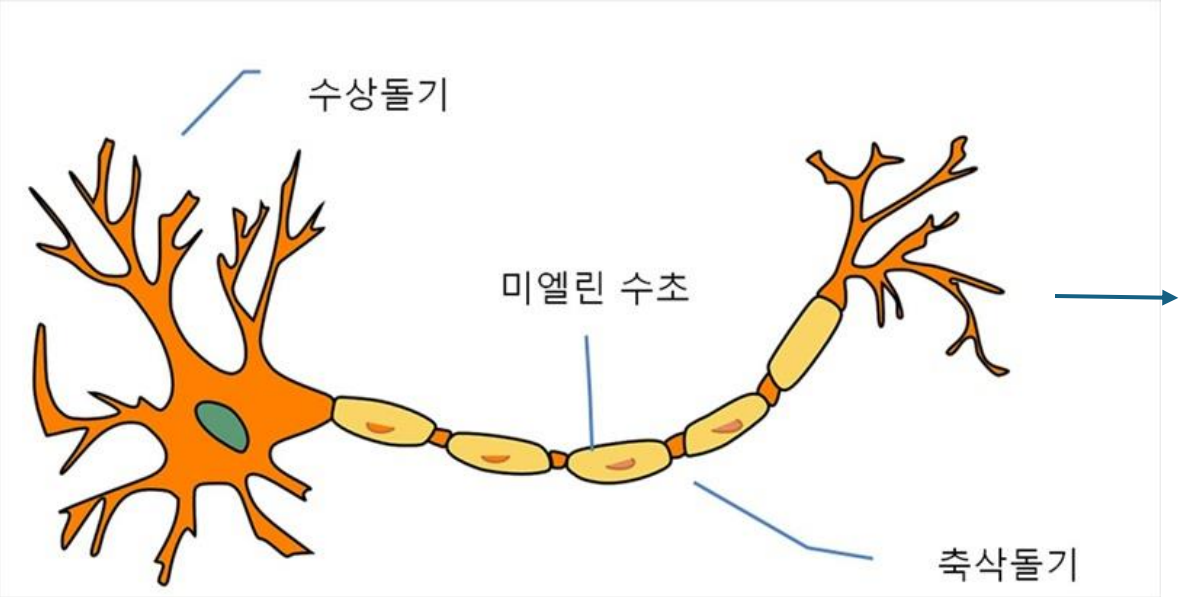
3장
신경망의 기본 구성 요소

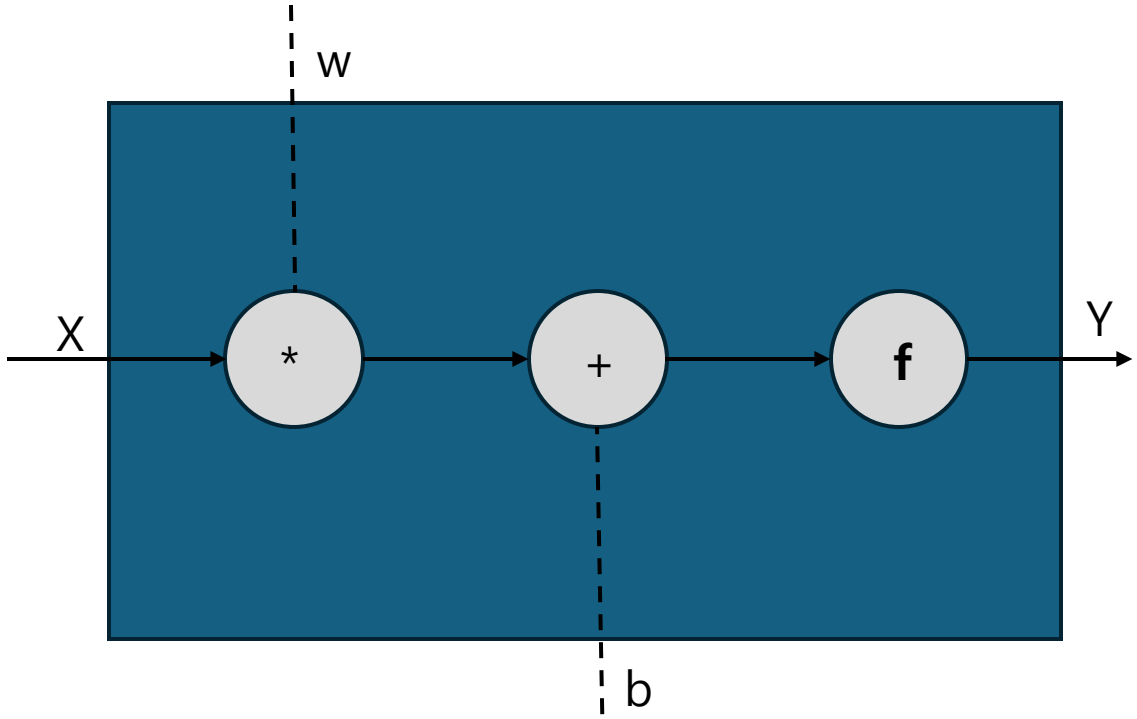
목차

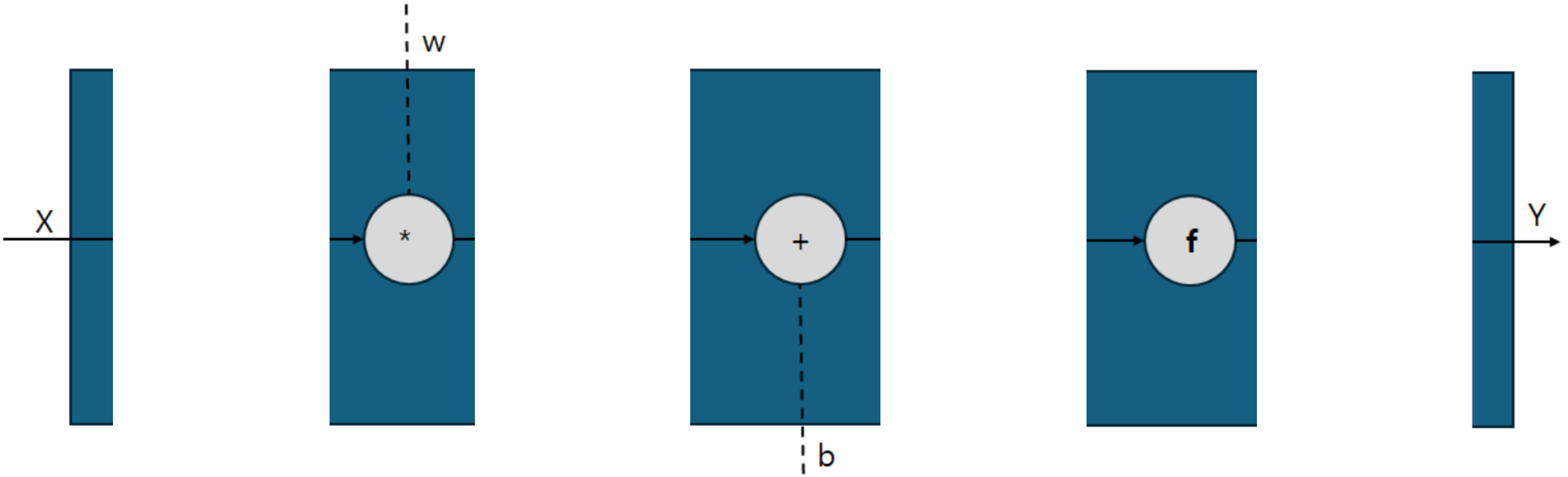
- 퍼셉트론
- 활성화함수
- 손실함수
- 지도학습 훈련 알아보기
- 부가적인 훈련 개념
- 실습
  - Yelp 감성 분류기(예제)

퍼셉트론

- 생물학적 뉴런을 본떠 만든 가장 간단한 신경망







입력 (Input)
여러 개의 입력 벡터

가중치(weight)
와 입력 벡터의
dot product
데이터에서 학습

절편(bias)
데이터에서 학습

활성화 함수(f)
f(w*x+b)
설계자가 결정

출력 (output)

아핀변환
w*x+b

퍼셉트론

```python
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)).squeeze()
```

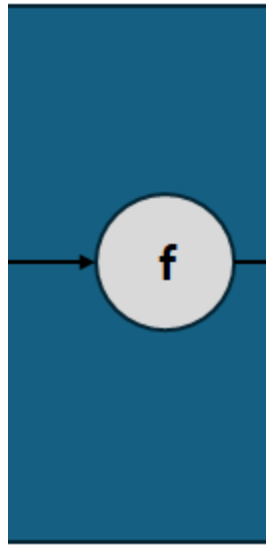
코드 3-1

이 코드의 예제로 알 수 있는 점.

- Linear
  정방향 계산에 필요한 작업 수행
  (아핀 변환)

활성화 함수

- 데이터의 복잡한 관계를 감지하는 데 사용
- 선형 함수 vs 비선형 함수.
  - 시그모이드
  - 탄젠트 하이퍼 볼릭
  - 렐루
  - 소프트맥스



활성화 함수(f)

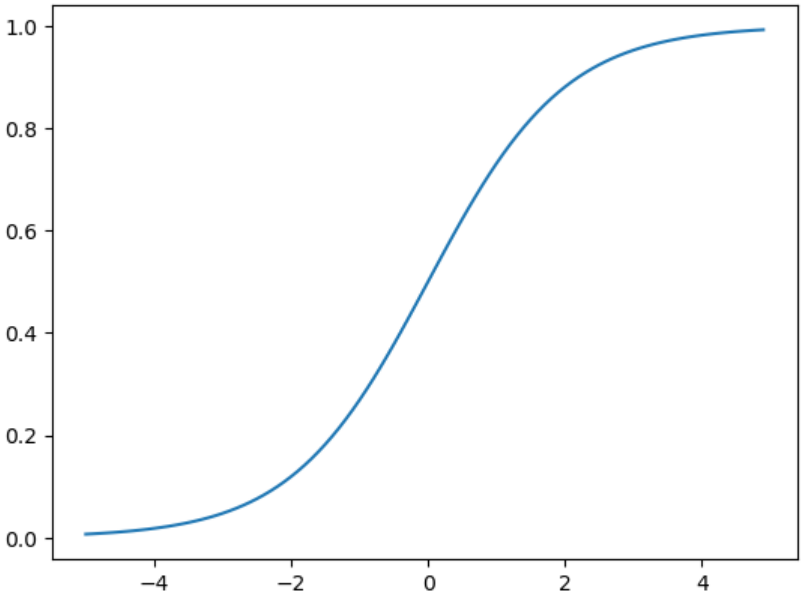
시그모이드 sigmoid

- 초창기 멤버.
- 출력을 확률로 압축할 때 사용
  - 그레디언트 소실
  - 그레디언트 폭주

공식: $sigmoid(x) = \frac{1}{e^{-x}+1}$

```python
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()
```

코드 3-2



코드 3-2 실행 결과

하이퍼볼릭 탄젠트 tanh

- 시그모이드 함수의 변환
- 음수까지 출력 가능

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

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

$$= \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$= \frac{e^x(1 - e^{-2x})}{e^x(1 + e^{-2x})}$$

$$= sigmoid(2x) - \frac{e^{-2x}}{1 + e^{-2x}}$$

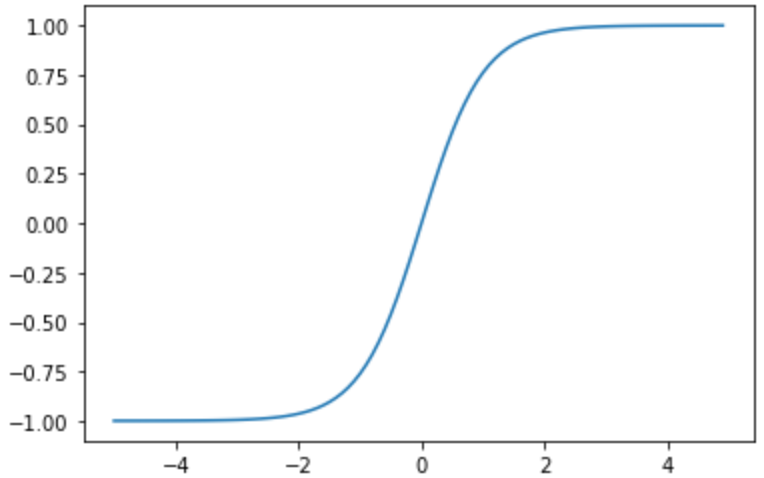
$$= sigmoid(2x) - \frac{e^{-2x} + 1 - 1}{1 + e^{-2x}}$$

$$= 2 \times sigmoid(2x) - 1$$

Tanh 공식 정리

```python
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()
```

코드 3-3



코드 3-3 실행 결과

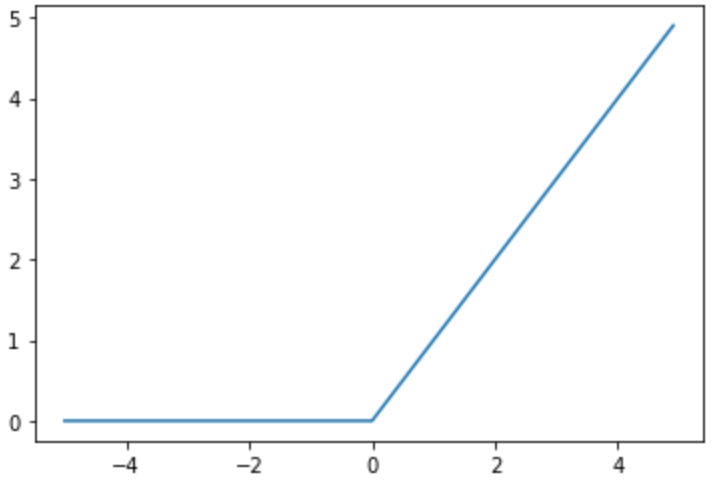
렐루 ReLU

- 음수 값을 자르는 활성화 함수.
  - 해결 : 그레디언트 소실 문제
  - 문제 : 죽은 ReLU
    (출력이 0이 되면 끝)

공식 : $ReLU(x) = \max(0, x)$

```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()
```

코드 3-4



코드 3-4 실행 결과

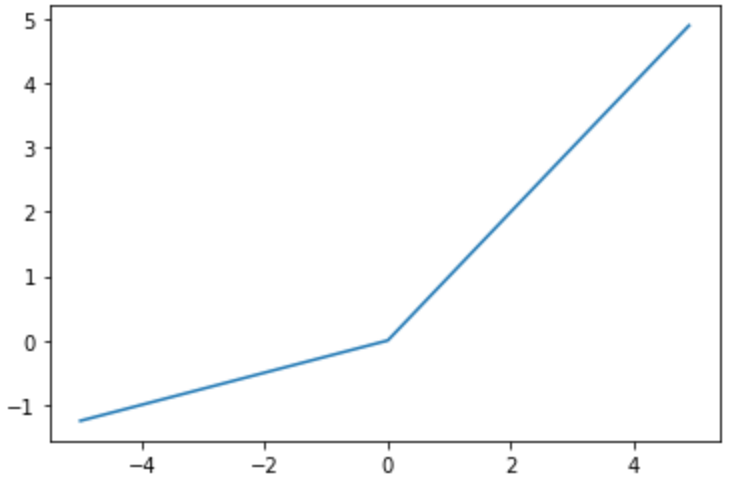
렐루 PReLU

- Dying relu를 해결하기 위해 개발
- 학습되는 파라미터 α를 가짐.

공식 : $PReLU(x) = \max(x, \alpha x)$

```python
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()
```

코드 3-5



코드 3-5 실행 결과

소프트 맥스 Softmax

- K개 클래스에 대한 이산확률 분포를 출력
- 출력의 합이 1
- 분류 작업의 출력 해석에 유용
- 한계점
  - 지수 함수를 사용하기에 시그모이드 함수가 가지는 단점을 가질 수 있다.

공식 : $Softmax(x_i) = \frac{e^{xmax}}{\sum_{f=1}^{k} e^{x_f}} = \frac{e^{x_i}\, e^{xmax}}{\sum_{f=1}^{k} e^{x_f}\, e^{xmax}} = \frac{e^{x_i - e^{xmax}}}{\sum_{f=1}^{k} e^{x_f - e^{xmax}}}$

```
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))
```

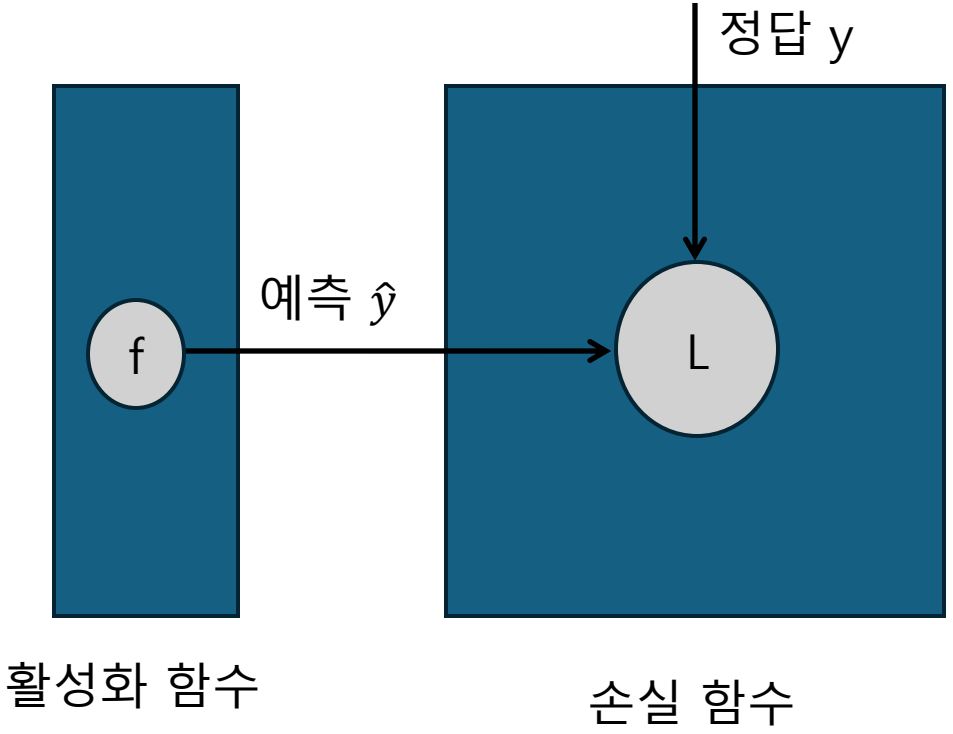
코드 3-6

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

코드 3-6 실행결과

손실함수

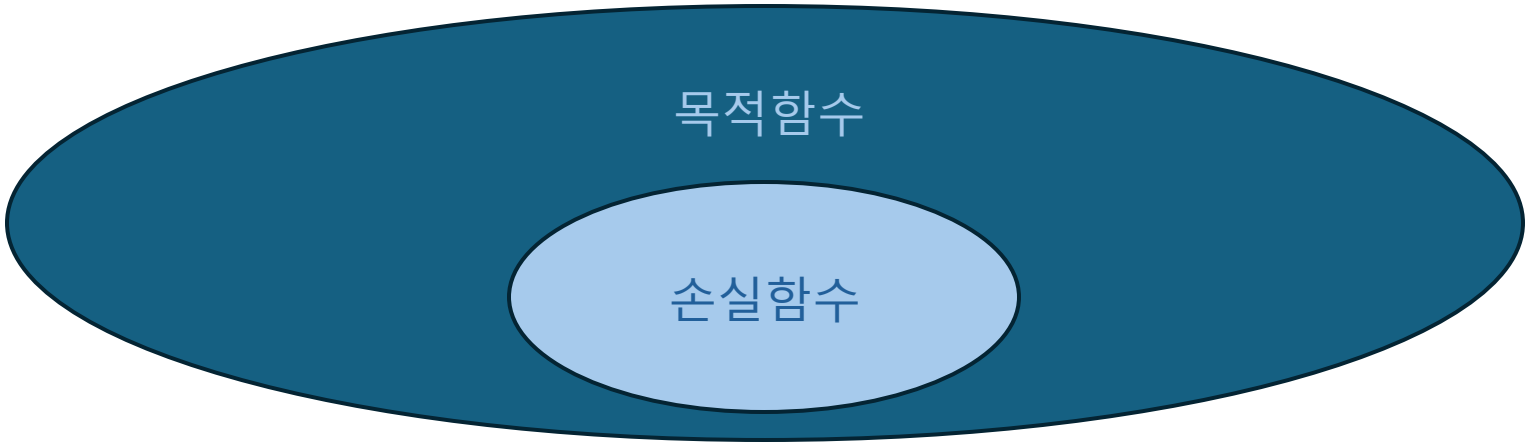
- 정답과 예측이 다른 정도를 결과로 출력
  - 평균 제곱 오차
  - 범주형 크로스 엔트로피 손실
  - 이진 크로스 엔트로피 손실



손실함수? 목적함수?

- 목적 함수 : 모델이 최적화 하고자 하는 전체적인 목표
- 손실 함수 : 모델이 특정 작업에서 최적으로 학습되도록 유도

-> 범위가 다르다.



평균 제곱오차 손실 MSE

- 정답과 예측이 연속 값인 회귀 문제에서 사용
- 정답과 예측의 차이를 제곱하여 평균한 값을 출력

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

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

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

```
tensor(1.6031, grad_fn=<MseLossBackward0>)
```

코드 3-7

범주형 크로스 엔트로피 손실 CE

- 클래스 소속 확률에 대한 예측, 다중 분류 문제에 사용.

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

ce_loss = nn.CrossEntropyLoss()
outputs = torch.randn(3, 5, requires_grad=True)
targets = torch.tensor([1, 0, 3], dtype=torch.int64)
loss = ce_loss(outputs, targets)
loss.backward()
print (loss)
```

```
tensor(2.5949, grad_fn=<NllLossBackward0>)
```

공식 : $L_{cross\_entropy}(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$

코드 3-8

로그 소프트 맥스 LogSoftmax

- K개 클래스에 대한 이산확률 분포를 출력
- 분류 작업의 출력 해석에 유용

공식 : $\log Softmax_{(x_i)} = log\left(\frac{e^{x_i} - e^{x_{max}}}{\sum_{f=1}^{k} e^{x_f} - e^{x_{max}}}\right)$

$$= (x_i - x_{max}) - log \sum_{f=1}^{k} e^{x_f} - e^{x_{max}}$$

이진 크로스 엔트로피 손실 BCE

- 크로스 엔트로피 손실과 달리 이진 분류 문제에 사용.
- 이진 확률 벡터, 정답 벡터를 사용해 손실 계산

```python
bce_loss = nn.BCELoss()
sigmoid = nn.Sigmoid()
probabilities = sigmoid(torch.randn(4, 1, requires_grad=True))
print(probabilities)

targets = torch.tensor([1, 0, 1, 0], dtype=torch.float32).view(4, 1)
loss = bce_loss(probabilities, targets)
loss.backward()
print(loss)
```

```
tensor([[0.5344],
        [0.4040],
        [0.6256],
        [0.5822]], grad_fn=<SigmoidBackward0>)
tensor(0.6215, grad_fn=<BinaryCrossEntropyBackward0>)
```

공식 : $L_{BCE}(y, \hat{y}) = -(y\log\hat{y} + (1-y)\log(1-\hat{y}))$

코드 3-9

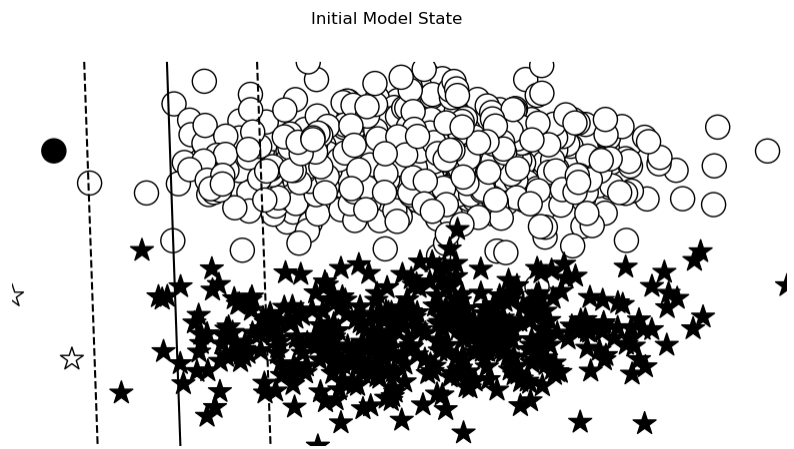
지도 학습 훈련 알아보기

- 지도 학습
  - 타깃에 새로운 샘플을 매핑하는 방법을 학습하는 방식.
- 실습 요약
  - 모델 예측, 손실함수를 사용해 모델의 파라미터를 그레디언트 기반의 방법으로 최적화
  - 2차원 데이터 포인트를 클래스 두 개 중 하나로 분류

예제 데이터 만들기

```
def get_toy_data(batch_size, left_center=LEFT_CENTER,right_center=RIGHT_CENTER):
    x_data = []
    y_targets = np.zeros(batch_size)
    for batch_i in range(batch_size):
        if np.random.random() > 0.5:
            x_data.append(np.random.normal(loc=left_center))
        else:
            x_data.append(np.random.normal(loc=right_center))
            y_targets[batch_i] = 1
    return torch.tensor(x_data, dtype=torch.float32)#
            ,torch.tensor(y_targets, dtype=torch.float32)
```

get_toy_data 코드



생성한 데이터 시각화

- get_toy_data() 함수를 통해 데이터를 제작합니다.
  - 분포 평균과 batch size를 입력 받습니다
  - 0과 1 사이의 무작위 생성 값이 0.5보다 큰지 확인하여 각 분포 평균을 평균으로 하는 정규 분포 데이터를 생성하여 추가합니다.

모델 선택

```python
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))
```

Perceptron 모델코드

- 사용한 모델 : 퍼셉트론
- 활성화 함수 : 시그모이드

확률을 클래스로 변환하기

```
def visualize_results(perceptron, x_data, y_truth, n_samples=1000,
                      ax=None, epoch=None, title='',
                      levels=[0.3, 0.4, 0.5], linestyles=['--', '-', '--']):
    y_pred = perceptron(x_data)
    y_pred = (y_pred > 0.5).long().data.numpy().astype(np.int32)

    x_data = x_data.data.numpy()
    y_truth = y_truth.data.numpy().astype(np.int32)

    n_classes = 2
```

Visualize_results 코드

- 예측 확률이 0.5이상이면 True 아니라면, False로 클래스로 변환 하는 부분.

손실 함수 선택

```
perceptron = Perceptron(input_dim=input_dim)
optimizer = optim.Adam(params=perceptron.parameters(), lr=lr)
bce_loss = nn.BCELoss()
```

손실함수 선택 부분 코드

- 활성화 함수로 시그모이드 함수를 사용하였으므로,
  이진 분류 문제에
  주로 사용되는
  BCE 이진 교차 엔트로피 손실 함수를 선택.

옵티마이저 선택

```
lr = 0.01
input_dim = 2

perceptron = Perceptron(input_dim=input_dim)
optimizer = optim.Adam(params=perceptron.parameters(), lr=lr)
bce_loss = nn.BCELoss()
```

옵티마이저 선택 부분 코드

- 일반적으로 가장 효과적으로 동작하는 옵티마이저인 Adam 옵티마이저를 사용
- 예측과 타깃 사이의 오차 신호를 이용해 모델의 가중치 갱신
- 기본 학습률(learning rate)는 0.001

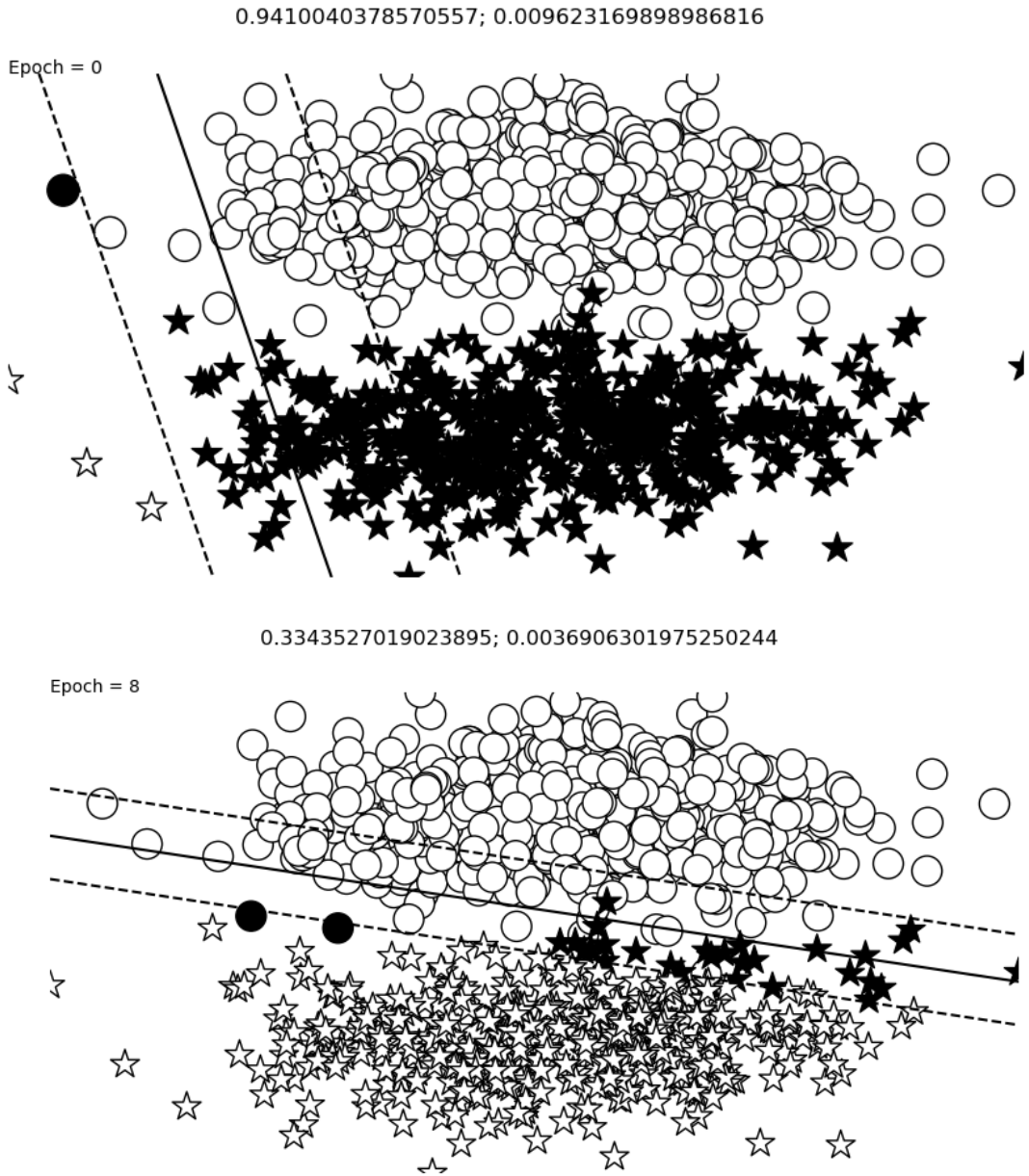
모두 합치기

```
change = 1.0
last = 10.0
epsilon = 1e-3
epoch = 0
while change > epsilon or epoch < n_epochs or last > 0.3:
#for epoch in range(n_epochs):
    for _ in range(n_batches):

        optimizer.zero_grad()
        x_data, y_target = get_toy_data(batch_size)
        y_pred = perceptron(x_data).squeeze()
        loss = bce_loss(y_pred, y_target)
        loss.backward()
        optimizer.step()

        loss_value = loss.item()
        losses.append(loss_value)

        change = abs(last - loss_value)
        last = loss_value
            |
    fig, ax = plt.subplots(1, 1, figsize=(10,5))
    visualize_results(perceptron, x_data_static, y_truth_static,
                      ax=ax, epoch=epoch,
                      title=f"{loss_value}; {change}")
    plt.axis('off')
    epoch += 1
```

- 그레디언트 기반의 지도학습
  - 손실의 그레디언트
= 손실값의 순간 변화율.
= 파라미터를 바꿔야하는 정도
  - Pytorch loss객체의 backward()를 이용, 손실을 반복해서 전파하여 각 파라미터에 대한 그레디언트를 계산
  - Adam 옵티마이저 step() 함수로 파라미터에 그레디언트를 갱신하는 방법을 지시.

훈련 과정 및 결과

- 흰색이 바르게 분류한 것이고, 검은 색으로 표시된 것이 틀리게 분류한 것
- 점점 손실값을 줄이는 방향으로 결정 경계가 변경되고 있음.



부가적인 훈련개념

- 훈련 과정에 중요한 부가 개념
  - 평가지표
  - 데이터 분할
  - 조기종료
  - 하이퍼파라미터
  - 규제

모델 성능 올바르게 측정하기 : 평가지표

- 모델의 훈련에 사용하지 않은 데이터로 성능 측정
- 평가지표
  - 정확도 : 전체 예측 중 정답의 비율
  - 정밀도 : a라 예측한 샘플 중 실제 a인 샘플의 비율
  - 재현율 : 실제 a인 샘플중 정확히 a라 예측한 샘플의 비율
  - F1 score : 재현율과 정밀도의 조화평균, 불균형한 클래스 분포에 사용
  - ROC-AUC : 이진분류에서 사용.
  - 손실 함수 : 최소화 되도록.

모델 성능 올바르게 측정하기 :데이터 분할

- 일반화를 잘 시키는 것이 최종 목표.
- 유한한 샘플에 과적합되지 않도록 데이터를 분할하여 사용
  - 훈련, 검증, 테스트 3개의 데이터셋으로 랜덤하게 샘플링
  - K-fold 교차검증

훈련 중지 시점 파악하기

- 시간 절약과 모델의 과적합을 피하기위해 훈련 중지 시점 설정
  - 조기종료 : 에포크마다 성능을 기록하여 성능이 좋아지지 않으면 훈련 종료
  - 인내 : 훈련을 조기 종료하기 전 기다린 에포크 횟수.

최적의 하이퍼파라미터 찾기

- 하이퍼파라미터 : 모델의 파라미터 개수와 값에 영향을 미치는 모든 모델 설정
  - 손실함수
  - 옵티마이저, 옵티마이저 학습률
  - 인내할 에포크 수
  - 층 크기
  - 규제

규제

- 모델의 복잡성 제어와 과적합 방지 위한 정규화 기법
  - L1 규제 (Lasso Regression) : 특정 특성 외에 무시
    간단한 모델 유도, 희소한 솔루션을 만드는데 사용.
  - L2 규제 (Ridge Regression) : 모든 특성을 반영
    들쑥 날쑥하기보다는 완만하고 부드럽게 만드는 제약.
  - Elastic net : L1+L2

Yelp 레스토랑 리뷰 감성 분류하기

- Pytorch Yelp review dataset
- Vocabulary, Vectorizer, DataLoader 클래스
- 퍼셉트론 분류기
- 모델 훈련
- 평가, 추론, 분석

Pytorch Yelp review dataset

- Torch의 dataset 클래스를 상속하여 ReviewDataset 클래스를 제작.
- 데이터 포인트 하나에 벡터화 로직을 적용하는 데이터셋을 구현

```
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}
```

ReviewDataset 클래스 코드 일부분

Vocabulary 클래스

- 토큰을 정수로 매핑
- UNK 토큰을 통해 훈련에 본적 없는 토큰 처리와 Vocabulary가 사용하는 메모리를 제한합니다.

```
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:
        index = self._token_to_idx[token]
    else:
        index = len(self._token_to_idx)
        self._token_to_idx[token] = index
        self._idx_to_token[index] = token
    return index
```

Vocabulary 클래스 코드 일부분

Vectorizer 클래스

- 입력 데이터 포인트의 토큰을 순회하며 토큰을 정수로 변환
- Cutoff에 지정한 수보다 빈도가 높은 토큰만 Vocabulary 객체에 추가
- One-hot encoding을 사용

```python
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)
```

Vectorizer 클래스 코드 일부분

DataLoader 클래스

- 변환한 데이터들을 훈련하기 편하도록 미니배치로 모아줌

```
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
```

DataLoader 클래스 코드 일부분

퍼셉트론 분류기

- 이진 분류 문제이므로 out_features =1
- 이진 분류 문제에 적합한 손실 함수는 이진 크로스 엔트로피 손실 함수 이므로 안정된 계산을 하기위해 apply_sigmoid=False로 설정 BCEWithLogitsLoss()라는 손실 함수를 대신 사용.

```
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
```

ReviewClassifier 클래스 코드 일부분

모델 훈련

- 헬퍼 함수
  - 모델의 훈련 상태를 저장하고 갱신하는 함수
  - 성능에 따라 모델을 저장하거나 훈련을 조기종료 하기도 함

```python
# 적어도 한 번 모델을 저장합니다
if train_state['epoch_index'] == 0:
    torch.save(model.state_dict(), train_state['model_filename'])
    train_state['stop_early'] = False

# 성능이 향상되면 모델을 저장합니다
elif train_state['epoch_index'] >= 1:
    loss_tm1, loss_t = train_state['val_loss'][-2:]

    # 손실이 나빠지면
    if loss_t >= train_state['early_stopping_best_val']:
        # 조기 종료 단계 업데이트
        train_state['early_stopping_step'] += 1
    # 손실이 감소하면
    else:
        # 최상의 모델 저장
        if loss_t < train_state['early_stopping_best_val']:
            torch.save(model.state_dict(), train_state['model_filename'])

        # 조기 종료 단계 재설정
        train_state['early_stopping_step'] = 0

    # 조기 종료 여부 확인
    train_state['stop_early'] = \
        train_state['early_stopping_step'] >= args.early_stopping_criteria

return train_state
```

update_train_state 함수 코드 일부분

모델 훈련

- 훈련하던 모델이 없으면 데이터 셋과 모델을 형성
- 손실 함수, 옵티마이저 초기화

```
if args.reload_from_files:
    # 체크포인트에서 훈련을 다시 시작
    print("데이터셋과 Vectorizer를 로드합니다")
    dataset = ReviewDataset.load_dataset_and_load_vectorizer(args.review_csv,
                                                             args.vectorizer_file)
else:
    print("데이터셋을 로드하고 Vectorizer를 만듭니다")
    # 데이터셋과 Vectorizer 만들기
    dataset = ReviewDataset.load_dataset_and_make_vectorizer(args.review_csv)
    dataset.save_vectorizer(args.vectorizer_file)
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)
```

초기화 코드 일부분

모델 훈련

- 훈련 반복(.train() method)
  - 앞서 설정한 값에 따라
  - 데이터 셋을 순회
  - 입력 데이터에서 모델의 출력 계산
  - 손실 계산
  - 손실에 비례하여 모델 수정 (Adam 옵티마이저)

```
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()
    # --------------------------------------
```

훈련반복 코드 일부분

모델 훈련

- 모델 검증 (.eval() method)
  - Val 데이터셋을 사용
  - 모델의 파라미터 수정 불가
  - 모델의 성능 측정
  - 과대적합 시
    (훈련 세트와 검증 세트의 성능 차이가 크다면)
    조기종료

```
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)

train_state = update_train_state(args=args, model=classifier,
                                 train_state=train_state)

scheduler.step(train_state['val_loss'][-1])

train_bar.n = 0
val_bar.n = 0
epoch_bar.update()

if train_state['stop_early']:
    break
```

훈련반복 코드 일부분

평가

- 모델 평가 (.eval() method)
  - Test 데이터셋 사용

```
# 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
```

모델평가 코드 일부분

추론

- 모델에서 새로운 데이터를 잘 추론하는지 평가.

```python
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)
```

모델추론 코드 일부분

분석

- 가중치를 분석하여 올바른 값인지 평가

```python
# 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]))
```

모델가중치 분석 코드 일부분