

분류와 회귀

주요 내용

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

머신러닝 주요 용어

한글	영어	뜻
샘플, 입력값	sample, input	모델 훈련에 사용되는 데이터
배치	batch	16, 32, 64, 128 등의 개수의 샘플로 구성된 묶음(배치). 훈련 루프의 스텝에 사용되는 훈련셋.
예측값, 출력값	prediction, output	모델이 계산한 예측값
타깃	target	모델이 맞춰야 하는 값
라벨	label	분류 모델에서 타깃을 가리키는 표현
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(범주)	class	분류 모델에서 각각의 샘플이 속하는 범주(클래스)
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, 토키 등 여러 종의 포함 여부 확인
(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.

이진 분류: 영화 후기 분류

영화 후기의 긍정/부정 여부를 판단하는 이진 분류 모델을 구성한다.

데이터 준비: IMDB 데이터셋

- 긍정 후기와 부정 후기 각각 25,000개
- [IMDB\(Internet Movie Database\)](#) 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

단어 사용 빈도가 높은 10,000개 단어만 사용한다.

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

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

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

- **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)
```

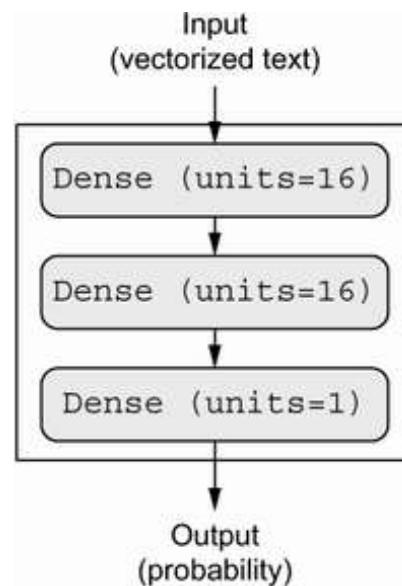
모델 구성

- MNIST 데이터셋을 이용했을 때처럼 `Dense` 층과 `Sequential` 클래스를 이용하여 단순한 모델을 구성한다.
- `Dense` 층으로 신경망을 구성할 때 다음 두 가지를 정해야 한다.
 - 몇 개의 층을 사용하는가?
 - 각 층마다 몇 개의 유닛_{unit}을 사용하는가?
- 여기서는 세 개의 `Dense` 층으로 순차 모델을 구성한다.

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

은닉층과 출력층

- 출력층: 딥러닝 신경망 모델에 사용된 층 중에서 모델의 예측값을 계산하는 마지막 층
- 은닉층: 출력층 이외의 다른 층



출력층의 유닛 수

- 이진 분류 모델의 출력층
 - 유닛 1개
 - 긍정과 부정 중의 하나, 양성과 음성 중의 하나를 결정하는 데에 사용될 하나의 값 저장
- 다중 클래스 분류 모델의 출력층
 - 범주 개수만큼의 유닛 사용
 - MNIST 모델 훈련: 유닛 10개
- 회귀 모델의 출력층
 - 유닛 1개
 - 하나의 예측값 저장

은닉층의 활성화 함수 선택

- 보통 음수값을 제거하는 `relu()` 함수가 사용

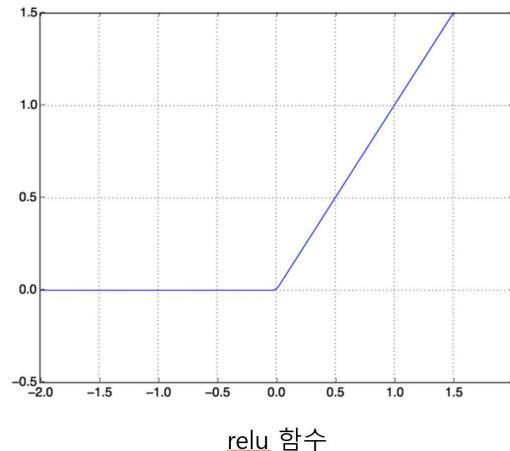
```
def relu(x):
    if x > 0:
        return x
    else:
        return 0
```

- 은닉층의 활성화 함수로 `relu()` 이외에 `prelu()`, `elu()`, `tanh()` 등이 많이 활용됨.

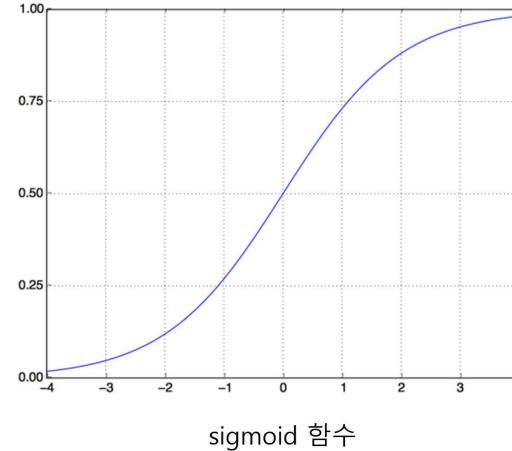
출력의 활성화 함수 선택

- 출력층의 활성화 함수: 모델의 종류에 따라 다르게 선택
- 영화 후기의 긍정/부정을 예측하는 이진 분류 모델의 출력층: 0과 1사이의 확률값을 계산하는 `sigmoid()` 함수 사용

```
def sigmoid(x):  
    ...  
    return 1/(1 + np.exp(-x))
```



relu 함수



sigmoid 함수

- 다중 클래스 분류 모델의 경우: `softmax()` 함수를 사용
- 회귀 모델의 경우: 일반적으로 활성화 함수를 사용하지 않음.

이진 분류 모델 컴파일

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

모델 훈련과 활용

```
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

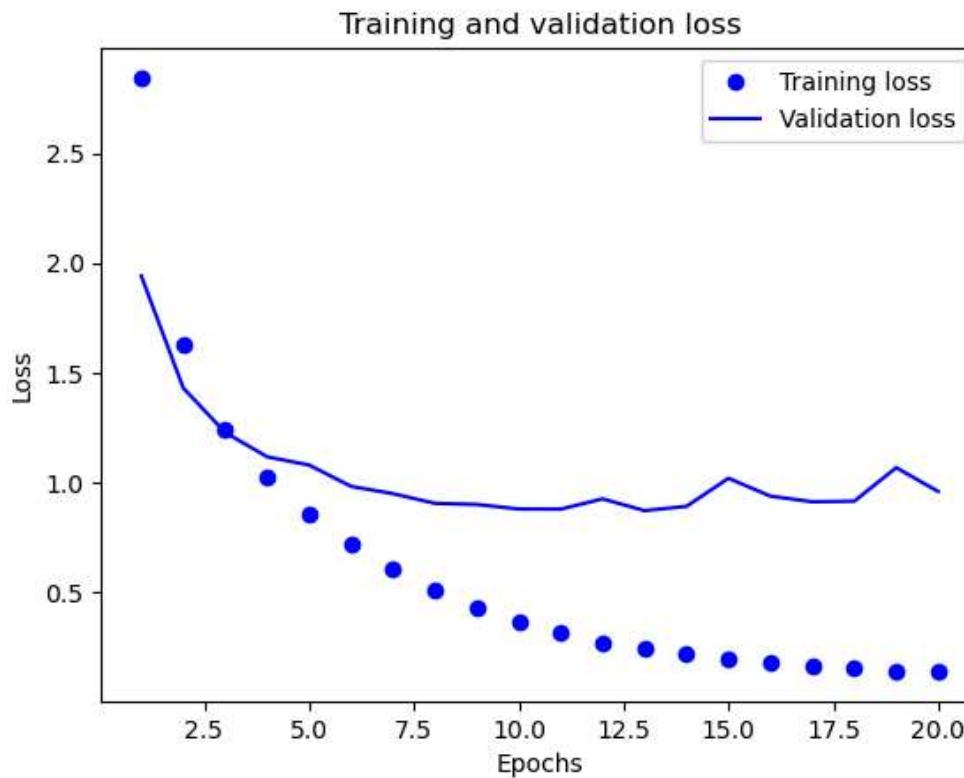
history = model.fit(partial_x_train,
                     partial_y_train,
                     epochs=20,
                     batch_size=512,
                     validation_data=(x_val, y_val) # 검증 데이터셋 지정
)
```

fit() 메서드 반환값: History 객체

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

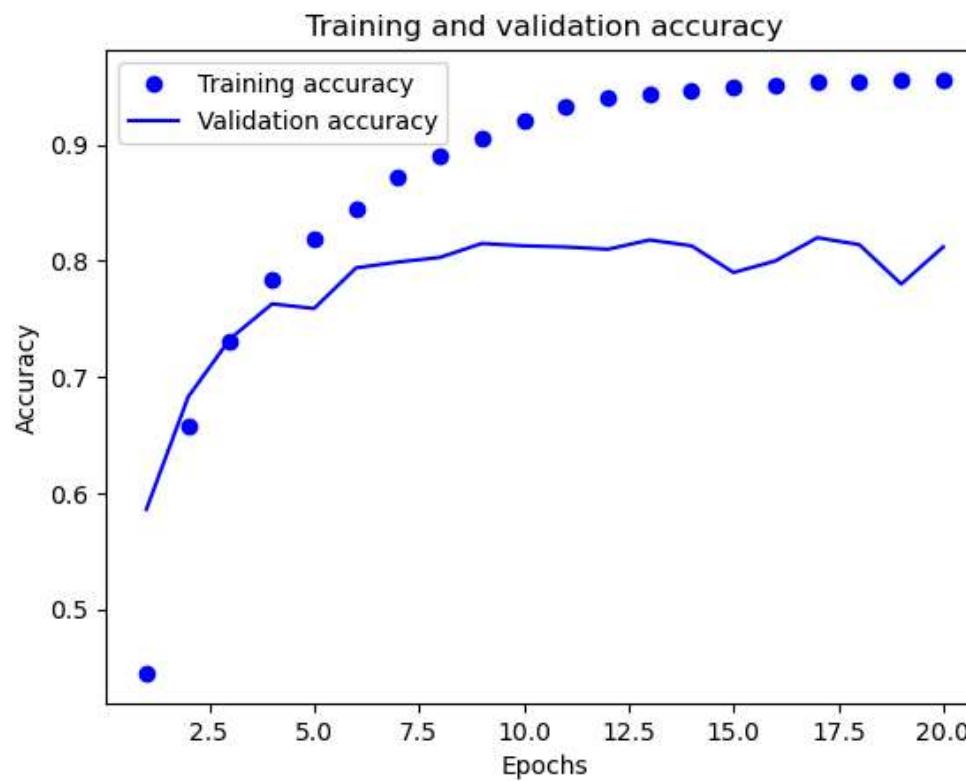
손실값 변화

```
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
```



정확도 변화

```
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
```



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 평가

```
>>> 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)
```