강사 윤영주
yj.youn1103@gmail.com

넷째마당
딥러닝 기본기 다지기

10장 딥러닝 모델 설계하기

1 모델의 정의

2 입력층, 은닉층, 출력층

3 모델 컴파일

4 모델 실행하기

딥러닝 모델 설계하기

- **딥러닝 모델 설계하기**
  - 가끔 어려운 수식이 나오기도 했지만, 모든 개념이 머릿속에 정리되었다면 여러분을 딥러닝의 세계로 성큼성큼 안내해 줄 텐서플로, 케라스와 함께 지금부터 초고속으로 전진할 것
  - 지금부터는 딥러닝의 기본 개념들이 실전에서는 어떤 방식으로 구현되는지, 왜 우리가 그 어려운 개념들을 익혀야 했는지 공부하겠음
  - 2장에서 소개한 '폐암 수술 환자의 생존율 예측하기' 예제를 기억하나요? 당시에는 딥러닝 모델 부분을 자세히 설명할 수 없었지만, 이제 설명할 수 있음
  - 여러분은 앞서 배운 내용이 이 짧은 코드 안에 모두 들어 있다는 사실에 감탄할지도 모름
  - 머릿속에 차곡차곡 들어찬 딥러닝의 개념들이 어떻게 활용되는지 지금부터 함께 알아보자

1 모델의 정의

1 모델의 정의

- **모델의 정의**
  - '폐암 수술 환자의 생존율 예측하기'의 딥러닝 코드를 다시 한 번 옮겨 보면 다음과 같음

```
# 텐서플로 라이브러리 안에 있는 케라스 API에서 필요한 함수들을 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 데이터를 다루는 데 필요한 라이브러리를 불러옵니다.
import numpy as np

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git
```

1 모델의 정의

- 모델의 정의

```
# 준비된 수술 환자 데이터를 불러옵니다.
Data_set = np.loadtxt("./data/ThoracicSurgery3.csv", delimiter=",")

X = Data_set[:,0:16]  # 환자의 진찰 기록을 X로 지정합니다.
y = Data_set[:,16]    # 수술 1년 후 사망/생존 여부를 y로 지정합니다.

# 딥러닝 모델의 구조를 결정합니다.
model = Sequential()
model.add(Dense(30, input_dim=16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

1 모델의 정의

- 모델의 정의

```
# 딥러닝 모델을 실행합니다.
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
history = model.fit(X, y, epochs=5, batch_size=16)
```

1 모델의 정의

- **모델의 정의**
  - 이 코드에서 데이터를 불러오고 다루는 부분은 2장에서 이미 살펴보았으므로 여기서는 실제로 딥러닝이 수행되는 부분을 더 자세히 알아보자
  - 딥러닝의 모델을 설정하고 구동하는 부분은 모두 **model**이라는 함수를 선언하며 시작
  - 먼저 model = Sequential()로 시작되는 부분은 딥러닝의 구조를 짜고 층을 설정하는 부분
  - 이어서 나오는 model.compile( ) 부분은 앞에서 정한 모델을 컴퓨터가 알아들을 수 있게끔 컴파일하는 부분
  - model.fit()으로 시작하는 부분은 모델을 실제로 수행하는 부분

2 입력층, 은닉층, 출력층

2 입력층, 은닉층, 출력층

- **입력층, 은닉층, 출력층**
  - 먼저 딥러닝의 구조를 짜고 층을 설정하는 부분을 살펴보면 다음과 같음

```
model = Sequential()
model.add(Dense(30, input_dim=16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

2 입력층, 은닉층, 출력층

- **입력층, 은닉층, 출력층**
  - 셋째 마당에서 딥러닝이란 입력층과 출력층 사이에 은닉층들을 차곡차곡 추가하면서 학습시키는 것임을 배웠음
  - 이 층들이 케라스에서는 Sequential( ) 함수를 통해 쉽게 구현
  - Sequential() 함수를 model로 선언해 놓고 model.add()라는 라인을 추가하면 새로운 층이 만들어짐
  - 코드에는 model.add( )로 시작되는 라인이 두 개 있으므로 층을 두 개 가진 모델을 만든 것
  - 맨 마지막 층은 결과를 출력하는 '출력층'이 됨
  - 나머지는 모두 '은닉층'의 역할을 함
  - 지금 만들어진 이 층 두 개는 각각 은닉층과 출력층
  - 각각의 층은 Dense라는 함수를 통해 구체적으로 그 구조가 결정

2 입력층, 은닉층, 출력층

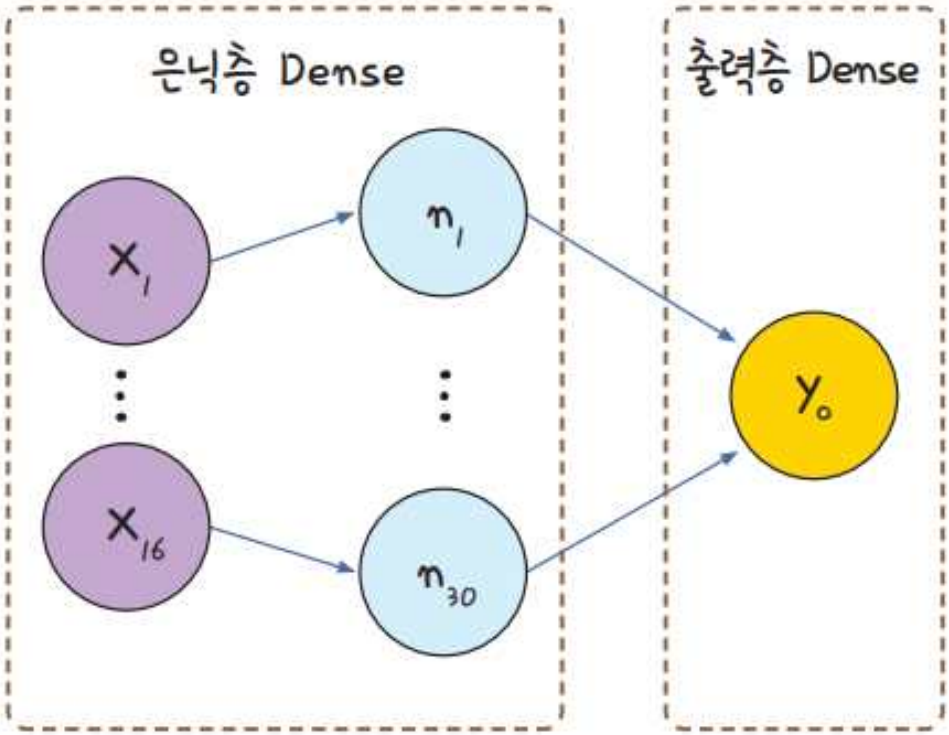
- **입력층, 은닉층, 출력층**
  - 이제 model.add(Dense(30, input_dim=16)) 부분을 더 살펴보자
  - model.add( ) 함수를 통해 새로운 층을 만들고 나면 Dense() 함수의 첫 번째 인자에 몇 개의 노드를 이 층에 만들 것인지 숫자를 적어 줌
  - 노드란 앞서 소개된 '가중합'에 해당하는 것으로 이전 층에서 전달된 변수와 가중치, 바이어스가 하나로 모이게 되는 곳
  - 하나의 층에 여러 개의 노드를 적절히 만들어 주어야 하는데, 30이라고 되어 있는 것은 이 층에 노드를 30개 만들겠다는 것

2 입력층, 은닉층, 출력층

- **입력층, 은닉층, 출력층**
  - 이어서 input_dim이라는 변수가 나옴
  - 이는 입력 데이터에서 몇 개의 값을 가져올지 정하는 것
  - keras는 입력층을 따로 만드는 것이 아니라, 첫 번째 은닉층에 input_dim을 적어 줌으로써 첫 번째 Dense가 은닉층 + 입력층의 역할을 겸함
  - 우리가 다루고 있는 폐암 수술 환자의 생존 여부 데이터에는 입력 값이 16개 있음
  - 데이터에서 값을 16개 받아 은닉층의 노드 30개로 보낸다는 의미

2 입력층, 은닉층, 출력층

▼ 그림 10-1 | 첫 번째 Dense는 입력층과 첫 번째 은닉층을, 두 번째 Dense는 출력층을 의미



2 입력층, 은닉층, 출력층

- **입력층, 은닉층, 출력층**
  - 이제 두 번째 나오는 model.add(Dense(1, activation='sigmoid'))를 보겠음
  - 마지막 층이므로 이 층이 곧 출력층이 됨
  - 출력 값을 하나로 정해서 보여 주어야 하므로 출력층의 노드 수는 한 개
  - 이 노드에서 입력받은 값은 활성화 함수를 거쳐 최종 출력 값으로 나와야 함
  - 여기서는 활성화 함수로 시그모이드(sigmoid) 함수를 사용

3 모델 컴파일

3 모델 컴파일

- **모델 컴파일**
  - 다음으로 model.compile 부분

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
```

3 모델 컴파일

- **모델 컴파일**
  - model.compile 부분은 앞서 지정한 모델이 효과적으로 구현될 수 있게 여러 가지 환경을 설정해 주면서 컴파일하는 부분
  - 먼저 어떤 오차 함수를 사용할지 정해야 함
  - 우리는 5장에서 손실 함수에는 두 가지 종류가 있음을 배웠음
  - 바로 선형 회귀에서 사용한 평균 제곱 오차와 로지스틱 회귀에서 사용한 교차 엔트로피 오차
  - 폐암 수술 환자의 생존율 예측은 생존과 사망, 둘 중 하나를 예측하므로 교차 엔트로피 오차 함수를 적용하기 위해 binary_crossentropy를 선택

3 모델 컴파일

- **모델 컴파일**
  - 손실 함수는 최적의 가중치를 학습하기 위해 필수적인 부분
  - 올바른 손실 함수를 통해 계산된 오차는 옵티마이저를 적절히 활용하도록 만들어 줌
  - 케라스는 쉽게 사용 가능한 여러 가지 손실 함수를 준비해 놓고 있음
  - 크게 평균 제곱 오차 계열과 교차 엔트로피 계열 오차로 나뉘는데, 이를 표 10-1에 정리해 놓았음
  - 선형 회귀 모델은 평균 제곱 계열 중 하나를, 이항 분류를 위해서는 binary_crossentropy를, 그리고 다항 분류에서는 categorical_crossentropy를 사용한다는 것을 기억하자

3 모델 컴파일

▼ 표 10-1 | 대표적인 오차 함수

* 실제 값을 y_t, 예측 값을 y_o라고 가정할 때

평균 제곱 계열 (선형 회귀 모델)	mean_squared_error	평균 제곱 오차 계산: mean(square(y_t - y_o))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(y_t - y_o))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(y_t - y_o)/abs(y_t)) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(y_o) + 1) - (log(y_t) + 1)))
교차 엔트로피 계열 (다항 분류, 이항 분류)	categorical_crossentropy	범주형 교차 엔트로피(다항 분류, 여럿 중 하나를 예측할 때)
	binary_crossentropy	이항 교차 엔트로피(이항 분류, 둘 중 하나를 예측할 때)

3 모델 컴파일

- **모델 컴파일**
  - 이어서 옵티마이저를 선택할 차례
  - 앞 장에서 현재 가장 많이 쓰는 옵티마이저는 adam이라고 했음
  - Optimizer란에 adam을 적어 주는 것으로 실행할 준비가 됨(예 optimizer='adam')
  - metrics() 함수는 모델이 컴파일될 때 모델 수행의 결과를 나타내게끔 설정하는 부분
  - accuracy라고 설정한 것은 학습셋에 대한 정확도에 기반해 결과를 출력하라는 의미(예 metrics=['accuracy'])
  - accuracy 외에 학습셋에 대한 손실 값을 나타내는 loss, 테스트셋을 적용할 경우 테스트셋에 대한 정확도를 나타내는 val_acc, 테스트셋에 대한 손실 값을 나타내는 val_loss 등을 사용할 수 있음

4 모델 실행하기

4 모델 실행하기

- **모델 실행하기**
  - 모델을 정의하고 컴파일하고 나면 이제 실행시킬 차례
  - 앞서 컴파일 단계에서 정해진 환경을 주어진 데이터를 불러 실행시킬 때 사용되는 함수는 다음과 같이 model.fit() 부분

```
history = model.fit(X, y, epochs=5, batch_size=16)
```

4 모델 실행하기

- **모델 실행하기**
  - 이 부분을 설명하기에 앞서 용어를 다시 한 번 정리해 보자
  - 주어진 폐암 수술 환자의 수술 후 생존 여부 데이터는 총 470명의 환자에게서 16개의 정보를 정리한 것
  - 이때 각 정보를 '속성'이라고 함
  - 생존 여부를 클래스, 가로 한 줄에 해당하는 각 환자의 정보를 각각 '샘플'이라고 함
  - 주어진 데이터에는 총 470개의 샘플이 각각 16개씩의 속성을 가지고 있는 것이라고 앞서 설명한 바 있음

4 모델 실행하기

▼ 그림 10-2 | 폐암 환자 생존율 예측 데이터의 샘플, 속성, 클래스 구분

속성 / 클래스

	정보 1	정보 2	...	정보 16	생존 여부
1번째 환자	2	2.88	...	60	0
2번째 환자	2	3.4	...	51	0
3번째 환자	2	2.76	...	59	0
...	...	...	...	...	...
470번째 환자	2	4.72	...	51	0

샘플

4 모델 실행하기

- **모델 실행하기**
  - 이 용어는 출처마다 조금씩 다름
  - 예를 들어 샘플을 instance 또는 example이라고도 하며, 속성 대신 피처(feature) 또는 특성이라고도 함
  - 이 책에서는 '속성'과 '샘플'로 통일해서 사용

4 모델 실행하기

- **모델 실행하기**
  - 학습 프로세스가 모든 샘플에 대해 한 번 실행되는 것을 1 epoch('에포크'라고 읽음)라고 함
  - 코드에서 epochs=5로 지정한 것은 각 샘플이 처음부터 끝까지 다섯 번 재사용될 때까지 실행을 반복하라는 의미
  - batch_size는 샘플을 한 번에 몇 개씩 처리할지 정하는 부분으로 batch_size=16은 전체 470개의 샘플을 16개씩 끊어서 집어넣으라는 의미
  - batch_size가 너무 크면 학습 속도가 느려지고, 너무 작으면 각 실행 값의 편차가 생겨서 전체 결괏값이 불안정해질 수 있음
  - 자신의 컴퓨터 메모리가 감당할 만큼의 batch_size를 찾아 설정해 주는 것이 좋음