CHAPTER 09 텍스트를 위한 인공 신경망

혼자 공부하는 머신러닝+딥러닝(개정판)

한국공학대학교 게임공학과
이재영

시작하기전에

지은이 / 박해선

기계공학을 전공했으나 졸업 후엔 줄곧 코드를 읽고 쓰는 일을 했다. 머신러닝과 딥러닝에 관한 책을 집필하고 번역하면서 소프트웨어와 과학의 경계를 흥미롭게 탐험하고 있다.

『핸즈온 머신러닝 2판』(한빛미디어, 2020)을 포함해서 여러 권의 머신러닝, 딥러닝 책을 우리말로 옮겼고 『Do it! 딥러닝 입문』(이지스퍼블리싱, 2019)을 집필했다.

- **교재의 모든 코드는 웹 브라우저에서 파이썬 코드를 실행할 수 있는 구글 코랩(Colab)을 사용하여 작성했습니다.**
- **사용할 실습 환경은 네트워크에 연결된 컴퓨터와 구글 계정입니다.**

학습 로드맵

머신러닝편 **01~06장**

딥러닝만 먼저 배우고
싶다면 01~04장을 읽은 후
07장으로 건너뛰어도 좋습니다.

딥러닝편 **07~10장**

07장을 읽은 후 08장과 09장은
순서대로 읽지 않아도 괜찮습니다.
10장을 읽기 전에 07장과 09장을
읽는 것이 좋습니다.

난이도 ●●●●●

START

01 나의 첫 머신러닝 ●○○○○

02 데이터 다루기 ●●○○○

03 회귀 알고리즘과 모델 규제

2번 보기

04 다양한 분류 알고리즘 ●●○○○

05 트리 알고리즘 ●●●○○

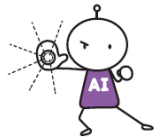
06 비지도 학습 ●●●○○

07 딥러닝을 시작합니다 ●●●●○

08 이미지를 위한 인공 신경망 ●●●●●

09 텍스트를 위한 인공 신경망 ●●●●●

10 언어 모델을 위한 신경망 ●●●●●



GOAL

이 책의 학습 목표

- **CHAPTER 01: 나의 첫 머신러닝**
  - 인공지능, 머신러닝, 딥러닝의 차이점을 이해합니다.
  - 구글 코랩 사용법을 배웁니다.
  - 첫 번째 머신러닝 프로그램을 만들고 머신러닝의 기본 작동 원리를 이해합니다.
- **CHAPTER 02: 데이터 다루기**
  - 머신러닝 알고리즘에 주입할 데이터를 준비하는 방법을 배웁니다.
  - 데이터 형태가 알고리즘에 미치는 영향을 이해합니다.
- **CHAPTER 03: 회귀 알고리즘과 모델 규제**
  - 지도 학습 알고리즘의 한 종류인 회귀 알고리즘에 대해 배웁니다.
  - 다양한 선형 회귀 알고리즘의 장단점을 이해합니다.
- **CHAPTER 04: 다양한 분류 알고리즘**
  - 로지스틱 회귀, 확률적 경사 하강법과 같은 분류 알고리즘을 배웁니다.
  - 이진 분류와 다중 분류의 차이를 이해하고 클래스별 확률을 예측합니다.
- **CHAPTER 05: 트리 알고리즘**
  - 성능이 좋고 이해하기 쉬운 트리 알고리즘에 대해 배웁니다.
  - 알고리즘의 성능을 최대화하기 위한 하이퍼파라미터 튜닝을 실습합니다.
  - 여러 트리를 합쳐 일반화 성능을 높일 수 있는 앙상블 모델을 배웁니다.

이 책의 학습 목표

- **CHAPTER 06: 비지도 학습**
  - 타깃이 없는 데이터를 사용하는 비지도 학습과 대표적인 알고리즘을 소개합니다.
  - 대표적인 군집 알고리즘인 k-평균과 DBSCAN을 배웁니다.
  - 대표적인 차원 축소 알고리즘인 주성분 분석(PCA)을 배웁니다.
- **CHAPTER 07: 딥러닝을 시작합니다**
  - 딥러닝의 핵심 알고리즘인 인공 신경망을 배웁니다.
  - 대표적인 인공 신경망 라이브러리인 텐서플로와 케라스를 소개합니다.
  - 인공 신경망 모델의 훈련을 돕는 도구를 익힙니다.
- **CHAPTER 08: 이미지를 위한 인공 신경망**
  - 이미지 분류 문제에 뛰어난 성능을 발휘하는 합성곱 신경망의 개념과 구성 요소에 대해 배웁니다.
  - 케라스 API로 합성곱 신경망을 만들어 패션 MNIST 데이터에서 성능을 평가해 봅니다.
  - 합성곱 층의 필터와 활성화 출력을 시각화하여 합성곱 신경망이 학습한 내용을 고찰해 봅니다.
- **CHAPTER 09: 텍스트를 위한 인공 신경망**
  - 텍스트와 시계열 데이터 같은 순차 데이터에 잘 맞는 순환 신경망의 개념과 구성 요소에 대해 배웁니다.
  - 케라스 API로 기본적인 순환 신경망에서 고급 순환 신경망을 만들어 영화 감상평을 분류하는 작업에 적용해 봅니다.
  - 순환 신경망에서 발생하는 문제점과 이를 극복하기 위한 해결책을 살펴봅니다.
- **CHAPTER 10: 언어 모델을 위한 신경망**
  - 어텐션 메커니즘과 트랜스포머에 대해 배웁니다.
  - 트랜스포머로 상품 설명 요약하기를 학습합니다.
  - 대규모 언어 모델로 텍스트 생성하기를 익힙니다.

Contents

- CHAPTER 09: 텍스트를 위한 인공 신경망

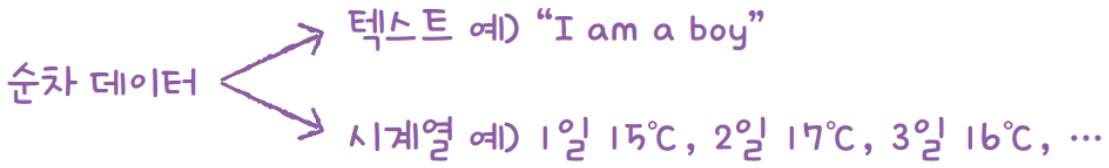
CHAPTER 09 텍스트를 위한 인공 신경망

한빛 마켓의 댓글을 분석하라!

- 텍스트와 시계열 데이터 같은 순차 데이터에 잘 맞는 순환 신경망의 개념과 구성 요소에 대해 배웁니다.
- 케라스 API로 기본적인 순환 신경망에서 고급 순환 신경망을 만들어 영화 감상평을 분류하는 작업에 적용해 봅니다.
- 순환 신경망에서 발생하는 문제점과 이를 극복하기 위한 해결책을 살펴봅니다.

SECTION 9-1 순차 데이터와 순환 신경망(1)

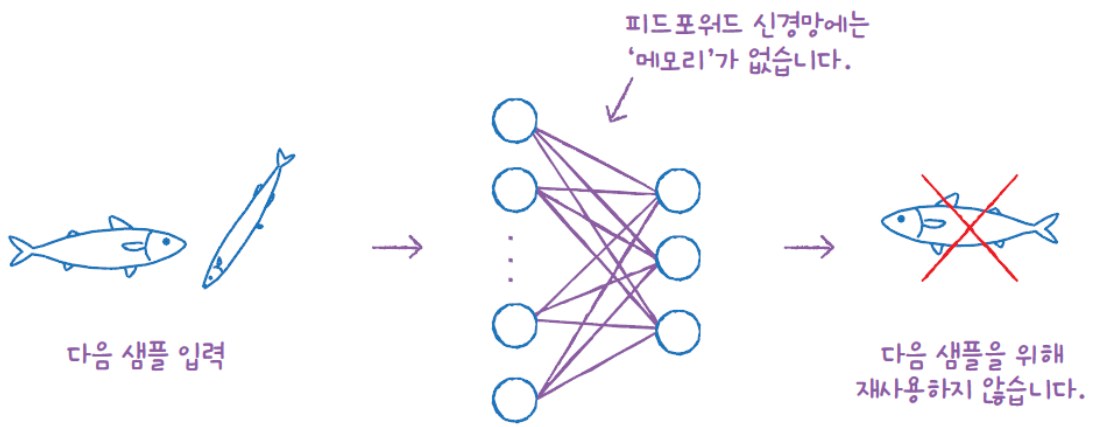
- 순차 데이터(sequential data)
  - 순차 데이터는 텍스트나 시계열 데이터(time series data)와 같이 순서에 의미가 있는 데이터

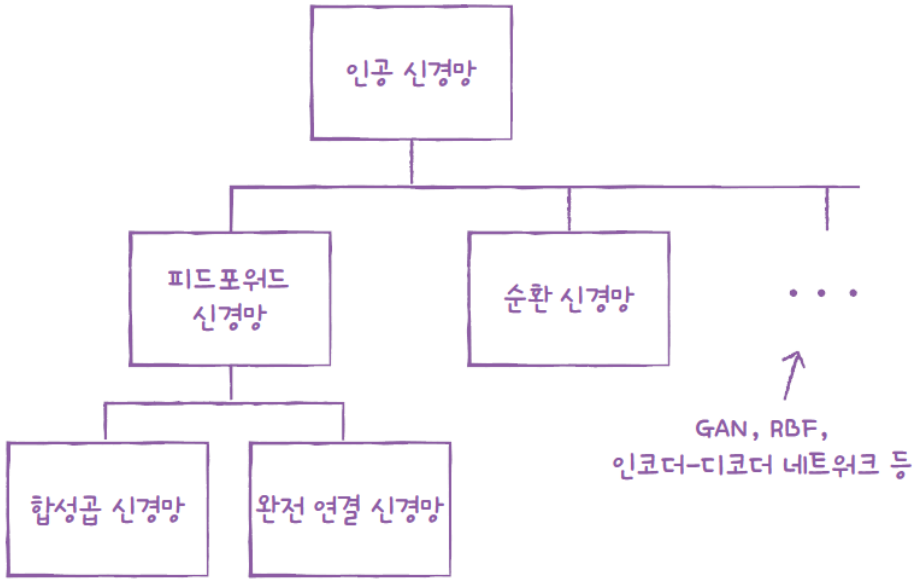


  - 앞에서 공부한 생선 데이터나 패션 MNIST 데이터는 어떤 샘플이 먼저 주입되어도 모델의 학습에 큰 영향을 주지 않았음
  - 텍스트 데이터는 단어의 순서가 중요한 순차 데이터
    - 예를 들어 "별로지만 추천해요"에서 "추천해요"가 입력될 때 "별로지만"을 기억하고 있어야 이 댓글을 무조건 긍정적이라고 판단하지 않을 것임

SECTION 9-1 순차 데이터와 순환 신경망(2)

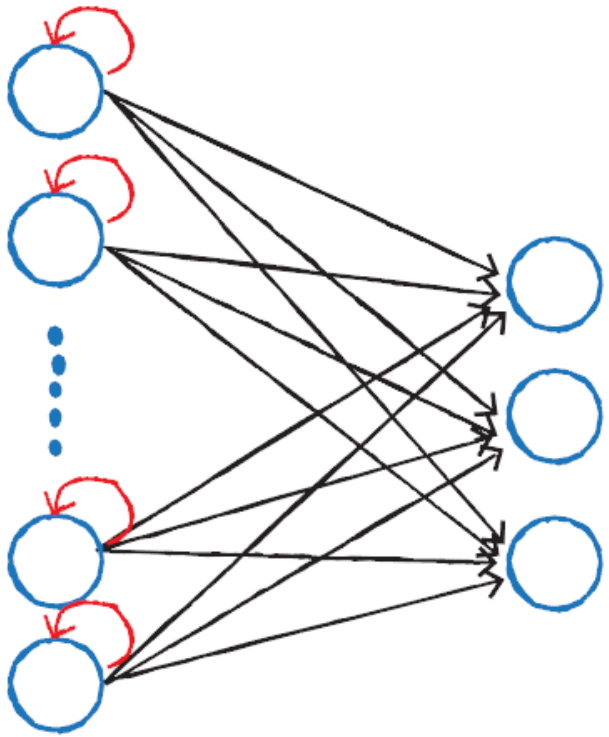
- 순차 데이터(sequential data)
  - 피드포워드 신경망(feedforward neural network, FFNN)
    - 입력 데이터의 흐름이 앞으로만 전달되는 신경망
    - 완전 연결 신경망이나 합성곱 신경망은 하나의 샘플(또는 하나의 배치)을 사용하여 정방향 계산을 수행하고 나면 그 샘플은 버려지고 다음 샘플을 처리할 때 재사용하지 않음





SECTION 9-1 순차 데이터와 순환 신경망(3)

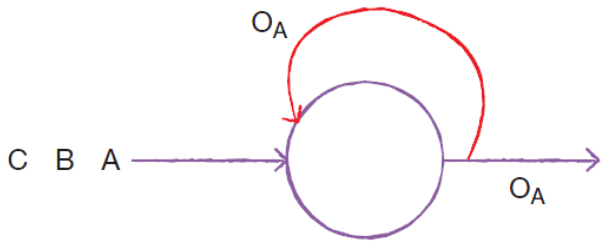
- 순환 신경망(recurrent neural network, RNN)
  - 순환 신경망은 일반적인 완전 연결 신경망과 거의 비슷함
  - 완전 연결 신경망에 이전 데이터의 처리 흐름을 순환하는 고리 하나만 추가하면 됨
    - 뉴런의 출력이 다시 자기 자신으로 전달. 즉 어떤 샘플을 처리할 때 바로 이전에 사용했던 데이터를 재사용



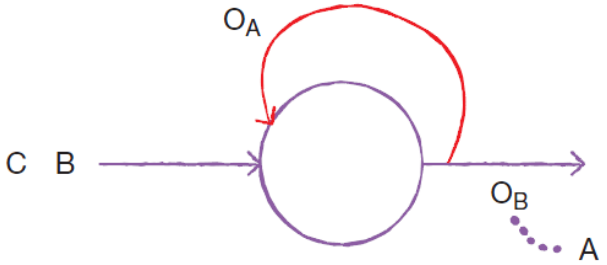
순환 신경망

SECTION 9-1 순차 데이터와 순환 신경망(4)

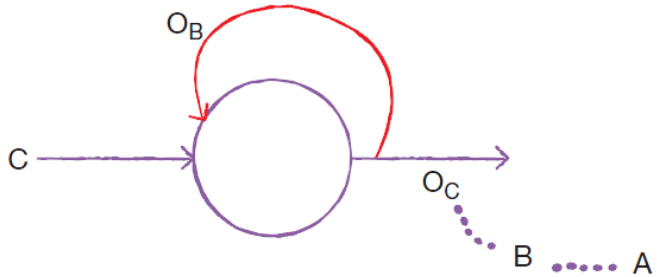
- 순환 신경망(recurrent neural network, RNN)



첫 번째 샘플 A를 처리하고 난 출력(O_A)이 다시 뉴런으로 들어감



그다음 B를 처리할 때 앞에서 A를 사용해 만든 출력 O_A를 함께 사용

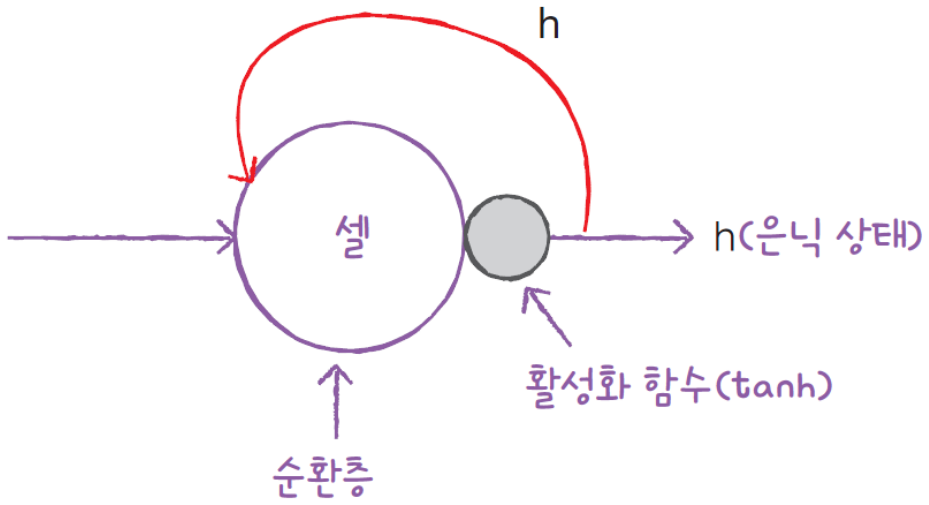


그다음 C를 처리할 때는 O_B를 함께 사용

▲ A, B, C, 3개의 샘플을 처리하는 순환 신경망의 뉴런이 있다고 가정
(O는 출력된 결과)

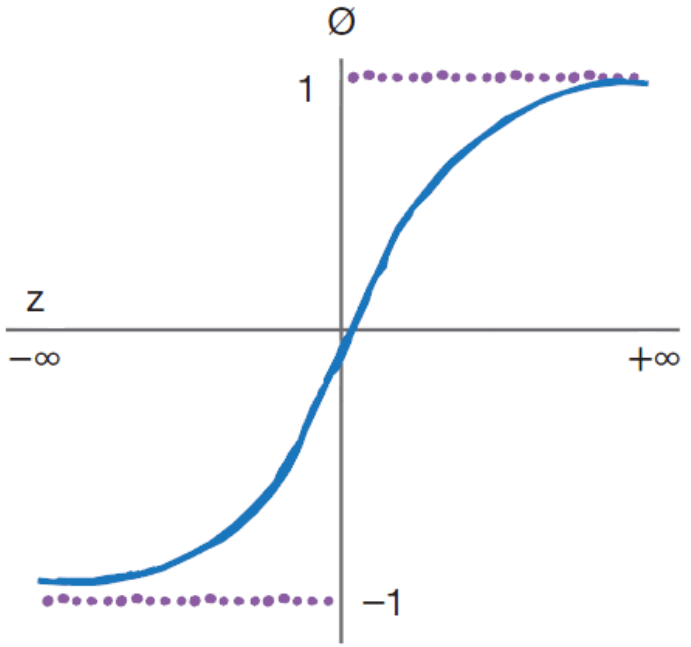
SECTION 9-1 순차 데이터와 순환 신경망(5)

- 순환 신경망(recurrent neural network, RNN)
  - 타임스텝(timestep): 순환 신경망에서 샘플을 처리하는 한 단계
    - 순환 신경망은 이전 타임스텝의 샘플을 기억하지만 타임스텝이 오래될수록 순환되는 정보는 희미해짐
  - 셀(cell): 순환 신경망에서는 특별히 층을 셀이라고 지칭
    - 한 셀에는 여러 개의 뉴런이 있지만 완전 연결 신경망과 달리 뉴런을 모두 표시 않고 하나의 셀로 층을 표현
  - 은닉 상태(hidden state): 셀의 출력



SECTION 9-1 순차 데이터와 순환 신경망(6)

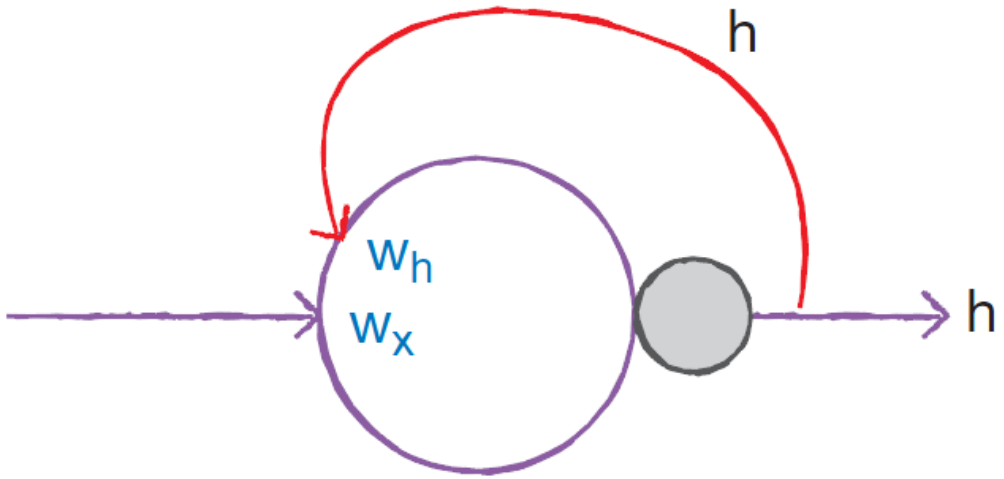
- 순환 신경망(recurrent neural network, RNN)
  - 은닉층의 활성화 함수로는 하이퍼볼릭 탄젠트(hyperbolic tangent) 함수인 tanh가 많이 사용
    - tanh 함수는 시그모이드 함수와는 달리 -1~1 사이의 범위를 가짐



tanh 함수

SECTION 9-1 순차 데이터와 순환 신경망(7)

- 순환 신경망(recurrent neural network, RNN)
  - 합성곱 신경망과 같은 피드포워드 신경망에서 뉴런은 입력과 가중치를 곱하듯, 순환 신경망에서도 동일하지만, 순환 신경망의 뉴런은 가중치가 하나 더 있음
    - 이전 타임스텝의 은닉 상태에 곱해지는 가중치
    - 셀은 입력과 이전 타임스텝의 은닉 상태를 사용하여 현재 타임스텝의 은닉 상태를 만듦

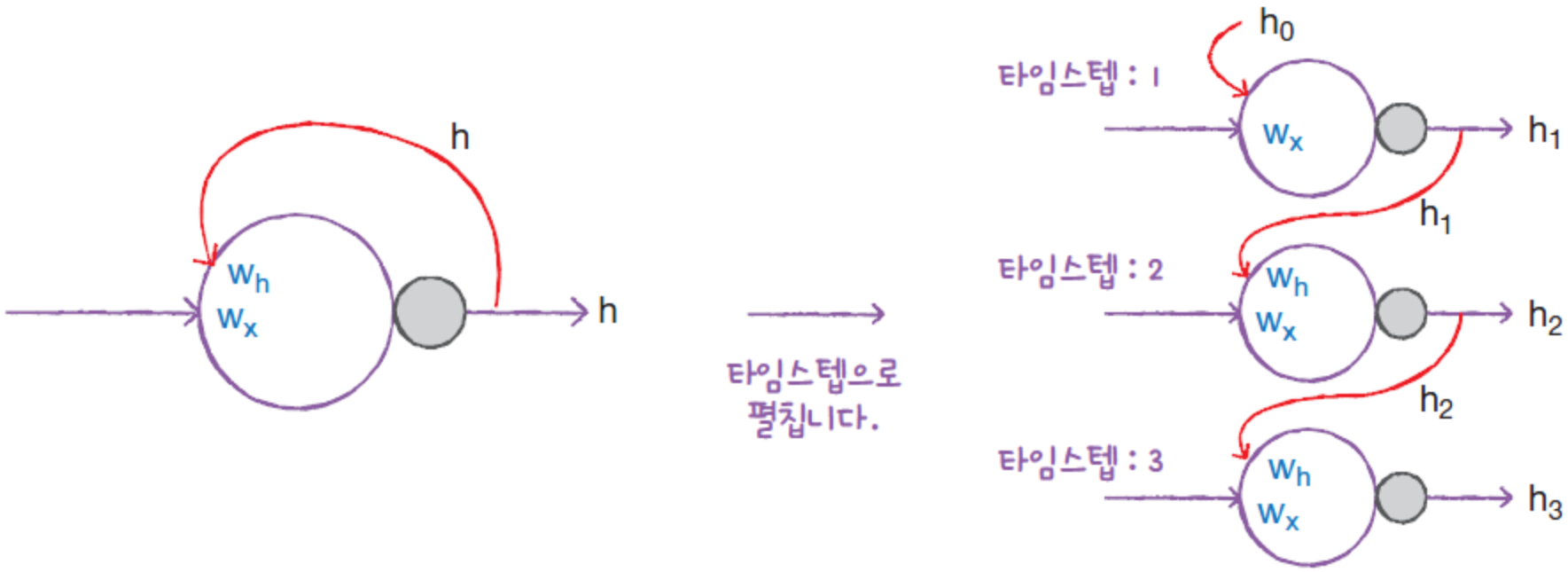


- w_x: 입력에 곱해지는 가중치
- w_h: 이전 타임스텝의 은닉 상태에 곱해지는 가중치

SECTION 9-1 순차 데이터와 순환 신경망(8)

◦ 순환 신경망(recurrent neural network, RNN)

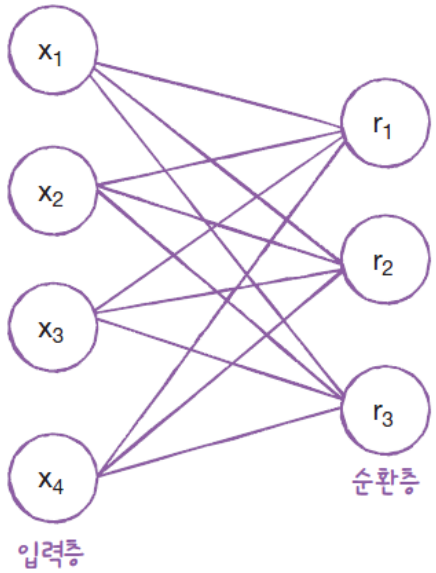
- 셀의 출력(은닉 상태)이 다음 타임스텝에 재사용되므로 타임스텝으로 셀을 나누어 그릴 수 있음

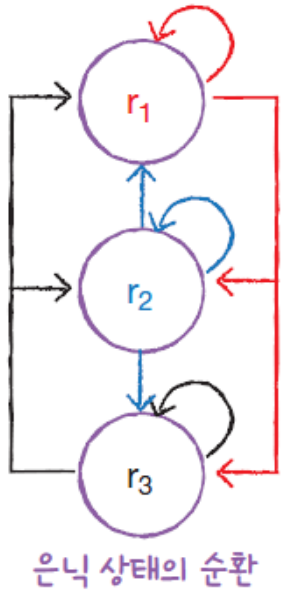


- 모든 타임스텝에서 사용되는 가중치는 w_h 하나
- 가중치 w_h는 타임스텝에 따라 변화되는 뉴런의 출력을 학습

SECTION 9-1 순차 데이터와 순환 신경망(9)

- 셀의 가중치와 입출력
  - 순환 신경망의 셀에서 필요한 가중치 크기 계산
    - 다음 그림처럼 순환층에 입력되는 특성의 개수가 4개이고 순환층의 뉴런이 3개라고 가정





- W_x의 크기: 입력층과 순환층의 뉴런이 모두 완전 연결되므로 가중치 W_x의 크기는 $4 \times 3 = 12$개
- 순환층에서 은닉 상태를 위한 가중치 W_h는 $3 \times 3 = 9$개

SECTION 9-1 순차 데이터와 순환 신경망(10)

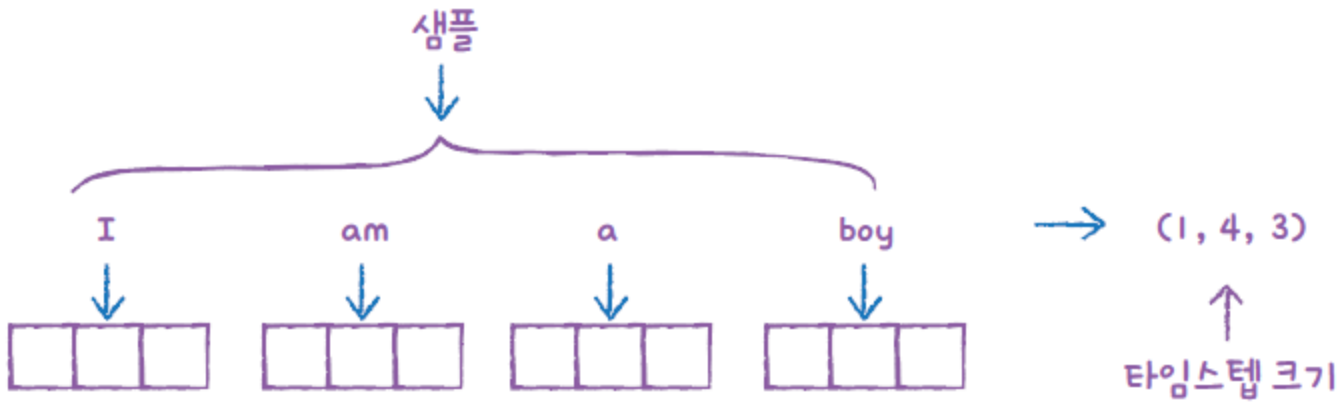
- 셀의 가중치와 입출력
  - 모델 파라미터 개수를 계산
    - 가중치에 절편을 더해줌
    - 각 뉴런마다 하나의 절편이 있으므로 이 순환층은 모두 12 + 9 + 3 = 24개의 모델 파라미터를 가짐

$$\text{모델 파라미터 수} = W_x + W_h + \text{절편} = 12 + 9 + 3 = 24$$

  - 순환층의 입력과 출력
    - 순환층은 일반적으로 샘플마다 2개의 차원
    - 보통 하나의 샘플을 하나의 시퀀스(sequence)라고 함
    - 시퀀스 안에는 여러 개의 아이템이 들어 있으며, 여기에서 시퀀스의 길이가 바로 타임스텝 길이가 됨

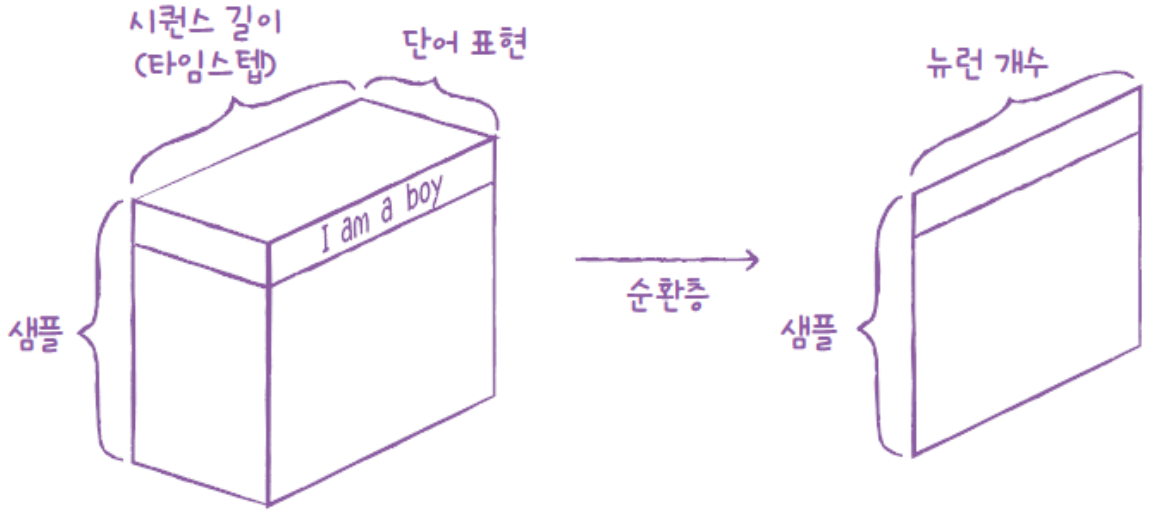
SECTION 9-1 순차 데이터와 순환 신경망(11)

- 셀의 가중치와 입출력
  - 순환층의 입력과 출력
    - 순환층은 일반적으로 샘플마다 2개의 차원
    - 보통 하나의 샘플을 하나의 시퀀스(sequence)라고 함
    - 시퀀스 안에는 여러 개의 아이템이 들어 있으며, 여기에서 시퀀스의 길이가 바로 타임스텝 길이가 됨
    - 어떤 샘플에 "I am a boy"란 문장이 들어 있다고 가정. 이 샘플은 4개의 단어로 이루어져 있음
    - 각 단어를 3개의 어떤 숫자로 표현한다고 가정



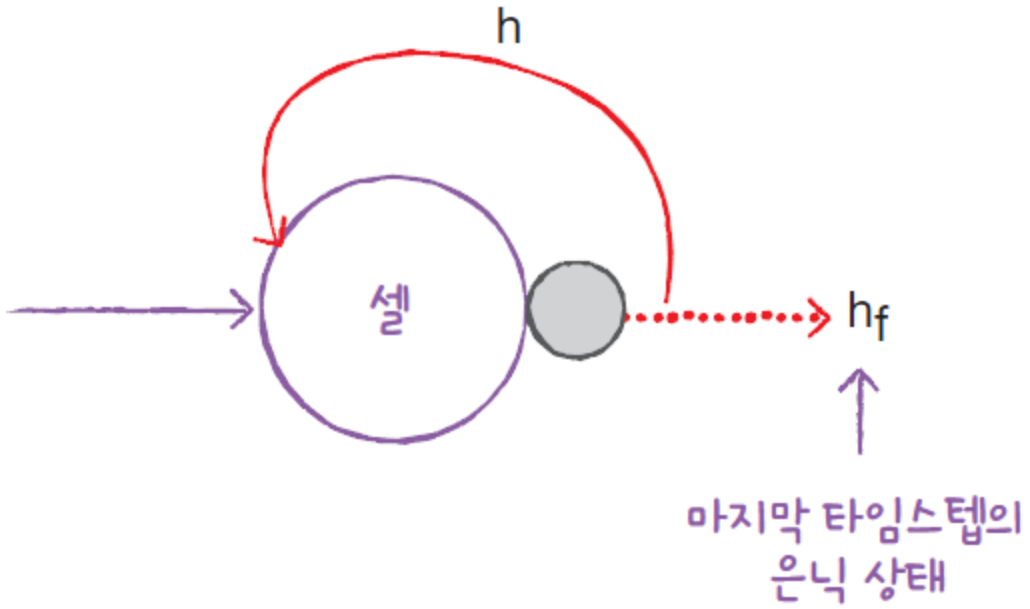
SECTION 9-1 순차 데이터와 순환 신경망(12)

- 셀의 가중치와 입출력
  - 순환층의 입력과 출력
    - 입력이 순환층을 통과하면 두 번째, 세 번째 차원이 사라지고 순환층의 뉴런 개수만큼 출력
    - 하나의 샘플은 시퀀스 길이(여기에서는 단어 개수)와 단어 표현의 2차원 배열이며, 순환층을 통과하면 1차원 배열로 바뀜
    - 이 1차원 배열의 크기는 순환층의 뉴런 개수에 의해 결정



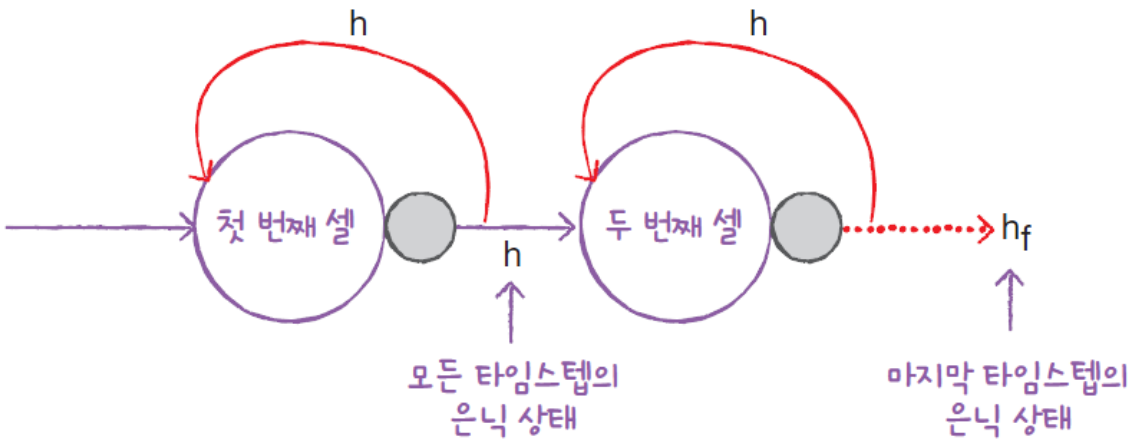
SECTION 9-1 순차 데이터와 순환 신경망(13)

- 셀의 가중치와 입출력
  - 순환층의 입력과 출력
    - 순환층은 기본적으로 마지막 타임스텝의 은닉 상태만 출력
    - 마치 입력된 시퀀스 길이를 모두 읽어서 정보를 마지막 은닉 상태에 압축하여 전달하는 것과 같음



SECTION 9-1 순차 데이터와 순환 신경망(14)

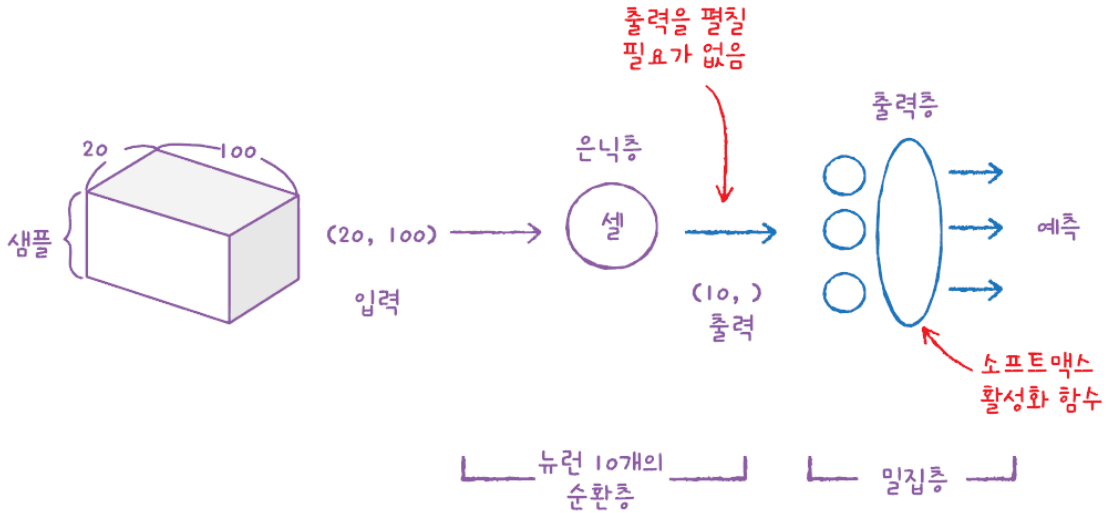
- 셀의 가중치와 입출력
  - 순환층의 입력과 출력
    - 순환층을 여러 개 쌓았을 때는 셀의 출력
    - 셀의 입력은 샘플마다 타임스텝과 단어 표현으로 이루어진 2차원 배열이어야 하므로 첫 번째 셀이 마지막 타임스텝의 은닉 상태만 출력해서는 안 됨
    - 이런 경우에는 마지막 셀을 제외한 다른 모든 셀은 모든 타임스텝의 은닉 상태를 출력



▲ 첫 번째 셀은 모든 타임스텝의 은닉 상태를 출력하고, 두 번째 셀은 마지막 타임스텝의 은닉 상태만 출력

SECTION 9-1 순차 데이터와 순환 신경망(15)

- 셀의 가중치와 입출력
  - 출력층의 구성
    - 합성곱 신경망과 마찬가지로 순환 신경망도 마지막에는 밀집층을 두어 클래스를 분류
    - 다중 분류일 경우에는 출력층에 클래스 개수만큼 뉴런을 두고 소프트맥스 활성화 함수를 사용
    - 이진 분류일 경우에는 하나의 뉴런을 두고 시그모이드 활성화 함수를 사용
    - 합성곱 신경망과 다른 점은 마지막 셀의 출력이 1차원이기 때문에 Flatten 클래스로 펼칠 필요가 없음. 셀의 출력을 그대로 밀집층에 사용할 수 있음



▲ 다중 분류 문제에서 입력 샘플의 크기가 (20, 100)일 경우 하나의 순환층을 사용하는 순환 신경망의 구조

SECTION 9-1 순차 데이터와 순환 신경망(16)

◦ 순환 신경망으로 순환 데이터 처리(문제해결 과정)

- 학습

  • 순차 데이터와 순환 신경망을 소개
  - 순차 데이터의 특징
  - 순환 신경망의 개념과 주요 구성 요소(순환층, 셀, 은닉 상태 등)

  • 순환층은 순서를 가진 데이터를 처리하기 위해 밀집 신경망이나 합성곱 신경망과는 계산하는 방식이 다름
  - 은닉층의 출력을 다음 층으로만 보내지 않고 다음 순서에 다시 재사용하는 순환 구조
  - 거시적인 구조는 다른 신경망과 크게 다르지 않음
  - 입력에 가중치를 곱하고 절편을 더한 다음 활성화 함수를 통과시켜 다음 층으로 전달
  - 다만 순환층은 이전 타임스텝의 출력을 입력으로 함께 사용
  - 마지막 타임스텝의 출력만 다음 층으로 전달

SECTION 9-1 마무리

- 키워드로 끝나는 핵심 포인트
  - 순차 데이터는 텍스트나 시계열 데이터와 같이 순서에 의미가 있는 데이터
    - 대표적인 순차 데이터로는 글, 대화, 일자별 날씨, 일자별 판매 실적 등
  - 순환 신경망은 순차 데이터에 잘 맞는 인공 신경망의 한 종류
    - 순차 데이터를 처리하기 위해 고안된 순환층을 1개 이상 사용한 신경망을 순환 신경망이라고 함
  - 순환 신경망에서는 종종 순환층을 셀이라 부름
    - 일반적인 인공 신경망과 마찬가지로 하나 의 셀은 여러 개의 뉴런으로 구성
  - 순환 신경망에서는 셀의 출력을 특별히 은닉 상태라고 함
    - 은닉 상태는 다음 층으로 전달될 뿐만 아니라 셀이 다음 타임스텝의 데이터를 처리할 때 재사용

SECTION 9-1 확인 문제(1)

1. 다음 중 순차 데이터로 처리하기 어려운 작업은?

   ① 환자의 검사 결과를 바탕으로 질병 예측하기
   ② 월별 주택 가격을 바탕으로 다음 달의 주택 가격 예측하기
   ③ 태풍의 이동 경로를 바탕으로 다음 경로 예측하기
   ④ 노래 악보를 바탕으로 다음 음표를 예측하기

2. 순환 신경망에서 순환층을 부르는 다른 말과 순환층의 출력을 나타내는 용어를 올바르게 짝지은 것은?

   ① 셸(shell)-셸 상태
   ② 셸(shell)-은닉 상태
   ③ 셀(cell)-셀 상태
   ④ 셀(cell)-은닉 상태

SECTION 9-1 확인 문제(2)

3. 순환 신경망에서 한 셀에 있는 뉴런의 개수가 10개. 이 셀의 은닉 상태가 다음 타임스텝에 사용될 때 곱해지는 가중치 W_h의 크기는?

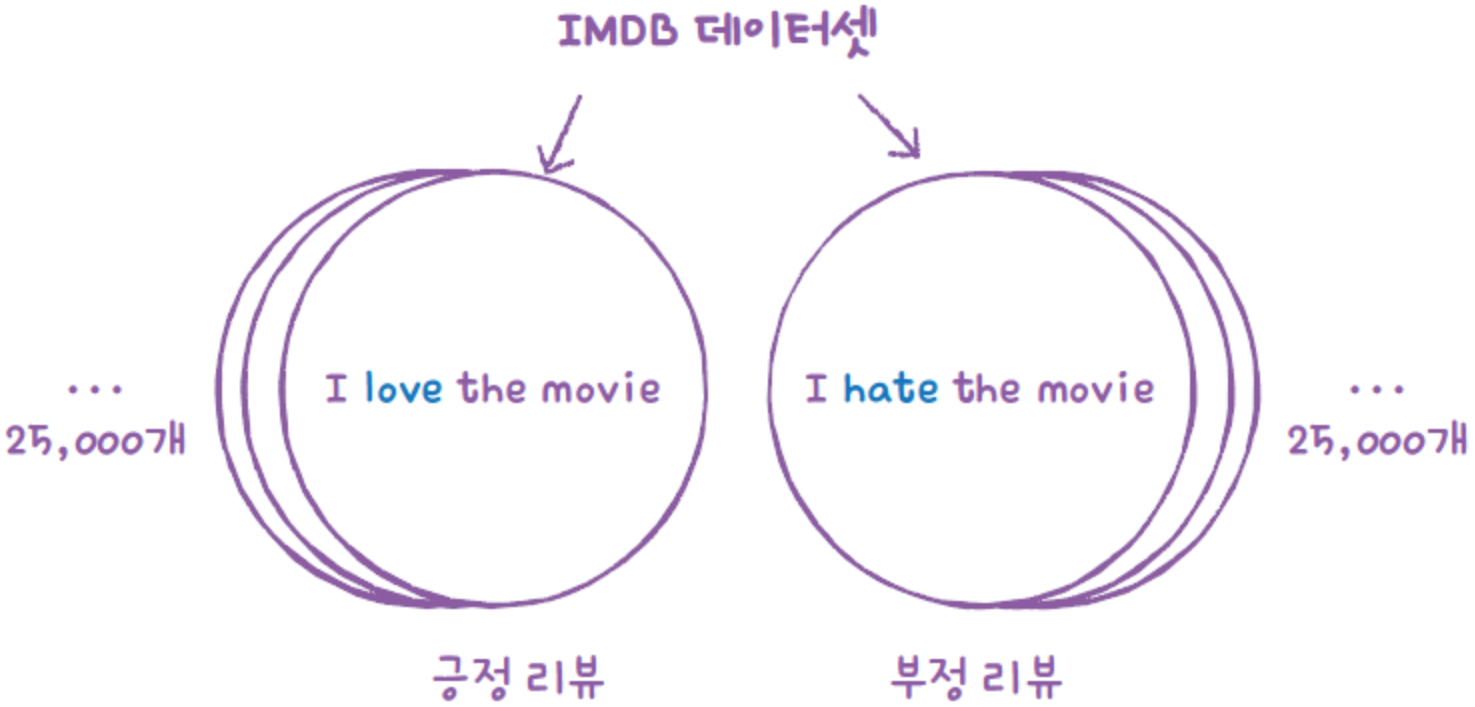
① (10,)
② (10, 10)
③(10, 10, 10)
④ 알 수 없음

4. 기본 순환 신경망에서 널리 사용되는 활성화 함수는?

① ReLU
② max
③ softmax
④ tanh

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(1)

- IMDB 리뷰 데이터셋을 사용해 가장 간단한 순환 신경망 모델을 훈련



SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(2)

- IMDB 리뷰 데이터셋
  - IMDB 리뷰 데이터셋은 유명한 인터넷 영화 데이터베이스인 imdb.com에서 수집한 리뷰를 감상평에 따라 긍정과 부정으로 분류해 놓은 데이터셋
  - 총 50,000개의 샘플로 이루어져 있고 훈련 데이터와 테스트 데이터가 각각 25,000개
  - 텍스트 데이터의 경우 단어를 숫자 데이터로 바꾸는 일반적인 방법은 단어마다 고유한 정수를 부여
  - 토큰(token): 영어 문장은 모두 소문자로 바꾸고 구둣점을 삭제한 다음 공백을 기준으로 분리된 단어
    - 하나의 샘플은 여러 개의 토큰으로 이루어져 있고 1개의 토큰이 하나의 타임스탬프에 해당
    - 토큰에 할당하는 정수 중에 몇 개는 특정한 용도로 예약되어 있는 경우가 많음. 예를 들어 0은 패딩, 1은 문장의 시작, 2는 어휘 사전에 없는 토큰을 나타냄

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(3)

◦ IMDB 리뷰 데이터셋

- 실제 IMDB 리뷰 데이터셋은 영어로 된 문장이지만 케라스에는 이미 정수로 바꾼 데이터가 포함
- keras.datasets 패키지 아래 imdb 모듈을 임포트하여 이 데이터를 적재
  - 여기에서는 전체 데이터셋에서 가장 자주 등장하는 단어 200개만 사용

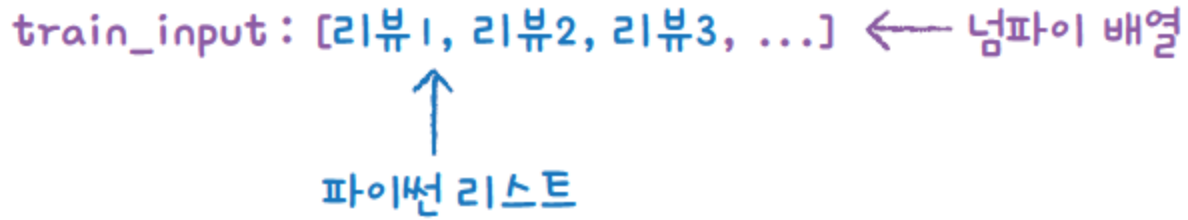
```
from keras.datasets import imdb

(train_input, train_target), (test_input, test_target) = imdb.load_data(
                                num_words=200)
```

- 훈련 세트와 테스트 세트의 크기를 확인

```
print(train_input.shape, test_input.shape)
```

→ (25000,) (25000,)

- IMDB 리뷰 텍스트는 길이가 제각각이므로 고정 크기의 2차원 배열에 담기 보다는 리뷰마다 별도의 파이썬 리스트로 담아야 메모리를 효율적으로 사용할 수 있음



SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(4)

- IMDB 리뷰 데이터셋
  - 첫 번째 리뷰의 길이를 출력

```
print(len(train_input[0]))
```

→ 218

  - 두 번째 리뷰의 길이를 확인

```
print(len(train_input[1]))
```

→ 189

◀ 하나의 리뷰가 하나의 샘플

  - 첫번째 리뷰에 담긴 내용을 출력

```
print(train_input[0])
```

→

[1, 14, 22, 16, 43, 2, 2, 2, 2, 65, 2, 2, 66, 2, 4, 173, 36, 256, 5, 25, 100,
43, 2, 112, 50, 2, 2, 9, 35, 2, 284, 5, 150, 4, 172, 112, 167, 2, 2, 2, 39,
4, 172, 2, 2, 17, 2, 38, 13, 2, 4, 192, 50, 16, 6, 147, 2, 19, 14, 22, 4, 2,
2, 2, 4, 22, 71, 87, 12, 16, 43, 2, 38, 76, 15, 13, 2, 4, 22, 17, 2, 17, 12,
16, 2, 18, 2, 5, 62, 2, 12, 8, 2, 8, 106, 5, 4, 2, 2, 16, 2, 66, 2, 33, 4,
130, 12, 16, 38, 2, 5, 25, 124, 51, 36, 135, 48, 25, 2, 33, 6, 22, 12, 215,
28, 77, 52, 5, 14, 2, 16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 2, 5,
2, 36, 71, 43, 2, 2, 26, 2, 2, 46, 7, 4, 2, 2, 13, 104, 88, 4, 2, 15, 297,
98, 32, 2, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 2, 26, 2, 5,
144, 30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 2, 88, 12,
16, 283, 5, 16, 2, 113, 103, 32, 15, 16, 2, 19, 178, 32]

▲ num_words=200으로 지정했기 때문에 어휘 사전에는 200개의 단어만 들어 있음
따라서 어휘 사전에 없는 단어는 모두 2로 표시

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(5)

◦ IMDB 리뷰 데이터셋

- 타깃 데이터를 출력
  - 타깃값이 0(부정)과 1(긍정)

```
print(train_target[:20])
```

→ [1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1]

- 훈련 세트에서 검증 세트를 떼어 놓기
  - 훈련 세트의 크기가 25,000개였으므로 20%를 검증 세트로 떼어 놓으면 훈련 세트의 크기는 20,000개

```
from sklearn.model_selection import train_test_split

train_input, val_input, train_target, val_target = train_test_split(
        train_input, train_target, test_size=0.2, random_state=42)
```

- 넘파이 리스트 내포를 사용해 train_input의 원소를 순회하면서 길이를 측정
  - 평균적인 리뷰의 길이와 가장 짧은 리뷰의 길이 그리고 가장 긴 리뷰의 길이를 확인

```
import numpy as np

lengths = np.array([len(x) for x in train_input])
```

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(6)

◦ IMDB 리뷰 데이터셋

- 넘파이 mean( ) 함수와 median( ) 함수를 사용해 리뷰 길이의 평균과 중간값 계산

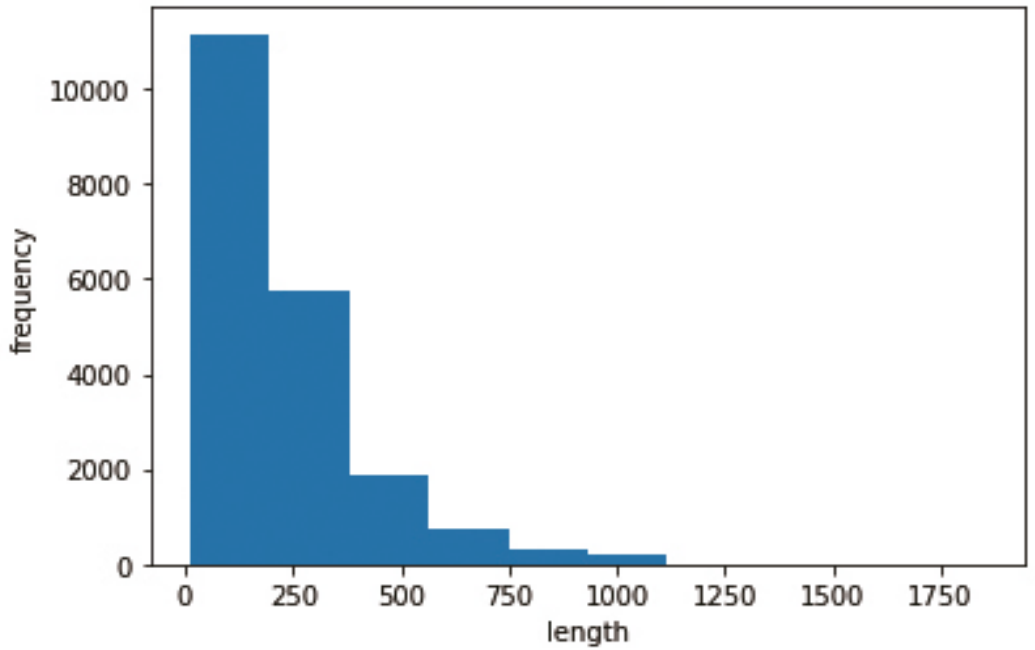
```
print(np.mean(lengths), np.median(lengths))
```

→ 239.00925 178.0

- lengths 배열을 히스토그램으로 표현

```
import matplotlib.pyplot as plt

plt.hist(lengths)
plt.xlabel('length')
plt.ylabel('frequency')
plt.show()
```

→



SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(7)

◦ IMDB 리뷰 데이터셋

- 예제에서는 중간값보다 훨씬 짧은 100개의 단어만 사용
  - 길거나 짧은 리뷰들의 길이를 100에 맞추기 위해 패딩이 필요
  - 보통 패딩을 나타내는 토큰으로는 0을 사용
- 시퀀스 데이터의 길이를 맞추는 pad_sequences( ) 함수를 사용해 train_input의 길이를 100으로 조정

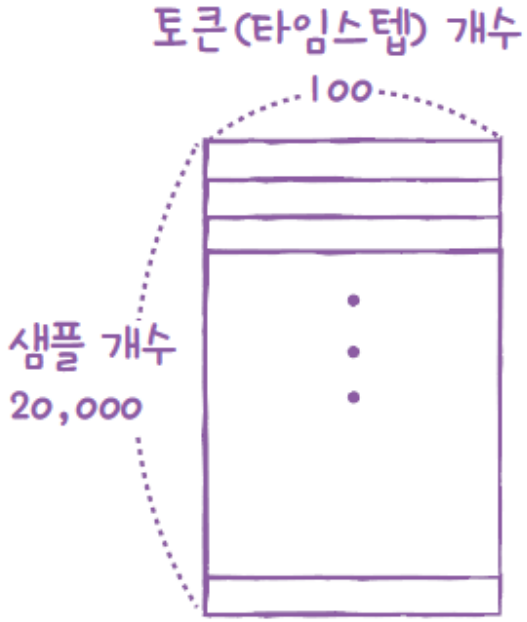
```
from keras.preprocessing.sequence import pad_sequences

train_seq = pad_sequences(train_input, maxlen=100)
```

- 패딩 된 결과 중 먼저 train_seq의 크기를 확인

```
print(train_seq.shape)
```

→ (20000, 100)

- train_input은 파이썬 리스트의 배열이었지만 길이를 100으로 맞춘 train_seq는 이제 (20000, 100) 크기의 2차원 배열이 되었음



SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(8)

◦ IMDB 리뷰 데이터셋

- train_seq에 있는 첫 번째 샘플을 출력

```
print(train_seq[0])
```

→

```
[ 10 4 20 9 2 2 2 5 45 6 2 2 33 269 8 2 142 2
5 2 17 73 17 204 5 2 19 55 2 2 92 66 104 14 20 93
76 2 151 33 4 58 12 188 2 151 12 215 69 224 142 73 237 6
2 7 2 2 188 2 103 14 31 10 10 2 7 2 5 2 80 91
2 30 2 34 14 20 151 50 26 131 49 2 84 46 50 37 80 79
6 2 46 7 14 20 10 10 2 158]
```

- 음수 인덱스와 슬라이싱을 사용해 train_input[0]에 있는 마지막 10개의 토큰을 출력
  - 이 샘플의 앞뒤에 패딩값 0이 없는 것으로 보아 100보다는 길었을 것으로 추정
  - 원래 샘플의 앞부분 혹은 뒷부분이 잘렸는지 train_input에 있는 원본 샘플의 끝을 확인

```
print(train_input[0][-10:])
```

→ [6, 2, 46, 7, 14, 20, 10, 10, 2, 158]

▲ pad_sequences ( ) 함수는 기본으로 maxlen보다 긴 시퀀스의 앞부분을 잘라냄
일반적으로 시퀀스의 뒷부분의 정보가 더 유용하리라 기대하기 때문임
시퀀스의 뒷부분을 잘라내고 싶다면 pad_sequences( ) 함수의 truncating 매개변수의 값을
기본값 'pre'가 아닌 'post'로 변경

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(9)

◦ IMDB 리뷰 데이터셋

- train_seq에 있는 여섯 번째 샘플을 출력
  - 앞부분에 0이 있는 것으로 보아 이 샘플의 길이는 100이 안 되는 것으로 추정
  - 패딩 토큰은 시퀀스의 뒷부분이 아니라 앞부분에 추가됨

```
print(train_seq[5])
```

→

```
[ 0 0 0 0 1 2 195 19 49 2 2 190 4 2 2 2 183 10
10 13 82 79 4 2 36 71 269 8 2 25 19 49 7 4 2 2
2 2 2 10 10 48 25 40 2 11 2 2 40 2 2 5 4 2
2 95 14 238 56 129 2 10 10 21 2 94 2 2 2 2 11 190
24 2 2 7 94 205 2 10 10 87 2 34 49 2 7 2 2 2
2 2 290 2 46 48 64 18 4 2]
```

- 검증 세트의 길이도 100으로 조정

```
val_seq = pad_sequences(val_input, maxlen=100)
```

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(10)

◦ 순환 신경망 만들기

- 케라스의 Sequential 클래스로 만든 신경망 코드(SimpleRNN 클래스 사용)
  - Input 함수의 입력 차원을 (100, 200)으로 지정 - 첫 번째 차원은 샘플 길이 100
  - SimpleRNN 클래스에 사용할 뉴런의 개수를 지정
    - SimpleRNN 클래스의 activation 매개변수의 기본값은 'tanh'로 하이퍼볼릭 탄젠트 함수 사용

```
import keras

model = keras.Sequential()
model.add(keras.layers.Input(shape=(100,200)))
model.add(keras.layers.SimpleRNN(8))
model.add(keras.layers.Dense(1, activation='sigmoid'))
```

- to_categorical ( ) 함수로 정수 배열을 입력하면 자동으로 원-핫 인코딩된 배열을 반환

```
train_oh = keras.utils.to_categorical(train_seq)
```

- train_oh 배열의 크기 출력

```
print(train_oh.shape)
```

→ (20000, 100, 200)

▲ 샘플 데이터의 크기가 1차원 정수 배열 (100,)에서 2차원 배열 (100, 200)로 바꿔야 하므로 SimpleRNN 클래스의 input_shape 매개변수의 값을 (100, 200)으로 지정

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(11)

◦ 순환 신경망 만들기

- train_oh의 첫 번째 샘플의 첫 번째 토큰 10 출력하여 인코딩 확인

```
print(train_oh[0][0][:12])
```

→ [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]

- 처음 12개 원소를 출력해 보면 열한 번째 원소가 1 나머지 원소는 모두 0인지 확인하기 위해 넘파이 sum( ) 함수로 모든 원소의 값을 더하기

```
print(np.sum(train_oh[0][0]))
```

→ 1.0

- 같은 방식으로 val_seq도 원-핫 인코딩으로 변환

```
val_oh = keras.utils.to_categorical(val_seq)
```

- 앞서 만든 모델의 구조를 출력

```
model.summary()
```

→

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 8)	1,672
dense (Dense)	(None, 1)	9

Total params: 1,681 (6.57 KB)
Trainable params: 1,681 (6.57 KB)
Non-trainable params: 0 (0.00 B)

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(12)

- 순환 신경망 만들기
  - SimpleRNN에 전달할 샘플의 크기는 (100, 200)이지만 이 순환층은 마지막 타임스텝의 은닉 상태만 출력
    - 이 때문에 출력 크기가 순환층의 뉴런 개수와 동일한8 임을 확인
  - 순환층에 사용된 모델 파라미터의 개수
    - 입력 토큰은 200차원의 원-핫 인코딩 배열
    - 이 배열이 순환층의 뉴런 8개와 완전히 연결되기 때문에 총 200 × 8 = 1,600개의 가중치
    - 순환층의 은닉 상태는 다시 다음 타임스텝에 사용되기 위해 또 다른 가중치와 곱해짐
    - 이 은닉 상태도 순환층의 뉴런과 완전히 연결되기 때문에 8(은닉 상태 크기) × 8(뉴런 개수) =64개의 가중치가 필요
    - 마지막으로 뉴런마다 하나의 절편이 있음
      - 따라서 모두 1,600 + 64 + 8 = 1,672개의 모델 파라미터가 필요

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(13)

◦ 순환 신경망 훈련하기

- 순환 신경망의 훈련은 완전 연결 신경망이나 합성곱 신경망과 모델을 만드는 것은 달라도 훈련하는 방법은 모두 같음
- Adam 옵티마이저를 사용하고 이진 분류 문제이므로 손실 함수로 'binary_crossentropy'를 사용
- 에포크 횟수를 100으로 늘리고 배치 크기는 64개로 설정

```
model.compile(optimizer='adam', loss='binary_crossentropy',
                        metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-simplernn-model.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)
history = model.fit(train_oh, train_target, epochs=100, batch_size=64,
                    validation_data=(val_oh, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

→ ▶ 출력 결과는 다음 쪽에 이어짐

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(14)

◦ 순환 신경망 훈련하기

- 이 훈련은 열 세 번째 에포크에서 조기 종료 -검증 세트에 대한 정확도는 약 73% 정도

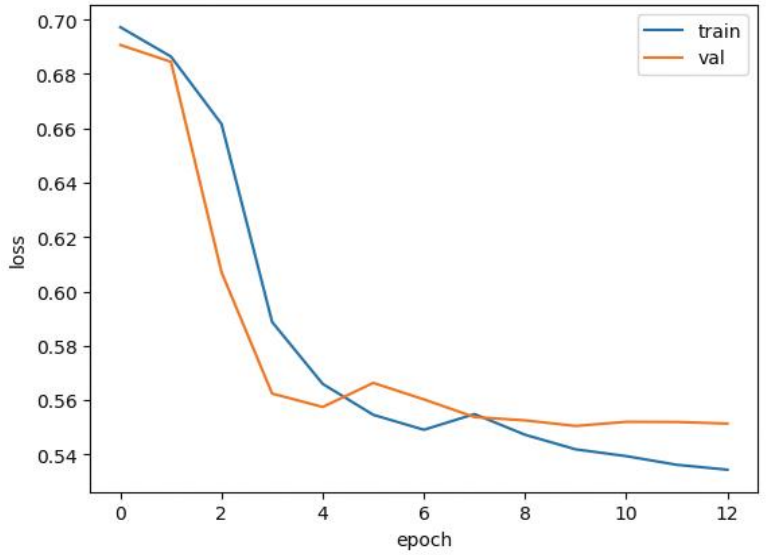
◀ 앞쪽 코드 출력 결과

```
Epoch 1/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14s 31ms/step - accuracy:
0.5031 - loss: 0.7030 - val_accuracy: 0.5284 - val_loss: 0.6907
Epoch 2/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - accuracy:
0.5437 - loss: 0.6871 - val_accuracy: 0.5508 - val_loss: 0.6844
Epoch 3/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - accuracy:
0.5760 - loss: 0.6759 - val_accuracy: 0.6800 - val_loss: 0.6070
(중략)
Epoch 10/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy:
0.7301 - loss: 0.5443 - val_accuracy: 0.7256 - val_loss: 0.5505
Epoch 11/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy:
0.7349 - loss: 0.5386 - val_accuracy: 0.7228 - val_loss: 0.5520
Epoch 12/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy:
0.7358 - loss: 0.5371 - val_accuracy: 0.7268 - val_loss: 0.5520
Epoch 13/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy:
0.7363 - loss: 0.5348 - val_accuracy: 0.7270 - val_loss: 0.5514
```

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(15)

- 순환 신경망 훈련하기
  - 훈련 손실과 검증 손실 그래프

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```



  - 입력 데이터가 엄청 커지는 것이 원-핫 인코딩의 단점
    train_seq 배열의 크기는 약 7.6M 정도인데 train_oh의 크기는 3GB에 달함

```
print(train_seq.nbytes, train_oh.nbytes)
```

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(16)

- 단어 임베딩(word embedding) 사용하기
  - 순환 신경망에서 텍스트를 처리할 때 즐겨 사용하는 단어 임베딩은 각 단어를 고정된 크기의 실수 벡터로 바꿔줌

'cat'의 단어 임베딩 벡터

0.2	0.1	1.3	0.8	0.2	0.4	1.1	0.9	0.2	0.1

  - 단어 임베딩으로 만들어진 벡터는 원-핫 인코딩된 벡터보다 훨씬 의미 있는 값으로 채워져 있기 때문에 자연어 처리에서 더 좋은 성능을 내는 경우가 많음
  - 케라스에서는 keras.layers 패키지 아래 Embedding 클래스로 임베딩 기능을 제공
  - 단어 임베딩의 장점: 입력으로 정수 데이터를 받음
  - 원-핫 인코딩으로 변경된 train_oh 배열이 아니라 train_seq를 사용할 수 있어서 메모리를 훨씬 효율적으로 활용

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(17)

- 단어 임베딩(word embedding) 사용하기
  - 많이 등장하는 500개의 단어까지 선택해서 IMDB 데이터셋을 다시 준비

```
(train_input, train_target), (test_input, test_target) = imdb.load_data(
                                        num_words=500)
train_input, val_input, train_target, val_target = train_test_split(
                train_input, train_target, test_size=0.2, random_state=42)
train_seq = pad_sequences(train_input, maxlen=100)
val_seq = pad_sequences(val_input, maxlen=100)
```

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(18)

◦ 단어 임베딩(word embedding) 사용하기

- Embedding 클래스를 SimpleRNN 층 앞에 추가한 두 번째 순환 신경망을 만들기

```
model_emb = keras.Sequential()
model_emb.add(keras.layers.Input(shape=(100,)))
model_emb.add(keras.layers.Embedding(500, 16))
model_emb.add(keras.layers.SimpleRNN(8))
model_emb.add(keras.layers.Dense(1,
activation='sigmoid'))
```

- 모델의 구조를 출력

```
model_emb.summary()
```

→

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 16)	8,000
simple_rnn_1 (SimpleRNN)	(None, 8)	200
dense_1 (Dense)	(None, 1)	9

Total params: 8,209 (32.07 KB)
Trainable params: 8,209 (32.07 KB)
Non-trainable params: 0 (0.00 B)

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(19)

- 단어 임베딩(word embedding) 사용하기
  - 모델 파라미터 개수
    - Embedding 클래스는 200개의 각 토큰을 크기가 16인 벡터로 변경하기 때문에 총 200 × 16 = 3,200개의 모델 파라미터
    - impleRNN 층은 임베딩 벡터의 크기가 16이므로 8개의 뉴런과 곱하기 위해 필요한 가중치 16 × 8 = 128개
    - 은닉 상태에 곱해지는 가중치 8 × 8 = 64개
    - 마지막으로 8개의 절편이 있으므로 이 순환층에 있는 전체 모델 파라미터의 개수는 128 + 64 + 8 = 200개
    - Dense 층의 가중치 개수는 이전과 동일하게 9개

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(20)

- 단어 임베딩(word embedding) 사용하기
  - 모델 훈련 과정

```
model_emb.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-embedding-model.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)
history = model_emb.fit(train_seq, train_target, epochs=100, batch_size=64,
                        validation_data=(val_seq, val_target),
                        callbacks=[checkpoint_cb, early_stopping_cb])
```

→

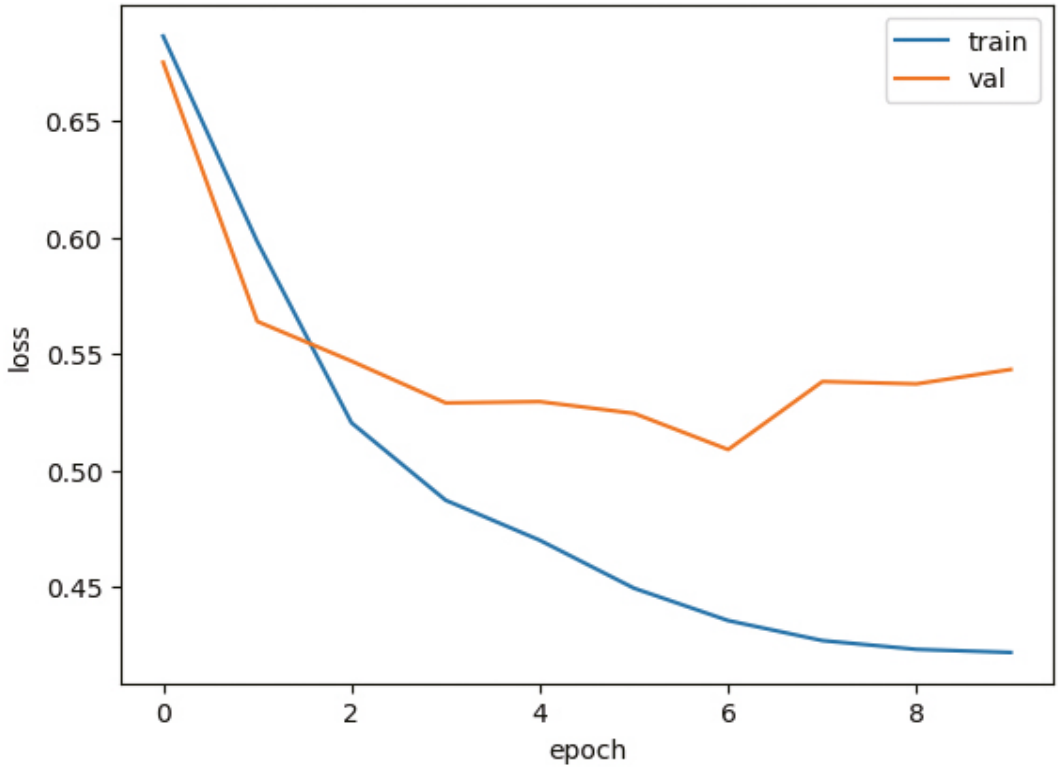
```
Epoch 1/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53s 161ms/step - accuracy:
0.5242 - loss: 0.6913 - val_accuracy: 0.5782 - val_loss: 0.6750
Epoch 2/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49s 156ms/step - accuracy:
0.6491 - loss: 0.6349 - val_accuracy: 0.7262 - val_loss: 0.5638
(중략)
Epoch 9/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49s 156ms/step - accuracy:
0.8087 - loss: 0.4268 - val_accuracy: 0.7582 - val_loss: 0.5370
Epoch 10/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49s 155ms/step - accuracy:
0.8131 - loss: 0.4257 - val_accuracy: 0.7550 - val_loss: 0.5432
```

SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(21)

- 단어 임베딩(word embedding) 사용하기
  - 훈련 손실과 검증 손실 그래프

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

→



SECTION 9-2 순환 신경망으로 IMDB 리뷰 분류하기(22)

- 케라스 API로 순환 신경망 구현(문제해결 과정)
  - 학습
    - 순환 신경망의 개념을 실제 모델을 만들어 보면서 구체화
    - 텐서플로와 케라스는 완전 연결 신경망, 합성곱 신경망뿐만 아니라 다양한 순환층 클래스를 제공하기 때문에 손쉽게 순환 신경망을 만들 수 있음
    - 순환 신경망의 MNIST 데이터셋으로 생각할 수 있는 유명한 IMDB 리뷰 데이터셋을 사용
      - 이 작업은 리뷰의 감상평을 긍정과 부정으로 분류하는 이진 분류 작업
    - 두 가지 모델을 훈련
      1) 입력 데이터를 원-핫 인코딩으로 변환하여 순환층에 직접 주입하는 방법
      2) 정수 시퀀스를 그대로 사용하기 위해 모델 처음에 Embedding 층을 추가
    - 단어 임베딩은 단어마다 실수로 이루어진 밀집 벡터를 학습하므로 단어를 풍부하게 표현할 수 있음

SECTION 9-2 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - 말뭉치는 자연어 처리에서 사용하는 텍스트 데이터의 모음, 훈련 데이터셋
  - 토큰은 텍스트에서 공백으로 구분되는 문자열
    - 종종 소문자로 변환하고 구둣점은 삭제
  - 원-핫 인코딩은 어떤 클래스에 해당하는 원소만 1이고 나머지는 모두 0인 벡터
    - 정수로 변환된 토큰을 원-핫 인코딩으로 변환하려면 어휘 사전 크기의 벡터가 만들어짐
  - 단어 임베딩은 정수로 변환된 토큰을 비교적 작은 크기의 실수 밀집 벡터로 변환
    - 이런 밀집 벡터는 단어 사이의 관계를 표현할 수 있기 때문에 자연어 처리에서 좋은 성능을 발휘

SECTION 9-2 마무리(2)

- 핵심 패키지와 함수
  - Keras
    - pad_sequences( ): 시퀀스 길이를 맞추기 위해 패딩을 추가
    - to_categorical ( ): 정수 시퀀스를 원-핫 인코딩으로 변환
    - SimpleRNN: 케라스의 기본 순환층 클래스
    - Embedding: 단어 임베딩을 위한 클래스

SECTION 9-2 확인 문제(1)

1. pad_sequences(5, padding='post', truncating='pre')로 했을 때 만들어질 수 없는 시퀀스는 무엇인가?

   ① [10, 5, 7, 3, 8]
   ② [0, 0, 10, 5, 7]
   ③ [5, 7, 3, 8, 0]
   ④ [7, 3, 8, 0, 0]

2. 케라스에서 제공하는 가장 기본적인 순환층 클래스는 무엇인가?

   ① RNN
   ② BaseRNN
   ③ PlainRNN
   ④ SimpleRNN

SECTION 9-2 확인 문제(2)

3. 어떤 순환층에 (100, 10) 크기의 입력이 주입. 이 순환층의 뉴런 개수는 16개. 이 층에 필요한 모델 파라미터 개수는 몇 개인가?

   ① 192
   ② 416
   ③ 432
   ④ 1,872

4. Embedding 클래스에 대해 올바르게 설명한 것은?

   ① 토큰 정숫값을 실수 벡터로 바꾸어 줌
   ② Embedding 층의 벡터는 사전에 결정된 값으로 훈련 도중 바뀌지 않음
   ③ 만들어진 벡터의 길이는 토큰마다 다름
   ④ 일반적으로 임베딩 벡터의 크기는 원-핫 인코딩 벡터보다 큼

SECTION 9-2 파이토치 버전 살펴보기(1)

- 파이토치로 순환 신경망 만들기
  - 케라스 모델과 비교하기 위해 본문과 동일한 IMDB 데이터셋을 사용해 파이토치 모델을 구현
  - 먼저 케라스에서 IMDB 데이터를 로드하고 각 샘플의 길이를 100으로 맞춤

```
from keras.datasets import imdb
from sklearn.model_selection import train_test_split

(train_input, train_target), (test_input, test_target) = imdb.load_data(
      num_words=500)
train_input, val_input, train_target, val_target = train_test_split(
     train_input, train_target, test_size=0.2, random_state=42)

from keras.preprocessing.sequence import pad_sequences

train_seq = pad_sequences(train_input, maxlen=100)
val_seq = pad_sequences(val_input, maxlen=100)
```

SECTION 9-2 파이토치 버전 살펴보기(2)

- train_seq에는 20,000개의 샘플이 있으며, 각 샘플의 길이는 100
- train_seq와 train_target의 크기 확인

```
print(train_seq.shape, train_target.shape)
```

→ (20000, 100) (20000,)

SECTION 9-2 파이토치 버전 살펴보기(3)

- 케라스로 준비한 이 데이터는 넘파이 배열
  - 파이토치의 데이터로더에 사용하려면 이를 파이토치 텐서로 변환해야 함
  - torch.tensor ( ) 함수를 사용하면 리스트나 튜플과 같은 파이썬 데이터 타입을 파이토치 텐서로 변환

```
train_seq = torch.tensor(train_seq)
val_seq = torch.tensor(val_seq)
```

  - 타깃값을 텐서로 변환
    - 먼저, 넘파이 배열의 dtype 속성을 사용해 train_target의 데이터 타입을 확인

```
print(train_target.dtype)
```

→ int64

SECTION 9-2 파이토치 버전 살펴보기(4)

- 앞의 출력 결과를 보면 train_target은 64비트 정수
  - train_target은 긍정 또는 부정을 나타내는 1과 0으로 채워져 있음
  - 그런데 파이토치 손실 함수는 입력으로 실숫값을 기대
  - 따라서 train_target과 val_target을 실수형 텐서로 변환

```
train_target = torch.tensor(train_target, dtype=torch.float32)
val_target = torch.tensor(val_target, dtype=torch.float32)
```

  - 데이터 타입이 32비트 부동소수점으로 바뀌었음을 확인

```
print(train_target.dtype)
```

→ torch.float32

SECTION 9-2 파이토치 버전 살펴보기(5)

- 데이터 로더로 훈련 세트와 검증 세트를 준비

```
from torch.utils.data import TensorDataset, DataLoader

train_dataset = TensorDataset(train_seq, train_target)
val_dataset = TensorDataset(val_seq, val_target)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

SECTION 9-2 파이토치 버전 살펴보기(6)

- 모델을 구현
  - 파이토치의 RNN 층은 모든 타임스텝의 출력과 최종 은닉 상태, 두 가지 값을 반환
    - 따라서 Sequential 클래스를 사용하여 모델을 구현하기에는 어려움이 있음
    - 대신 nn.Module의 서브 클래스를 만들어 모델을 구현하면 손쉽게 RNN 모델을 만들 수 있음
    - nn.Module 클래스를 상속하는 방법은 RNN 이외에도 다양한 구조의 모델을 만들 수 있기 때문에 파이토치 개발자들이 즐겨 사용

```
import torch.nn as nn

class IMDBRnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.embedding = nn.Embedding(500, 16)
        self.rnn = nn.RNN(16, 8, batch_first=True)
        self.dense = nn.Linear(8, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x = self.embedding(x)
        _, hidden = self.rnn(x)
        outputs = self.dense(hidden[-1])
        return self.sigmoid(outputs)
```

SECTION 9-2 파이토치 버전 살펴보기(7)

- 구현 코드 설명
  - nn.Module 클래스를 상속한 IMDBRnn 클래스를 선언
    - 생성자인 __init__( )와 정방향 계산을 담당할 forward( )
  - 파이토치에서 임베딩 층은 nn.Embedding 클래스에 구현
  - 파이토치의 기본 순환층인 nn.RNN 클래스의 객체
    - 배치 차원이 맨 앞이라는 것을 알리기 위해 batch_first 매개변수를 True로 지정 - 이 매개변수의 기본값은 False
  - nn.Linear 층 - 입력의 크기가 8이고 출력 크기가 1
  - 시그모이드 활성화 함수를 놓기 위한 nn.Sigmoid 층
  - forward( ) 메서드에는 __init__( ) 메서드에서 정의한 층을 사용해 입력에서 출력까지 층 객체를 호출
  - 각 타임스텝의 은닉 상태는 batch_first=True로 지정된 경우
    - 크기가 (배치 크기, 시퀀스 길이, 뉴런개수)이고 batch_first=False이면 (시퀀스 길이, 배치 크기, 뉴런 개수)
    - 두 번째 반환 값인 최종 은닉 상태의 크기는 (층 개수, 배치 크기, 뉴런 개수)
    - 예제는 각 샘플의 길이가 100이고 1개의 순환층과 64개의 배치를 사용하므로 첫 번째 출력의 크기는 (64, 100, 8)이고, 두 번째 출력의 크기는 (1, 64, 8)
  - hidden[-1]로 마지막 층에 해당하는 은닉 상태를 선택해 nn.Linear 객체에 전달

SECTION 9-2 파이토치 버전 살펴보기(8)

- IMDBRnn 클래스로 모델 객체를 만들고 GPU로 전달

```
model = IMDBRnn()

import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

SECTION 9-2 파이토치 버전 살펴보기(9)

- 손실 함수와 옵티마이저를 정의
  - 이진 분류의 경우 모델의 마지막 층으로 시그모이드 함수를 추가했기 때문에 nn.BCELoss를 사용
  - 만약 모델이 마지막 출력을 만들기 위해 시그모이드 함수를 사용하지 않는다면 nn.BCEWithLogitsLoss을 손실 함수로 사용
  - 옵티마이저는 이전과 동일하게 Adam을 사용하되 학습률을 기본값보다 조금 낮춰 2e-4로 설정

```
import torch.optim as optim

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=2e-4)
```

SECTION 9-2 파이토치 버전 살펴보기(10)

- 모델을 훈련하고 결과를 확인

```
train_hist = []
val_hist = []
patience = 2
best_loss = -1
early_stopping_counter = 0

epochs = 100
for epoch in range(epochs):
    model.train()
    train_loss = 0
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), targets)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
(이하 코드 생략)
```

→

SECTION 9-2 파이토치 버전 살펴보기(11)

- 출력 결과

```
에포크:1, 훈련 손실:0.7088, 검증 손실:0.7030
에포크:2, 훈련 손실:0.6992, 검증 손실:0.6970
에포크:3, 훈련 손실:0.6941, 검증 손실:0.6934
에포크:4, 훈련 손실:0.6907, 검증 손실:0.6909
에포크:5, 훈련 손실:0.6883, 검증 손실:0.6893
에포크:6, 훈련 손실:0.6865, 검증 손실:0.6875
에포크:7, 훈련 손실:0.6847, 검증 손실:0.6861
에포크:8, 훈련 손실:0.6829, 검증 손실:0.6850
에포크:9, 훈련 손실:0.6812, 검증 손실:0.6834
(중략)
에포크:49, 훈련 손실:0.5259, 검증 손실:0.5592
에포크:50, 훈련 손실:0.5237, 검증 손실:0.5593
에포크:51, 훈련 손실:0.5220, 검증 손실:0.5678
51번째 에포크에서 조기 종료되었습니다.
```

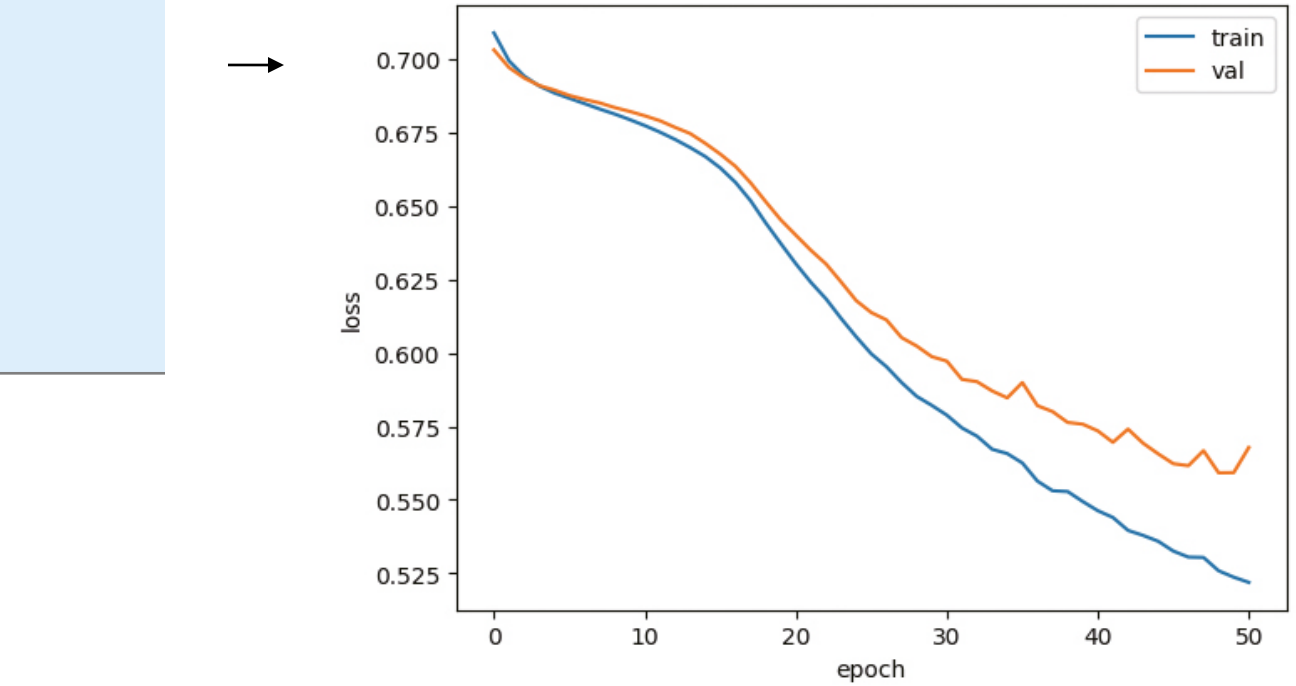
49번째 에포크가 최상의 성능을 기록했으며, 51번째 에포크에서 조기 종료

SECTION 9-2 파이토치 버전 살펴보기(12)

- 훈련 손실과 검증 손실 그래프

```
import matplotlib.pyplot as plt

plt.plot(train_hist, label='train')
plt.plot(val_hist, label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

→



검증 손실이 훈련 손실과 함께 감소하지만, 에포크가 진행될수록 두 손실 값 사이의 간격이 점점 벌어짐
마지막에는 검증 손실이 크게 증가하는데, 이를 통해 모델이 적절한 시점에 조기 종료된 것을 알 수 있음

SECTION 9-2 파이토치 버전 살펴보기(13)

- 검증 세트에 대한 모델의 정확도 확인
  - 훈련 과정에서 저장한 best_rnn_model.pt 파일을 로드한 다음 이전과 동일하게 val_loader를 사용해 올바르게 예측한 개수를 누적
  - 이 모델의 출력은 시그모이드 함수가 만든 양성 클래스에 대한 확률 값
  - 따라서 모델의 출력 값이 0.5보다 크면 양성 클래스이고, 그렇지 않으면 음성 클래스로 판별

SECTION 9-2 파이토치 버전 살펴보기(14)

- 비교를 위한 불리언 텐서 만들기
  - 출력 값 outputs와 0.5를 비교하여 모델의 출력이 양성 클래스인지 음성 클래스인지를 기록한 불리언 텐서
- 이 불리언 텐서를 squeeze( ) 메서드로 1차원 텐서로 줄인 다음 targets와 비교하여 올바른 예측 개수를 카운트

```
model.load_state_dict(torch.load('best_rnn_model.pt', weights_only=True))

model.eval()
corrects = 0
with torch.no_grad():
    for inputs, targets in val_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        outputs = model(inputs)
        predicts = outputs > 0.5
        corrects += (predicts.squeeze() == targets).sum().item()

accuracy = corrects / len(val_dataset)
print(f"검증 정확도: {accuracy:.4f}")
```

→ 검증 정확도: 0.7272

SECTION 9-2 파이토치 버전 살펴보기(15)

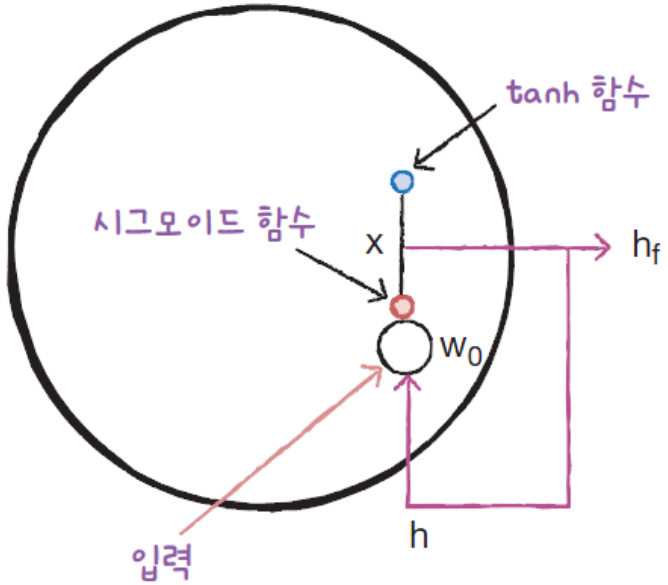
- 비교를 위한 불리언 텐서 만들기
  - 출력 값 outputs와 0.5를 비교하여 모델의 출력이 양성 클래스인지 음성 클래스인지를 기록한 불리언 텐서
- 이 불리언 텐서를 squeeze( ) 메서드로 1차원 텐서로 줄인 다음 targets와 비교하여 올바른 예측 개수를 카운트

```
model.load_state_dict(torch.load('best_rnn_model.pt', weights_only=True))

model.eval()
corrects = 0
with torch.no_grad():
    for inputs, targets in val_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        outputs = model(inputs)
        predicts = outputs > 0.5
        corrects += (predicts.squeeze() == targets).sum().item()

accuracy = corrects / len(val_dataset)
print(f"검증 정확도: {accuracy:.4f}")
```

SECTION 9-3 LSTM과 GRU 셀(1)

◦ LSTM 구조

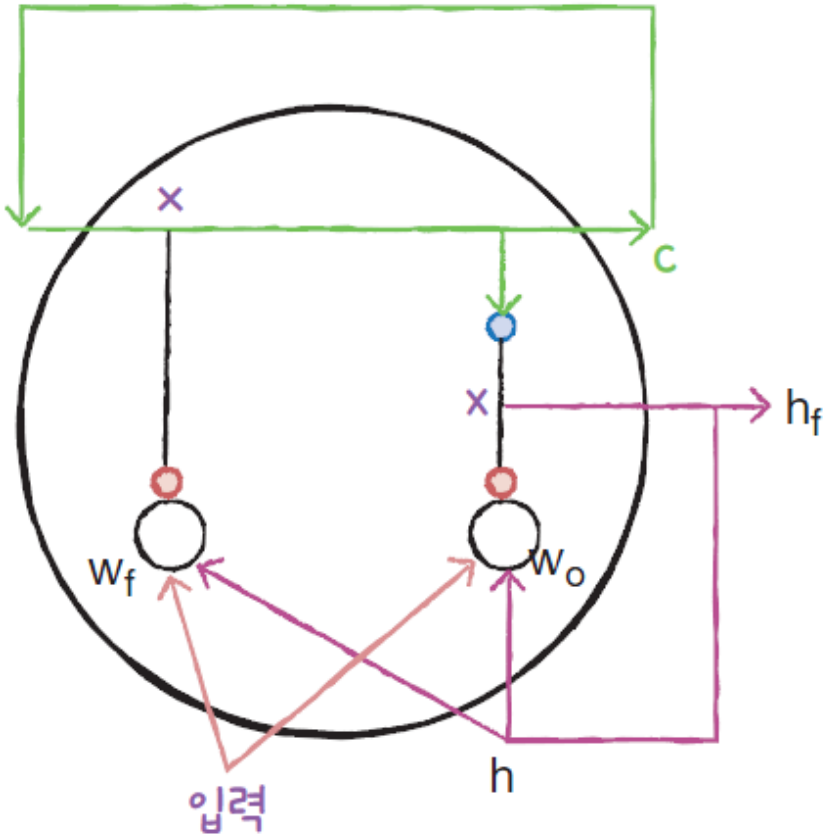
- LSTM은 Long Short-Term Memory의 약자. 단기 기억을 오래 기억하기 위해 고안
- LSTM은 입력과 가중치를 곱하고 절편을 더해 활성화 함수를 통과시키는 구조를 여러 개 가지고 있으며, 이런 계산 결과는 다음 타임스텝에 재사용
- 은닉 상태
  - 입력과 이전 타임스텝의 은닉 상태를 가중치에 곱한 후 활성화 함수를 통과시켜 다음 은닉 상태를 만듦 이때 기본 순환층과는 달리 시그모이드 활성화 함수를 사용
  - tanh 활성화 함수를 통과한 어떤 값과 곱해져서 은닉 상태를 만듦



SECTION 9-3 LSTM과 GRU 셀(2)

◦ LSTM 구조

- 셀 상태(cell state)
  - 은닉 상태와 달리 셀 상태는 다음 층으로 전달되지 않고 LSTM 셀에서 순환만 되는 값



셀 상태 계산 과정

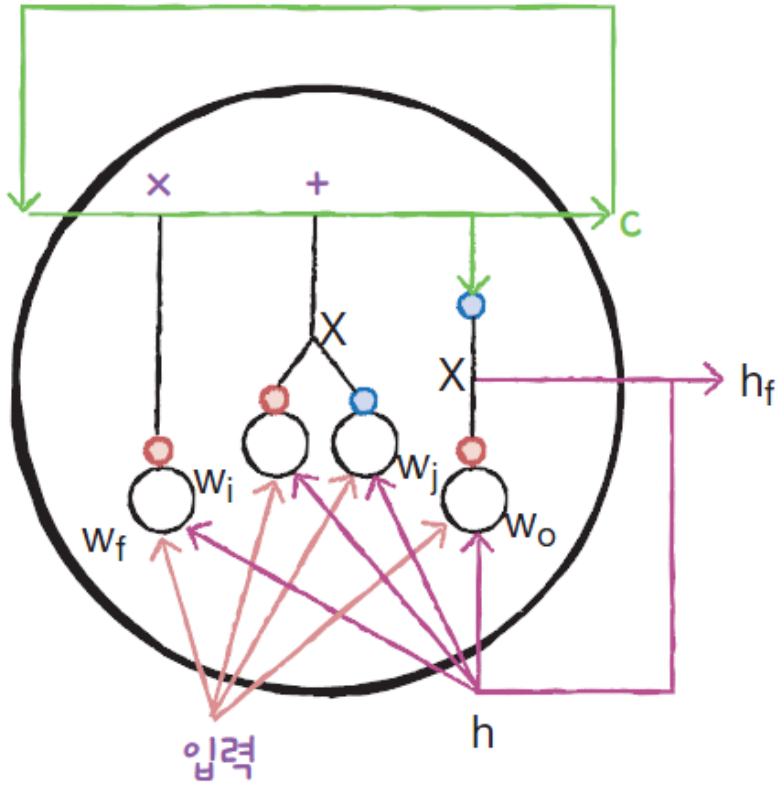
1) 먼저 입력과 은닉 상태를 또 다른 가중치 wf에 곱한 다음 시그모이드 함수를 통과시킴
2) 이전 타임스텝의 셀 상태와 곱하여 새로운 셀 상태를 만듦
3) 이 셀 상태가 오른쪽에서 tanh 함수를 통과하여 새로운 은닉 상태를 만드는 데 기여

SECTION 9-3 LSTM과 GRU 셀(3)

◦ LSTM 구조

- 셀 상태(cell state)

• LSTM은 마치 작은 셀을 여러 개 포함하고 있는 큰 셀 같음
  - 입력과 은닉 상태에 곱해지는 가중치 W_o와 W_f가 다름



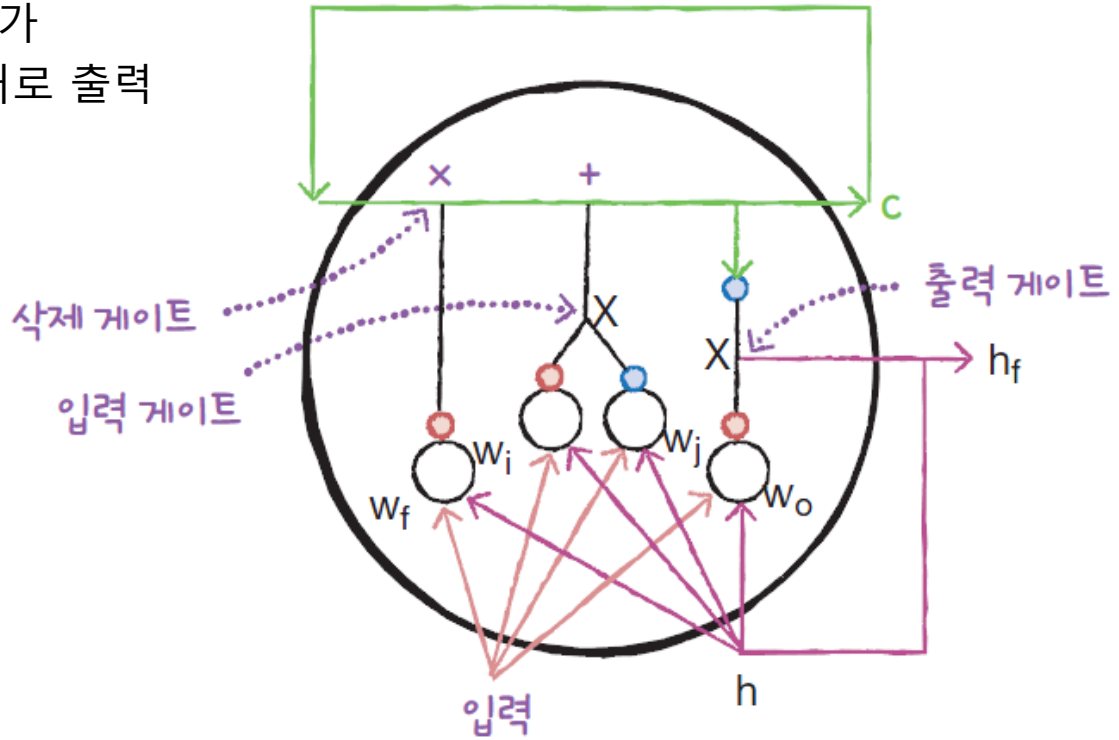
셀 상태 계산 과정

1) 입력과 은닉 상태를 각기 다른 가중치에 곱한 다음, 하나는 시그모이드 함수를 통과시키고 다른 하나는 tanh 함수를 통과시킴
2) 그다음 두 결과를 곱한 후 이전 셀 상태와 더함
3) 이 결과가 최종적인 다음 셀 상태가 됨

SECTION 9-3 LSTM과 GRU 셀(4)

◦ LSTM 구조

- 셀 상태(cell state)
  - 삭제 게이트, 입력 게이트, 출력 게이트
    - 삭제 게이트: 셀 상태에 있는 정보를 제거
    - 입력 게이트: 새로운 정보를 셀 상태에 추가
    - 출력 게이트: 이 셀 상태가 다음 은닉 상태로 출력



SECTION 9-3 LSTM과 GRU 셀(5)

◦ LSTM 신경망 훈련하기

- IMDB 리뷰 데이터를 로드하고 훈련 세트와 검증 세트로 나누기

```
from keras.datasets import imdb
from sklearn.model_selection import train_test_split

(train_input, train_target), (test_input, test_target) = imdb.load_data(
        num_words=500)
train_input, val_input, train_target, val_target = train_test_split(
        train_input, train_target, test_size=0.2, random_state=42)
```

- 케라스의 pad_sequences( ) 함수로 각 샘플의 길이를 100에 맞추고 부족할 때는 패딩을 추가

```
from keras.preprocessing.sequence import pad_sequences

train_seq = pad_sequences(train_input, maxlen=100)
val_seq = pad_sequences(val_input, maxlen=100)
```

SECTION 9-3 LSTM과 GRU 셀(6)

◦ LSTM 신경망 훈련하기

- LSTM 셀을 사용한 순환층을 만들기(SimpleRNN 클래스를 LSTM 클래스로 바꾸면 됨)

```
import keras

model_lstm = keras.Sequential()
model_lstm.add(keras.layers.Input(shape=(100,)))
model_lstm.add(keras.layers.Embedding(500, 16))
model_lstm.add(keras.layers.LSTM(8))
model_lstm.add(keras.layers.Dense(1, activation='sigmoid'))
```

SECTION 9-3 LSTM과 GRU 셀(7)

◦ LSTM 신경망 훈련하기

- 모델 구조 출력 - SimpleRNN 대신에 LSTM을 사용

```
model_lstm.summary()
```

→

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 16)	8,000
lstm (LSTM)	(None, 8)	800
dense (Dense)	(None, 1)	9

Total params: 8,809 (34.41 KB)

Trainable params: 8,809 (34.41 KB)

Non-trainable params: 0 (0.00 B)

▲ SimpleRNN 클래스의 모델 파라미터 개수는 200개였음
LSTM 셀에는 작은 셀이 4개 있으므로 정확히 4배가 늘어
모델 파라미터 개수는 800개

SECTION 9-3 LSTM과 GRU 셀(8)

◦ LSTM 신경망 훈련하기

- 모델을 컴파일하고 훈련 -이전과 같이 배치 크기는 64개, 에포크 횟수는 100으로 지정

```
model_lstm.compile(optimizer='adam', loss='binary_crossentropy',
                   metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-lstm-model.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)
history = model_lstm.fit(train_seq, train_target, epochs=100, batch_size=64,
                         validation_data=(val_seq, val_target),
                         callbacks=[checkpoint_cb, early_stopping_cb])
```

▶ 출력 결과는 다음 쪽에 이어짐

SECTION 9-3 LSTM과 GRU 셀(9)

◦ LSTM 신경망 훈련하기

- 모델을 컴파일하고 훈련 -이전과 같이 배치 크기는 64개, 에포크 횟수는 100으로 지정
- 검증 세트에 대한 정확도를 보면 약 80% 정도로 SimpleRNN 클래스를 사용했을 때보다 향상됨

◀ 앞쪽 코드 출력 결과

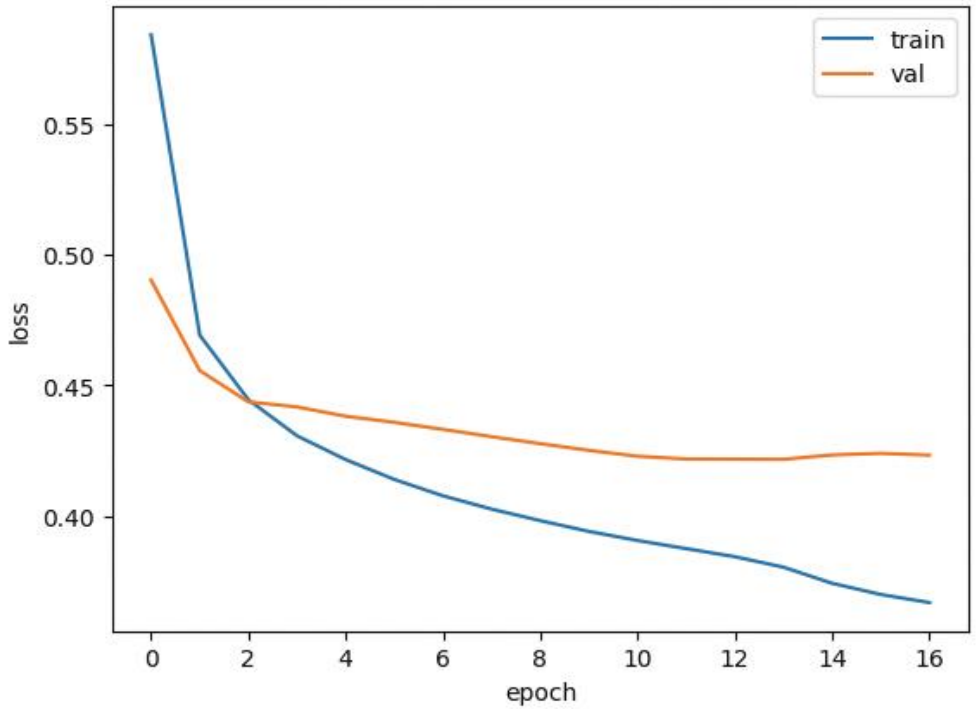
```
Epoch 1/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step -
accuracy:
0.6052 - loss: 0.6481 - val_accuracy: 0.7632 - val_loss: 0.4902
Epoch 2/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy:
0.7781 - loss: 0.4808 - val_accuracy: 0.7874 - val_loss: 0.4556
Epoch 3/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy:
0.7947 - loss: 0.4495 - val_accuracy: 0.7922 - val_loss: 0.4437
(중략)
Epoch 16/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy:
0.8272 - loss: 0.3718 - val_accuracy: 0.8086 - val_loss: 0.4240
Epoch 17/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy:
0.8291 - loss: 0.3672 - val_accuracy: 0.8088 - val_loss: 0.4233
```

SECTION 9-3 LSTM과 GRU 셀(10)

◦ LSTM 신경망 훈련하기

- 훈련 손실과 검증 손실 그래프

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```



훈련 손실이 잘 줄어들고 있지만 과대적합을 잘 억제하지 못한 것 같음

SECTION 9-3 LSTM과 GRU 셀(11)

◦ 순환층에 드롭아웃 적용하기

- 순환층은 자체적으로 드롭아웃 기능을 제공
  - dropout 매개변수는 셀의 입력에 드롭아웃을 적용
  - recurrent_dropout은 순환되는 은닉 상태에 드롭아웃을 적용
    -하지만 기술적인 문제로 인해 recurrent_dropout을 사용하면 GPU를 사용하여 모델의 훈련 속도가 크게 느려짐

  - LSTM 클래스에 dropout 매개변수를 0.2으로 지정하여 20%의 입력을 드롭아웃

```
model_dropout = keras.Sequential()
model_dropout.add(keras.layers.Input(shape=(100,)))
model_dropout.add(keras.layers.Embedding(500, 16))
model_dropout.add(keras.layers.LSTM(8, dropout=0.2))
model_dropout.add(keras.layers.Dense(1, activation='sigmoid'))
```

SECTION 9-3 LSTM과 GRU 셀(12)

◦ 순환층에 드롭아웃 적용하기

- 이전과 동일 조건으로 모델 훈련

```
model_dropout.compile(optimizer='adam', loss='binary_crossentropy',
                      metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-dropout-model.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)
history = model_dropout.fit(train_seq, train_target, epochs=100, batch_size=64,
                            validation_data=(val_seq, val_target),
                            callbacks=[checkpoint_cb, early_stopping_cb])
```

▶ 출력 결과는 다음 쪽에 이어짐

SECTION 9-3 LSTM과 GRU 셀(13)

◦ 순환층에 드롭아웃 적용하기

- 드롭아웃을 추가했더니 모델의 성능이 약간 줄어든 것 같음

◀ 앞쪽 코드 출력 결과

```
Epoch 1/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy:
0.5852 - loss: 0.6662 - val_accuracy: 0.7528 - val_loss: 0.5231
Epoch 2/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy:
0.7606 - loss: 0.5058 - val_accuracy: 0.7748 - val_loss: 0.4804
Epoch 3/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy:
0.7853 - loss: 0.4649 - val_accuracy: 0.7868 - val_loss: 0.4625
(중략)
Epoch 8/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy:
0.8107 - loss: 0.4132 - val_accuracy: 0.7950 - val_loss: 0.4418
Epoch 9/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy:
0.8134 - loss: 0.4100 - val_accuracy: 0.7964 - val_loss: 0.4382
```

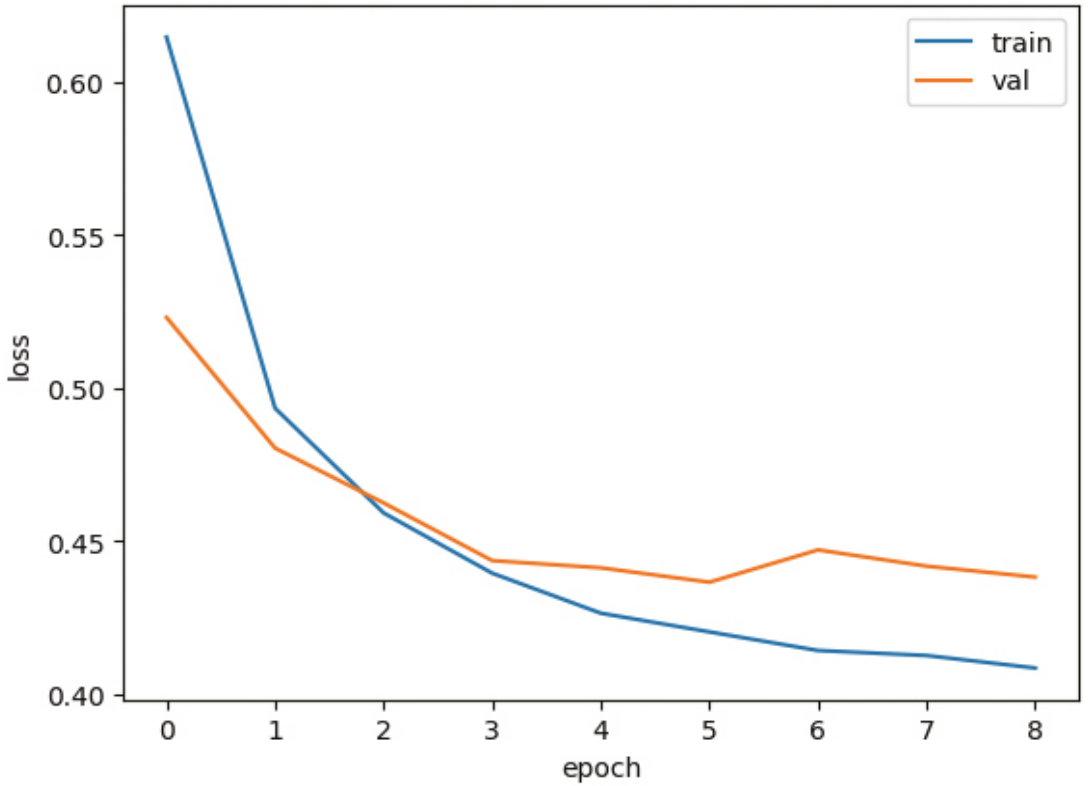
SECTION 9-3 LSTM과 GRU 셀(14)

◦ 순환층에 드롭아웃 적용하기

- 훈련 손실과 검증 손실 그래프

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

→

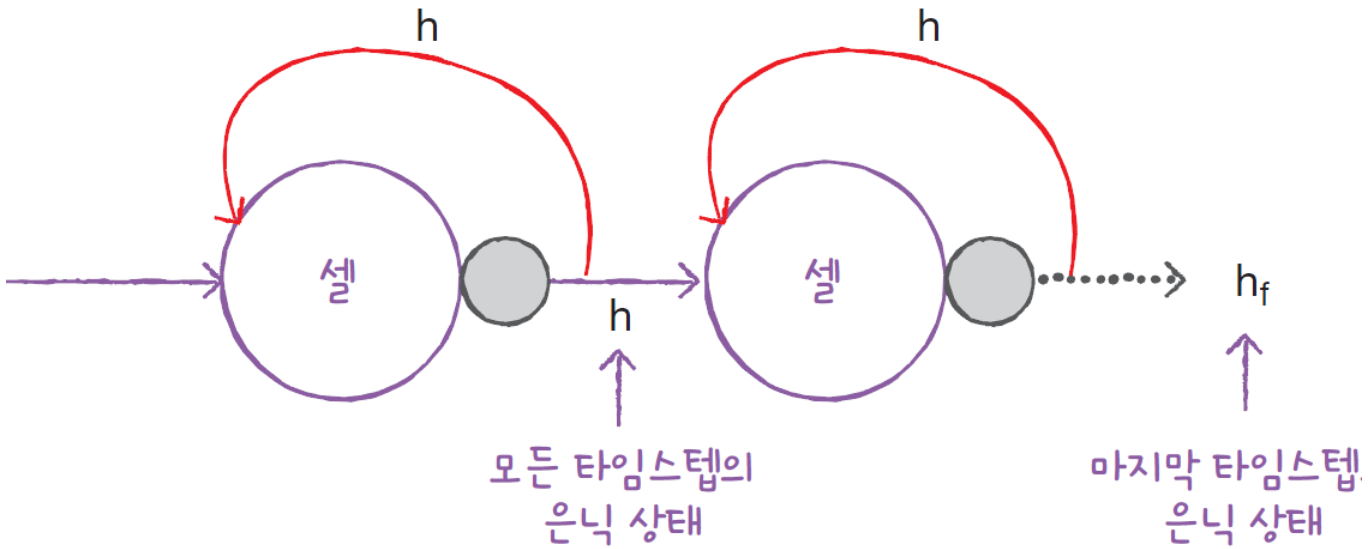


LSTM 층에 적용한 드롭아웃 덕분에 훈련 손실이 줄어드는 것을
조금 억제했지만 검증 손설이 더 나아지지는 않음

SECTION 9-3 LSTM과 GRU 셀(15)

◦ 2개의 층을 연결하기

- 순환층의 은닉 상태는 샘플의 마지막 타임스텝에 대한 은닉 상태만 다음 층으로 전달
- 하지만 순환층을 쌓게 되면 모든 순환층에 순차 데이터가 필요
- 따라서 앞쪽의 순환층이 모든 타임스텝에 대한 은닉 상태를 출력해야 함
- 마지막 순환층만 마지막 타임스텝의 은닉 상태를 출력해야 함



SECTION 9-3 LSTM과 GRU 셀(16)

◦ 2개의 층을 연결하기

- 케라스의 순환층에서 모든 타임스텝의 은닉 상태를 출력
  - 마지막을 제외한 다른 모든 순환층에서 return_sequences 매개변수를 True로 지정

```
odel_2lstm = keras.Sequential()
model_2lstm.add(keras.layers.Input(shape=(100,)))
model_2lstm.add(keras.layers.Embedding(500, 16))
model_2lstm.add(keras.layers.LSTM(8, dropout=0.2, return_sequences=True))
model_2lstm.add(keras.layers.LSTM(8, dropout=0.2))
model_2lstm.add(keras.layers.Dense(1, activation='sigmoid'))
model_2lstm.summary()
```

- 2개의 LSTM 층을 쌓았고 모두 드롭아웃을 0.2으로 지정
- 첫 번째 LSTM 클래스에는 return_sequences 매개변수를 True로 지정

SECTION 9-3 LSTM과 GRU 셀(17)

◦ 2개의 층을 연결하기

- summary( ) 메서드의 결과 확인

```
model_2lstm.summary()
```

→

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 100, 16)	8,000
lstm_2 (LSTM)	(None, 100, 8)	800
lstm_3 (LSTM)	(None, 8)	544
dense_2 (Dense)	(None, 1)	9

Total params: 9,353 (36.54 KB)

Trainable params: 9,353 (36.54 KB)

Non-trainable params: 0 (0.00 B)

SECTION 9-3 LSTM과 GRU 셀(18)

◦ 2개의 층을 연결하기

- 모델을 훈련

```
model_2lstm.compile(optimizer='adam', loss='binary_crossentropy',
                    metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-2lstm-model.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)
history = model_2lstm.fit(train_seq, train_target, epochs=100, batch_size=64,
                    validation_data=(val_seq, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

▶ 출력 결과는 다음 쪽에 이어짐

SECTION 9-3 LSTM과 GRU 셀(19)

◦ 2개의 층을 연결하기

• 모델을 훈련

◀ 앞쪽 코드 출력 결과

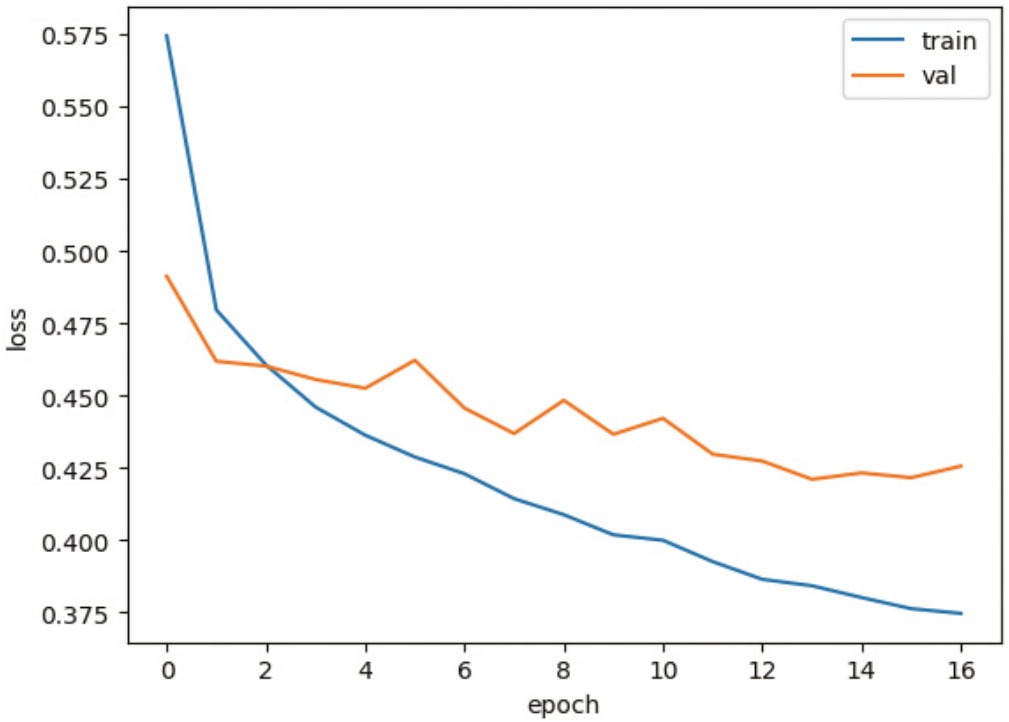
```
Epoch 1/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy:
0.6153 - loss: 0.6340 - val_accuracy: 0.7670 - val_loss: 0.4913
Epoch 2/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy:
0.7712 - loss: 0.4857 - val_accuracy: 0.7862 - val_loss: 0.4619
(중략)
Epoch 15/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 13ms/step - accuracy:
0.8317 - loss: 0.3782 - val_accuracy: 0.8050 - val_loss: 0.4233
Epoch 16/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy:
0.8346 - loss: 0.3715 - val_accuracy: 0.8042 - val_loss: 0.4216
Epoch 17/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy:
0.8349 - loss: 0.3723 - val_accuracy: 0.8094 - val_loss: 0.4256
```

SECTION 9-3 LSTM과 GRU 셀(20)

◦ 2개의 층을 연결하기

- 손실 그래프로 훈련 과정 확인

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

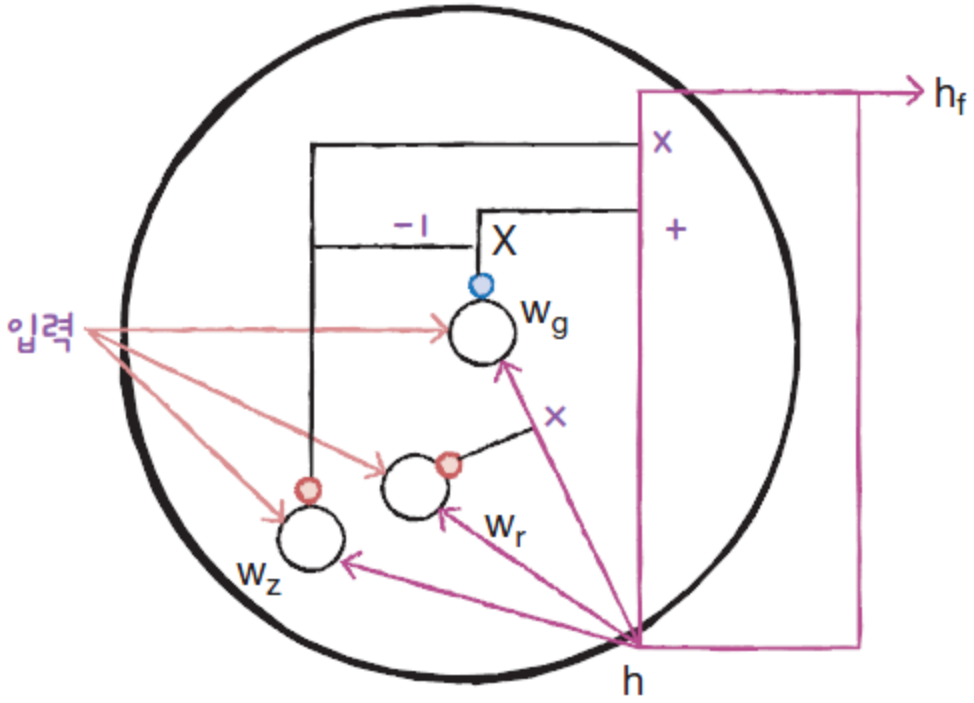


과대적합을 억제하기 위해 노력하면서 손실을 최대한 낮춤

SECTION 9-3 LSTM과 GRU 셀(21)

◦ GRU 구조

- GRU(Gated Recurrent Unit)는 LSTM을 간소화한 버전
- 이 셀은 LSTM처럼 셀 상태를 계산하지 않고 은닉 상태 하나만 포함



SECTION 9-3 LSTM과 GRU 셀(22)

◦ GRU 신경망 훈련하기

```
model_gru = keras.Sequential()
model_gru.add(keras.layers.Input(shape=(100,)))
model_gru.add(keras.layers.Embedding(500, 16))
model_gru.add(keras.layers.GRU(8, dropout=0.2))
model_gru.add(keras.layers.Dense(1, activation='sigmoid'))
```

- 모델의 구조 확인

```
model_gru.summary()
```

→

Model: "sequential_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 100, 16)	8,000
gru (GRU)	(None, 8)	624
dense_3 (Dense)	(None, 1)	9

Total params: 8,633 (33.72 KB)

Trainable params: 8,633 (33.72 KB)

Non-trainable params: 0 (0.00 B)

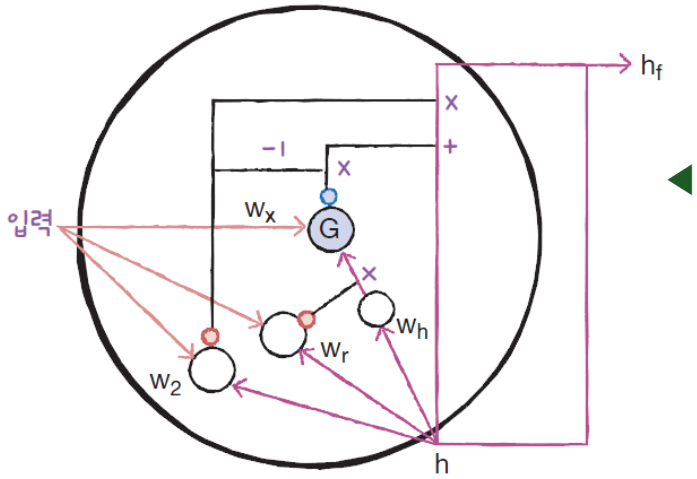
SECTION 9-3 LSTM과 GRU 셀(23)

◦ GRU 신경망 훈련하기

- GRU 층의 모델 파라미터 개수를 계산

  • GRU 셀에는 3개의 작은 셀이 있고, 작은셀에는 입력과 은닉 상태에 곱하는 가중치와 절편
  - 입력에 곱하는 가중치는 16 × 8 =128개
  - 은닉 상태에 곱하는 가중치는 8 × 8 = 64개
  - 절편은 뉴런마다 하나씩이므로 8개
  - 모두 더하면 128 + 64 + 8 = 200개
  - 이런 작은 셀이 3개이므로 모두 600개 의 모델 파라미터가 필요

  • summary( ) 메서드의 출력은 624개



◀ 입력과 은닉 상태에 곱해지는 가중치를 wx와 wh로 나누어 계산하면 은닉 상태에 곱해지는 가중치 외에 절편이 별도로 필요
따라서 작은 셀마다 하나씩 절편이 추가되고 8개의 뉴런이 있으므로 총 24개의 모델 파라미터가 더해짐
따라서 GRU 층의 총 모델 파라미터 개수는 624개

SECTION 9-3 LSTM과 GRU 셀(24)

◦ GRU 신경망 훈련하기

• GRU 셀을 사용한 순환 신경망 훈련

```
model_gru.compile(optimizer='adam', loss='binary_crossentropy',
                  metrics=['accuracy'])
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-gru-model.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)
history = model_gru.fit(train_seq, train_target, epochs=100, batch_size=64,
                        validation_data=(val_seq, val_target),
                        callbacks=[checkpoint_cb, early_stopping_cb])
```

▶ 출력 결과는 다음 쪽에 이어짐

SECTION 9-3 LSTM과 GRU 셀(25)

- GRU 신경망 훈련하기
  - LSTM와 거의 비슷한 성능을 보임
    - 앞쪽 코드 출력 결과

```
Epoch 1/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy:
0.5616 - loss: 0.6737 - val_accuracy: 0.7552 - val_loss: 0.5116
Epoch 2/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy:
0.7562 - loss: 0.5036 - val_accuracy: 0.7536 - val_loss: 0.5063
(중략)
Epoch 15/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step -
accuracy:
0.8248 - loss: 0.3905 - val_accuracy: 0.8058 - val_loss: 0.4320
Epoch 16/100
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy:
0.8283 - loss: 0.3842 - val_accuracy: 0.8034 - val_loss: 0.4320
```

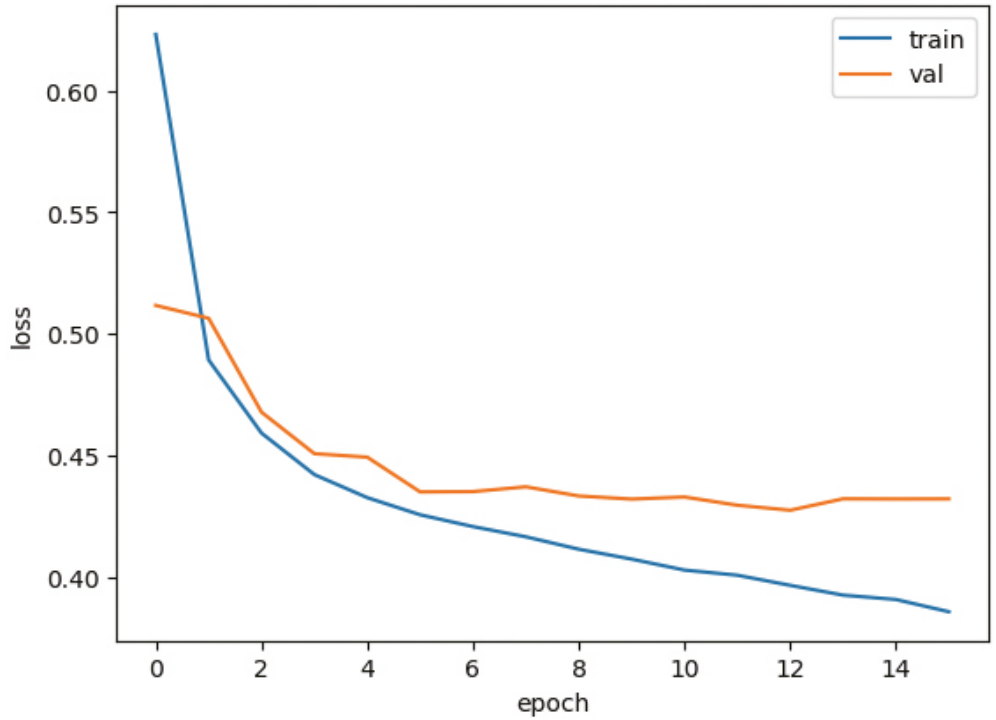
SECTION 9-3 LSTM과 GRU 셀(26)

◦ GRU 신경망 훈련하기

• 모델의 손실 그래프

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

→



SECTION 9-3 LSTM과 GRU 셀(27)

◦ LSTM과 GRU 셀로 훈련(문제해결 과정)

- 학습
  - 순환 신경망에서 가장 인기 있는 LSTM과 GRU 셀
  - 순환층에 드롭아웃 적용과 순환층을 쌓는 방법
  - 마지막에 훈련한 GRU 모델을 다시 로드하여 테스트 세트에 대한 성능을 확인
    - 테스트 세트를 훈련 세트와 동일한 방식으로 변환
    - load_model( ) 함수로 best-2rnn-model.h5 파일을 읽고 evaluate ( ) 메서드로 테스트 세트에서 성능을 계산

```
test_seq = pad_sequences(test_input, maxlen=100)
best_model = keras.models.load_model('best-gru-model.keras')
best_model.evaluate(test_seq, test_target)
```

→

782/782 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8082 - loss: 0.4215
[0.4206896126270294, 0.8072400093078613]

SECTION 9-3 마무리

- 키워드로 끝나는 핵심 포인트
  - LSTM 셀은 타임스텝이 긴 데이터를 효과적으로 학습하기 위해 고안된 순환층
    - 입력 게이트, 삭제 게이트, 출력 게이트 역할을 하는 작은 셀이 포함
  - LSTM 셀은 은닉 상태 외에 셀 상태를 출력
    - 셀 상태는 다음 층으로 전달되지 않으며 현재 셀에서만 순환
  - GRU 셀은 LSTM 셀의 간소화 버전으로 생각할 수 있지만 LSTM 셀에 못지않는 성능

- 핵심 패키지와 함수
  - TensorFlow
    - LSTM: LSTM 셀을 사용한 순환층 클래스
    - GRU: GRU 셀을 사용한 순환층 클래스

SECTION 9-3 확인 문제(1)

1. 다음 중 텐서플로에서 제공하는 순환층 클래스가 아닌 것은?

   ① SimpleRNN
   ② LSTM
   ③ GRU
   ④ Conv2D

2. LSTM 층에 있는 게이트가 아닌 것은?

   ① 순환 게이트
   ② 삭제 게이트
   ③ 입력 게이트
   ④ 출력 게이트

SECTION 9-3 확인 문제(2)

3. 순환층을 2개 이상 쌓을 때 마지막 층을 제외하고는 모든 타임스텝의 은닉 상태를 출력하기 위해 지정해야 할 매개변수는?

① return_seq
② return_sequences
③ return_series
④ return_hidden

4. GRU 셀에 대해 잘못 설명한 것은?

① 순차 데이터에 적용할 수 있는 순환 신경망의 한 종류임
② LSTM 셀의 간소화 버전으로 생각할 수 있음
③ LSTM 셀보다 모델 파라미터가 더 많음
④ 케라스는 과대적합을 억제하기 위한 드롭아웃 옵션을 제공

SECTION 9-3 파이토치 버전 살펴보기(1)

- 파이토치로 LSTM 모델 훈련하기
  - 파이토치에서 두 개의 LSTM 층을 쌓아 IMDB 리뷰 데이터를 분류
  - 이전과 동일한 방식으로 훈련 데이터를 준비
    - 케라스에서 IMDB 데이터를 로드
    - 훈련 세트와 테스트 세트로 나눔
    - 패딩을 추가하여 파이토치 텐서로 변환
    - 훈련 세트와 검증 세트를 위한 데이터로더를 생성

```
from keras.datasets import imdb
from sklearn.model_selection import train_test_split

(train_input, train_target), (test_input, test_target) = imdb.load_data(
        num_words=500)
train_input, val_input, train_target, val_target = train_test_split(
        train_input, train_target, test_size=0.2, random_state=42)
from keras.preprocessing.sequence import pad_sequences
(이하 생략)
```

SECTION 9-3 파이토치 버전 살펴보기(2)

- 모델 구현

```
import torch.nn as nn

class IMDBLstm(nn.Module):
    def __init__(self):
        super().__init__()
        self.embedding = nn.Embedding(500, 16)
        self.lstm = nn.LSTM(16, 8, batch_first=True, num_layers=2, dropout=0.2)
        self.dense = nn.Linear(8, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.lstm(x)
        outputs = self.dense(hidden[-1])
        return self.sigmoid(outputs)
```

SECTION 9-3 파이토치 버전 살펴보기(3)

- 모델 훈련과 결과
  - 모델을 만들고 훈련하는 코드는 이전과 동일
  - 이번에는 Adam 옵티마이저의 학습률로 기본값을 그대로 사용

```
model = IMDBLstm()

import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

import torch.optim as optim

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters())
(이하 생략)
```

→

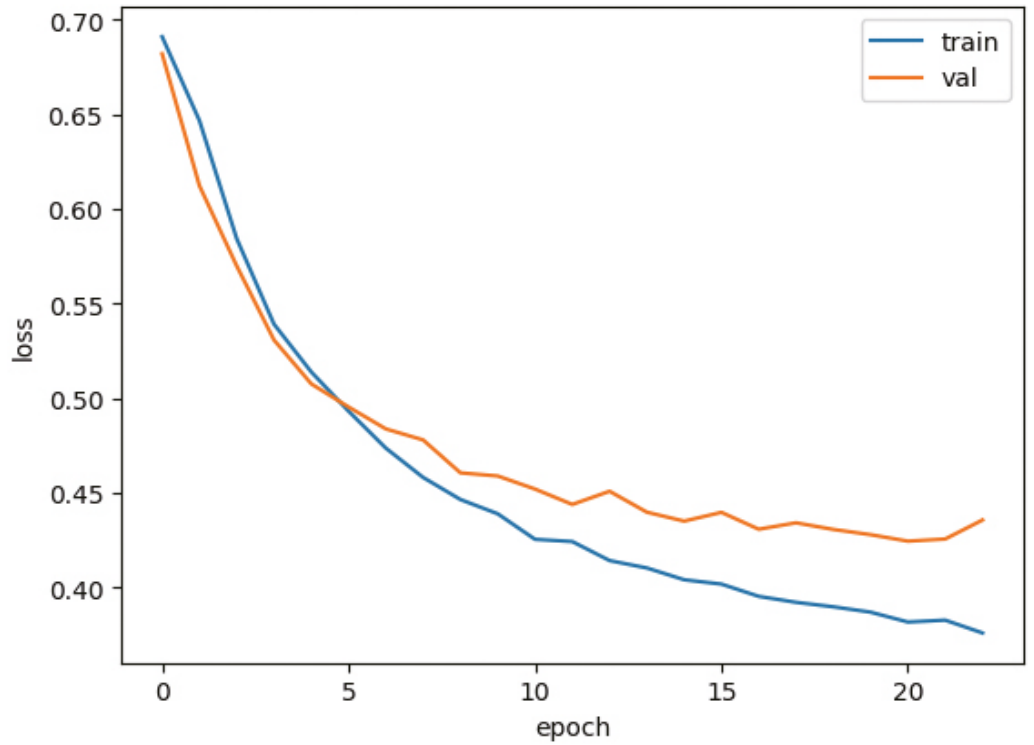
```
에포크:1, 훈련 손실:0.6910, 검증 손실:0.6820
에포크:2, 훈련 손실:0.6467, 검증 손실:0.6123
에포크:3, 훈련 손실:0.5843, 검증 손실:0.5699
(중략)
에포크:22, 훈련 손실:0.3827, 검증 손실:0.4256
에포크:23, 훈련 손실:0.3760, 검증 손실:0.4356
23번째 에포크에서 조기 종료되었습니다.
```

SECTION 9-3 파이토치 버전 살펴보기(4)

- 훈련 손실과 검증 손실 그래프

```
import matplotlib.pyplot as plt

plt.plot(train_hist, label='train')
plt.plot(val_hist, label='val')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```

→



훈련 손실과 검증 손실이 어느정도 간격을 유지하면서 감소하다가, 21번째 에포크 이후부터 두 값의 차이가 크게 벌어지는 것을 확인
- 과대적합이 발생하기 전에 적절하게 훈련이 멈춘 것

SECTION 9-3 파이토치 버전 살펴보기(5)

- 검증 세트에 대한 정확도 확인
  - best_2lstm_model.pt 파일을 로드

```
model.load_state_dict(torch.load('best_2lstm_model.pt', weights_only=True))

model.eval()
corrects = 0
with torch.no_grad():
    for inputs, targets in val_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        outputs = model(inputs)
        predicts = outputs > 0.5
        corrects += (predicts.squeeze() == targets).sum().item()

accuracy = corrects / len(val_dataset)
print(f"검증 정확도: {accuracy:.4f}")
```

→ 검증 정확도: 0.8014

SECTION 9-3 파이토치 버전 살펴보기(6)

- 테스트 세트에 대한 모델의 성능 확인
  - 앞서 test_input과 test_target으로 데이터를 분리
  - 길이가 100이 되도록 자른 다음 패딩을 추가하고 파이토치 텐서로 변환
  - 데이터로더를 만들어 모델에 전달해서 정확도를 계산

```
test_seq = pad_sequences(test_input, maxlen=100)
test_seq = torch.tensor(test_seq)
test_target = torch.tensor(test_target, dtype=torch.float32)

test_dataset = TensorDataset(test_seq, test_target)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

model.eval()
corrects = 0
with torch.no_grad():
    for inputs, targets in test_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        outputs = model(inputs)
        predicts = outputs > 0.5
        corrects += (predicts.squeeze() == targets).sum().item()

accuracy = corrects / len(test_dataset)
print(f"테스트 정확도: {accuracy:.4f}")
```

→ 테스트 정확도: 0.8072

SECTION 9-3 자주하는 질문

- 09-2절에서 train_seq에 있는 정숫값이 Embedding 층에 있는 벡터로 어떻게 변환되는 것인가?
- 09-2절에서 입력을 제로 패딩하여 모델에 주입하면 훈련할 때 문제가 되지 않나?
- 09-3절에서 GRU 셀의 모델 파라미터 개수를 계산할 때 절편을 추가한 이유가 무엇인가?