐서. tensor.shapesms (batch,)
```

Torch.nn.Module -> Neural Network의 모든 것을 포괄하는 모든 신경망 모델의 Base Class.

.nn.Linear(입력 차원, 출력 차원) -> 선형 회귀 모델.

활성화 함수 - 시그모이드

```
import torch
import matplotlib.pyplot as plt

x = torch.arange(-5., 5., 0.1)
y = torch.sigmoid(x)
plt.plot(x.numpy(), y.detach().numpy())
plt.show()
```



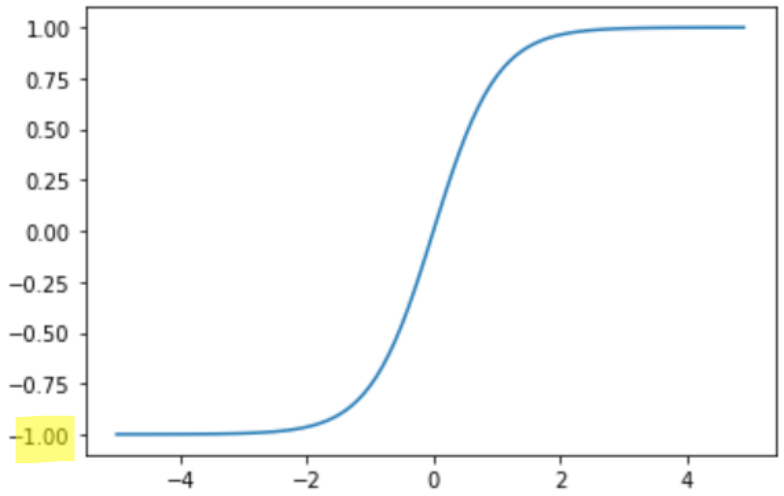
그레이디언트 소실 문제

그레이디언트 폭주 문제

-> 출력층에서 출력을 확률로 압축하는 데 사용

활성화 함수 – 하이퍼볼릭 탄젠트

```
import torch
import matplotlib.pyplot as plt

x = torch.arange(-5., 5., 0.1)
y = torch.tanh(x)

plt.plot(x.numpy(), y.detach().numpy())
plt.show()
```

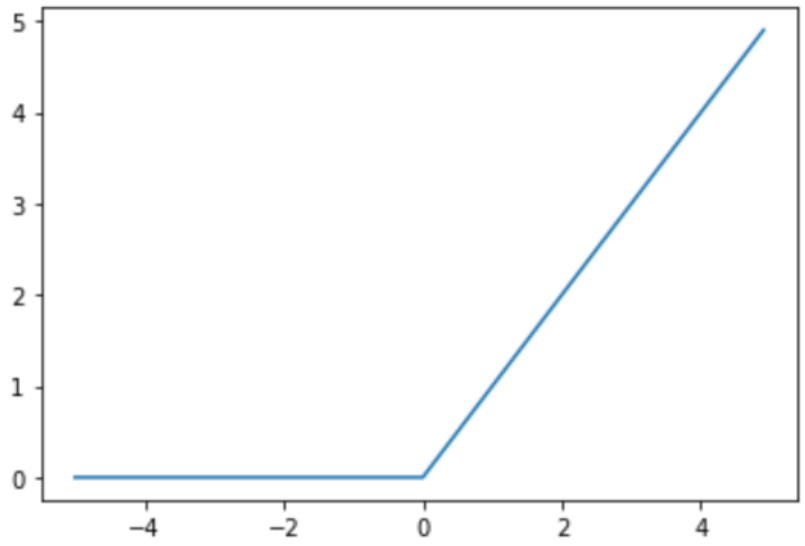


시그모이드 함수 변종

[0, 1] -> [-1, 1]

활성화 함수 – 렐루(ReLU)

```python
import torch
import matplotlib.pyplot as plt

relu = torch.nn.ReLU()
x = torch.arange(-5., 5., 0.1)
y = relu(x)

plt.plot(x.numpy(), y.detach().numpy())
plt.show()
```



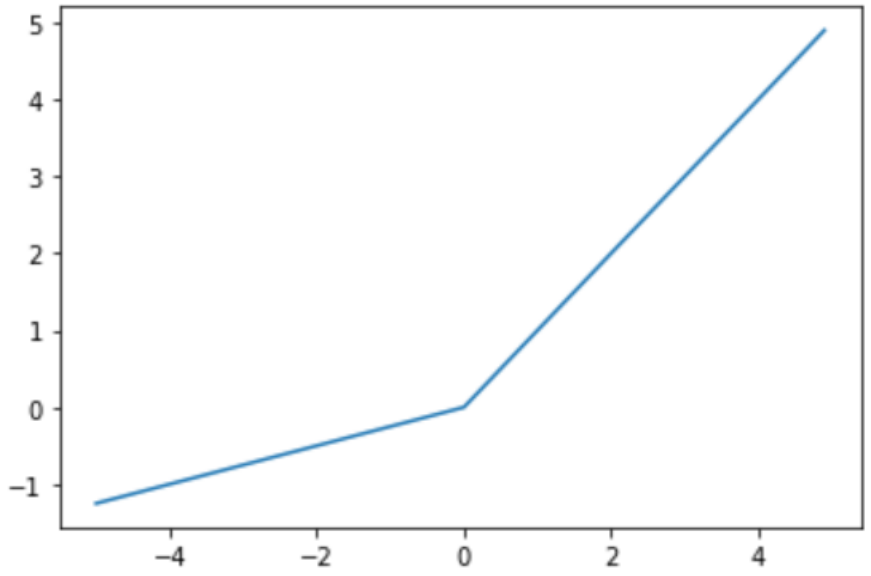
가장 중요한 활성화 함수

음수값을 0으로

-> 특정 출력이 0이 되면 돌아오지 않음 (= 죽은 렐루 문제)

활성화 함수 – PReLU

```
import torch.nn as nn
import matplotlib.pyplot as plt

prelu = nn.PReLU(num_parameters=1)
x = torch.arange(-5., 5., 0.1)
y = prelu(x)

plt.plot(x.numpy(), y.detach().numpy())
plt.show()
```



렐루의 변종

활성화 함수 - 소프트맥스

```
softmax = nn.Softmax(dim=1)
x_input = torch.randn(1, 3)
y_output = softmax(x_input)
print(x_input)
print(y_output)
print(torch.sum(y_output, dim=1))
```

```
tensor([[-2.0260, -2.0655, -1.2054]])
tensor([[0.2362, 0.2271, 0.5367]])
tensor([1.])
```

다중 분류에서 출력을 확률로 변환하는데 유용

손실 함수(Loss Function)

모델의 **예측**과 타깃에 대한 오차

-> 손실 함수 값이 높을 수록 예측 성능 **나쁨**

손실 함수의 값이 충분히 낮아질 때까지 파라미터를 반복 업데이트

손실 함수 – 평균 제곱 오차 손실 (MSE)

신경망의 출력과 타깃값의 차이를 제곱하여 평균 -> **회귀**에서 널리 사용

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

(Mean Square Error, MSE)

```python
import torch
import torch.nn as nn

mse_loss = nn.MSELoss() # 평균 제곱 오차, mean squared error(MSE)
outputs = torch.randn(3, 5, requires_grad=True) # 예측 (gradient)
targets = torch.randn(3, 5) # 타깃
loss = mse_loss(outputs, targets)
print(outputs)
print(targets)
print(loss)
```

```
tensor([[-0.1713,  0.3444,  0.0806,  0.3825, -0.0695],
        [-2.1203, -0.8431,  0.8352, -0.9471,  1.0058],
        [ 1.0583, -0.1974, -1.3686,  1.1271, -0.2524]], requires_grad=True)
tensor([[ 1.9709,  0.2994, -0.3736,  0.2663,  1.6074],
        [-0.2983,  0.5600, -1.1969,  0.3254, -0.0441],
        [-0.7714,  1.5879,  1.3588,  0.9168, -0.6727]])
tensor(2.2638, grad_fn=<MseLossBackward0>)
```

손실 함수 – 범주형 크로스 엔트로피 손실

(모든 클래스의 다항 분포를 나타내는 원소 n개로 이루어진 벡터)타깃 x 예측(클래스 소속 확률)의 로그

*다항 분포 벡터 -> 모든 **원소의 합 1** & 모든 원소는 **양수**

-> **다중 분류**에서 사용

$$-\sum_{i=1}^{n} y_i log(a_i)$$

(Categorical Cross-entropy)

```python
# 크로스 엔트로피 손실
import torch
import torch.nn as nn

ce_loss = nn.CrossEntropyLoss()
outputs = torch.randn(3,5, requires_grad=True) # 신경망의 출력
targets = torch.tensor([1, 0, 3], dtype=torch.int64) # 각 샘플의 정답클래스 인덱스
loss = ce_loss(outputs, targets)
print(outputs)
print(targets)
print(loss)
```

```
tensor([[-0.8560,  0.4839,  0.9075,  0.4101, -0.3021],
        [-0.5788,  0.9395,  2.0644, -0.6267,  1.8642],
        [-0.5552, -0.8513,  0.6529, -0.4432, -0.4517]], requires_grad=True)
tensor([1, 0, 3])
tensor(2.2585, grad_fn=<NllLossBackward0>)
```

손실 함수 – 이진 크로스 엔트로피 손실

타깃은 1(양성클래스)과 0(음성클래스)로 구성

타깃
1 -> -log(예측확률)
0 -> -log(1-예측확률)

->**이진 분류**에서 사용

$$-\frac{1}{n}\sum_{i=1}^{n}(y_i log\hat{y}_i + (1-y_i)log(1-\hat{y}_i))$$

(Binary Cross-entropy)

```
[ ]  # 이진 크로스 엔트로피 손실
     bce_loss = nn.BCELoss()
     sigmoid = nn.Sigmoid()
     probabilities = sigmoid(torch.randn(4,1, requires_grad=True)) # 시그모이드 함수를 적용해 확률 벡터를 만들어줌.
     target = torch.tensor([1,0,1,0], dtype=torch.float32).view(4,1)
     loss = bce_loss(probabilities, target)
     print(probabilities)
     print(loss)

     tensor([[0.8115],
             [0.3148],
             [0.4450],
             [0.4306]], grad_fn=<SigmoidBackward0>)
     tensor(0.4900, grad_fn=<BinaryCrossEntropyBackward0>)
```

지도 학습 훈련

모델의 **예측**과 **손실 함수**를 사용해 모델의 **파라미터**를 **그레이디언트** 기반의 방법으로 최적화

필요한 것

- 훈련 데이터(샘플과 타깃 쌍)
- 모델 → 샘플에 대한 예측 계산
- 손실 함수 → 예측과 타깃의 오차 계산
- **최적화 알고리즘 → 모델의 파라미터 조정 ➔ *가능한 한 낮은 손실***

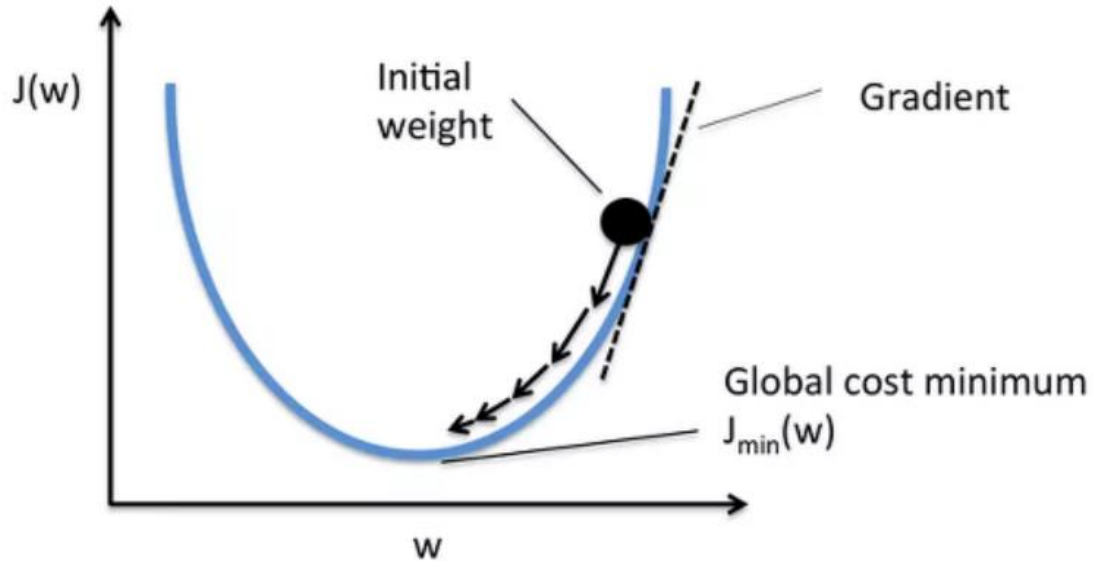
지도 학습 훈련 과정

데이터 준비 ➔ 모델 선택 ➔ **손실함수** 선택 ➔ **최적화 알고리즘(옵티마이저)** 설정 ➔ 모두 합쳐 실행

옵티마이저

그레이디언트 기반의 최적화 알고리즘. 손실 함수의 오차 신호로 모델의 **가중치 업데이트**.

"각 파라미터에 대한 손실값의 순간 변화율 (미분계수)" → 모델 파라미터를 얼마나 많이 바꿔야 하는지



- SGD (확률적 경사 하강법)
- **Adam**

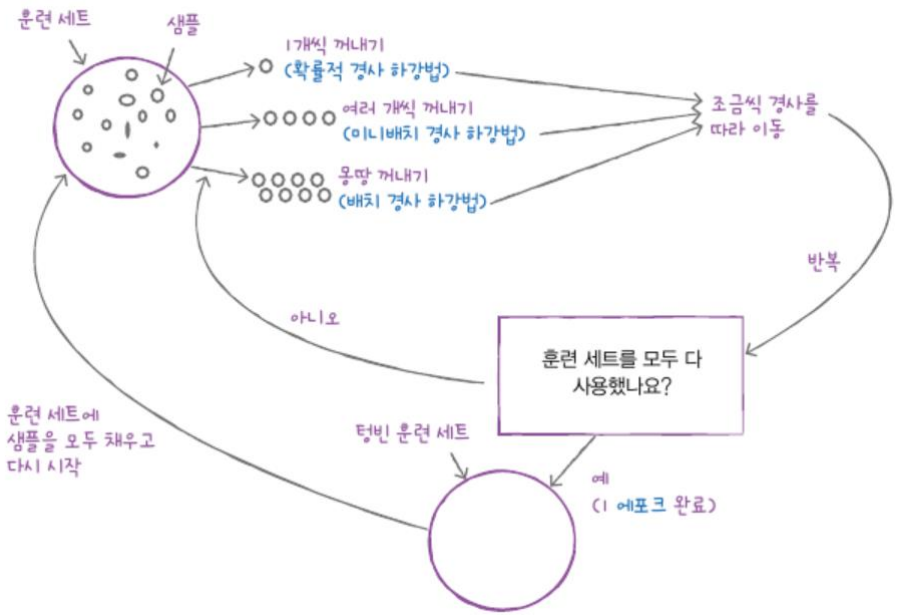
학습 훈련 반복

퍼셉트론 -> 이진분류 반복 과정

```
[ ]  # 그레이디언트 업데이트 알고리즘

     for epoch_i in range(n_epochs):
       for batch_i in range(n_batches):

         # 0단계: 데이터 가져오기
         x_data, y_target = get_toy_data(batch_size)

         # 1단계: 그레이디언트 초기화
         perceptron.zero_grad()

         # 2단계: 모델의 정방향 계산 수행
         y_pred = perceptron(x_data, apply_sigmoid=True)

         # 3단계: 최적하려는 손실 계산하기
         loss = bce_loss(y_pred, y_target)

         # 4단계: 손실 신호를 거꾸로 전파하기
         loss.backward()

         #5단계: 옵티마이저로 업데이트
         optimizer.step()
```

지정한 에포크수 or 종료 조건(성능이 더 좋아지지 않음) 만큼 반복.

배치 개수 = 전체 훈련 데이터셋 / batch_size (하이퍼 파라미터)



모델 성능 평가하기

평가 지표

정확도 : 훈련하는 동안 만나지 못한 데이터에 대해 올바르게 예측한 비율

일반화가 잘 된 모델 → 훈련 데이터의 샘플뿐만 아니라 본 적없는 샘플에서도 오차를 줄 일 때

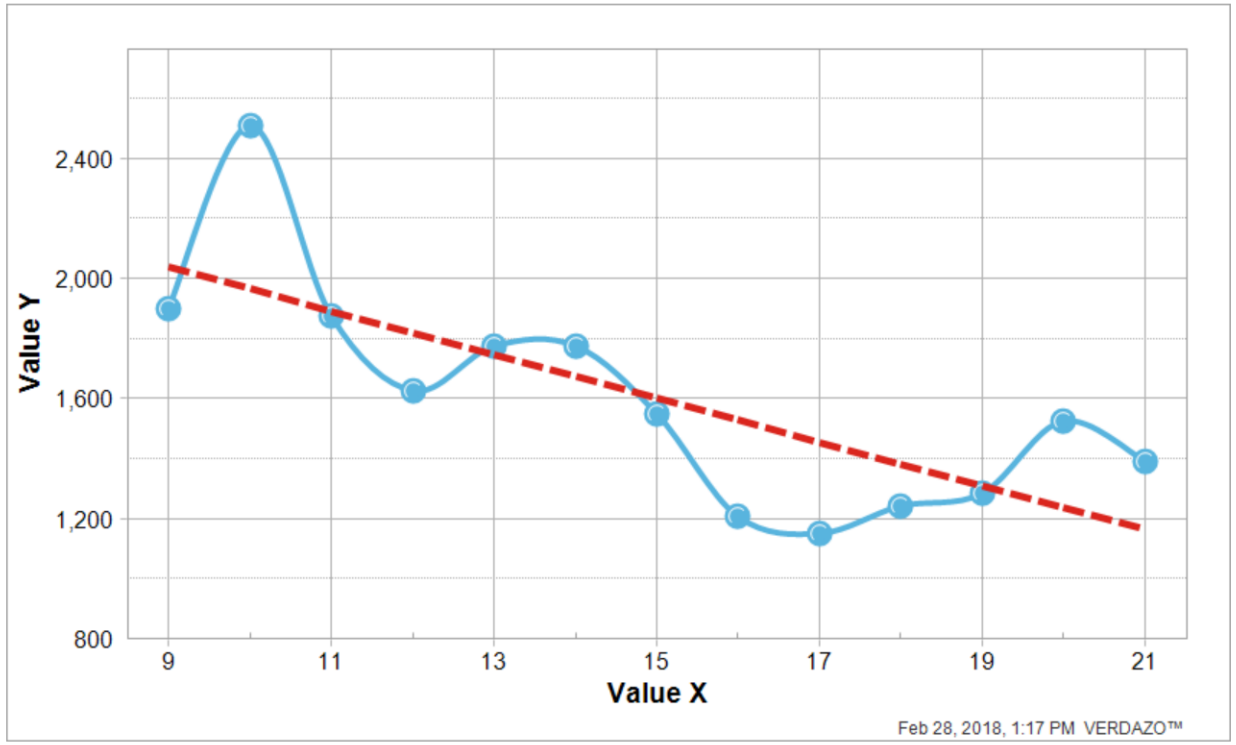
- 랜덤하게 샘플링하여 3개로 나누기 (훈련 세트 70%, 검증 세트 15%, 테스트 세트 15%)
- k-겹 교차 검증

학습데이터	학습데이터	학습데이터	학습데이터	테스트 데이터	결과1
학습데이터	학습데이터	학습데이터	테스트 데이터	학습데이터	결과2
학습데이터	학습데이터	테스트 데이터	학습데이터	학습데이터	결과3
학습데이터	테스트 데이터	학습데이터	학습데이터	학습데이터	결과4
테스트 데이터	학습데이터	학습데이터	학습데이터	학습데이터	결과5

규제

오컴의 면도날 이론

'같은 현상을 설명하는 두 개의 주장이 있다면, 간단한 쪽을 선택하라(given two equally accurate theories, choose the one that is less complex)'

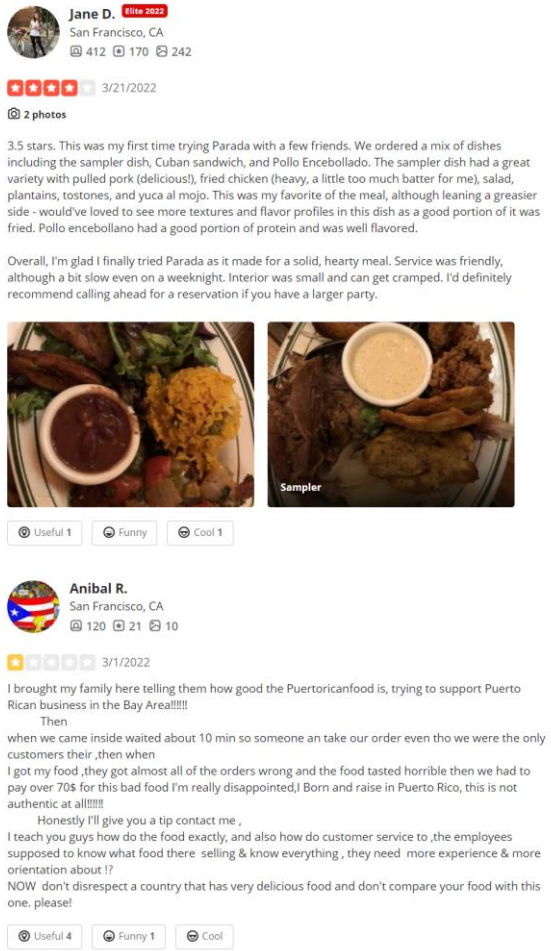


Lasso (L1)	Ridge (L2)
가중치를 0으로, 특성 무력화	가중치를 0에 가깝게, 특성들의 영향력 감소
일부 특성이 중요하다면	특성의 중요도가 전체적으로 비슷하다면
sparse model	Non-sparse model
feature selection	

Practice – 레스토랑 리뷰 감성(긍정/부정)분류

퍼셉트론과 지도 학습 훈련 방법을 사용해

옐프(Yelp)의 레스토랑 리뷰가 **긍정적**인지 **부정적**인지 분류하는 작업 ➔ 이진 분류



별 1,2 ➔ 음성클래스(부정적)

별 3개 이상 ➔ 양성클래스(긍정적)

옐프 리뷰 데이터셋

	rating	review
0	1	Unfortunately, the frustration of being Dr. Go...
1	2	Been going to Dr. Goldberg for over 10 years. ...
2	1	I don't know what Dr. Goldberg was like before...
3	1	I'm writing this review to give you a heads up...
4	2	All the food is great here. But the best thing...
...	...	...
165881	1	Just went back this weekend. Not as good as 1s...
165882	1	After lasting one night in that spooky L Motel...
165883	1	DO NOT EAT HERE!₩n₩nMy friend and I had breakf...
165884	2	Have to say that after a long search my good f...
165885	2	I get cleanings every 6 months & visited a new...

[165886 rows × 2 columns]

1이 부정적, 2가 긍정적

훈련 : 560,000개, 테스트 : 38,000개

10%만 사용
➔ (훈련/검증/테스트)

옐프 리뷰 데이터셋 전처리

```
[25] final_reviews.head()
```

	rating	review	split
0	negative	andrew here said it best . . . food served by ...	train
1	negative	well , the delivery was fast at least . everyt...	train
2	negative	this is not a review on this gym . it is a rev...	train
3	negative	normally one would think , one chain restauran...	train
4	negative	unfortunately , the service we were given ruin...	train

```
[26] final_reviews.tail()
```

	rating	review	split
2110	positive	the food at joe s rusty nail is great . as the...	test
2111	positive	a cool if mildly painfully hipster ish decor p...	test
2112	positive	at this time , i have officially had every mea...	test
2113	positive	a classic old school joint . the bar dates bac...	NaN
2114	positive	best pizza in the north boros . i can t make a...	NaN

| 데이터셋 클래스 : ReviewDataset

```python
class ReviewDataset(Dataset):
    def __init__(self, review_df, vectorizer):
        """
        매개변수:
            review_df (pandas.DataFrame): 데이터셋
            vectorizer (ReviewVectorizer): ReviewVectorizer 객체
        """
        self.review_df = review_df
        self._vectorizer = vectorizer

        self.train_df = self.review_df[self.review_df.split=='train']
        self.train_size = len(self.train_df)

        self.val_df = self.review_df[self.review_df.split=='val']
        self.validation_size = len(self.val_df)

        self.test_df = self.review_df[self.review_df.split=='test']
        self.test_size = len(self.test_df)

        self._lookup_dict = {'train': (self.train_df, self.train_size),
                             'val': (self.val_df, self.validation_size),
                             'test': (self.test_df, self.test_size)}

        self.set_split('train')
```

```python
    def __len__(self):
        return self._target_size

    def __getitem__(self, index):
        """ 파이토치 데이터셋의 주요 진입 메서드

        매개변수:
            index (int): 데이터 포인트의 인덱스
        반환값:
            데이터 포인트의 특성(x_data)과 레이블(y_target)로 이루어진 딕셔너리
        """
        row = self._target_df.iloc[index]

        review_vector = ₩
            self._vectorizer.vectorize(row.review)

        rating_index = ₩
            self._vectorizer.rating_vocab.lookup_token(row.rating)

        return {'x_data': review_vector,
                'y_target': rating_index}
```

Dataset 클래스 상속 → getitem, len override

getitem() : { 'x_data' : 벡터화한 리뷰 데이터, 'y_target' :벡터 타깃(1:positive or 0:negetive) }

len() : 데이터셋 전체 길이

어휘사전 클래스 : Vocabulary

어휘 사전을 만드는 클래스. 토큰을 정수로 매핑 (일대일 매핑)

→ 딕셔너리에 토큰을 추가하면 자동으로 인덱스 증가

→ UNK(unknown) 토큰 관리 – 훈련에서 본적 없는 토큰 처리

```python
class Vocabulary(object):
    """ 매핑을 위해 텍스트를 처리하고 어휘 사전을 만드는 클래스 """

    def __init__(self, token_to_idx=None, add_unk=True, unk_token="<UNK>"): # UNK('unknown'을 의미, 훈련에서 본 적이 없는 토큰)
        """
        매개변수:
            token_to_idx (dict): 기존 토큰-인덱스 매핑 딕셔너리
            add_unk (bool): UNK 토큰을 추가할지 지정하는 플래그
            unk_token (str): Vocabulary에 추가할 UNK 토큰
        """
```

```python
def add_token(self, token):
    """ 토큰을 기반으로 매핑 딕셔너리를 업데이트합니다

    매개변수:
        token (str): Vocabulary에 추가할 토큰
    반환값:
        index (int): 토큰에 상응하는 정수
    """
    if token in self._token_to_idx: # 기존 토큰-인덱스 매핑 딕셔너리에 있는 경우
        index = self._token_to_idx[token] # 해당 인덱스 반환
    else: # 기존 토큰-인덱스 매핑 딕셔너리에 있는 경우
        index = len(self._token_to_idx) # 마지막 인덱스 + 1
        self._token_to_idx[token] = index # 토큰 - 정수 매핑
        self._idx_to_token[index] = token
    return index
```

```python
def lookup_token(self, token):
    """ 토큰에 대응하는 인덱스를 추출합니다.
```

```python
def lookup_index(self, index):
    """ 인덱스에 해당하는 토큰을 반환합니다.
```

벡터화 클래스 : Vectorizer

입력 데이터 포인트(토큰화된 텍스트 샘플)의 토큰들을 순회하면서 **정수로 바꾸기**

→ 텍스트 샘플을 **원-핫 벡터**로 변경

```
def vectorize(self, review):
    """ 리뷰에 대한 원-핫 벡터를 만듭니다

    매개변수:
        review (str): 리뷰
    반환값:
        one_hot (np.ndarray): 원-핫 벡터
    """
    one_hot = np.zeros(len(self.review_vocab), dtype=np.float32)
                어휘 사전의 크기만큼 0으로 채우기
    for token in review.split(" "):
        if token not in string.punctuation:
            one_hot[self.review_vocab.lookup_token(token)] = 1
                                    해당 토큰의 인덱스의 값을 1로 지정
    return one_hot
```

[0, 0, 1, 0, … , 0, 1, 0]

DataLoader

벡터로 변환한 데이터 포인트를 미니배치로 모으는 작업을 편하게 해줌.

```python
def generate_batches(dataset, batch_size, shuffle=True,
                     drop_last=True, device="cpu"):
    """
    파이토치 DataLoader를 감싸고 있는 제너레이터 함수.
    걱 텐서를 지정된 장치로 이동합니다.
    """
    dataloader = DataLoader(dataset=dataset, batch_size=batch_size,
                            shuffle=shuffle, drop_last=drop_last)

    for data_dict in dataloader:
        out_data_dict = {}
        for name, tensor in data_dict.items():
            out_data_dict[name] = data_dict[name].to(device)
        yield out_data_dict
```

퍼셉트론 분류기 : ReviewClassifier

```python
class ReviewClassifier(nn.Module):
    """ 간단한 퍼셉트론 기반 분류기 """
    def __init__(self, num_features):
        """
        매개변수:
            num_features (int): 입력 특성 벡트의 크기
        """
        super(ReviewClassifier, self).__init__()
        self.fc1 = nn.Linear(in_features=num_features,
                             out_features=1)

    def forward(self, x_in, apply_sigmoid=False):
        """ 분류기의 정방향 계산

        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
                x_in.shape는 (batch, num_features)입니다.
            apply_sigmoid (bool): 시그모이드 활성화 함수를 위한 플래그
                크로스-엔트로피 손실을 사용하려면 False로 지정합니다
        반환값:
            결과 텐서. tensor.shape은 (batch,)입니다.
        """
        y_out = self.fc1(x_in).squeeze()
        if apply_sigmoid:
            y_out = torch.sigmoid(y_out)
        return y_out
```

하나의 선형층

이진 분류 이므로 출력은 하나

apply_sigmoid = False
→ 시그모이드 적용하지 않고 이진 크로스 엔트로피 손실 BCELoss() 적용

모델 훈련 - 준비

```
args = Namespace(
    # 날짜와 경로 정보
    frequency_cutoff=25,
    model_state_file='model.pth',
    review_csv='data/yelp/reviews_with_splits_lite.csv',
    # review_csv='data/yelp/reviews_with_splits_full.csv',
    save_dir='model_storage/ch3/yelp/',
    vectorizer_file='vectorizer.json',
    # 모델 하이퍼파라미터 없음
    # 훈련 하이퍼파라미터
    batch_size=128,
    early_stopping_criteria=5,
    learning_rate=0.001,
    num_epochs=100,
    seed=1337,
```

하이퍼 파라미터 지정

모델 훈련 – 데이터셋, 모델, 손실, 옵티마이저

```
vectorizer = dataset.get_vectorizer()

classifier = ReviewClassifier(num_features=len(vectorizer.review_vocab))
classifier = classifier.to(args.device)

loss_func = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(classifier.parameters(), lr=args.learning_rate)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer,
                                                 mode='min', factor=0.5,
                                                 patience=1)
train_state = make_train_state(args)
```

```
def make_train_state(args):
    return {'stop_early': False,
            'early_stopping_step': 0,
            'early_stopping_best_val': 1e8,
            'learning_rate': args.learning_rate,
            'epoch_index': 0,
            'train_loss': [],
            'train_acc': [],
            'val_loss': [],
            'val_acc': [],
            'test_loss': -1,
            'test_acc': -1,
            'model_filename': args.model_state_file}
```

훈련 상태 딕셔너리

모델 훈련 – 훈련 반복

```
try:
    for epoch_index in range(args.num_epochs):
        train_state['epoch_index'] = epoch_index

        # 훈련 세트에 대한 순회

        # 훈련 세트와 배치 제너레이터 준비, 손실과 정확도를 0으로 설정
        dataset.set_split('train')
        batch_generator = generate_batches(dataset,
                                           batch_size=args.batch_size,
                                           device=args.device)
        running_loss = 0.0
        running_acc = 0.0
        classifier.train()

        for batch_index, batch_dict in enumerate(batch_generator):
            # 훈련 과정은 5단계로 이루어집니다

            # --------------------------------------
            # 단계 1. 그레이디언트를 0으로 초기화합니다
            optimizer.zero_grad()

            # 단계 2. 출력을 계산합니다
            y_pred = classifier(x_in=batch_dict['x_data'].float())

            # 단계 3. 손실을 계산합니다
            loss = loss_func(y_pred, batch_dict['y_target'].float())
            loss_t = loss.item()
            running_loss += (loss_t - running_loss) / (batch_index + 1)

            # 단계 4. 손실을 사용해 그레이디언트를 계산합니다
            loss.backward()

            # 단계 5. 옵티마이저로 가중치를 업데이트합니다
            optimizer.step()
            # --------------------------------------

            # 정확도를 계산합니다
            acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
            running_acc += (acc_t - running_acc) / (batch_index + 1)
```

```
# 검증 세트에 대한 순회

# 검증 세트와 배치 제너레이터 준비, 손실과 정확도를 0으로 설정
dataset.set_split('val')
batch_generator = generate_batches(dataset,
                                   batch_size=args.batch_size,
                                   device=args.device)
running_loss = 0.
running_acc = 0.
classifier.eval()

for batch_index, batch_dict in enumerate(batch_generator):

    # 단계 1. 출력을 계산합니다
    y_pred = classifier(x_in=batch_dict['x_data'].float())

    # 단계 2. 손실을 계산합니다
    loss = loss_func(y_pred, batch_dict['y_target'].float())
    loss_t = loss.item()
    running_loss += (loss_t - running_loss) / (batch_index + 1)

    # 단계 3. 정확도를 계산합니다
    acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
    running_acc += (acc_t - running_acc) / (batch_index + 1)
```

```
training routine: 99% 99/100 [12:10<00:07, 7.17s/it]
split=train: 100% 271/272 [12:21<00:03, 3.76s/it, acc=98.5, epoch=50, loss=0.0656]
split=val: 57% 33/58 [12:23<00:01, 22.13it/s, acc=95, epoch=50, loss=0.152]
```

모델 평가 – 테스트 세트로 평가

[40]
```
# 가장 좋은 모델을 사용해 테스트 세트의 손실과 정확도를 계산합니다
classifier.load_state_dict(torch.load(train_state['model_filename']))
classifier = classifier.to(args.device)

dataset.set_split('test')
batch_generator = generate_batches(dataset,
                                   batch_size=args.batch_size,
                                   device=args.device)
running_loss = 0.
running_acc = 0.
classifier.eval()

for batch_index, batch_dict in enumerate(batch_generator):
    # 출력을 계산합니다
    y_pred = classifier(x_in=batch_dict['x_data'].float())

    # 손실을 계산합니다
    loss = loss_func(y_pred, batch_dict['y_target'].float())
    loss_t = loss.item()
    running_loss += (loss_t - running_loss) / (batch_index + 1)

    # 정확도를 계산합니다
    acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
    running_acc += (acc_t - running_acc) / (batch_index + 1)

train_state['test_loss'] = running_loss
train_state['test_acc'] = running_acc
```

[41]
```
print("테스트 손실: {:.3f}".format(train_state['test_loss']))
print("테스트 정확도: {:.2f}".format(train_state['test_acc']))
```

```
테스트 손실: 0.469
테스트 정확도: 82.81
```

모델 동작 확인

```
[42] def preprocess_text(text):
         text = text.lower()
         text = re.sub(r"([.,!?])", r" #1 ", text)
         text = re.sub(r"[^a-zA-Z.,!?]+", r" ", text)
         return text
```

```
[43] def predict_rating(review, classifier, vectorizer, decision_threshold=0.5):
         """ 리뷰 점수 예측하기

         매개변수:
             review (str): 리뷰 텍스트
             classifier (ReviewClassifier): 훈련된 모델
             vectorizer (ReviewVectorizer): Vectorizer 객체
             decision_threshold (float): 클래스를 나눌 결정 경계
         """
         review = preprocess_text(review)

         vectorized_review = torch.tensor(vectorizer.vectorize(review))
         result = classifier(vectorized_review.view(1, -1))

         probability_value = torch.sigmoid(result).item()
         index = 1
         if probability_value < decision_threshold:
             index = 0

         return vectorizer.rating_vocab.lookup_index(index)
```

```
[44] test_review = "this is a pretty awesome book"

     classifier = classifier.cpu()
     prediction = predict_rating(test_review, classifier, vectorizer, decision_threshold=0.5)
     print("{} -> {}".format(test_review, prediction))
```

```
this is a pretty awesome book -> positive
```

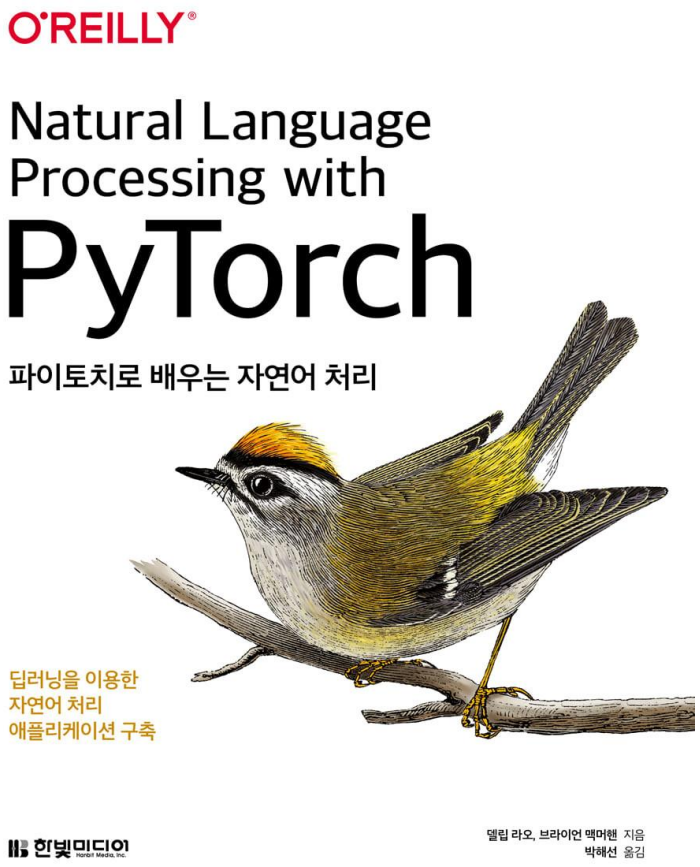
가중치 분석

```
[45]  # 가중치 정렬
      fc1_weights = classifier.fc1.weight.detach()[0]
      _, indices = torch.sort(fc1_weights, dim=0, descending=True)
      indices = indices.numpy().tolist()

      # 긍정적인 상위 20개 단어
      print("긍정 리뷰에 영향을 미치는 단어:")
      print("----------------------------------------")
      for i in range(20):
          print(vectorizer.review_vocab.lookup_index(indices[i]))

      print("====₩n₩n₩n")

      # 부정적인 상위 20개 단어
      print("부정 리뷰에 영향을 미치는 단어:")
      print("----------------------------------------")
      indices.reverse()
      for i in range(20):
          print(vectorizer.review_vocab.lookup_index(indices[i]))
```

```
긍정 리뷰에 영향을 미치는 단어:
----------------------------------------
great
amazing
love
delicious
recommend
able
excellent
favorite
chocolate
lunch
burgh
bit
quick
little
selection
perfect
highly
always
stop
awesome
====
```

```
부정 리뷰에 영향을 미치는 단어:
----------------------------------------
terrible
worst
awful
poor
cold
sorry
water
no
dirty
told
horrible
sad
okay
attitude
overpriced
guess
minutes
bad
better
elsewhere
```

What's next?



『파이토치로 배우는 자연어 처리』

4장 자연어 처리를 위한 피드 포워드 신경망 (다층 퍼셉트론, 합성곱 신경망)

5장 단어와 타입 임베딩