



3장 신경망의 기본 구성 요소

최혜원

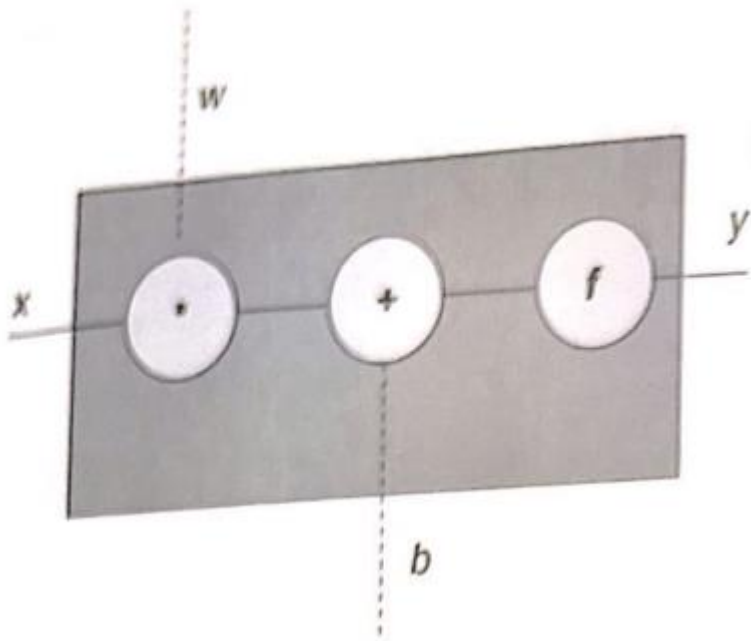
주로 다룰 내용

- ▶ 활성화 함수
- ▶ 손실 함수
- ▶ 옵티마이저
- ▶ 지도 학습 훈련 방법

주로 다룰 내용

- ▶ **활성화 함수**
- ▶ 손실 함수
- ▶ 옵티마이저
- ▶ 지도 학습 훈련 방법

3.1 퍼셉트론 : 가장 간단한 신경망



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

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

```
class Perceptron(nn.Module):
```

```
    """ 퍼셉트론은 하나의 선형 층입니다 """
```

```
    def __init__(self, input_dim):
```

```
        """
```

```
        매개변수:
```

```
            input_dim (int): 입력 특성의 크기
```

```
        """
```

```
        super(Perceptron, self).__init__()
```

```
        self.fc1 = nn.Linear(input_dim, 1)
```

아핀 변환 수행

```
    def forward(self, x_in):
```

```
        """ 퍼셉트론의 정방향 계산
```

```
        매개변수:
```

```
            x_in (torch.Tensor): 입력 데이터 텐서
```

```
            x_in.shape는 (batch, num_features)입니다.
```

```
        반환값:
```

```
            결과 텐서. tensor.shape는 (batch,)입니다.
```

```
        """
```

```
        return torch.sigmoid(self.fc1(x_in)).squeeze()
```

비선형 함수인 시그모이드 함수

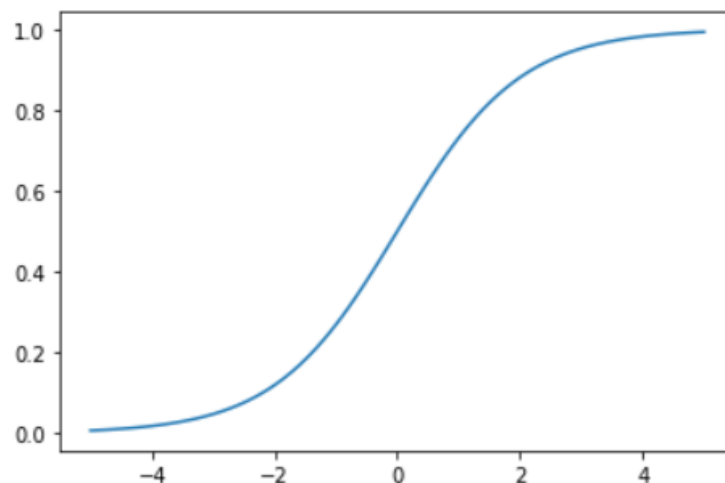
3.2 활성화 함수

▶ 비선형 함수, 신경망에서 데이터의 복잡한 관계를 감지.

1) 시그모이드

```
1 import torch
2 import matplotlib.pyplot as plt
3
4 x = torch.range(-5., 5., 0.1)
5 y = torch.sigmoid(x)
6
7 plt.plot(x.numpy(), y.numpy())
8 plt.show()
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py
after removing the cwd from sys.path.



$$f(x) = \frac{1}{1 + e^{-x}}$$

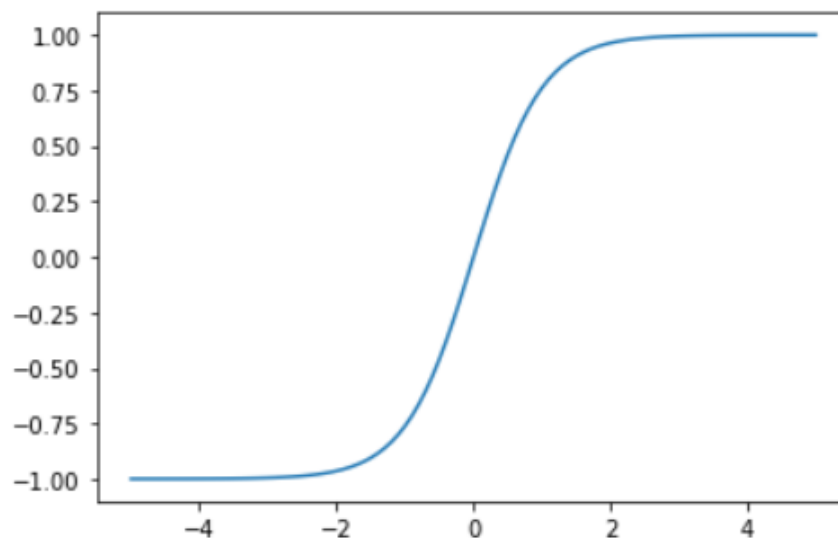
3.2 활성화 함수

2) 하이퍼볼릭 탄젠트 : 시그모이드의 변형

$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

```
1 x = torch.range(-5., 5., 0.1)
2 y = torch.tanh(x)
3
4 plt.plot(x.numpy(), y.numpy())
5 plt.show()
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
"""Entry point for launching an IPython kernel.



3.2 활성화 함수

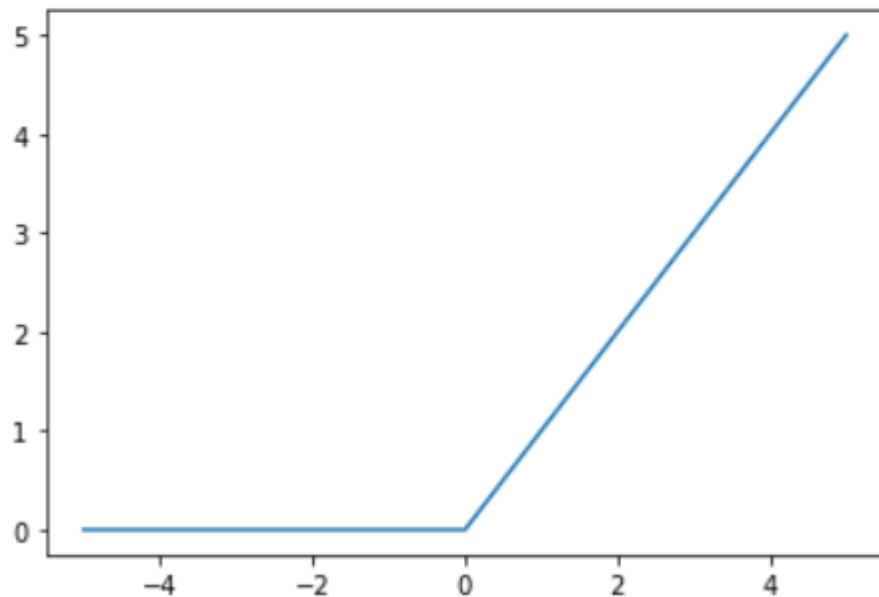
3) ReLU

:음수값을 0으로 자름

```
1 relu = torch.nn.ReLU()  
2 x = torch.range(-5., 5., 0.1)  
3 y = relu(x)  
4  
5 plt.plot(x.numpy(), y.numpy())  
6 plt.show()
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher

$$f(x) = \max(0, x)$$



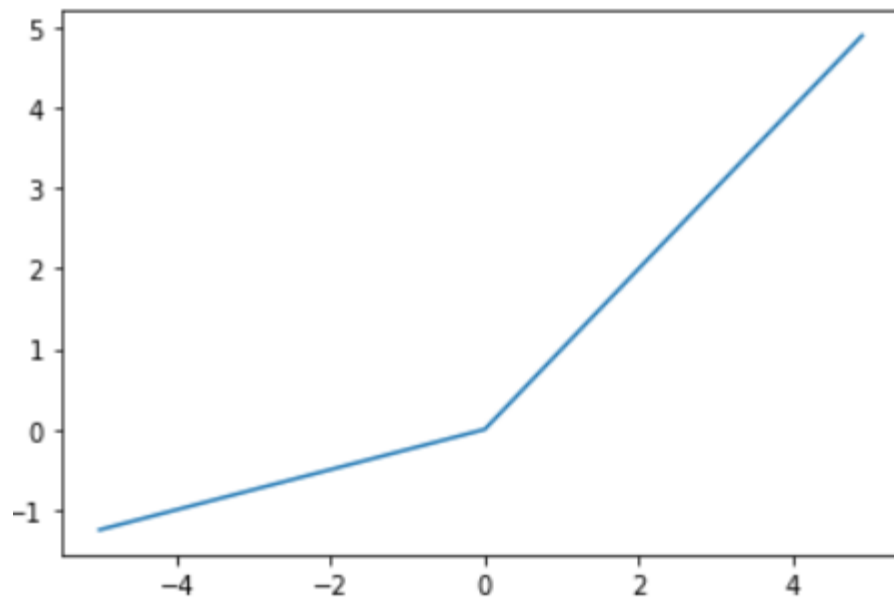
Q. 시간이 지나서 신경망의 특정 출력이 0이 되면 다시 돌아오지 않는다는 문제 존재 : '죽은 렐루'

3.2 활성화 함수

3-1) PReLU

$$f(x) = \max(x, ax)$$

```
1 import torch.nn as nn
2 import matplotlib.pyplot as plt
3
4 prelu = nn.PReLU(num_parameters=1)
5 x = torch.arange(-5., 5., 0.1)
6 y = prelu(x)
7
8 plt.plot(x.numpy(), y.detach().numpy())
9 plt.show()
```



3.2 활성화 함수

4) softmax

분류작업의 출력을 해석할 때 유용

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{f=1}^k e^{x_f}}$$

```
1 import torch.nn as nn
2 import torch
3
4 softmax = nn.Softmax(dim=1)
5 x_input = torch.randn(1, 3)
6 y_output = softmax(x_input)
7 print(x_input)
8 print(y_output)
9 print(torch.sum(y_output, dim = 1))
```

```
tensor([[ 0.2208,  3.0630, -1.2055]])
tensor([[0.0544, 0.9326, 0.0131]])
tensor([1.])
```

총 합: 1

주로 다룰 내용

- ▶ 활성화 함수
- ▶ 손실 함수
- ▶ 옵티마이저
- ▶ 지도 학습 훈련 방법

3.3 손실 함수

- ▶ 훈련 데이터에 대한 예측이 타겟과 얼마나 멀리 떨어져 있는 지 비교하는 함수

1) 평균 제곱 오차 손실(mean squared error) MSE

$$L_{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2$$

```
1 mse_loss = nn.MSELoss()
2 outputs = torch.randn(3, 5, requires_grad=True)
3 targets = torch.randn(3, 5)
4 loss = mse_loss(outputs, targets)
5 print(loss)
```

```
tensor(2.4218, grad_fn=<MseLossBackward>)
```

질문 ! Requires_grad = True
Autograd 에 모든 operation을 추적
할 수 있도록 한다.

3.3 손실 함수

- 2) 범주형 크로스 엔트로피 손실(categorical cross-entropy)
: 일반적으로 출력을 클래스 소속 확률에 대한 예측으로 이해할 수 있는 다중 분류 문제에 사용.

Categorical cross entropy

$$L_{cross_entropy}(y, \hat{y}) = - \sum_i y_i \log \hat{y}_i \quad Loss = - \frac{1}{N} \sum_{j=1}^N \sum_{i=1}^C t_{ij} \log(y_{ij})$$

```
1 import torch
2 import torch.nn as nn
3
4 ce_loss = nn.CrossEntropyLoss()
5 outputs = torch.randn(3, 5, requires_grad = True)
6 print(outputs)
7 targets = torch.tensor([1, 0, 3], dtype= torch.int64)
8 loss = ce_loss(outputs, targets)
9 print(loss)
```

```
tensor([[ 0.5045, -0.3446,  0.9249,  0.3126, -0.2593],
        [-1.1265,  1.1821,  1.0050,  1.2374,  1.1338],
        [-0.5891, -0.5461,  0.6047, -0.5647,  1.2783]], requires_grad=True)
tensor(2.8346, grad_fn=<NLLossBackward>)
```

3.3 손실 함수

3) 이진 크로스 엔트로피 손실(binary cross-entropy) : 이진 분류에 유용

```
1  bce_loss = nn.BCELoss()
2  sigmoid = nn.Sigmoid()
3  probabilities = sigmoid(torch.randn(4, 1, requires_grad=True))
4  targets = torch.tensor([1, 0, 1, 0], dtype=torch.float32).view(4, 1)
5  loss = bce_loss(probabilities, targets)
6  print(probabilities)
7  print(loss)
```



```
tensor([[0.1487],
        [0.6546],
        [0.6048],
        [0.6790]], grad_fn=<SigmoidBackward>)
tensor([[0.1487],
        [0.6546],
        [0.6048],
        [0.6790]], grad_fn=<SigmoidBackward>)
tensor(1.1520, grad_fn=<BinaryCrossEntropyBackward>)
```

3.4 지도 학습 훈련 알아보기

지도 학습 : 레이블된 데이터를 이용하여 지정된 타겟에 새로운 샘플을 매핑하는 방법을 학습.

목표 : 별클래스와 원클래스를 구별하는 것.

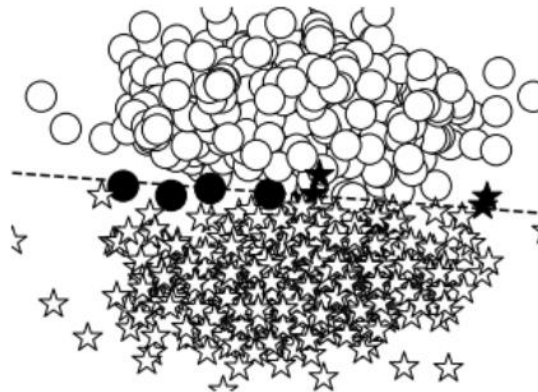
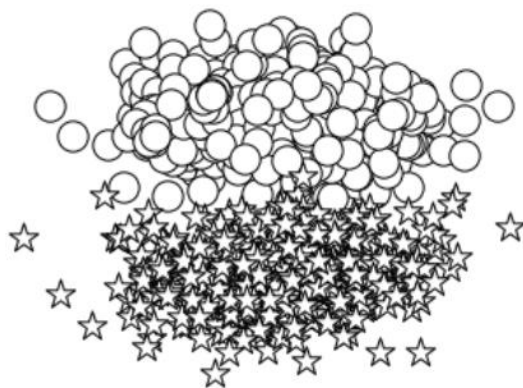
데이터 생성

모델 선택

손실함수 선택

최적화 알고리즘
설정(옵티마이저)

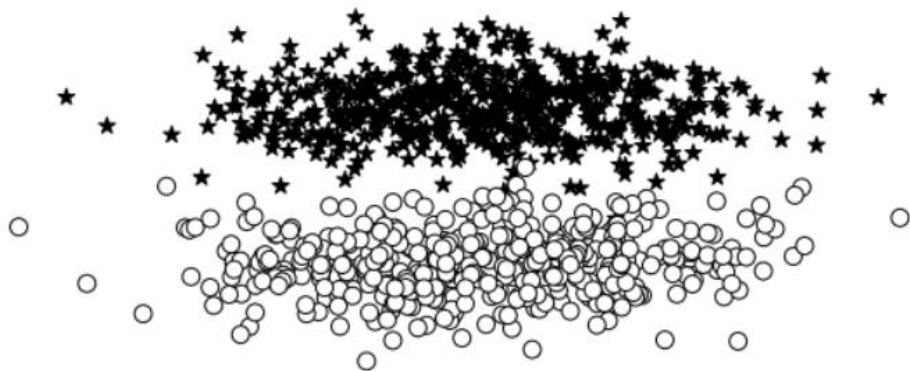
모두 실행



3.4 지도 학습 훈련 알아보기

▶ 데이터 생성

```
1 def get_toy_data(batch_size, left_center=LEFT_CENTER, right_center=RIGHT_CENTER):
2     x_data = []
3     y_targets = np.zeros(batch_size)
4     for batch_i in range(batch_size):
5         if np.random.random() > 0.5:
6             x_data.append(np.random.normal(loc=left_center))
7         else:
8             x_data.append(np.random.normal(loc=right_center))
9             y_targets[batch_i] = 1
10    return torch.tensor(x_data, dtype=torch.float32), torch.tensor(y_targets, dtype=torch.float32)
```



3.4 지도 학습 훈련 알아보기

- ▶ 모델 선택
: 퍼셉트론 사용
출력 = $P(y=1|x)$
- ▶ 확률을 클래스로 변환
결정경계 δ 적용
 $P(y=1|x) > \delta$ 이면 1 //아니면 0

3.4 지도 학습 훈련 알아보기

- ▶ 손실함수 선택
: 이진 크로스 엔트로피
- ▶ 옵티마이저 선택
: 모델의 가중치 업데이트
: Adam 사용

3.4 지도 학습 훈련 알아보기

▶ 모두 실행하여 적용해보기

```
class Perceptron(nn.Module):
    """ 퍼셉트론은 하나의 선형 층입니다 """

    def __init__(self, input_dim):
        """
        매개변수:
            input_dim (int): 입력 특성의 크기
        """
        super(Perceptron, self).__init__()
        self.fc1 = nn.Linear(input_dim, 1)

    def forward(self, x_in):
        """퍼셉트론의 정방향 계산

        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
            x_in.shape는 (batch, num_features)입니다.
        반환값:
            결과 텐서. tensor.shape는 (batch,)입니다.
        """
        return torch.sigmoid(self.fc1(x_in))
```

```
1  lr = 0.01
2  input_dim = 2
3
4  batch_size = 1000
5  n_epochs = 12
6  n_batches = 5
7
8  seed = 1337
9
10 torch.manual_seed(seed)
11 torch.cuda.manual_seed_all(seed)
12 np.random.seed(seed)
13
14 perceptron = Perceptron(input_dim=input_dim)
15 optimizer = optim.Adam(params=perceptron.parameters(), lr=lr)
16 bce_loss = nn.BCELoss()
17
18 losses = []
19
20 x_data_static, y_truth_static = get_toy_data(batch_size)
21 fig, ax = plt.subplots(1, 1, figsize=(10,5))
22 visualize_results(perceptron, x_data_static, y_truth_static, ax=ax, title='Initial Model State')
23 plt.axis('off')
24 #plt.savefig('initial.png')
25
```

3.4 지도 학습 훈련 알아보기

▶ 모두 실행하여 적용해보기

```
26 change = 1.0
27 last = 10.0
28 epsilon = 1e-3
29 epoch = 0
30 while change > epsilon or epoch < n_epochs or last > 0.3:
31     #for epoch in range(n_epochs):
32         for _ in range(n_batches):
33
34             optimizer.zero_grad()
35             x_data, y_target = get_toy_data(batch_size)
36             y_pred = perceptron(x_data).squeeze()
37             loss = bce_loss(y_pred, y_target)
38             loss.backward()
39             optimizer.step()
40
41             loss_value = loss.item()
42             losses.append(loss_value)
43
44             change = abs(last - loss_value)
45             last = loss_value
46
47     fig, ax = plt.subplots(1, 1, figsize=(10,5))
48     visualize_results(perceptron, x_data_static, y_truth_static, ax=ax, epoch=epoch,
49                       title=f"{loss_value}; {change}")
50     plt.axis('off')
51     epoch += 1
52     #plt.savefig('epoch{}_toylearning.png'.format(epoch))
```

3.5 부가적인 훈련 개념

- ▶ 모델 성능 올바르게 측정하기 : 평가 지표
 - 정확도
- ▶ 모델 성능 올바르게 측정하기 : 데이터 분할
 - 훈련, 검증, 테스트셋
 - K겹 교차 검증
- ▶ 훈련 중지 시점 파악하기
 - 조기 종료 : 성능이 더는 좋아지지 않을 때
- ▶ 최적의 파라미터 찾기
 - 하이퍼 파라미터 개선
- ▶ 규제
 - 간단한 것이 복잡한 것보다 낫다.
 - 드롭아웃

3.6 레스토랑 리뷰 감성 분류하기

▶ 데이터

```
args = Namespace(  
    raw_train_dataset_csv="data/yelp/raw_train.csv",  
    raw_test_dataset_csv="data/yelp/raw_test.csv",  
    proportion_subset_of_train=0.1,  
    train_proportion=0.7,  
    val_proportion=0.15,  
    test_proportion=0.15,  
    output_munged_csv="data/yelp/reviews_with_splits_lite.csv",  
    seed=1337  
)
```

원본 데이터

```
# 원본 데이터를 읽습니다  
train_reviews = pd.read_csv(args.raw_train_dataset_csv, header=None, names=['rating', 'review'])
```

```
# 리뷰 클래스 비율이 동일하도록 만듭니다  
by_rating = collections.defaultdict(list)  
for _, row in train_reviews.iterrows():  
    by_rating[row.rating].append(row.to_dict())  
  
review_subset = []  
  
for _, item_list in sorted(by_rating.items()):  
    n_total = len(item_list)  
    n_subset = int(args.proportion_subset_of_train * n_total)  
    review_subset.extend(item_list[:n_subset])  
  
review_subset = pd.DataFrame(review_subset)
```

```
review_subset.head()
```

	rating	review
0	1	Unfortunately, the frustration of being Dr. Go...
1	1	I don't know what Dr. Goldberg was like before...
2	1	I'm writing this review to give you a heads up...
3	1	Wing sauce is like water. Pretty much a lot of...
4	1	Owning a driving range inside the city limits ...

3.6 레스토랑 리뷰 감성 분류하기

▶ 데이터

```
args = Namespace(  
    raw_train_dataset_csv="data/yelp/raw_train.csv",  
    raw_test_dataset_csv="data/yelp/raw_test.csv",  
    proportion_subset_of_train=0.1,  
    train_proportion=0.7,  
    val_proportion=0.15,  
    test_proportion=0.15,  
    output_munged_csv="data/yelp/reviews_with_splits_lite.csv",  
    seed=1337  
)
```

분할 데이터

훈련, 검증, 테스트를 만들기 위해 별점을 기준으로 나눕니다.
by_rating = collections.defaultdict(list) **Key값이 없을 경우 미리지정해놓은 default 값 반환**
for _, row *in* review_subset.iterrows(): **여기선 list 메소드 반환**
 by_rating[row.rating].append(row.to_dict())

분할 데이터를 만듭니다.

final_list = []
np.random.seed(args.seed)

for _, item_list *in* sorted(by_rating.items()):

np.random.shuffle(item_list)

n_total = len(item_list)
 n_train = int(args.train_proportion * n_total)
 n_val = int(args.val_proportion * n_total)
 n_test = int(args.test_proportion * n_total)

데이터 포인터에 분할 속성을 추가합니다

for item *in* item_list[:n_train]:
 item['split'] = 'train'

for item *in* item_list[n_train:n_train+n_val]:
 item['split'] = 'val'

for item *in* item_list[n_train+n_val:n_train+n_val+n_test]:
 item['split'] = 'test'

최종 리스트에 추가합니다

final_list.extend(item_list)

분할 데이터를 데이터 프레임으로 만듭니다

final_reviews = pd.DataFrame(final_list)

3.6 레스토랑 리뷰 감성 분류하기

▶ 데이터

```
# 리뷰를 전처리합니다
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

final_reviews.review = final_reviews.review.apply(preprocess_text)
```

```
final_reviews['rating'] = final_reviews.rating.apply({1: 'negative', 2: 'positive'}.get)
```

```
final_reviews.head()
```

	rating	review	split
0	negative	all i can say is that a i had no other option ...	train
1	negative	i went here once when my long time stylist mov...	train
2	negative	i don t know why i stopped here for lunch this...	train
3	negative	did i order the wrong thing ? or maybe it was ...	train
4	negative	i went here for restaurant week . the restaura...	train

3.6 레스토랑 리뷰 감성 분류하기

▶ Vocabulary

```
def __init__(self, token_to_idx=None, add_unk=True, unk_token="<UNK>"):
    """
    매개 변수:
        token_to_idx (dict): 기존 토큰-인덱스 매핑 딕셔너리
        add_unk (bool): UNK 토큰을 추가할지 지정하는 플래그
        unk_token (str): Vocabulary에 추가할 UNK 토큰
    """

    if token_to_idx is None:
        token_to_idx = {}
    self._token_to_idx = token_to_idx

    self._idx_to_token = {idx: token
                          for token, idx in self._token_to_idx.items()}

    self._add_unk = add_unk
    self._unk_token = unk_token

    self.unk_index = -1
    if add_unk:
        self.unk_index = self.add_token(unk_token)
```

```
38 def add_token(self, token):
39     """ 토큰을 기반으로 매핑 딕셔너리를 업데이트합니다
40
41     매개 변수:
42         token (str): Vocabulary에 추가할 토큰
43     반환값:
44         index (int): 토큰에 상응하는 정수
45     """
46     if token in self._token_to_idx:
47         index = self._token_to_idx[token]
48     else:
49         index = len(self._token_to_idx)
50         self._token_to_idx[token] = index
51         self._idx_to_token[index] = token
52     return index
```

```
def lookup_token(self, token):
    """ 토큰에 대응하는 인덱스를 추출합니다.
    토큰이 없으면 UNK 인덱스를 반환합니다.
```

```
매개 변수:
    token (str): 찾을 토큰
반환값:
    index (int): 토큰에 해당하는 인덱스
노트:
```

```
UNK 토큰을 사용하려면 (Vocabulary에 추가하기 위해)
`unk_index`가 0보다 커야 합니다.
```

```
"""
if self.unk_index >= 0:
    return self._token_to_idx.get(token, self.unk_index)
else:
    return self._token_to_idx[token]
```

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

```
매개 변수:
    index (int): 찾을 인덱스
반환값:
    token (str): 인덱스에 해당하는 토큰
예러:
    KeyError: 인덱스가 Vocabulary에 없을 때 발생합니다.
```

```
"""
if index not in self._idx_to_token:
    raise KeyError("Vocabulary에 인덱스(%d)가 없습니다." % index)
return self._idx_to_token[index]
```


3.6 레스토랑 리뷰 감성 분류하기

▶ Vectorizer

토큰을 순회하며 각 토큰을 정수로 바꾸기

@classmethod

```
def from_dataframe(cls, review_df, cutoff=25):
    """ 데이터셋 데이터프레임에서 Vectorizer 객체를 만듭니다

    매개변수:
        review_df (pandas.DataFrame): 리뷰 데이터셋
        cutoff (int): 빈도 기반 필터링 설정값
    반환값:
        ReviewVectorizer 객체
    """

    review_vocab = Vocabulary(add_unk=True)
    rating_vocab = Vocabulary(add_unk=False)

    # 점수를 추가합니다
    for rating in sorted(set(review_df.rating)):
        rating_vocab.add_token(rating)

    # count > cutoff인 단어를 추가합니다
    word_counts = Counter()
    for review in review_df.review:
        for word in review.split(" "):
            if word not in string.punctuation:
                word_counts[word] += 1

    for word, count in word_counts.items():
        if count > cutoff:
            review_vocab.add_token(word)

    return cls(review_vocab, rating_vocab)
```

```
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)

    for token in review.split(" "):
        if token not in string.punctuation:
            one_hot[self.review_vocab.lookup_token(token)] = 1

    return one_hot
```

3.6 레스토랑 리뷰 감성 분류하기

▶ 퍼셉트론 분류기

```
1 class ReviewClassifier(nn.Module):
2     """ 간단한 퍼셉트론 기반 분류기 """
3     def __init__(self, num_features):
4         """
5         매개변수:
6         | num_features (int): 입력 특성 벡트의 크기
7         """
8         super(ReviewClassifier, self).__init__()
9         self.fc1 = nn.Linear(in_features=num_features,
10                               | out_features=1)
11
12     def forward(self, x_in, apply_sigmoid=False):
13         """ 분류기의 정방향 계산
14
15         매개변수:
16         | x_in (torch.Tensor): 입력 데이터 텐서
17         |   x_in.shape는 (batch, num_features)입니다.
18         | apply_sigmoid (bool): 시그모이드 활성화 함수를 위한 플래그
19         |   크로스-엔트로피 손실을 사용하려면 False로 지정합니다
20         반환값:
21         | 결과 텐서. tensor.shape은 (batch,)입니다.
22         """
23         y_out = self.fc1(x_in).squeeze()
24         if apply_sigmoid:
25             y_out = torch.sigmoid(y_out)
26         return y_out
```

3.6 레스토랑 리뷰 감성 분류하기

▶ 모델 하러

```

1  args = Namespace(
2      # 날짜와 경로 정보
3      frequency_cutoff=25,
4      model_state_file='model.pth',
5      review_csv='data/yelp/reviews_with_splits_lite.csv',
6      # review_csv='data/yelp/reviews_with_splits_full.csv',
7      save_dir='model_storage/ch3/yelp/',
8      vectorizer_file='vectorizer.json',
9      # 모델 하이퍼파라미터 없음
10     # 훈련 하이퍼파라미터
11     batch_size=128,
12     early_stopping_criteria=5,
13     learning_rate=0.001,
14     num_epochs=100,
15     seed=1337,
16     # 실행 옵션
17     catch_keyboard_interrupt=True,
18     cuda=True,
19     expand_filepaths_to_save_dir=True,
20     reload_from_files=False,
21 )
22

```

```

1  if args.reload_from_files:
2      # 체크포인트에서 훈련을 다시 시작
3      print("데이터셋과 Vectorizer를 로드합니다")
4      dataset = ReviewDataset.load_dataset_and_load_vectorizer(args.review_csv,
5                                                                args.vectorizer_file)
6  else:
7      print("데이터셋을 로드하고 Vectorizer를 만듭니다")
8      # 데이터셋과 Vectorizer 만들기
9      dataset = ReviewDataset.load_dataset_and_make_vectorizer(args.review_csv)
10     dataset.save_vectorizer(args.vectorizer_file)
11     vectorizer = dataset.get_vectorizer()
12
13     classifier = ReviewClassifier(num_features=len(vectorizer.review_vocab))
14     classifier = classifier.to(args.device)
15
16     loss_func = nn.BCEWithLogitsLoss()
17     optimizer = optim.Adam(classifier.parameters(), lr=args.learning_rate)
18     scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer,
19                                                       mode='min', factor=0.5,
20                                                       patience=1)
21     train_state = make_train_state(args)

```

3.6 레스토랑 리뷰 감성 분류하기

▶ 훈련반복

```
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)

# 진행 바 업데이트
train_bar.set_postfix(loss=running_loss,
                      acc=running_acc,
                      epoch=epoch_index)
train_bar.update()

train_state['train_loss'].append(running_loss)
train_state['train_acc'].append(running_acc)
```

3.6 레스토랑 리뷰 감성 분류하기

▶ 평가

```
# 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)

    val_bar.set_postfix(loss=running_loss,
                        acc=running_acc,
                        epoch=epoch_index)
    val_bar.update()

train_state['val_loss'].append(running_loss)
train_state['val_acc'].append(running_acc)
```

3.6 레스토랑 리뷰 감성 분류하기

▶ 테스트

```
1  # 가장 좋은 모델을 사용해 테스트 세트의 손실과 정확도를 계산합니다
2  classifier.load_state_dict(torch.load(train_state['model_filename']))
3  classifier = classifier.to(args.device)
4
5  dataset.set_split('test')
6  batch_generator = generate_batches(dataset,
7                                   batch_size=args.batch_size,
8                                   device=args.device)
9  running_loss = 0.
10 running_acc = 0.
11 classifier.eval()
12
13 for batch_index, batch_dict in enumerate(batch_generator):
14     # 출력을 계산합니다
15     y_pred = classifier(x_in=batch_dict['x_data'], float())
16
17     # 손실을 계산합니다
18     loss = loss_func(y_pred, batch_dict['y_target'], float())
19     loss_t = loss.item()
20     running_loss += (loss_t - running_loss) / (batch_index + 1)
21
22     # 정확도를 계산합니다
23     acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
24     running_acc += (acc_t - running_acc) / (batch_index + 1)
25
26 train_state['test_loss'] = running_loss
27 train_state['test_acc'] = running_acc
```

3.6 레스토랑 리뷰 감성 분류하기

▶ 예측 출력

```
1 def predict_rating(review, classifier, vectorizer, decision_threshold=0.5):
2     """ 리뷰 점수 예측하기
3
4     매개변수:
5     review (str): 리뷰 텍스트
6     classifier (ReviewClassifier): 훈련된 모델
7     vectorizer (ReviewVectorizer): Vectorizer 객체
8     decision_threshold (float): 클래스를 나눌 결정 경계
9     """
10    review = preprocess_text(review)
11
12    vectorized_review = torch.tensor(vectorizer.vectorize(review))
13    result = classifier(vectorized_review.view(1, -1))
14
15    probability_value = torch.sigmoid(result).item()
16    index = 1
17    if probability_value < decision_threshold:
18        index = 0
19
20    return vectorizer.rating_vocab.lookup_index(index)
```

```
1 test_review = "this is a pretty awesome book"
2
3 classifier = classifier.cpu()
4 prediction = predict_rating(test_review, classifier, vectorizer, decision_threshold=0.5)
5 print("{} -> {}".format(test_review, prediction))
```

this is a pretty awesome book -> positive

3.7 요약

- ▶ 가장 간단한 신경망 모델인 퍼셉트론
- ▶ 활성화함수, 손실함수
- ▶ 엘프의 레스토랑 리뷰