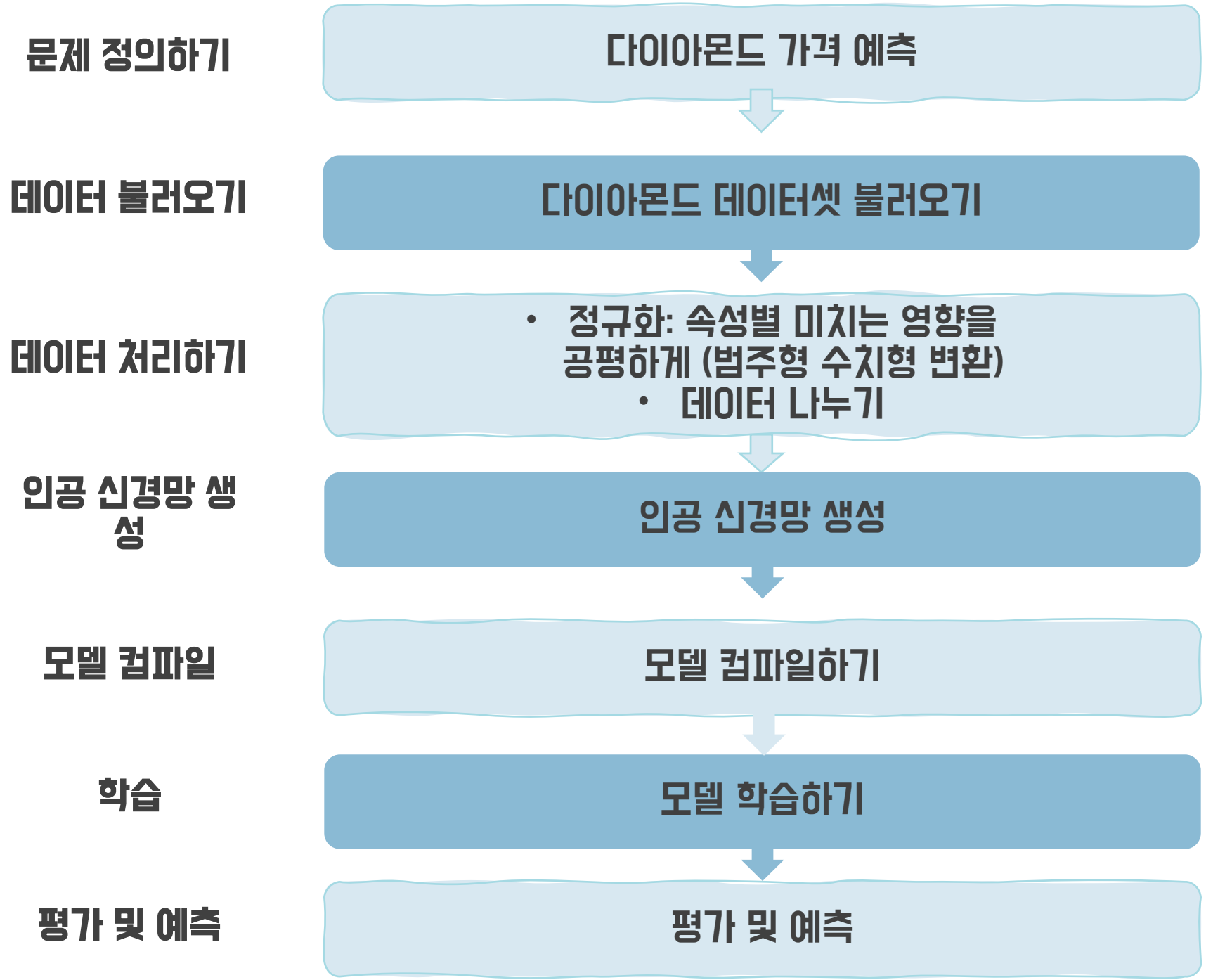
다이아몬드 가격을 예측하다

다이아몬드 가격을 예측하다

다이아몬드 가격에 영향을 미치는 다양한 요소들이 있기 때문에 전문가가 아닌 이상 일반인이 그 가격을 예측하기는 쉽지 않습니다.

그래서 간혹 너무 비싸게 주고 살 때가 있습니다.

이번 활동에서는 딥러닝으로 문제를 해결하는 핵심 단계를 토대로, **다층 퍼셉트론을 이용하여 다이아몬드 가격을 예측하는 인공지능 모델을 만들어 봅니다.**


문제 정의하기
다이아몬드 가격 예측
데이터 불러오기
다이아몬드 데이터셋 불러오기
데이터 처리하기
• 정규화: 속성별 미치는 영향을 공평하게 (범주형 수치형 변환)
• 데이터 나누기
인공 신경망 생성
인공 신경망 생성
모델 컴파일
모델 컴파일하기
학습
모델 학습하기
평가 및 예측
평가 및 예측

문제 정의하기

문제 상황 이해하기

전문가가 아닌 일반인은 다이아몬드 가격을 예측하기 어렵지만 인공지능을 이용하면 가능해집니다.
인공지능은 어떻게 다이아몬드 가격을 예측할 수 있을까요?

문제 해결에 필요한 정보

문제 해결 과정에서 필요한정보를 미리 살펴봅시다.

- 다이아몬드 데이터 셋을 사용하며, 시본(seaborn) 라이브러리를 통해 쉽게 불러올 수 있습니다.

문제 정의하기

2. 다이아몬드 데이터 셋을 모델 학습에 사용하려면 어떻게 해야 할까요?

- 모델 학습에 사용할 수 있도록 범주형 데이터는 원-핫 인코딩 처리를 합니다. 또한 다이아몬드가격 예측에 영향을 주는 독립 변수 및 독립 변수의 영향을 받는 종속 변수를 설정한 후 속성값의 범위를 일정하게 해 주는 정규화 작업이 필요합니다.

3. 모델 생성에서 필요한 활성화 함수는 무엇일까요?

- 인공 신경망의 은닉층 활성화 함수는 렐루 함수입니다.

문제 정의하기

4. 모델 컴파일에서 해야 할 작업은 무엇일까요?

- 정확도, 정밀도, 재현율, F1 Score 등을 사용합니다.

5. 회귀 모델 성능을 나타내는 평가 지표는 무엇일까요?

- 평균절대오차(MAE)와 0~1 사이의 값으로 표현하는 결정계수(R2)를 사용합니다.

데이터 불러오기

데이터 셋 소개

- 다이아몬드의 가치는 무게 (캐럿, carat)뿐 아니라 커팅 품질(cut), 색상(color), 투명도(clarity) 등의 영향을 받음.
- 다이아몬드를 커팅하는 것은 매우 어려운 일이며, 어떻게 커팅하느냐에 따라 광채가 달라지기 때문에 가격에 영향을 미침.
- 다이아몬드는 색이 없을수록 귀한 것이라 여기지만, 간혹 특별한 색은 그 가치가 높아지기도 함.

데이터 불러오기

데이터 셋 소개

다이아몬드 데이터 셋

- 다이아몬드 무게(carat), 커팅 품질(cut), 색상(color), 투명도(clarity), 가격(price), 크기(x, y, z)
- 총 53,900개의 데이터

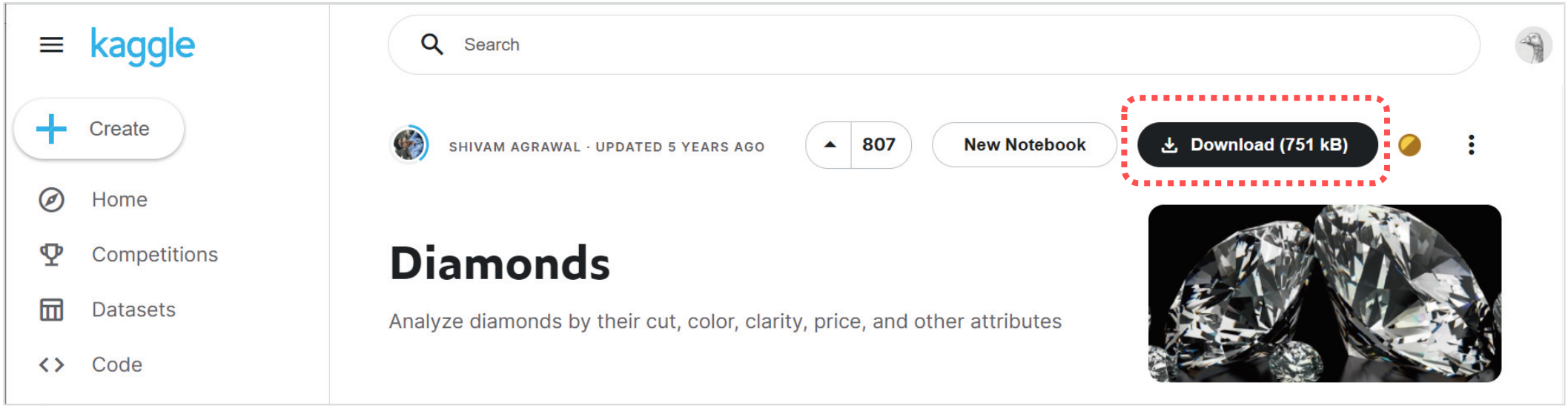
	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47

데이터 불러오기

보충

다이아몬드 데이터 셋 다운로드하기

캐글 검색창에서 'diamond'를 검색하여 내려받은 압축 파일을 풀어 'diamond.csv' 확인



데이터 불러오기

데이터 셋 불러오기

- 시본(seaborn) 라이브러리에서 불러오기

시본(seaborn) 라이브러리 불러오기

시본 라이브러리를 불러와서 sns라는 별명으로 사용할 것이라고 선언한다.

```
1 import seaborn as sns
```

데이터 불러오기

다이아몬드 데이터 셋 불러오기

데이터 셋은 시본 라이브러리에서 가져올 수 있으므로 다음과 같이 데이터 셋을 불러온다.

```
데이터 셋 객체 = 시본 라이브러리.load_dataset('데이터 셋명')
```

데이터 불러오기

다이아몬드 데이터 셋 불러오기

다이아몬드 데이터 셋의 이름은 'diamonds'이므로 sns.load_dataset(diamonds)을 통해 데이터 셋을 불러와 df라는 이름의 객체로 사용한다.
(반드시 df라는 이름을 사용하지 않아도 됨.)

```
1 import seaborn as sns
2 df = sns.load_dataset('diamonds')
```

데이터 불러오기

보충

시본(Seaborn) 라이브러리

- 시본 라이브러리는 본래 데이터를 쉽게 시각화할 수 있도록 돕는 라이브러리로, 맷플롯립(matplotlib) 라이브러리를 기반으로 만들어졌기 때문에 그 확장판임.
- 시본 라이브러리는 데이터를 이용하여 손쉽게 그래프를 만들 뿐 아니라, 멋지게도 만들 수 있어서 많은 사용자들이 활용하고 있고, 그 이유는 이름 그대로 마치 바다의 부드럽고 아름다운 장면들이 그래프로 표현되기 때문임.

데이터 불러오기

보충

시본(Seaborn) 라이브러리

- 시본 라이브러리는 애너그램, 자동차 사고, 다이아몬드, 항공편, 간헐천, 홍채, 자동차 연비, 펭귄, 행성, 택시, 타이타닉 등 다양한 데이터 셋을 포함함.
- 데이터 셋의 목록을 확인하는 방법

```
1 import seaborn as sns
2 sns.get_dataset_names( )
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'taxis',
 'tips',
 'titanic']
```

데이터 탐색 및 전처리하기

전처리 방법 알아보기

- 데이터를 불러온 후 데이터 셋에 어떤 속성과 값들이 포함되어 있는지 살펴보기
- 인공신경망으로 처리하기 위해 다음의 세 가지 전처리 수행

1) 범주형 속성값 변환
2) 정규화(normalization)
3) 훈련 데이터와 테스트 데이터 분리

데이터 탐색 및 전처리하기

범주형 속성값 변환

- 인공지능은 숫자 데이터만 처리하기 때문에 범주형 데이터들을 수치형 데이터로 변환한다.
- 범주형 데이터의 각 범주끼리 순서를 표현할 수 있다면(예: 다이아몬드 커팅/색상/투명도 등급) 숫자값 간의 위계가 있도록 1, 2, 3과 같은 순서 형태의 값으로 변환한다.
- 범주형 데이터의 각 범주끼리 순서를 표현할 수 없다면(예: 남성과 여성) 0과 1로 구성하는 원-핫 인코딩(one-hot encoding) 방법으로 변환한다.

데이터 탐색 및 전처리하기

정규화

- 범위가 다른 속성값을 같은 범위의 값으로 변환한다.
- 인공지능 모델은 데이터를 숫자로 처리하기 때문에 숫자가 클수록 더욱 중요하다고 인식한다.
- 따라서 정확한 예측을 위해 모든 속성값의 범위를 0.0~1.0 사이의 실수로 변환한다.

데이터 탐색 및 전처리하기

훈련 데이터와 테스트 데이터 분리

- 데이터 셋을 인공지능 모델을 학습하는데 사용하는 **훈련 데이터**
- 인공지능 모델이 학습을 잘했는지 테스트하는데 사용하는 **테스트 데이터**

데이터 탐색 및 전처리하기

데이터 살펴보기

다이아몬드 데이터의 개수와 속성의 개수, 어떤 속성이 있는지 살펴본다.
판다스 라이브러리의 info( ) 메소드를 사용한다.

데이터프레임 객체.info( )

info( ) 메소드를 통해 데이터 개수, 속성 개수, 속성명, 결측치,
속성의 데이터 타입 등 확인

데이터 탐색 및 전처리하기

데이터 정보 확인하기

```
1 df.info( )
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  category
 2   color    53940 non-null  category
 3   clarity  53940 non-null  category
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: category(3), float64(6), int64(1)
memory usage: 3.0 MB
```

속성명	설명
carat	다이아몬드 무게(1캐럿: 0.2g)
cut	커팅 품질(Fair<Good<Very Good<Premium<Ideal)
color	색상(J<I<H<G<F<E<D)
clarity	투명도(I1<SI2<SI1<VS2<VS1<VVS2<VVS1<IF)
depth	z / mean(x, y)
table	상단의 가장 넓은 지점의 너비
price	가격($)
x	길이(mm)
y	너비(mm)
z	깊이(mm)

데이터 탐색 및 전처리하기

데이터 정보 확인하기

- 총 53,940개의 데이터가 있고 속성은 10개
- 각 속성명은 Column열에 carat, cut, color, …, z로 표현되어 있음.
- 각 속성별로 53,940개의 값을 가진 것으로 보아 결측치가 없다는 것을 알 수 있음.

결측치가 있다면 해당 행을 삭제할 것인지, 결측값을 다른 값으로 대체할 것인지를 고민해야 하지만 여기서는 그럴 필요가 없다.

데이터 탐색 및 전처리하기

속성별로 데이터의
유형을 말해 보거라.

무게(carat),
깊이(depth), 테이블(table),
길이(x), 너비(y), 깊이(z)의 속성은
실수형(float64)이고, 가격(price)은
정수형(int64)이며, 커팅 품질(cut),
색상(color), 투명도(clarity)는
범주형(category)인 것을
알 수 있어요.

데이터 탐색 및 전처리하기

데이터 일부 살펴보기

- 실제로 데이터가 어떻게 생겼는지 확인해 보기 위해 전체 데이터를 불러오지 않고 head( ) 메소드를 사용하여 일부 데이터만 살펴본다.

```
1  df.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

데이터 탐색 및 전처리하기

데이터 일부 살펴보기

- 다이아몬드 데이터 셋의 0~4행까지 데이터를 데이터프레임 형태로 살펴볼 수 있다.

데이터 탐색 및 전처리하기

데이터 일부 살펴보기

- describe( ) 메소드를 사용하여 데이터의 통계량(개수, 평균, 중앙값, 표준편차, 최솟값, 최댓값, 4분위수 등)을 살펴본다.
- 수칫값을 갖는 속성에 대해 통계량을 출력하며 결측치는 제외된다.
- 따라서 커팅 품질(cut), 색상(color), 투명도(clarity)는 문자열이므로 출력되지 않는다.

데이터 탐색 및 전처리하기

데이터 일부 살펴보기

```
1  df.describe()
```

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

데이터 탐색 및 전처리하기

데이터 일부 살펴보기

- 다이아몬드는 0.2캐럿에서 5.01캐럿까지 분포
- 다이아몬드의 절반이 0.2~0.7캐럿 사이에 분포
- 75%가 1.04캐럿 이하
- 가격은 최저 326$에서 최고 18,823$ 사이
- 길이, 너비, 깊이의 최솟값은 0. 다이아몬드가 존재하는데도 불구하고 0이라는 값은 잘못된 값 즉, 이상치가 있음을 의미함.

데이터 탐색 및 전처리하기

수치형 데이터 변환

원-핫 인코딩 방법

- 원-핫 인코딩은 범주의 개수만큼 속성을 만들고 범주마다 0이나 1을 입력하는 방법
- 1개의 속성만 1로 표기하고 나머지 속성에는 0을 표기한다고 해서 '원-핫(one-hot)'이라는 이름이 붙임.

원-핫 인코딩

범주의 개수만큼 속성을 만들고 범주마다 0이나 1을 입력하는 방법

데이터 탐색 및 전처리하기

원-핫 인코딩 방법

해당하는 범주에는 1, 아닌 범주에는 0을 표기

변환 예시 dog 데이터 값은 PET_dog에 해당하므로 이 속성에만 1을 표기하고

나머지 PET_cat, PET_rabbit 속성에는 0을 표기(나머지도 동일하게 적용)

PET
dog
cat
rabbit
dog

➡

PET_dog	PET_cat	PET_rabbit
1	0	0
0	1	0
0	0	1
1	0	0

데이터 탐색 및 전처리하기

원-핫 인코딩 적용

- 커팅 품질(cut), 색상(color), 투명도(clarity) 속성은 순서가 있는 범주형 데이터이므로 원-핫 인코딩 수행
- 원-핫 인코딩을 수행하는 여러 방법 중에서 판다스 라이브러리의 get_dummies( ) 사용

판다스 객체.get_dummies(데이터프레임 객체, columns = [인코딩할 속성명])

특정 속성만 원-핫 인코딩을 할 때는 columns에 속성명 제시

데이터 탐색 및 전처리하기

원-핫 인코딩 적용

- 판다스 라이브러리를 불러와 다이아몬드 데이터 셋 원-핫 인코딩 수행하기

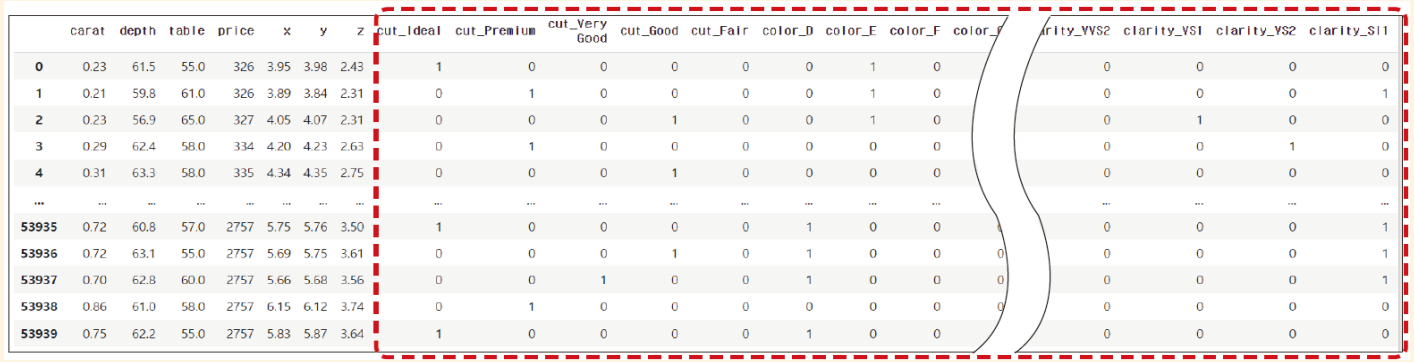
```
1 import pandas as pd
2 df = pd.get_dummies(df)
3 df
```

	carat	depth	table	price	x	y	z	cut_Ideal	cut_Premium	cut_Very Good	cut_Good	cut_Fair	color_D	color_E	color_F	color_G	clarity_VVS2	clarity_VS1	clarity_VS2	clarity_SI1
0	0.23	61.5	55.0	326	3.95	3.98	2.43	1	0	0	0	0	0	1	0		0	0	0	0
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0	1	0	0	0	0	1	0		0	0	0	1
2	0.23	56.9	65.0	327	4.05	4.07	2.31	0	0	0	1	0	0	1	0		0	1	0	0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0	1	0	0	0	0	0	0		0	0	1	0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	0	0	0	1	0	0	0	0		0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...		...	...	...	...
53935	0.72	60.8	57.0	2757	5.75	5.76	3.50	1	0	0	0	0	1	0	0	0	0	0	0	1
53936	0.72	63.1	55.0	2757	5.69	5.75	3.61	0	0	0	1	0	1	0	0	0	0	0	0	1
53937	0.70	62.8	60.0	2757	5.66	5.68	3.56	0	0	1	0	0	1	0	0	0	0	0	0	1
53938	0.86	61.0	58.0	2757	6.15	6.12	3.74	0	1	0	0	0	0	0	0	0	0	0	0	0
53939	0.75	62.2	55.0	2757	5.83	5.87	3.64	1	0	0	0	0	1	0	0	0	0	0	0	0

원-핫 인코딩 부분

데이터 탐색 및 전처리하기

원-핫 인코딩 적용



	carat	depth	table	price	x	y	z	cut_Ideal	cut_Premium	cut_Very Good	cut_Good	cut_Fair	color_D	color_E	color_F	color_G	clarity_VVS2	clarity_VS1	clarity_VS2	clarity_SI1
0	0.23	61.5	55.0	326	3.95	3.98	2.43	1	0	0	0	0	0	1	0		0	0	0	0
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0	1	0	0	0	0	1	0		0	0	0	1
2	0.23	56.9	65.0	327	4.05	4.07	2.31	0	0	0	1	0	0	1	0		0	1	0	0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0	1	0	0	0	0	0	0		0	0	1	0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	0	0	0	1	0	0	0	0		0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...		...	...	...	...
53935	0.72	60.8	57.0	2757	5.75	5.76	3.50	1	0	0	0	0	1	0	0		0	0	0	1
53936	0.72	63.1	55.0	2757	5.69	5.75	3.61	0	0	0	1	0	1	0	0	0	0	0	0	1
53937	0.70	62.8	60.0	2757	5.66	5.68	3.56	0	0	1	0	0	1	0	0	0	0	0	0	1
53938	0.86	61.0	58.0	2757	6.15	6.12	3.74	0	1	0	0	0	0	0	0	0	0	0	0	0
53939	0.75	62.2	55.0	2757	5.83	5.87	3.64	1	0	0	0	0	1	0	0	0	0	0	0	0

- 모든 범주형 데이터가 범주의 개수만큼 새로운 속성들이 생성되었고, 해당하는 범주에만 1이 체크된 것 확인
- get_dummies( )를 사용하여 원-핫 인코딩된 후에는 기존의 범주형 데이터 속성(cut, color, clarity)은 삭제

데이터 탐색 및 전처리하기

독립 변수와 종속 변수

- **독립 변수**: 영향을 미치는 속성
- **종속 변수**: 독립 변수의 영향을 받는 속성

다이아몬드 데이터 셋 속성 구분	
독립 변수	무게(carat), 커팅 품질(cut), 색상(color), 투명도(clarity), 테이블(table) 등의 속성
종속 변수	가격 (price) 속성

데이터 탐색 및 전처리하기

독립 변수와 종속 변수

- 원-핫 인코딩 결과에서 가격(price) 속성은 종속 변수로, 나머지 속성은 독립 변수로 설정
- 인덱스로 속성을 추출하는 iloc 메소드와 슬라이싱([:]) 사용

- 독립 변수는 가격(price) 속성을 제외한 나머지 변수이므로, 0~2번째까지 속성과 4번째 이후 속성 전체가 해당함.
- 0~2번째 속성을 추출하여 X1으로 설정하고 4번째 이후 속성을 추출하여 X2로 설정함.
- 0~2번째와 4번째 이후 속성을 한 번에 추출할 수 없기 때문에 각각 추출한 이후 합쳐야 함.

데이터 탐색 및 전처리하기

독립 변수와 종속 변수

- 데이터프레임 X1과 X2를 합치기 위해서는 판다스 라이브러리의 concat( )을 사용
- X1과 X2는 열 방향으로 합쳐야 하므로 axis = 1로 설정

데이터프레임 개체.iloc[범위]

판다스 객체.concat([데이터프레임명, 데이터프레임명], axis = 0 / 1)

axis = 0이면 행 방향(아래)으로 합침.(기본값), axis = 1이면 열 방향(오른쪽)으로 합침.

데이터 탐색 및 전처리하기

독립 변수와 종속 변수

- 독립 변수만 추출하기

```
1 X1 = df.iloc[:, 0:3] # 모든 행의 0, 1, 2번째 속성 선택
2 X2 = df.iloc[:, 4:] # 모든 행의 4번째 이후 속성 선택
3 X = pd.concat([X1, X2], axis = 1) # X1과 X2를 열 방향으로 합침.
4 X
```

carat	depth	table	x	y	z	cut_Ideal	cut_Premium	cut_Very Good	cut_Good	...	color_I	color_J	clarity_IF	clarity_VVS1
0.23	61.5	55.0	3.95	3.98	2.43	1	0	0	0	...	0	0	0	0
0.21	59.8	61.0	3.89	3.84	2.31	0	1	0	0	...	0	0	0	0
0.23	56.9	65.0	4.05	4.07	2.31	0	0	0	1	...	0	0	0	0
0.29	62.4	58.0	4.20	4.23	2.63	0	1	0	0	...	1	0	0	0
0.31	63.3	58.0	4.34	4.35	2.75	0	0	0	1	...	0	1	0	0

데이터 탐색 및 전처리하기

독립 변수와 종속 변수

- 종속 변수인 가격(price) 속성만 추출하기

```
1 Y = df.iloc[:, 3] # 모든 행의 3번째 속성 선택
```

```
0          326
1          326
2          327
3          334
4          335
          ...
53935     2757
53936     2757
53937     2757
53938     2757
53939     2757
Name: price, Length: 53940, dtype: int64
```

데이터 탐색 및 전처리하기

정규화

인공지능 모델이 결과를 보다 잘 예측하려면?

- 너무 크거나 작은 범위의 속성값들이 널뛰고 있지 않아야 함.
- 일정 범위 안에 표현되어야 함.
- 같은 의미를 가진 값들을 표현한 범위가 다르다면 통일시킴.

데이터 탐색 및 전처리하기

정규화

예시

한 학생의 국어 성적은 100점 만점 중 80점이고, 수학 성적은 10점 만점에 점일 때, 80이 9보다 큰 수라고 해서 수학보다 국어를 잘했다고 볼 수 없다.

수학 성적의 총점을 100점 만점으로 변환하거나 국어 성적의 총점을 10점으로 변환해야 한다.

이때 여러 속성값의 범위를 일정하게 맞춰 주는 방법이 **정규화**와 **표준화**이며, 이번활동에서는 **정규화** 방법을 이용한다.

데이터 탐색 및 전처리하기

정규화

구분	변환값		장단점	
정규화	특징	0.0~1.0 사이의 실숫값으로 변환	장점	모든 속성의 범위를 0.0 ~ 1.0으로 통일
	공식	• $x = \frac{x - \text{최솟값}}{\text{최댓값} - \text{최솟값}}$	단점	이상치를 처리하기 어려움.
표준화	특징	평균이 0, 분산이 1인 정규분포가 되도록 변환	장점	이상치를 잘 처리함.
	공식	• $Z = \frac{X(\text{속성값}) - \mu(\text{속성값 평균})}{\sigma(\text{표준편차})}$	단점	동일한 범위로 통일할 수 없음.

데이터 탐색 및 전처리하기

사이킷런 라이브러리 불러오기

- 정규화를 위한 함수는 **사이킷런(sklearn) 라이브러리**에서 지원한다.
- 사이킷런에서 필요한 모듈을 불러온다.
- 우리가 사용할 함수는 **최솟값과 최댓값을 이용하는 정규화 함수**이다.

```
1 from sklearn.preprocessing import MinMaxScaler
```

사이킷런에서 전처리를 지원하는 preprocessing의 MinMaxScaler를 불러왔다.
참고로 정규화를 **스케일링**이라고도 하는데, 사이킷런에서는 스케일러로 사용한다.

데이터 탐색 및 전처리하기

보충

사이킷런의 전처리 모듈

- 사이킷런의 preprocessing 모듈에서 제공하는 데이터 전처리에 필요한 기능(인코딩, 정규화, 스케일링 등)은 다음과 같이 python의 dir( ) 함수를 사용하여 조회할 수 있다.

```
1 import sklearn.preprocessing
2 dir(sklearn.preprocessing)
```

```
['Binarizer',
 'FunctionTransformer',
        ⋮
 'quantile_transform',
 'robust_scale',
 'scale']
```

데이터 탐색 및 전처리하기

독립 변수 정규화

- 정규화를 위해서는 MinMaxScaler( )로 정규화 객체를 만든 후에 속성값의 범위를 실숫값인 0.0~1.0 사이로 변환
- fit_transform( ) 메소드 사용

정규화 객체 = MinMaxScaler( )

정규화 객체.fit_transform(정규화할 데이터프레임명)

데이터 탐색 및 전처리하기

독립 변수 정규화

- 정규화 객체에 scaler라는 이름을 붙이기
- 정규화해야 하는 데이터프레임은 다이아몬드 가격에 영향을 미치게 되는 독립 변수가 X이므로 '정규화할 데이터프레임명' 위치에 X를 입력하기
- 정규화한 결과는 X_scaled로 사용하여 출력하기

데이터 탐색 및 전처리하기

독립 변수
정규화

- 정규화 출력하기

```
1 from sklearn.preprocessing import MinMaxScaler
2 scaler = MinMaxScaler()
3 X_scaled = scaler.fit_transform(X)
4 X_scaled
```

```
array([[0.00623701, 0.51388889, 0.23076923, ..., 0.        , 1.        ,
        0.        ],
       [0.002079  , 0.46666667, 0.34615385, ..., 1.        , 0.        ,
        0.        ],
       [0.00623701, 0.38611111, 0.42307692, ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.1039501 , 0.55      , 0.32692308, ..., 1.        , 0.        ,
        0.        ],
       [0.13721414, 0.5       , 0.28846154, ..., 0.        , 1.        ,
        0.        ],
       [0.11434511, 0.53333333, 0.23076923, ..., 0.        , 1.        ,
        0.        ]])
```

데이터 탐색 및 전처리하기

독립 변수 정규화

- 정규화한 결과를 살펴보면 모든 속성값이 0.0~1.0 사이의 실숫값
- 데이터프레임이 아니라 'array'라고 출력된 것을 확인

정규화를 하면 배열(array)로 바뀌게 된다. 따라서 지금부터는 X_scaled를 데이터프레임이라 부르지 않고 배열이라고 부름.

데이터 탐색 및 전처리하기

훈련 데이터 테스트 데이터 나누기

- 인공지능 모델을 학습시키기 전에 다이아몬드 데이터 셋을 훈련 데이터와 테스트 데이터로 나누기
- 일반적으로 훈련 데이터와 테스트 데이터는 7:3의 비율

사이킷런 라이브러리 불러오기

사이킷런 라이브러리의 모듈 중 model_selection에서 train_test_split을 불러오면 데이터를 쉽게 나눌 수 있다.

```
1 from sklearn.model_selection import train_test_split
```

데이터 탐색 및 전처리하기

훈련 데이터와 테스트 데이터 나누기

- 훈련 데이터와 테스트 데이터로 나누기 위해서는 train_test_split( ) 사용
- train_test_split( )을 실행시킬 때마다 전체 데이터 중에서 임의로 데이터를 추출하여 훈련 데이터와 테스트 데이터 만들기

학습에 사용할 독립 변수 객체, 테스트에 사용할 독립 변수 객체, 학습에 사용할 종속 변수 객체,

테스트에 사용할 종속 변수 객체 = train_test_split(독립 변수, 종속 변수, test_size =

테스트 데이터 비율)

테스트 데이터 비율을 0.3으로 입력하면 테스트 데이터는 전체 데이터의 30%로 설정

데이터 탐색 및 전처리하기

훈련 데이터와 테스트 데이터 나누기

- 전체 데이터 중에서 테스트 데이터의 크기를 0.3(30%)으로 설정하면 훈련 데이터는 70%
- 학습에 사용하는 독립 변수들은 train_input으로 설정
- 테스트에 사용하는 독립 변수들은 test_input으로 설정
- 학습에 사용하는 종속 변수는 train_target으로 설정
- 테스트에 사용하는 종속 변수는 test_target으로 설정

데이터 탐색 및 전처리하기

훈련 데이터와 테스트 데이터 나누기

훈련 데이터(70%)		테스트 데이터(30%)	
독립 변수 (train_input)	종속 변수 (train_target)	독립 변수 (test_input)	종속 변수 (test_target)

```
1 from sklearn.model_selection import train_test_split
2 train_input, test_input, train_target, test_target = train_test_split(
3                                                 X_scaled, Y, test_size = 0.3)
```

데이터 탐색 및 전처리하기

훈련 데이터와 테스트 데이터 나누기

- shape를 이용하여 훈련 데이터의 독립 변수와 종속 변수, 테스트 데이터의 독립 변수와 종속 변수의 개수 확인하기

```
1 train_input.shape, train_target.shape, test_input.shape, test_target.shape
```

```
((37758, 26), (37758,), (16182, 26), (16182,))
```

shape로 배열의 차원(형태)을 확인하세요.

데이터 탐색 및 전처리하기

훈련 데이터와 테스트 데이터 나누기

- 훈련 데이터는 전체 53,940개 데이터의 70%에 해당하는 37,758개 할당
- 원-핫 인코딩으로 인해 학습에 사용하는 독립 변수의 개수는 총 26개가 되어 (37758, 26) 형태
- 학습에 사용하는 데이터 중 종속 변수에 해당하는 가격 속성(price)은 1개 뿐이기 때문에 (37758,)의 형태
- 테스트 데이터는 전체 53,940개 데이터의 30%에 해당하는 16,182개가 할당
- 훈련 데이터와 마찬가지로 원-핫 인코딩으로 독립 변수의 개수는 총 26개가 되어 (16182,26) 형태
- 테스트에 사용하는 종속 변수도 같은 맥락에서 (16182,) 형태

모델 생성하기

모델 구성

- 데이터프레임의 한행 1개의 다이아몬드에 대한 정보
- 입력층으로 생성 전처리 과정이지만 모델 생성 단계에서 할 수 있음.

모델 생성 단계

데이터프레임의 한 행을 입력층으로 생성하는 작업

+

은닉층의 개수를 정하고 생성하는 작업

+

각 은닉층별 노드의 개수를 정하는 작업

+

활성화 함수를 설정하는 작업

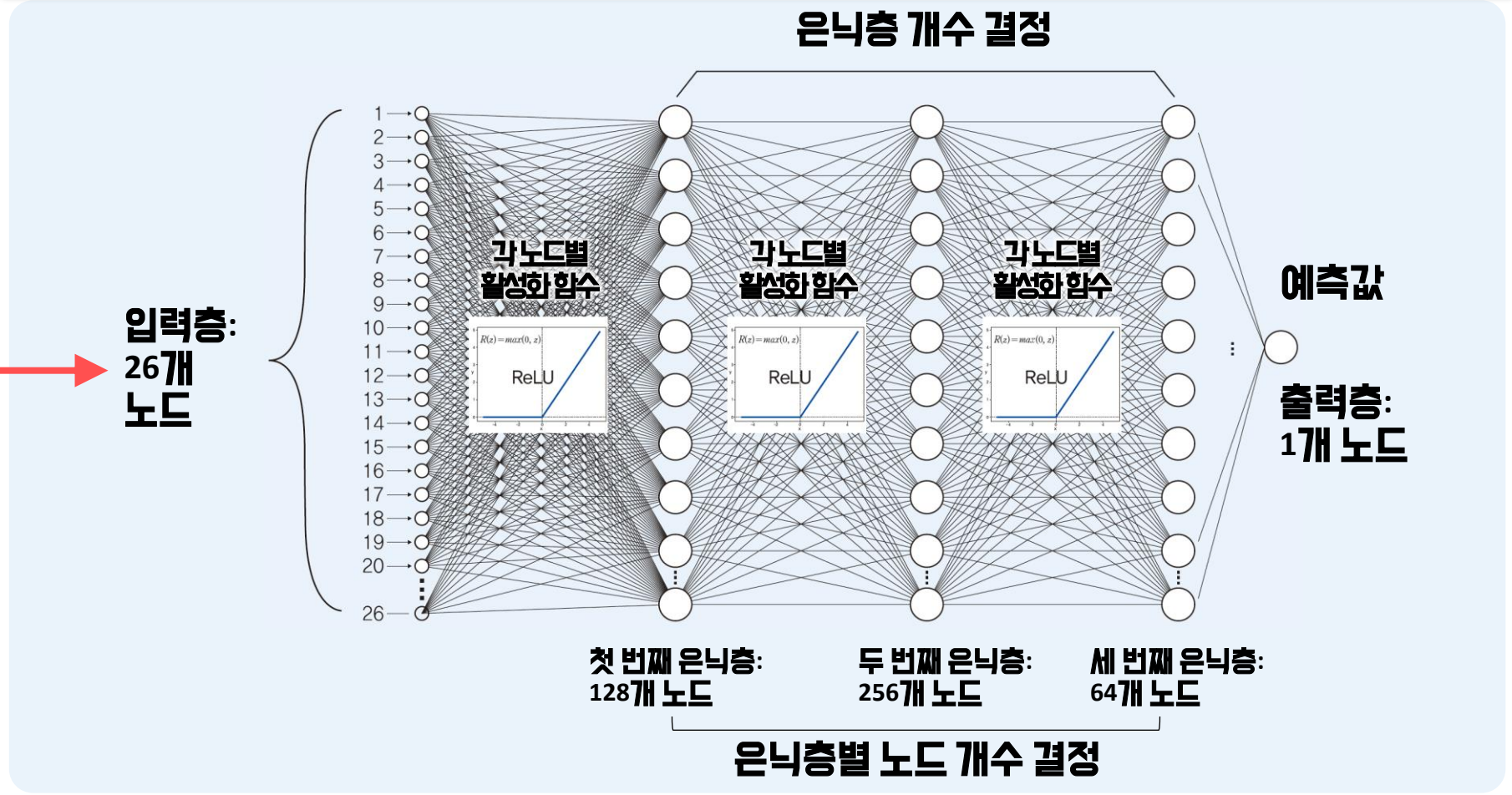
모델 생성하기

모델 구성

- 첫 번째 은닉층의 각 노드(원 모양)는 '26개(정규화된 배열 X_scaled의 한 행에 포함된 속성의 개수)의 입력층 노드 값×가중치+바이어스'의 계산 결과를 가짐.
- 각 노드는 활성화 함수를 통해 활성 여부(현재 노드가 다이아몬드 가격을 결정하는 데 중요한 영향을 미친다면 다음 노드로 값을 전달할 수 있도록 활성화시킴.)를 결정한 후 다음 은닉층의 노드(또는 출력층)로 전달됨.
- 딥러닝 프로그래밍 실행 초기에 가중치와 바이어스 값은 각각 임의의 값과 0으로 설정되었다가 학습하면서 조정하게 되며, 딥러닝을 통해 자동으로 결정됨.

모델 생성하기

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	price
0	0.006237	0.513889	0.230769	0.367784	0.067572	0.076415	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	326
1	0.002079	0.466667	0.346154	0.362197	0.065195	0.072642	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	326
2	0.006237	0.386111	0.423077	0.377095	0.069100	0.072642	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	327



모델 생성하기

입력층, 은닉층, 출력층 생성

- 일반적으로 인공 신경망의 층을 생성할 때는 입력층과 은닉층을 구분하여 만들거나 은닉층에 입력층을 포함하여 하나로 묶어서 만든다.
- 여기서는 후자를 이용한다.

모델 생성하기

입력층, 은닉층, 출력층 생성

- 323쪽 그림처럼 입력층은 데이터프레임의 한 행의 속성 개수만큼 입력받아야 하므로 26개의 노드가 필요하다.
- 1~3번째 은닉층은 사용자가 임의로 노드의 개수를 설정한다.
- 은닉층의 노드들 중 결과를 예측하는 데 중요한 역할을 하는 노드만 활성화시켜 다음 노드로 값을 전달할 수 있도록 활성화 함수를 'relu' 함수로 설정한다.
- 예측한 결괏값은 다이아몬드 가격 1개이므로 출력층 노드의 개수는 1개로 설정한다.

	입력층	1번째 은닉층	2번째 은닉층	3번째 은닉층	출력층
노드 개수	26	128	256	64	1
활성화 함수	-	relu	relu	relu	-

모델 생성하기

케라스 라이브러리 불러오기

- 입력층, 은닉층, 출력층을 만들기 위하여 케라스 라이브러리를 다음과 같이 불러온다.

```
1 from tensorflow import keras
```

모델 생성하기

케라스 라이브러리 불러오기

- 케라스 라이브러리는 다음과 같이 인공 신경망 모델을 생성하는 간단한 방법을 지원한다.

방법 1

```
모델 객체 = keras.models.Sequential([
keras.layers.Flatten(input_shape = (행의 개수, 열의 개수)), # 입력층 생성
keras.layers.Dense(은닉층 노드의 개수, activation = '활성화 함수명'), # 1번째 은닉층 생성
keras.layers.Dense(은닉층 노드의 개수, activation = '활성화 함수명'), # 2번째 은닉층 생성
                                                    # n번째 은닉층 생성
keras.layers.Dense(출력층 노드의 개수, activation = '활성화 함수명')]) # 출력층 생성
```

모델 생성하기

케라스 라이브러리 불러오기

- 이번 활동에서는 방법2의 keras.layers.Dense( )를 사용한다.

방법 2

```
모델 객체 = keras.Sequential( )
# keras.layers.Dense( )는 층을 만드는 명령, add는 모델에 층을 추가하는 명령
모델 객체.add(keras.layers.Dense(은닉층 노드의 개수, activation = '활성화 함수명',
            input_shape = (입력층 행의 개수, )))
모델 객체.add(keras.layers.Dense(은닉층 노드의 개수, activation = '활성화 함수명')
                                    ⋮
모델 객체.add(keras.layers.Dense(은닉층 노드의 개수, activation = '활성화 함수명')
모델 객체.add(keras.layers.Dense(출력층 노드의 개수, activation = '활성화 함수명')
```

모델 생성하기

각 층의 생성 과정

- 케라스 Sequential( ) 함수를 사용하여 순차적으로 노드값을 전달하는 모델의 객체model을 생성한다.
- model에 은닉층(첫 번째 은닉층에 입력층 포함)과 출력층을 추가한다.
- Sequential은 순차적, 연속적이라는 의미로 첫 번째 은닉층의 값이 두 번째 은닉층을 거쳐 세 번째 은닉층으로만 흘러가도록 할 때 적합하다.

모델 생성하기

각 층의 생성과정

model의 첫 번째 은닉층은 input_shape = (26,)을 통해 (26,) 형태의 데이터를 입력받기

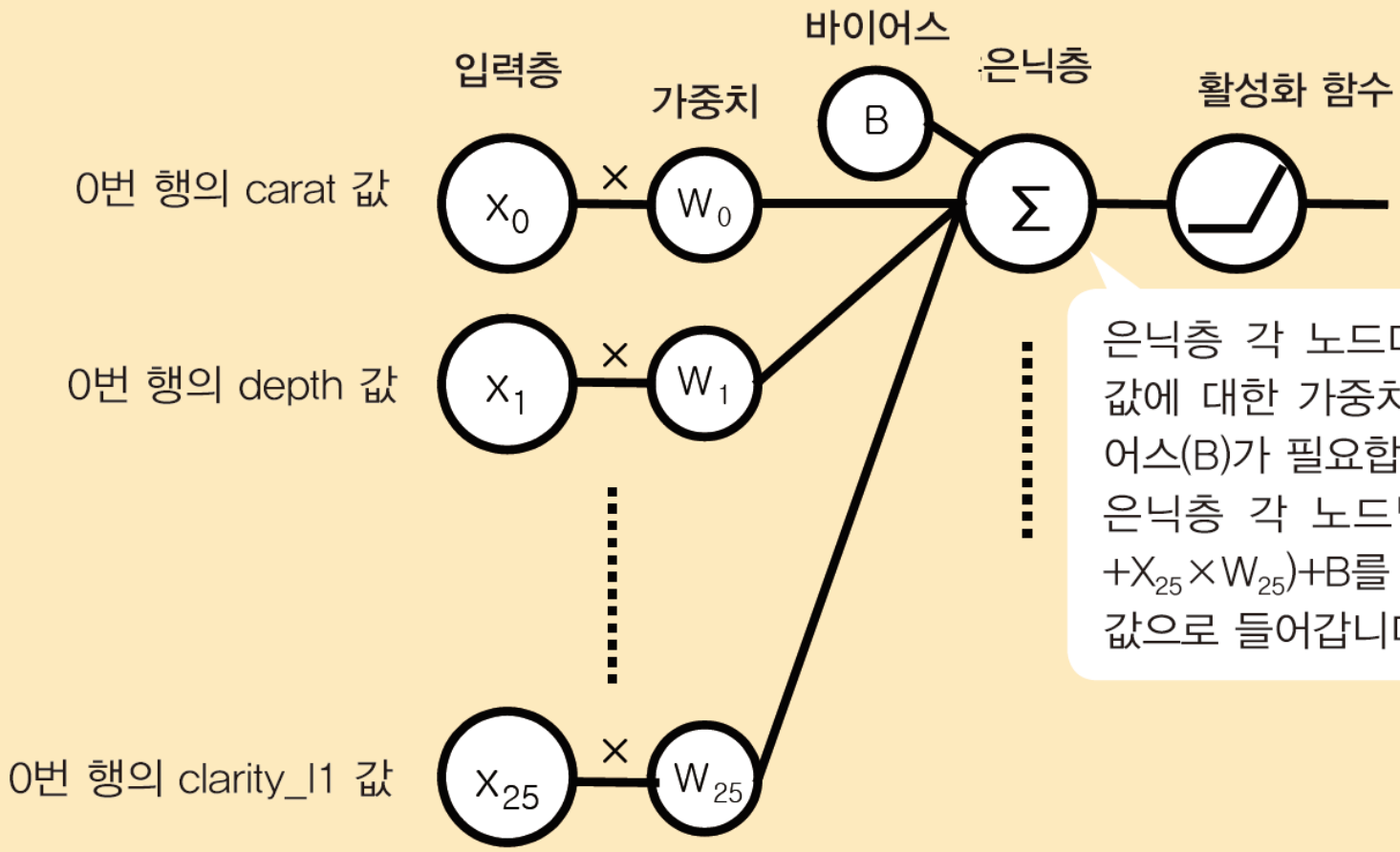
26개의 값을 첫 번째 은닉층의 128개 노드에 각각 전달하기

- 다이아몬드의 가격에 영향을 미치게 되는 노드들은 'relu' 활성화 함수를 통해 다음 은닉층의 노드들에 전달된다.

마지막 출력층은 '다이아몬드 가격'이라는 한 개의 값만 확인하면 되므로 노드의 개수가 1개이다.

모델 생성하기

각 층의 생성 과정



모델 생성하기

각 층의 생성 과정

- 위 내용을 토대로 모델을 생성하기

```
1 from tensorflow import keras
2 model = keras.Sequential( )
3 model.add(keras.layers.Dense(128, activation = 'relu', input_shape = (26, )))
4 model.add(keras.layers.Dense(256, activation = 'relu'))
5 model.add(keras.layers.Dense(64, activation = 'relu'))
6 model.add(keras.layers.Dense(1))
7 model.summary( ) # 모델을 간단하게 살펴보기
```

모델 생성하기

각 층의 생성 과정

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               3456

 dense_1 (Dense)             (None, 256)               33024

 dense_2 (Dense)             (None, 64)                16448

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 52,993
Trainable params: 52,993
Non-trainable params: 0
```

모델 생성하기

각 층의 생성 과정

첫 번째 은닉층

- 첫 번째 은닉층(dense)의 출력 형태(Output Shape)는 (None, 128)
- 1번째 은닉층 노드의 개수가 128개이므로 각 노드에 대한 결괏값도 128개
- 파라미터의 개수(Param #) 3,456은 다음 계산식에서 나온 값
  - 은닉층 각 노드에 다이아몬드 가격에 영향을 미치는 26개 입력값에 대한 가중치(W_0~W_{25}) 개수 26개.
  - 바이어스 값(B) 1개가 '(X_0×W_0+…+X_{25}×W_{25})+b'의 공식으로 계산되어 입력값
  - 은닉층의 노드는 총 128개이므로 '26개 가중치 파라미터×128개 노드 + 노드별 바이어스 파라미터×128개 노드'를 계산하여 3,456개가 나온 것

모델 생성하기

각 층의 생성 과정

두 번째 은닉층

- 두 번째 은닉층(dense_1)의 출력 형태(Output Shape)는 (None, 256). 두 번째 은닉층 노드의 개수가 256개이기 때문
- 파라미터 개수(Param #)는 33,024개
  - 첫 번째 은닉층 128개 노드에 대한 가중치 128개와 1개의 바이어스가 2번째 은닉층 각 노드(256개)의 입력값 계산에 필요
  - 따라서 '128개 가중치 파라미터×256개 노드 + 노드별 바이어스 파라미터 × 256개 노드'를 계산하여 33,024가 나온 것

모델 생성하기

각 층의 생성 과정

출력층 및 종합

- 출력층에는 활성화 함수가 없다.
- 이번 활동은 범주를 예측하는 분류 모델이 아니라 값을 예측하는 회귀 모델이므로 출력층에 값이 그대로 출력되기 때문이다.
- 그리고 모델을 생성했다고 해서 학습이 완료된 것이 아니다.
- 모델을 생성한 것은 신경망의 각 층을 생성한 것으로 사람의 뇌 구조를 구성했다고 생각하면 된다.

모델 생성하기

각 층의 생성 과정

- 은닉층의 개수와 은닉층의 노드 개수는 정해지지 않으면 데이터의 양, 처리할 수 있는 하드웨어 성능 등을 고려하여 조절한다.

- 데이터의 양이 적은데 은닉층이 많은 것은 좋지 않으며, 노드의 개수가 너무 많으면 훈련 데이터에 과도하게 잘 맞는 모델을 생성하게 되어 과대적합(훈련 데이터에만 과도하게 잘 맞고 테스트 데이터로는 잘 예측하지 못하는 현상)이 발생할 가능성이 높아진다.

모델 컴파일하기

모델 컴파일 단계

학습하기 전에 손실 함수와 최적화 함수, 평가 지표 설정

+

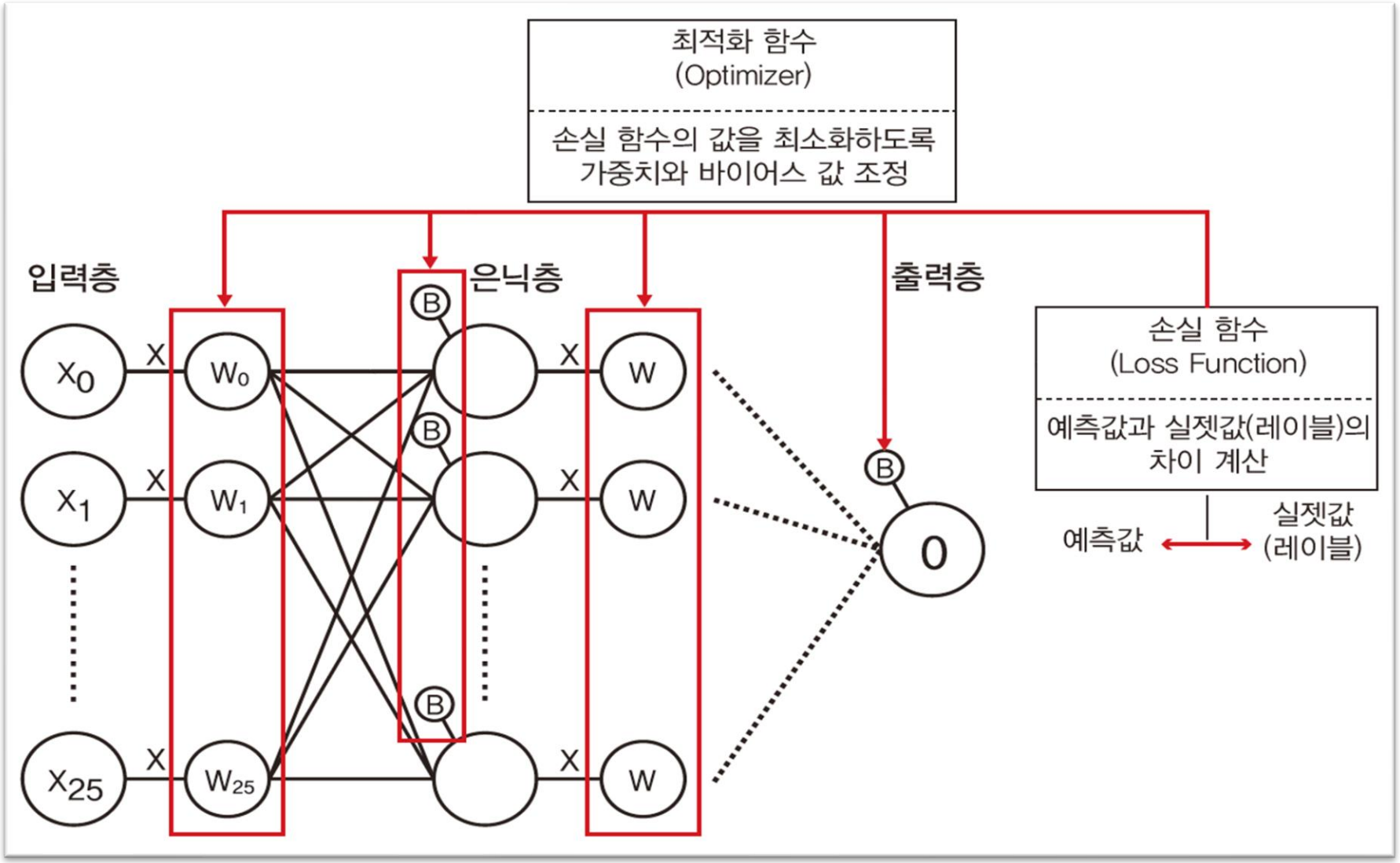
인공 신경망으로 학습한 후 예측한 값과 실제 정답 간의
차이가 얼마나 되는지 손실 함수를 통해 계산

+

이 차이를 줄이도록 최적화 함수가 가중치와 바이어스 값 수정

▶ 학습이 얼마나 잘되었는지에 대한 평가는 평가 지표에서 설정

모델 컴파일하기


최적화 함수
(Optimizer)
손실 함수의 값을 최소화하도록
가중치와 바이어스 값 조정
입력층
은닉층
출력층
손실 함수
(Loss Function)
예측값과 실젯값(레이블)의
차이 계산
예측값
실젯값
(레이블)
B
X
W
O

모델 컴파일하기

손실 함수 설정

손실 함수는 딥러닝의 예측값과 실젯값 사이의 차이(오차)를 수치로 나타내는 함수이다.

- 오차가 크면 손실 함수의 값이 크고, 오차가 작으면 손실 함수의 값이 작기 때문에 **손실 함수의 값이 작을수록 훈련 데이터를 잘 학습한 것**이라 파악 가능
- 회귀 모델에 사용하는 손실 함수는 **평균제곱오차**(MSE: Mean Squared Error)를 주로 사용

모델 컴파일하기

손실 함수 설정

- **MSE : 예측값과 실젯값의 차이를 제곱한 후 전부 더하여 평균을 낸 것**
- 328쪽 그림 : 예측값과 실젯값의 차이를 면적으로 만들어 평균을 낸 것을 그림으로 표현 것
- 예측값과 실젯값의 차이는 양수뿐만 아니라 음수가 나올 수 있으므로 합이 0이 되어버리는 오류를 막기 위해서 **제곱 연산 수행**

$$\frac{1}{n}\sum_{i=1}^{n}(\text{예측값}-\text{실젯값})^2 \qquad (n\text{: 데이터 개수})$$

모델 컴파일하기

최적화 함수 설정

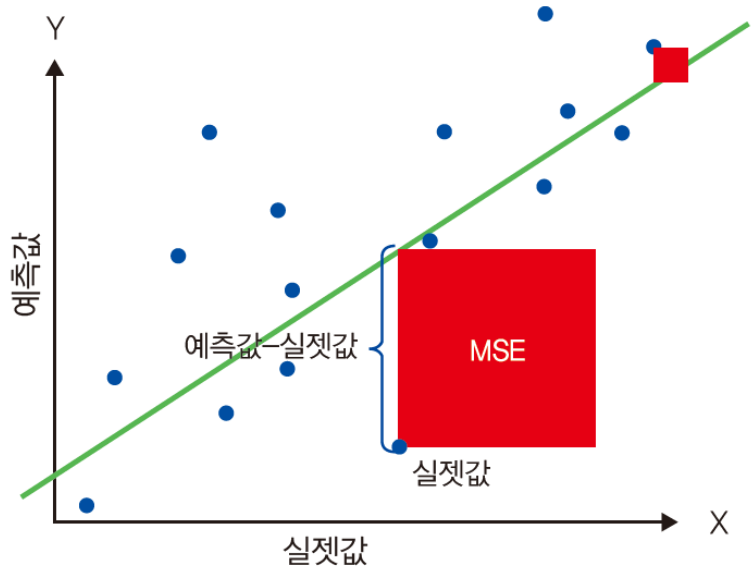
손실 함수의 값을 최소화하도록 가중치와 바이어스를 수정해 나가는 함수

- 초기에 설정한 임의의 가중치 값을 수정하면서 손실 함수의 값이 최소화되는 가중치를 찾는 역할
- 한 번에 가중치 값을 얼마나 변경시킬지, 어느 방향으로 이동할지 등이 손실 함수의 값을 찾는 데 영향을 미침.
- 이때 한 번에 가중치 값을 얼마나 변경시킬지 결정하는 것을 **학습률**이라고 함.

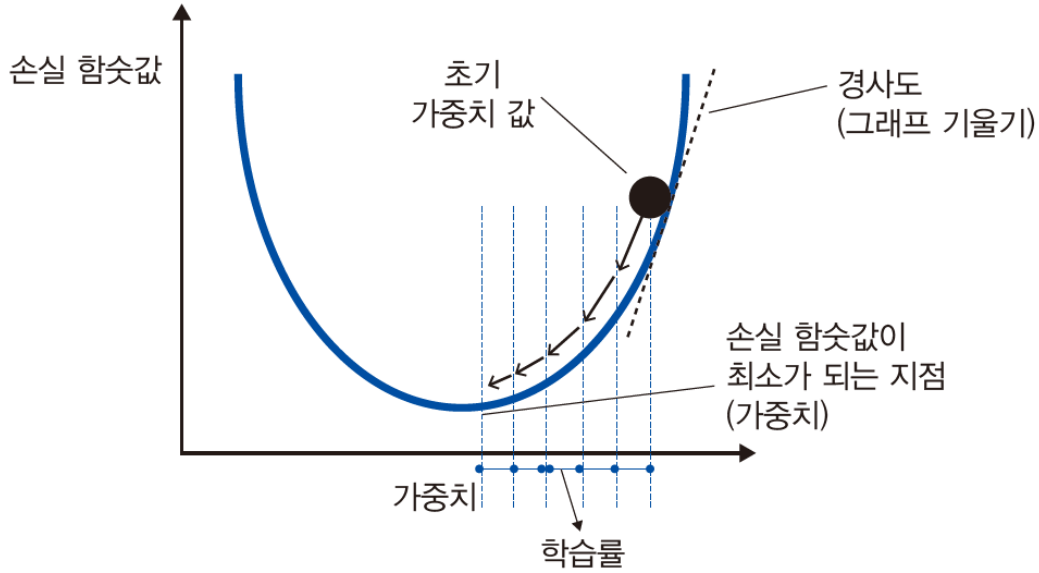
모델 컴파일하기

최적화 함수 설정

- 다음 그래프는 손실 함수와 최적화 함수의 원리 표현



손실 함수(MSE)



최적화 함수의 원리

모델 컴파일하기

평가 지표 설정

평가 지표는 회귀 모델이 결과를 얼마나 잘 예측할 수 있는지 성능을 평가하는 지표

- **평균절대오차**(MAE: Mean Absolute Error) 사용
- MAE는 오차의 절댓값을 모두 더한 것의 평균을 낸 것
- MAE는 오차의 절댓값을 사용하기 때문에 오차의 크기가 그대로 반영
- 따라서 오차가 얼마나 큰지를 쉽게 확인할 수 있어 회귀 모델의 평가 지표로 많이 사용

$$\frac{1}{n}\sum_{i=1}^{n}|\text{예측값}-\text{실젯값}| \qquad (n\text{: 데이터 개수})$$

모델 컴파일하기

모델 컴파일

회귀 모델에서는 분류 모델에서 사용하는 정확도의 개념을 사용하지 않음

- 정확하게 예측했는지 파악하는 것이 아니라 독립 변수로 종속 변수를 얼마나 잘 설명할 수 있는지를 결정계수 R^2(R-squared)값으로 살펴볼 수 있다.
- R^2값이 1에 가까우면 설명을 매우 잘할 수 있는 회귀 모델이고 0에 가까우면 설명을 할 수 없는 모델이다.

모델 컴파일하기

모델 컴파일

- 손실 함수로 MSE, 최적화 함수로 Adam, 평가 지표로 MAE를 사용한다.

모델 객체.compile(loss = '손실 함수명', optimizer = '최적화 함수명', metrics = ['평가 지표명'])

```
1 model.compile(loss = 'mse', optimizer = 'adam', metrics = ['mae'])
```

모델 학습하기

모델 학습

- 모델 학습을 할 때 우리가 설정해야 할 사항은 모델을 학습시킬 **훈련 데이터**, **훈련 데이터의 레이블**, 훈련 데이터를 **반복 학습하는 횟수**(epochs)이다.

모델 객체.fit(훈련 데이터, 훈련 데이터의 레이블, epochs = 반복 학습 횟수)

모델 학습하기

학습 방법
설정 및 학습

- train_input을 훈련 데이터로 train_target을 훈련 데이터의 레이블로 설정하고, 반복 학습 횟수는 50으로 설정한다.
- 여기서 history 객체에 모델 학습 결과를 저장한다.

```
1 history = model.fit(train_input, train_target, epochs = 50)
2 history
```

```
Epoch 1/50
1180/1180 [==============================] - 6s 4ms/step - loss: 8095385.0000 - mae: 1768.8564
Epoch 2/50
1180/1180 [==============================] - 6s 5ms/step - loss: 556845.5625 - mae: 401.2693
Epoch 3/50
1180/1180 [==============================] - 5s 5ms/step - loss: 488995.3438 - mae: 370.9196
                                    ⋮
Epoch 48/50
1180/1180 [==============================] - 2s 2ms/step - loss: 323414.1562 - mae: 316.3353
Epoch 49/50
1180/1180 [==============================] - 3s 2ms/step - loss: 324366.0312 - mae: 317.4190
Epoch 50/50
1180/1180 [==============================] - 2s 2ms/step - loss: 323235.1875 - mae: 315.2485
```

모델 학습하기

학습 방법
설정 및 학습

- 첫 번째 에포크(epoch)에서 손실 함수의 값이 8095385.0000, 평균절대 오차 값이 1768.8564였지만, 각 에포크(epoch)마다 훈련 데이터를 반복하여 학습
- 오차율을 낮추면서 마지막 50번째 에포크에서 손실 함수의 값은 323235.1875
- 평균절대오차 값은 315.2485로 낮아진 것 확인

모델 학습하기

보충

배치 사이즈와 미니 배치

배치 사이즈

- 데이터 1개를 학습시킬 때마다 실젯값과 비교하여 가중치를 갱신하는 것은 너무 비효율적이기 때문에 한 번에 여러 개의 데이터를 학습시킨 후 실젯값과 비교하여 가중치를 갱신
- 한 번에 학습시키는 데이터의 개수를 배치 사이즈라고 한다.
- 첫 번째 반복(epoch)의 1180은 배치 사이즈이다.

모델 학습하기

보충

배치 사이즈와 미니 배치

미니 배치

- 배치 사이즈만큼의 데이터 뭉치를 미니 배치라고 함.
- 케라스 라이브러리에서는 미니 배치의 개수를 별도로 설정하지 않으면 기본값이 32개가 됨.
- 여기서는 전체 훈련 데이터의 개수 37,758개를 32개의 미니 배치로 나누었으므로 배치 사이즈는 1180개가 됨.

모델 평가 및 예측하기

모델 평가

- 모델이 학습을 완료하고 나면 테스트 데이터와 테스트 데이터의 레이블을 사용하여 새로운 데이터를 얼마나 잘 예측할 수 있는지 확인
- 이때 evaluate( )를 사용하며, 사용하는 공식은 다음과 같다.

모델 객체.evaluate(테스트 데이터, 테스트 데이터의 레이블)

모델 평가 및 예측하기

모델 평가

- 테스트 데이터와 테스트 데이터의 레이블 사용하기
- 평가한 결과 중에서 손실 함수의 값과 MAE 값을 test_loss_score, test_mae_score에 저장하기

```
1 test_loss_score, test_mae_score = model.evaluate(test_input, test_target)
```

```
506/506 [==============================] - 1s 1ms/step - loss: 385635.2188 - mae: 364.5073
```

테스트 데이터의 평가 결과인 손실 함수의 값 loss와 평균절대오차 값 mae도 실행할 때마다 조금씩 달라질 수 있어요.

모델 평가 및 예측하기

모델 평가

- 테스트할 때 사용한 데이터는 506개
- 손실 함수의 값은 0.385635
- 평가 지표 MAE는 364.5073

모델 평가 및 예측하기

성능 평가하기

- 테스트 데이터를 사용하여 학습이 잘되었는지 평가
- 실젯값(테스트 데이터의 레이블)과 테스트 데이터로 예측한 결과 간의 관계를 결정계수로 출력

```
r2_score(실젯값, 예측값) # 예측값이 실젯값과 얼마나 일치하는지 확인
```

모델 평가 및 예측하기

성능 평가하기

- 결정계수인 R^2값을 확인하여 독립 변수가 종속 변수를 얼마나 잘 설명할 수 있는지 확인
- 이때 사이킷런 라이브러리의 metrics 사용

```
1 from sklearn.metrics import r2_score
2 r2 = r2_score(test_target, model.predict(test_input))
3 r2
```

```
0.9757539993955275
```

모델 평가 및 예측하기

성능 평가하기

- 결정계수 R2값이 0.9757…인 것 확인
- 독립 변수가 종속변수를 약 97.6% 설명할 수 있다는 것을 의미함.

모델 평가 및 예측하기

실젯값과 예측값 비교 그래프

- 맷플롯립 라이브러리를 사용하면 실젯값과 예측값을 비교하는 그래프와 오차값의 변화를 그래프로 그리는 시각화 표현
- 실젯값과 예측값 간의 관계를 산점도로 표현하기 위해 scatter( ) 함수 사용

맷플롯립 객체.scatter(x축 값, y축 값)

모델 평가 및 예측하기

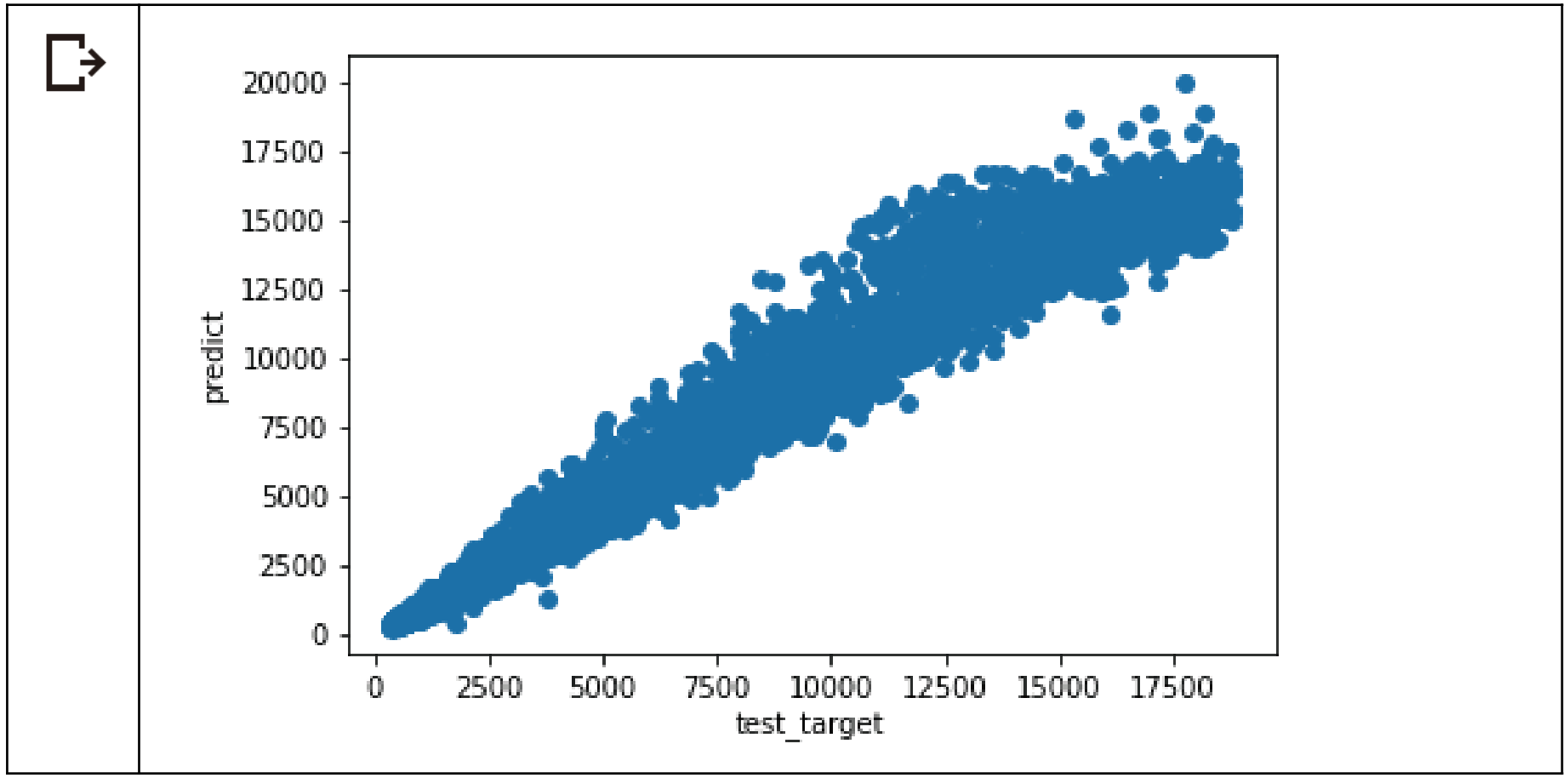
실젯값과 예측값 비교 그래프

- 파이플롯을 불러와 plt라는 별명으로 사용
- x축은 테스트 데이터의 레이블로, y축은 테스트 데이터로 예측한 값으로 설정
- x축명은 'test_target', y축명은 'predict'로 설정하여 산점도로 출력

```
1 import matplotlib.pyplot as plt
2 plt.scatter(test_target, model.predict(test_input))
3 plt.xlabel('test_target')
4 plt.ylabel('predict')
5 plt.show( )
```

모델 평가 및 예측하기

실젯값과 예측값 비교 그래프



모델 평가 및 예측하기

실젯값과 예측값 비교 그래프

- 결정계수 R^2값이 0.9757…이었던 것처럼,
- **실젯값과 예측값이 밀접한 관계**를 맺고 있다는 것을 확인할 수 있다.

모델 평가 및 예측하기

평균절대오차 그래프

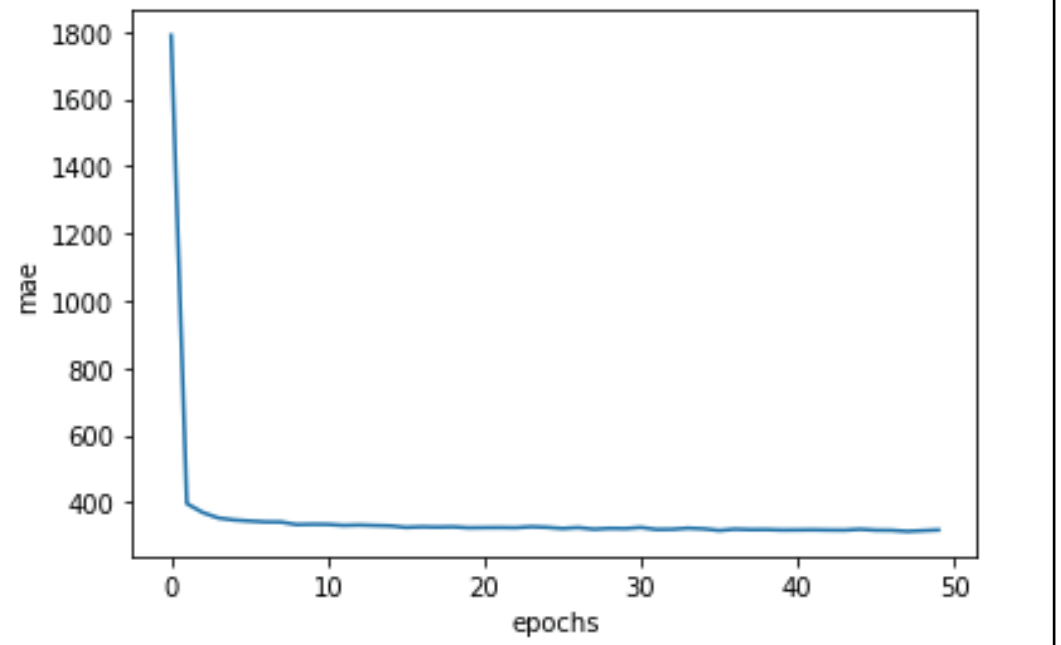
- 회귀 모델이 반복하여 학습(epochs)할 때마다 오차값이 어떻게 변했는지 확인한다.
- 모델을 학습할 때, 학습한 결과를 history라는 객체에 저장해 두었다.
- history에 저장된 MAE값의 리스트를 사용하여 그래프를 그리면 오차값의 변화를 쉽게 확인할 수 있다.

모델 평가 및 예측하기

평균절대오차 그래프

- plot( )을 사용하여 선그래프를 출력하기

```
1 import matplotlib.pyplot as plt
2 plt.plot(history.history['mae'])
3 plt.xlabel('epochs')
4 plt.ylabel('mae')
5 plt.show()
```



모델 평가 및 예측하기

평균절대오차 그래프

- history의 MAE값만 추출한 것을 확인할 수 있다. x축명은 'epochs'로 y축명은 'mae'로 설정
- 초반에 평균절대오차(MAE) 값이 급격히 줄었다가 추후 서서히 줄어드는 것 확인

모델 평가 및 예측하기

확인 문제

- 학습이 완료된 모델로 아래의 새로운 데이터에 대한 예측값 출력하기
- 아래 표 데이터의 범주형데이터 속성인 cut, color, clarity를 원-핫 인코딩하기
- 속성값을 정규화하여 모델이 예측한 가격 확인하기

carat	cut	color	clarity	depth	table	x	y	z
0.42	Very Good	E	SI2	57.2	55	4.33	4.35	2

배운 내용 정리하기

문제 정의하기

문제 상황 이해하기
해결할 문제: 다이아몬드 가격 예측

데이터 불러오기

시본(Seaborn)에서 다이아몬드 데이터 셋 불러오기

데이터 탐색 및 전처리하기

데이터 살펴보기(데이터 유형, 결측치, 속성명 등)
범주형 속성값을 수치형으로 변환하기(원-핫 인코딩)
독립 변수(영향을 미치는 변수)와 종속 변수(영향을 받는 변수) 구분하기
독립 변수 중 속성별 값을 0.0~1.0 사이의 실수로 변환하기(정규화)
훈련 데이터와 테스트 데이터 나누기

배운 내용 정리하기

모델 생성하기

입력층, 은닉층, 출력층 생성하기
활성화 함수(ReLU) 설정하기

모델 컴파일하기

손실 함수(MSE) 설정하기
최적화 함수(Adam) 설정하기
평가 지표(MAE) 설정하기

모델 학습하기

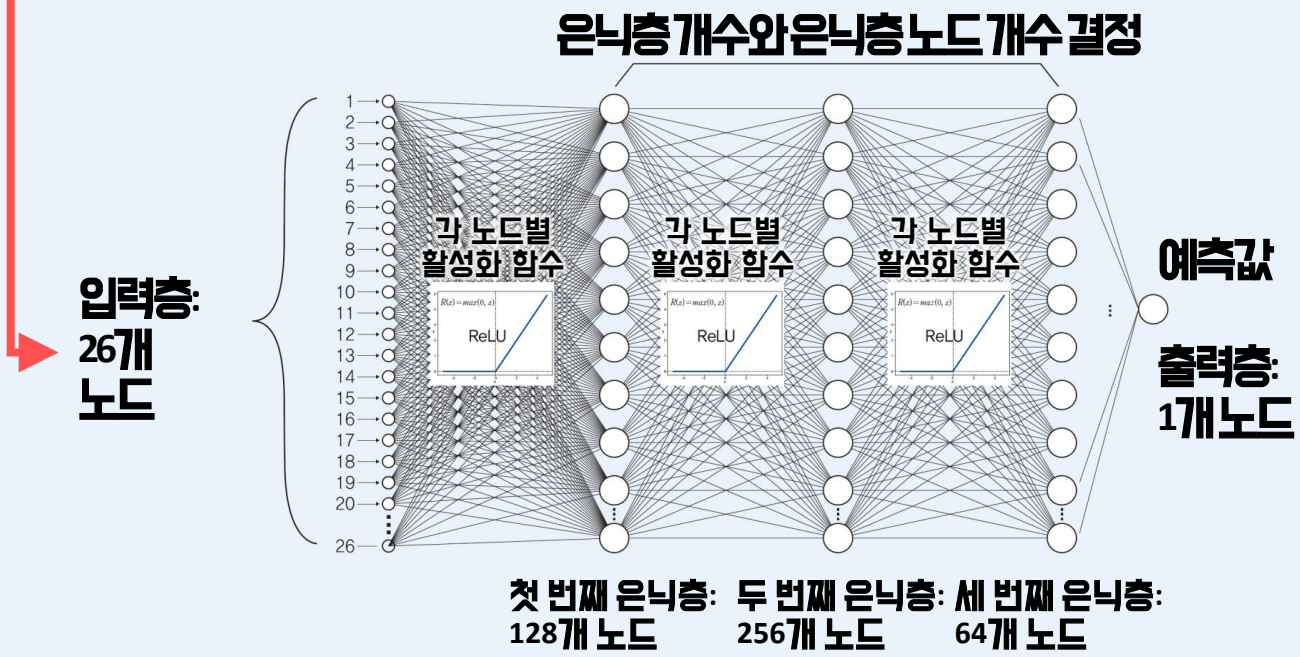
에포크 설정 및 학습하기(손실 함수의 값과 평균절대오차 값 확인)

모델 평가 및 예측하기

테스트 데이터로 모델 평가하기(결정계수 R^2, 평균절대오차 확인)
시각화로 결과 살펴보기

배운 내용 정리하기

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	price
0	0.006237	0.513889	0.230769	0.367784	0.067572	0.076415	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	326
1	0.002079	0.466667	0.346154	0.362197	0.065195	0.072642	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	326
2	0.006237	0.386111	0.423077	0.377095	0.069100	0.072642	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	327



배운 내용 정리하기

- 인공 신경망의 층의 수와 각 층별 노드의 개수를 설정하였습니다. 입력층의 경우 원-핫 인코딩 후 데이터 속성(독립 변수)의 개수와 동일한 26개를 생성하였으며, 출력층은 예측할 데이터 속성(종속 변수)이 다이아몬드 가격뿐이므로 1개의 노드로 구성
- 각 은닉층의 활성화 함수는 렐루 함수로 설정
- 출력층은 값 자체를 그대로 출력하므로 분류와 다르게 활성화 함수를 설정하지 않음.

배운 내용 정리하기

- 예측값과 실젯값의 차이를 계산하는 손실 함수는 평균제곱오차(MSE), 최적화 함수는 Adam, 회귀 모델이 다이아몬드 가격을 어느 정도 설명하는지 평가하는 지표는 평균절대오차(MAE)로 설정하여 모델 학습 진행

- 학습 횟수가 반복될수록 손실 함숫값은 낮아지는 반면, 정확도는 높아지는 학습 과정 확인