신경망 활용 처음부터 끝까지: 분류와 회귀

주요 내용

- 이진 분류 신경망 모델: 영화 후기 분류
- 다중 클래스 분류 신경망 모델: 뉴스 기사 분류
- 회귀 신경망 모델: 주택 가격 예측

머신러닝 주요 용어

한글	영어	뜻
샘플, 입력값	sample, input	모델 훈련에 사용되는 데이터
예측값, 출력값	prediction, output	모델이 계산한 예측값
타깃	target	모델이 맞춰야 하는 값
손실값, 비용, 예측 오차	loss value	타깃과 예측값 사이의 오차. 문제 유형에 따라 측정법 다름.
손실 함수, 비용 함수	loss function	손실값(비용)을 계산하는 함수.
클래스	class	분류 모델에서 각각의 샘플이 속하는 범주(클래스)
라벨	label	분류 모델에서 타깃 대신 사용하는 표현

한글	영어	뜻
이진 분류	binary classification	양성/음성, 긍정/부정 등 샘플을 두 개의 클래스로 분류.
다중 클래스 분류	multiclass classification	샘플을 세 개 이상의 클래스로 분류. 손글씨 숫자 분류 등.
다중 라벨 분류	multilabel classification	샘플에 대해 두 종류 이상의 라벨을 지정하는 분류. 한 장의 사진에 강아지, 고양이, 토끼 등 여러 종의 포함 여부 확인. 각각의 종에 대해 범부를 맞춰야 함.
(스칼라) 회귀	(scalar) regression	샘플 별로 하나의 값만 예측하기. 주택 가격 예측 등.
벡터 회귀	vector regression	샘플 별로 두 종류 이상의 값 예측하기. 네모 상자의 좌표 등.
배치	batch	보통 16, 32, 64, 128 등의 개수의 샘플로 구성된 묶음(배치). 훈련 루프의 스텝에 사용되는 훈련셋.

영화 후기: 이진 분류

영화 후기의 긍정/부정 여부를 판단하는 이진 분류 모델을 구성한다.

데이터 준비: IMDB 데이터셋

- 긍정 후기와 부정 후기 각각 25,000개
- IMDB(Internet Moview Database) 영화 후기 사이트

케라스 데이터셋 모듈

`tf.keras.datasets` 모듈이 몇 개의 연습용 데이터셋을 제공한다.

- MNIST 손글씨 숫자 분류 데이터셋
- CIFAR10 작은 이미지 분류 데이터셋
- CIFAR100 작은 이미지 분류 데이터셋
- IMDB 영화 후기 감성 분류 데이터셋
- Reuters 단문 기사 주제 분류 데이터셋
- 패션 MNIST(Fashion MNIST) dataset
- 보스턴 주택 가격(Boston Housing price) 회귀 데이터셋

케라스 데이터셋의 `load_data()` 함수

각 데이터셋의 `load_data()` 메서드를 활용하여 데이터셋을 불러온다. 반환값은 미리 구분된 훈련셋과 테스트셋을 담고 있는 넘파이 어레이 텐서다.

```
>>> from tensorflow.keras.datasets import imdb

>>> (train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)
```

단어 사용 빈도가 높은 10,000개 단어만 사용한다.

- 그 이외에는 사용 빈도가 너무 낮아 모델 훈련에 도움되지 않는다.
- `num_words=10000` 키워드 인자를 활용한다.

데이터 살펴보기

후기 샘플 각각에 사용되는 단어의 수는 일정하지 않다. 즉 각 후기 문장의 길이가 일정하지 않다.

예를 들어, 훈련셋의 첫째 후기 문장은 218개의 단어로, 둘째 후기 문장은 189개의 단어로 구성된다.

```
>>> len(train_data[0])
218

>>> len(train_data[1])
189
```

각각의 정수는 특정 단어를 가리킨다. 훈련셋의 0번 입력 샘플의 처음 10개 값(단어)은 다음과 같다.

```
>>> train_data[0][:10]
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
```

훈련셋 0번 샘플은 긍정 후기를 가리킨다.

```
>>> train_labels[0]
1
```

데이터 전처리: 벡터화와 멀티-핫 인코딩

- **벡터화**vectorization
  - 가장 긴 길이의 샘플에 맞춰 모든 샘플을 확장한다.
  - 확장에 사용되는 값은 기존 샘플에 사용되지 않은 값을 사용한다.
  - 예를 들어 여백을 의미하는 0을 사용할 수 있다.

- **멀티-핫 인코딩**multi-hot encoding
  - 0과 1로만 이루어진 일정한 길이의 벡터(1차원 어레이)로 변환한다.
  - 벡터의 길이는 사용된 단어의 총 수, 예를 들어 10,000을 사용한다.

영화 후기 멀티-핫 인코딩

- 어레이 길이: 10,000
- 항목: 0 또는 1
- 후기 샘플에 포함된 정수에 해당하는 인덱스의 항목만 1로 지정

예를 들어, `[1, 5, 9998]` 변환하기:

- 길이가 10,000인 1차원 어레이(벡터)로 변환
- 1번, 5번, 9998번 인덱스의 항목만 1이고 나머지는 0

```
[1, 5, 9998] => [0, 1, 0, 0, 0, 1, 0, ..., 0, 0, 1, 0]
```

멀티-핫 인코딩 함수

```
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))

    for i, seq in enumerate(sequences):
        for j in seq:
            results[i, j] = 1.
    return results
```

```
>>> x_train = vectorize_sequences(train_data).astype("float32")

>>> x_test = vectorize_sequences(test_data).astype("float32")
```

라벨 멀티-핫 인코딩: ❌

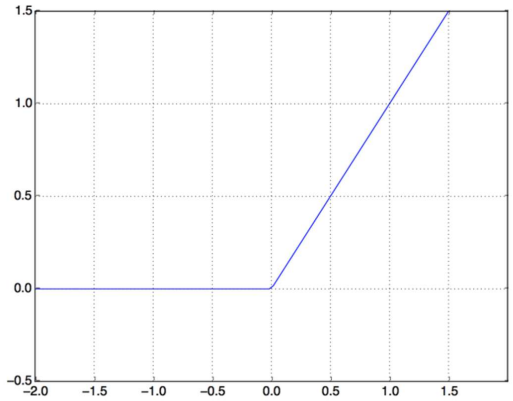
라벨(타깃)은 멀티-핫 인코딩을 적용하지 않는다. 다만 입력 샘플의 자료형과 맞추기 위해 `float32` 자료형으로 변환한다.

```
>>> y_train = np.asarray(train_labels).astype("float32")

>>> y_train
array([1., 0., 0., ..., 0., 1., 0.], dtype=float32)
```

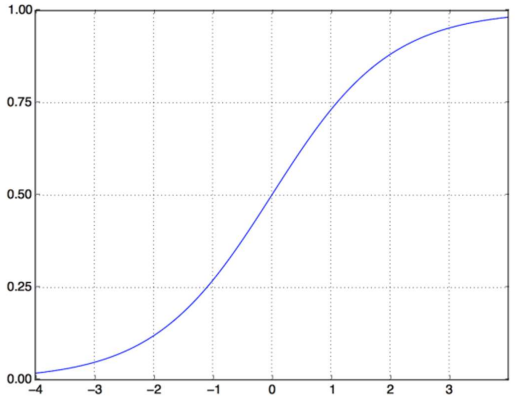
모델 구성

- 입력 샘플이 벡터(1차원 어레이)로 주어지고 라벨이 스칼라(하나의 숫자)일 때:
    - 밀집층densely-connected layer인 `Dense` 층
    - `Sequential` 모델 이용 추천
- 은닉층의 활성화 함수: `relu`, `prelu`, `elu`, `tanh` 등이 많이 사용됨. 일반적으로 `relu` 추천.
- 이진 분류 모델의 최상위 출력층의 활성화 함수: 0과 1사이의 확률값을 계산하는 `sigmoid` 함수
- 다중 클래스 분류 모델의 최상위 출력층의 활성화 함수: 클래스별 확률값을 계산하는 `softmax` 함수

relu와 sigmoid



relu 함수

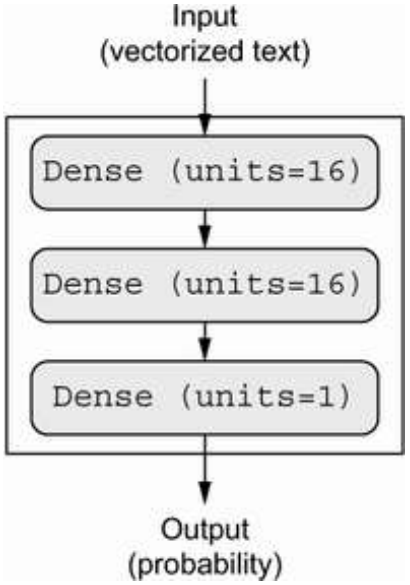


sigmoid 함수

Dense 층 활용

- 몇 개의 층을 사용하는가?
- 각 층마다 몇 개의 유닛unit을 사용하는가?

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
    ])
```



이진 분류 모델 컴파일

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"]
             )
```

모델 훈련과 활용

훈련 중인 모델을 에포크마다 검증하기 위해 검증셋을 따로 지정한다.

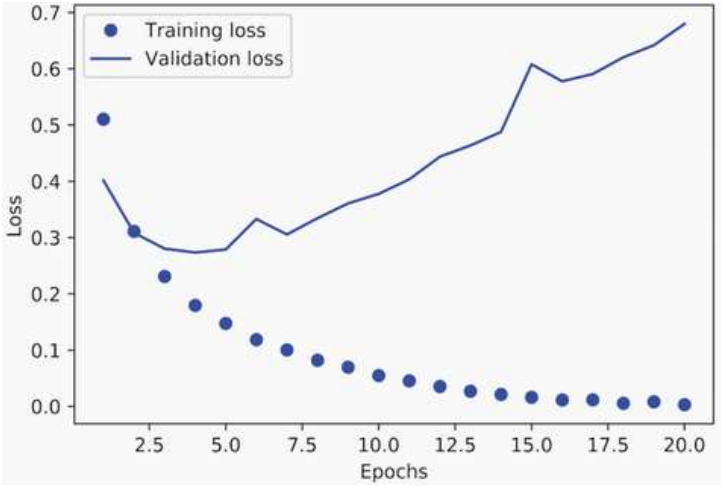
```
x_val = x_train[:10000]            # 검증용
partial_x_train = x_train[10000:]  # 훈련용
y_val = y_train[:10000]            # 검증용 타깃셋
partial_y_train = y_train[10000:]  # 훈련용 타깃셋

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val) # 검증 데이터셋 지정
                   )
```

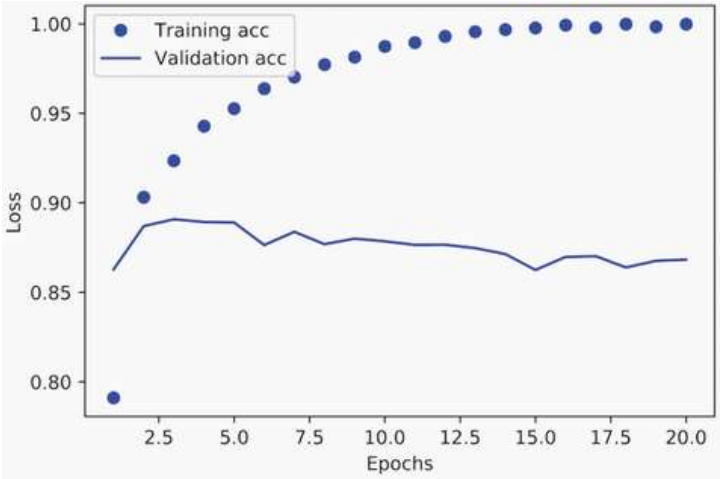
`fit()` 메서드 반환값: `History` 객체

```
>>> history_dict = history.history
>>> history_dict.keys()
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

손실값과 정확도의 변화

`history` 속성에 저장된 정보를 이용하여 손실값과 정확도의 변화를 확인할 수 있다.





과대적합 방지

- **과대적합**overfitting: 모델이 훈련셋에 익숙해져서 처음 보는 데이터에 대해서 성능이 더 이상 좋아지지 않거나 떨어지는 현상
- 4번째 에포크 이후로 과대적합 발생. 4번의 에포크만 훈련 반복을 진행하면 과대적합되지 않은 모델이 훈련됨
- 모델 재훈련: 모델 구성부터, 컴파일, 훈련을 모두 처음부터 다시 시작. 가중치와 편향이 초기화된 상태로 훈련이 다시 시작됨

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

model.fit(x_train, y_train, epochs=4, batch_size=512)
```

훈련 결과 테스트

```
>>> results = model.evaluate(x_test, y_test)
>>> results
[0.3139097988605499, 0.8770800232887268]
```

모델 활용

```
>>> model.predict(x_test, batch_size=512)
array([[0.25440323],
       [0.9999424 ],
       [0.95840394],
       ...,
       [0.17153329],
       [0.10725482],
       [0.6672551 ]], dtype=float32)
```

뉴스 기사: 다중 클래스 분류

로이터Reuter 통신사가 1986년에 작성한 단문 기사를 주제별로 분류한다.

데이터 준비: 로이터 데이터셋

- 총 11,228개의 단문 기사
    - 훈련셋 크기: 8,982
    - 테스트셋 크기: 2,246
- 기사 주제: 총 46 개
- 각각의 기사는 하나의 주제와 연관됨.
- **다중 클래스 분류**multiclass classification 모델 훈련

```
from tensorflow.keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) =
reuters.load_data(num_words=10000)
```

데이터 살펴보기

각 샘플은 정수들의 리스트이다.

```
>>> train_data[10]
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979,
3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]
```

각 샘플에 대한 라벨은 0부터 45까지의 정수로 표현된다. 3번 주제는 소득(earn)을 가리킨다.

```
>>> train_labels[10]
3
```

로이터 기사 주제

번호	주제	번호	주제	번호	주제	번호	주제
0	cocoa	1	grain	2	veg-oil	3	earn
4	acq	5	wheat	6	copper	7	housing
8	money-supply	9	coffee	10	sugar	11	trade
12	reserves	13	ship	14	cotton	15	carcass
16	crude	17	nat-gas	18	cpi	19	money-fx
20	interest	21	gnp	22	meal-feed	23	alum
24	oilseed	25	gold	26	tin	27	strategic-metal
28	livestock	29	retail	30	ipi	31	iron-steel
32	rubber	33	heat	34	jobs	35	lei
36	bop	37	zinc	38	orange	39	pet-chem
40	dlr	41	gas	42	silver	43	wpi
44	hog	45	lead				

입력 데이터셋 멀티-핫 인코딩

IMDB의 경우와 동일한 방식

```
>>> x_train = vectorize_sequences(train_data)
>>> x_test = vectorize_sequences(test_data)
```

라벨 데이터셋 원-핫 인코딩

라벨은 0부터 45 사이의 값이다. 3을 길이가 46인 벡터로 원-핫 인코딩one-hot encoding 하는 과정은 다음과 같다.

```
3 => [0, 0, 0, 1, 0, 0, ...., 0]
```

`to_categorical()` 함수

원-핫 인코딩을 담당한다.

```
>>> from tensorflow.keras.utils import to_categorical

>>> y_train = to_categorical(train_labels)
>>> y_test = to_categorical(test_labels)

>>> y_train[0]
array([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., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

모델 구성

- 은닉층: 64개의 유닛 사용.
    - 이진 분류보다 훨씬 많은 46개의 클래스로 분류하기 위해 보다 많은 정보 필요
    - 출력층의 유닛 수보다 커야 함.
- 다중 클래스 분류 모델의 출력층: 클래스 수 만큼의 유닛을 사용하는 `Dense` 밀집층을 사용
    - 활성화 함수: 모든 유닛에 대한 확률값의 합이 1이 되도록 하는 `softmax()`

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])
```

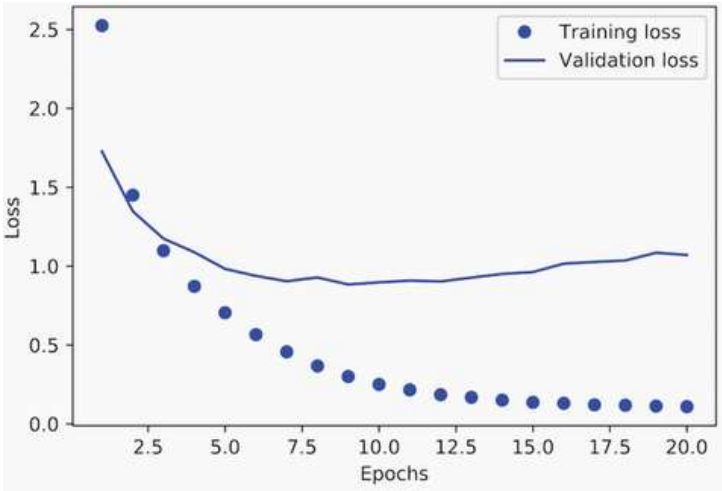
모델 컴파일

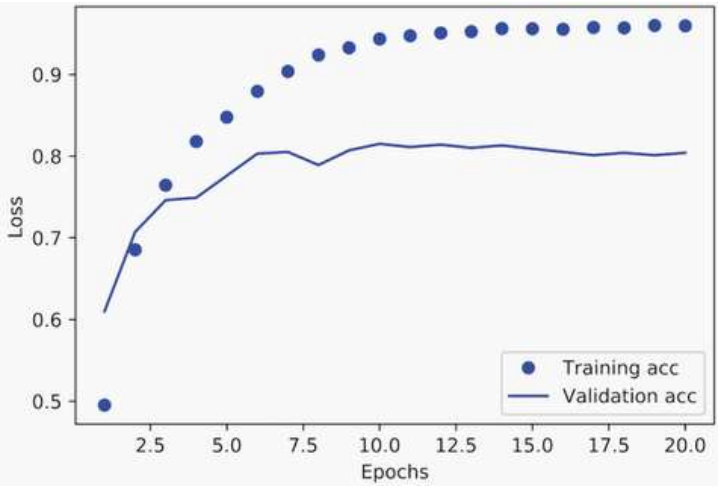
```
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

모델 훈련과 활용

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

손실값과 정확도의 변화


Training loss
Validation loss
Loss
Epochs
2.5
2.0
1.5
1.0
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training acc
Validation acc
Loss
Epochs
0.9
0.8
0.7
0.6
0.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

모델 재훈련

9번 에포크를 지나면서 과대적합이 발생한다.

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
model.fit(x_train,
          y_train,
          epochs=9,
          batch_size=512)
```

주택가격: 회귀

미국 보스턴Boston 시의 1970년대 중반의 주택가격을 예측하는 회귀 모델을 훈련시킨다.

데이터 준비: 보스턴 주택가격 데이터셋

- 1970년대 중반의 미국 보스턴 시 외곽의 총 506개 지역에서 수집된 통계 자료

특성	의미
CRIM	구역별 1인당 범죄율
ZN	25,000 평방 피트 이상의 주거 구역 비율
INDUS	구역별 비 소매 사업 면적(에이커) 비율
CHAS	Charles River 경계 접촉 여부
NOX	산화 질소 농도
RM	주택 당 평균 방 수
AGE	1940년 이전에 지어졌으면서 소유주가 살고 있는 주택 비율
DIS	보스턴 고용 센터 다섯 곳 까지의 가중(weighted) 거리
RAD	방사형 고속도로 접근성 지수
TAX	1만달러당 재산세율
PTRATIO	구역별 학생-교사 비율
B	1000(Bk - 0.63)^2 (Bk는구역별 흑인 비율)
LSTAT	구역별 하위 계층 인구 비율

보스턴 데이터셋의 윤리 문제

- 구역별로 범죄율, 흑인 비율, 하위 계층 비율 등을 포함
- 특히 흑인 비율을 사용하는 `B` 특성이 윤리적 논쟁을 불러 일으킴.
- 1970년대 미국에서 인종 차별이 여전히 주요 쟁점이었음을 단편적으로 보여줌.
- 여기서는 단순히 데이터 활용 차원에서만 보스턴 데이터셋을 이용할 뿐 다른 어떤 의도도 없음

데이터셋 준비

총 506개의 데이터 샘플로 구성된 매우 작은 데이터셋이다.

```
from tensorflow.keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()
```

타깃은 구역별 중앙 주택가격이며 부동소수점을 사용한다.

```
>>> train_targets
[ 15.2,  42.3,  50. ...  19.4,  19.4,  29.1]
```

데이터 전처리: 표준화

특성별 스케일을 통일시키기 위해 모든 특성별로 표준화를 사용한다.

$$\frac{x-\mu}{\sigma}$$

```
# 훈련셋의 특성별 평균값/표준편차
mean = train_data.mean(axis=0)
std = train_data.std(axis=0)

# 훈련셋 표준화
train_data -= mean
train_data /= std
```

테스트셋 표준화

- 테스트셋의 입력값도 표준화를 진행한다.
- 다만 훈련셋의 평균값과 표준편차를 사용한다.
- 이유는 테스트셋에 대한 어떤 정보도 미리 알 수 없다는 전제가 실현되야 하기 때문이다.

```
# 테스트셋 표준화
test_data -= mean
test_data /= std
```

모델 구성과 컴파일

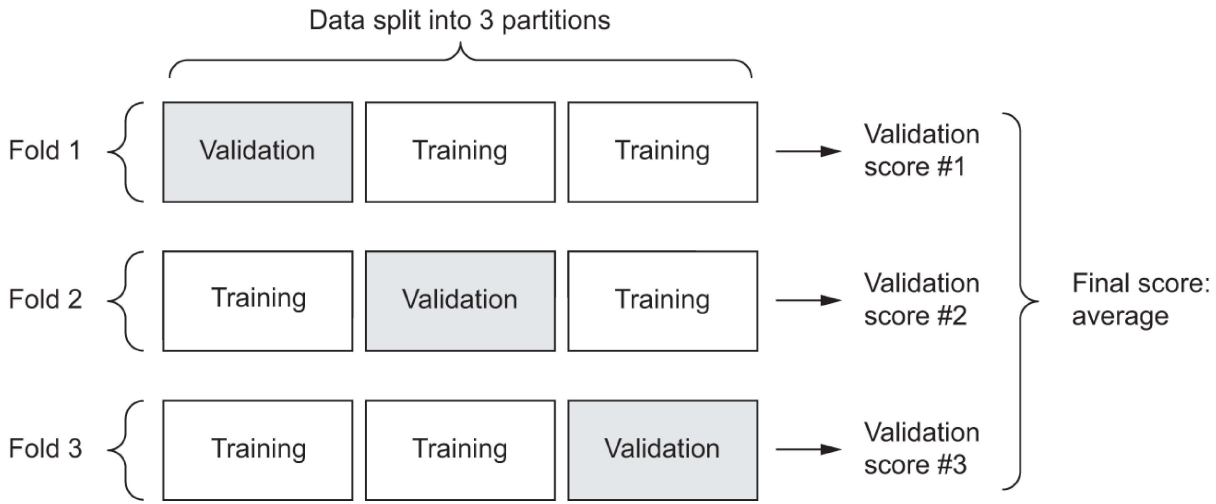
- 데이터셋이 작음
- 출력층을 제외하고 두 개 층만 사용
- 머신러닝 모델은 훈련셋이 작을 수록 과대적합을 보다 잘하기 때문에 보다 단순한 모델을 사용 권장

이번 훈련에서는 동일한 모델을 반복해서 재구성할 것이기에 모델 구성과 컴파일을 하나의 함수로 지정한다.

```
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
    return model
```

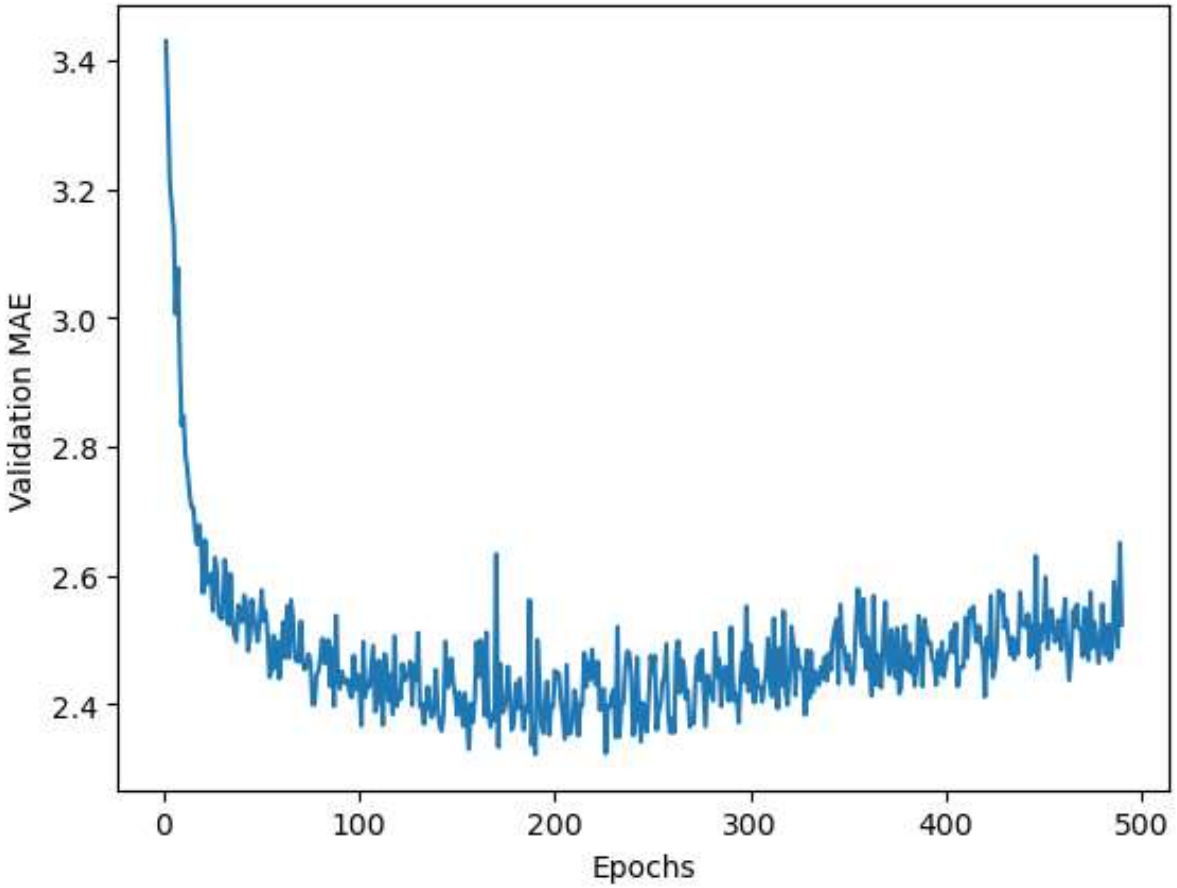
모델 훈련과 활용: K-겹 교차검증

훈련셋이 너무 작은 경우 검증셋을 별도로 지정하기 보다는 K-겹 교차검증을 이용한다.



과대 적합 발생

대략 200번 에포크 근처에서 과대적합이 발생한다.



모델 재훈련

mae가 최소가 되는 에포크를 확인하여 그만큼의 에포크만 사용해서 모델을 재훈련 하면 좋은 성능의 모델을 얻는다.

```
# 과대적합 발생하지 않는 최대 에포크
overfitting_epoch = np.argmin(average_mae_history)

# 모델 재훈련
model = build_model()
model.fit(train_data,
          train_targets,
          epochs=overfitting_epoch,
          batch_size=16,
          verbose=0)
```