뉴스 기사 분류: 다중 분류 문제

학습 내용

- 01 로이터 뉴스를 이용한 신경망 구현
- 02 다항분류 문제

In [140]:

```
import keras
keras.__version__
```

Out[140]:

```
'2.4.3'
```

로이터 뉴스를 46개의 상호 배타적인 토픽으로 분류하는 신경망

- 1986년에 로이터에서 공개한 짧은 뉴스 기사와 토픽의 집합인 로이터 데이터셋을 사용
- 46개의 토픽
- 각 토픽은 훈련 세트에 최소한 10개의 샘플

In [141]:

```
from keras.datasets import reuters

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

- IMDB 데이터셋에서처럼 num_words=10000 매개변수는 데이터에서 가장 자주 등장하는 단어 10,000개로 제한

In [142]:

```
len(train_data)
```

Out[142]:

```
8982
```

In [143]:

```
len(test_data)
```

Out[143]:

```
2246
```

In [144]:

```
print(train_labels)
```

```
[ 3  4  3 ... 25  3 25]
```

In [145]:

```
train_data[10][0:15]
```

Out[145]:

```
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74]
```

단어로 디코딩

In [146]:

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key)
                           for (key, value) in word_index.items()])

# 0, 1, 2는 '패딩', '문서 시작', '사전에 없음'을 위한 인덱스이므로 3을 뺍니다
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?')
                             for i in train_data[0]])
```

In [147]:

```
decoded_newswire
```

Out[147]:

```
'? ? ? said as a result of its december acquisition of space co it expects earnings
per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company
said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and ren
tal operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per
share this year should be 2 50 to three dlrs reuter 3'
```

샘플과 연결된 레이블

- 토픽의 인덱스로 0과 45사이의 정수

In [148]:

```
train_labels[10]
```

Out[148]:

```
3
```

데이터 준비

In [149]:

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

# 훈련 데이터 벡터 변환
x_train = vectorize_sequences(train_data)

# 테스트 데이터 벡터 변환
x_test = vectorize_sequences(test_data)
```

레이블을 벡터로 바꾸는 방법은 두 가지

- 레이블의 리스트를 정수 텐서로 변환하는 것과 원-핫 인코딩을 사용하는 것

In [150]:

```
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

# 출력(레이블)을 벡터 변환(원핫)
# 훈련 레이블 벡터 변환
one_hot_train_labels = to_one_hot(train_labels)
# 테스트 레이블 벡터 변환
one_hot_test_labels = to_one_hot(test_labels)
```

- MNIST 예제에서 이미 보았듯이 케라스에는 이를 위한 내장 함수

모델 구성

- 마지막 Dense 층의 크기가 46 : 각 입력 샘플에 대해서 46차원의 벡터를 출력
- 마지막 층에 softmax 활성화 함수을 사용
- 각 입력 샘플마다 46개의 출력 클래스에 대한 확률 분포를 출력
- 즉, 46차원의 출력 벡터를 만들며 output[i]는 어떤 샘플이 클래스 i에 속할 확률입니다. 46개의 값을 모두 더하면 1이 됩니다.
- 손실 함수는 categorical_crossentropy
    - 이 함수는 두 확률 분포의 사이의 거리를 측정
    - 네트워크가 출력한 확률 분포와 진짜 레이블의 분포 사이의 거리
        - 두 분포 사이의 거리를 최소화하면 진짜 레이블에 가능한 가까운 출력을 내도록 모델을 훈련

In [151]:

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

In [152]:

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

모델 검증

- 훈련 데이터에서 1,000개의 샘플을 따로 떼어서 검증 세트로 사용

In [153]:

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

In [154]:

```
# 4. 모델 학습시키기
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(patience = 20) # 조기종료 콜백함수 정의
```

In [155]:

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=50,
                    batch_size=512,
                    validation_data=(x_val, y_val),
                    callbacks=[early_stopping])
```

```
Epoch 1/50
16/16 [==============================] - 1s 48ms/step - loss: 3.1092 - accuracy: 0.4
084 - val_loss: 1.7030 - val_accuracy: 0.6560
Epoch 2/50
16/16 [==============================] - 1s 33ms/step - loss: 1.4936 - accuracy: 0.6
975 - val_loss: 1.3101 - val_accuracy: 0.7120
Epoch 3/50
16/16 [==============================] - 1s 32ms/step - loss: 1.0995 - accuracy: 0.7
617 - val_loss: 1.1436 - val_accuracy: 0.7540
Epoch 4/50
16/16 [==============================] - 1s 32ms/step - loss: 0.8673 - accuracy: 0.8
093 - val_loss: 1.1018 - val_accuracy: 0.7580
Epoch 5/50
16/16 [==============================] - 1s 32ms/step - loss: 0.7047 - accuracy: 0.8
465 - val_loss: 0.9751 - val_accuracy: 0.8010
Epoch 6/50
16/16 [==============================] - 1s 32ms/step - loss: 0.5270 - accuracy: 0.8
920 - val_loss: 0.9394 - val_accuracy: 0.7990
Epoch 7/50
16/16 [==============================] - 1s 33ms/step - loss: 0.4259 - accuracy: 0.9
127 - val_loss: 0.9274 - val_accuracy: 0.8090
Epoch 8/50
16/16 [==============================] - 0s 31ms/step - loss: 0.3555 - accuracy: 0.9
283 - val_loss: 0.9001 - val_accuracy: 0.8170
Epoch 9/50
16/16 [==============================] - 1s 33ms/step - loss: 0.2821 - accuracy: 0.9
396 - val_loss: 0.9329 - val_accuracy: 0.8130
Epoch 10/50
16/16 [==============================] - 0s 31ms/step - loss: 0.2298 - accuracy: 0.9
508 - val_loss: 0.9195 - val_accuracy: 0.8170
Epoch 11/50
16/16 [==============================] - 0s 31ms/step - loss: 0.1966 - accuracy: 0.9
548 - val_loss: 0.9113 - val_accuracy: 0.8170
Epoch 12/50
16/16 [==============================] - 1s 32ms/step - loss: 0.1686 - accuracy: 0.9
580 - val_loss: 0.9353 - val_accuracy: 0.8160
Epoch 13/50
16/16 [==============================] - 1s 32ms/step - loss: 0.1557 - accuracy: 0.9
566 - val_loss: 0.9703 - val_accuracy: 0.8150
Epoch 14/50
16/16 [==============================] - 0s 31ms/step - loss: 0.1394 - accuracy: 0.9
577 - val_loss: 0.9736 - val_accuracy: 0.8180
Epoch 15/50
16/16 [==============================] - 1s 33ms/step - loss: 0.1272 - accuracy: 0.9
615 - val_loss: 1.1096 - val_accuracy: 0.7840
Epoch 16/50
16/16 [==============================] - 1s 34ms/step - loss: 0.1230 - accuracy: 0.9
618 - val_loss: 1.0400 - val_accuracy: 0.8070
Epoch 17/50
16/16 [==============================] - 1s 32ms/step - loss: 0.1200 - accuracy: 0.9
604 - val_loss: 1.0236 - val_accuracy: 0.8110
```

```
Epoch 18/50
16/16 [==============================] - 1s 33ms/step - loss: 0.1118 - accuracy: 0.9
625 - val_loss: 1.0798 - val_accuracy: 0.8030
Epoch 19/50
16/16 [==============================] - 1s 33ms/step - loss: 0.1069 - accuracy: 0.9
592 - val_loss: 1.0416 - val_accuracy: 0.8160
Epoch 20/50
16/16 [==============================] - 1s 33ms/step - loss: 0.1111 - accuracy: 0.9
586 - val_loss: 1.1016 - val_accuracy: 0.8000
Epoch 21/50
16/16 [==============================] - 1s 34ms/step - loss: 0.0953 - accuracy: 0.9
640 - val_loss: 1.1554 - val_accuracy: 0.7960
Epoch 22/50
16/16 [==============================] - 1s 32ms/step - loss: 0.0892 - accuracy: 0.9
649 - val_loss: 1.1395 - val_accuracy: 0.8050
Epoch 23/50
16/16 [==============================] - 1s 33ms/step - loss: 0.0965 - accuracy: 0.9
608 - val_loss: 1.1828 - val_accuracy: 0.7910
Epoch 24/50
16/16 [==============================] - 1s 34ms/step - loss: 0.0952 - accuracy: 0.9
629 - val_loss: 1.1306 - val_accuracy: 0.8030
Epoch 25/50
16/16 [==============================] - 1s 32ms/step - loss: 0.0873 - accuracy: 0.9
625 - val_loss: 1.1466 - val_accuracy: 0.7990
Epoch 26/50
16/16 [==============================] - 0s 31ms/step - loss: 0.0863 - accuracy: 0.9
657 - val_loss: 1.1847 - val_accuracy: 0.7980
Epoch 27/50
16/16 [==============================] - 1s 34ms/step - loss: 0.0888 - accuracy: 0.9
622 - val_loss: 1.2242 - val_accuracy: 0.8070
Epoch 28/50
16/16 [==============================] - 1s 33ms/step - loss: 0.0926 - accuracy: 0.9
583 - val_loss: 1.1509 - val_accuracy: 0.8090
```

손실과 정확도 곡선

In [156]:

```python
import matplotlib.pyplot as plt
```

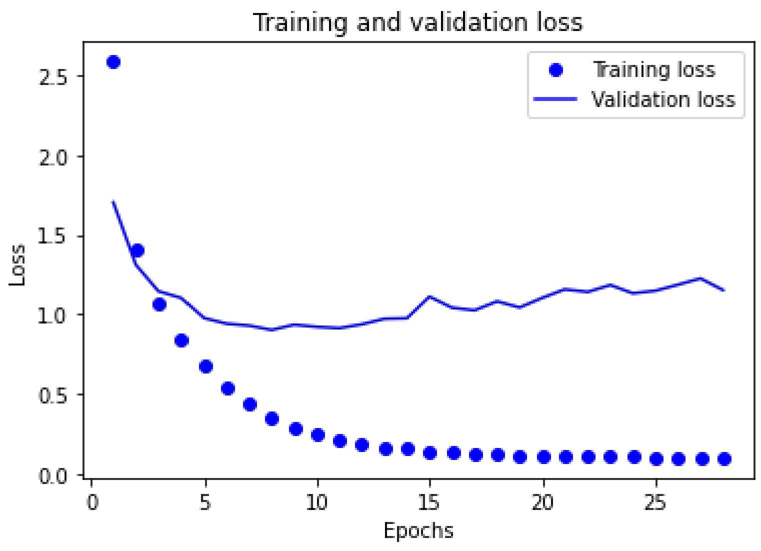
In [157]:

```
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')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



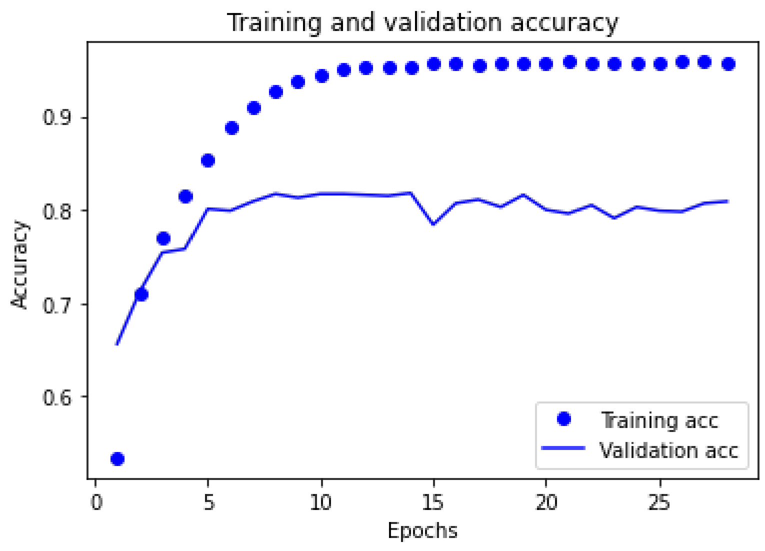
In [158]:

```
plt.clf()   # 그래프를 초기화합니다

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



9번째 에포크 이후에 과대적합 시작. 9번의 에포크로 새로운 모델 훈련과 테스트 세트에서 평가

In [159]:

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

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

model.fit(partial_x_train,
          partial_y_train,
          epochs=9,
          batch_size=512,
          validation_data=(x_val, y_val))

results = model.evaluate(x_test, one_hot_test_labels)
```

```
Epoch 1/9
16/16 [==============================] - 2s 48ms/step - loss: 3.0561 - accuracy: 0.3
845 - val_loss: 1.6750 - val_accuracy: 0.6470
Epoch 2/9
16/16 [==============================] - 1s 33ms/step - loss: 1.4413 - accuracy: 0.7
104 - val_loss: 1.2760 - val_accuracy: 0.7160
Epoch 3/9
16/16 [==============================] - 1s 33ms/step - loss: 1.0137 - accuracy: 0.7
884 - val_loss: 1.0943 - val_accuracy: 0.7730
Epoch 4/9
16/16 [==============================] - 1s 32ms/step - loss: 0.8205 - accuracy: 0.8
293 - val_loss: 1.0311 - val_accuracy: 0.7780
Epoch 5/9
16/16 [==============================] - 1s 36ms/step - loss: 0.6379 - accuracy: 0.8
687 - val_loss: 0.9730 - val_accuracy: 0.7890
Epoch 6/9
16/16 [==============================] - 1s 33ms/step - loss: 0.4955 - accuracy: 0.8
973 - val_loss: 0.9117 - val_accuracy: 0.8110
Epoch 7/9
16/16 [==============================] - 1s 33ms/step - loss: 0.4091 - accuracy: 0.9
150 - val_loss: 0.8921 - val_accuracy: 0.8170
Epoch 8/9
16/16 [==============================] - 1s 32ms/step - loss: 0.3380 - accuracy: 0.9
307 - val_loss: 0.9318 - val_accuracy: 0.8070
Epoch 9/9
16/16 [==============================] - 0s 30ms/step - loss: 0.2917 - accuracy: 0.9
354 - val_loss: 0.9500 - val_accuracy: 0.7950
71/71 [==============================] - 0s 3ms/step - loss: 1.0161 - accuracy: 0.77
96
```

In [160]:

```
results
```

Out[160]:

```
[1.016123652458191, 0.7796081900596619]
```

In [161]:

```
import copy

test_labels_copy = copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
float(np.sum(np.array(test_labels) == np.array(test_labels_copy))) / len(test_labels)
```

Out[161]:

```
0.19011576135351738
```

새로운 데이터로 예측

In [162]:

```
predictions = model.predict(x_test)
```

In [163]:

```
predictions[0].shape
```

Out[163]:

```
(46,)
```

In [164]:

```
np.sum(predictions[0])
```

Out[164]:

```
1.0000001
```

가장 큰 값이 예측 클래스가 된다.

In [165]:

```
np.argmax(predictions[0])
```

Out[165]:

```
3
```

정수 레이블을 그대로 사용할 때

In [166]:

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)

print(y_train.shape)
```

```
(8982,)
```

In [167]:

```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['acc'])
```

In [168]:

```
x_val = x_train[ :1000]
partial_x_train = x_train[1000:]

y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

In [169]:

```
partial_x_train.shape, partial_y_train.shape
```

Out[169]:

```
((7982, 10000), (7982,))
```

In [170]:

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

```
Epoch 1/20
16/16 [==============================] - 1s 47ms/step - loss: 0.2694 - acc: 0.9395
- val_loss: 0.9126 - val_acc: 0.8140
Epoch 2/20
16/16 [==============================] - 1s 33ms/step - loss: 0.1859 - acc: 0.9545
- val_loss: 0.9472 - val_acc: 0.8100
Epoch 3/20
16/16 [==============================] - 1s 32ms/step - loss: 0.1656 - acc: 0.9560
- val_loss: 0.9442 - val_acc: 0.8180
Epoch 4/20
16/16 [==============================] - 1s 32ms/step - loss: 0.1460 - acc: 0.9588
- val_loss: 0.9710 - val_acc: 0.8140
Epoch 5/20
16/16 [==============================] - 1s 33ms/step - loss: 0.1380 - acc: 0.9579
- val_loss: 1.0104 - val_acc: 0.8070
Epoch 6/20
16/16 [==============================] - 1s 32ms/step - loss: 0.1355 - acc: 0.9585
- val_loss: 1.0363 - val_acc: 0.8070
Epoch 7/20
16/16 [==============================] - 1s 31ms/step - loss: 0.1206 - acc: 0.9614
- val_loss: 1.0906 - val_acc: 0.7970
Epoch 8/20
16/16 [==============================] - 1s 32ms/step - loss: 0.1108 - acc: 0.9634
- val_loss: 1.0398 - val_acc: 0.8080
Epoch 9/20
16/16 [==============================] - 1s 32ms/step - loss: 0.1087 - acc: 0.9629
- val_loss: 1.0727 - val_acc: 0.8120
Epoch 10/20
16/16 [==============================] - 0s 31ms/step - loss: 0.1083 - acc: 0.9601
- val_loss: 1.0738 - val_acc: 0.8090
Epoch 11/20
16/16 [==============================] - 1s 32ms/step - loss: 0.0979 - acc: 0.9635
- val_loss: 1.0761 - val_acc: 0.7990
Epoch 12/20
16/16 [==============================] - 1s 32ms/step - loss: 0.0962 - acc: 0.9646
- val_loss: 1.0989 - val_acc: 0.8010
Epoch 13/20
16/16 [==============================] - 1s 33ms/step - loss: 0.0954 - acc: 0.9617
- val_loss: 1.0813 - val_acc: 0.8080
Epoch 14/20
16/16 [==============================] - 1s 33ms/step - loss: 0.0882 - acc: 0.9656
- val_loss: 1.0729 - val_acc: 0.8080
Epoch 15/20
16/16 [==============================] - 1s 32ms/step - loss: 0.0866 - acc: 0.9668
- val_loss: 1.1275 - val_acc: 0.8030
Epoch 16/20
16/16 [==============================] - 1s 35ms/step - loss: 0.0861 - acc: 0.9635
- val_loss: 1.1690 - val_acc: 0.7990
Epoch 17/20
16/16 [==============================] - 1s 33ms/step - loss: 0.0933 - acc: 0.9601
- val_loss: 1.2222 - val_acc: 0.7920
Epoch 18/20
```

```
16/16 [==============================] - 1s 32ms/step - loss: 0.0966 - acc: 0.9630
- val_loss: 1.1977 - val_acc: 0.8000
Epoch 19/20
16/16 [==============================] - 1s 32ms/step - loss: 0.0893 - acc: 0.9623
- val_loss: 1.1658 - val_acc: 0.8000
Epoch 20/20
16/16 [==============================] - 0s 30ms/step - loss: 0.0921 - acc: 0.9614
- val_loss: 1.1654 - val_acc: 0.8030
```

충분히 큰 중간층을 두기

In [175]:

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

In [176]:

```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [177]:

```
partial_x_train.shape, partial_y_train.shape
```

Out[177]:

```
((7982, 10000), (7982,))
```

In [178]:

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

```
Epoch 1/20
16/16 [==============================] - 2s 50ms/step - loss: 3.1570 - accuracy: 0.3
601 - val_loss: 1.6903 - val_accuracy: 0.5600
Epoch 2/20
16/16 [==============================] - 1s 34ms/step - loss: 1.5626 - accuracy: 0.6
098 - val_loss: 1.4705 - val_accuracy: 0.6570
Epoch 3/20
16/16 [==============================] - 1s 34ms/step - loss: 1.2729 - accuracy: 0.7
015 - val_loss: 1.3088 - val_accuracy: 0.6980
Epoch 4/20
16/16 [==============================] - 1s 34ms/step - loss: 1.0939 - accuracy: 0.7
301 - val_loss: 1.3052 - val_accuracy: 0.6930
Epoch 5/20
16/16 [==============================] - 1s 34ms/step - loss: 0.9641 - accuracy: 0.7
449 - val_loss: 1.2052 - val_accuracy: 0.7230
Epoch 6/20
16/16 [==============================] - 1s 34ms/step - loss: 0.8281 - accuracy: 0.7
939 - val_loss: 1.1829 - val_accuracy: 0.7360
Epoch 7/20
16/16 [==============================] - 1s 35ms/step - loss: 0.7417 - accuracy: 0.8
004 - val_loss: 1.2474 - val_accuracy: 0.7410
Epoch 8/20
16/16 [==============================] - 1s 35ms/step - loss: 0.6092 - accuracy: 0.8
455 - val_loss: 1.2330 - val_accuracy: 0.7500
Epoch 9/20
16/16 [==============================] - 1s 34ms/step - loss: 0.5599 - accuracy: 0.8
549 - val_loss: 1.2179 - val_accuracy: 0.7540
Epoch 10/20
16/16 [==============================] - 1s 35ms/step - loss: 0.4474 - accuracy: 0.8
868 - val_loss: 1.2634 - val_accuracy: 0.7520
Epoch 11/20
16/16 [==============================] - 1s 34ms/step - loss: 0.4141 - accuracy: 0.8
946 - val_loss: 1.5806 - val_accuracy: 0.7180
Epoch 12/20
16/16 [==============================] - 1s 35ms/step - loss: 0.4157 - accuracy: 0.8
972 - val_loss: 1.2615 - val_accuracy: 0.7610
Epoch 13/20
16/16 [==============================] - 1s 34ms/step - loss: 0.3258 - accuracy: 0.9
156 - val_loss: 1.4942 - val_accuracy: 0.7500
Epoch 14/20
16/16 [==============================] - 1s 35ms/step - loss: 0.3018 - accuracy: 0.9
209 - val_loss: 1.7386 - val_accuracy: 0.7360
Epoch 15/20
16/16 [==============================] - 1s 33ms/step - loss: 0.2953 - accuracy: 0.9
206 - val_loss: 1.3460 - val_accuracy: 0.7740
Epoch 16/20
16/16 [==============================] - 1s 35ms/step - loss: 0.1985 - accuracy: 0.9
503 - val_loss: 1.6846 - val_accuracy: 0.7530
Epoch 17/20
16/16 [==============================] - 1s 37ms/step - loss: 0.1967 - accuracy: 0.9
484 - val_loss: 2.1876 - val_accuracy: 0.7160
Epoch 18/20
16/16 [==============================] - 1s 35ms/step - loss: 0.2104 - accuracy: 0.9
```

```
402 - val_loss: 1.7678 - val_accuracy: 0.7560
Epoch 19/20
16/16 [==============================] - 1s 34ms/step - loss: 0.1788 - accuracy: 0.9
564 - val_loss: 1.5196 - val_accuracy: 0.7630
Epoch 20/20
16/16 [==============================] - 1s 32ms/step - loss: 0.1525 - accuracy: 0.9
551 - val_loss: 1.7107 - val_accuracy: 0.7620
```

층을 줄일 때,

In [181]:

```
model = models.Sequential()
model.add(layers.Dense(8, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

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

```
Epoch 1/20
16/16 [==============================] - 1s 33ms/step - loss: 3.6391 - accuracy: 0.1
660 - val_loss: 3.2453 - val_accuracy: 0.3530
Epoch 2/20
16/16 [==============================] - 0s 19ms/step - loss: 3.1270 - accuracy: 0.3
471 - val_loss: 2.8692 - val_accuracy: 0.3540
Epoch 3/20
16/16 [==============================] - 0s 19ms/step - loss: 2.7655 - accuracy: 0.3
465 - val_loss: 2.5852 - val_accuracy: 0.3540
Epoch 4/20
16/16 [==============================] - 0s 21ms/step - loss: 2.4857 - accuracy: 0.3
517 - val_loss: 2.3768 - val_accuracy: 0.3540
Epoch 5/20
16/16 [==============================] - 0s 20ms/step - loss: 2.2914 - accuracy: 0.3
576 - val_loss: 2.2113 - val_accuracy: 0.3610
Epoch 6/20
16/16 [==============================] - 0s 19ms/step - loss: 2.1186 - accuracy: 0.3
780 - val_loss: 2.0672 - val_accuracy: 0.3790
Epoch 7/20
16/16 [==============================] - 0s 20ms/step - loss: 1.9702 - accuracy: 0.3
838 - val_loss: 1.9389 - val_accuracy: 0.3850
Epoch 8/20
16/16 [==============================] - 0s 20ms/step - loss: 1.8238 - accuracy: 0.3
935 - val_loss: 1.8209 - val_accuracy: 0.4060
Epoch 9/20
16/16 [==============================] - 0s 20ms/step - loss: 1.7081 - accuracy: 0.4
671 - val_loss: 1.7162 - val_accuracy: 0.5490
Epoch 10/20
16/16 [==============================] - 0s 21ms/step - loss: 1.5858 - accuracy: 0.5
885 - val_loss: 1.6316 - val_accuracy: 0.5940
Epoch 11/20
16/16 [==============================] - 0s 22ms/step - loss: 1.4689 - accuracy: 0.6
304 - val_loss: 1.5663 - val_accuracy: 0.6130
Epoch 12/20
16/16 [==============================] - 0s 21ms/step - loss: 1.4515 - accuracy: 0.6
485 - val_loss: 1.5140 - val_accuracy: 0.6200
Epoch 13/20
16/16 [==============================] - 0s 20ms/step - loss: 1.3897 - accuracy: 0.6
601 - val_loss: 1.4700 - val_accuracy: 0.6310
Epoch 14/20
16/16 [==============================] - 0s 20ms/step - loss: 1.3177 - accuracy: 0.6
656 - val_loss: 1.4316 - val_accuracy: 0.6370
Epoch 15/20
16/16 [==============================] - 0s 20ms/step - loss: 1.2553 - accuracy: 0.6
```

```
737 - val_loss: 1.4023 - val_accuracy: 0.6440
Epoch 16/20
16/16 [==============================] - 0s 21ms/step - loss: 1.2015 - accuracy: 0.6
775 - val_loss: 1.3738 - val_accuracy: 0.6560
Epoch 17/20
16/16 [==============================] - 0s 20ms/step - loss: 1.1780 - accuracy: 0.6
937 - val_loss: 1.3456 - val_accuracy: 0.6650
Epoch 18/20
16/16 [==============================] - 0s 20ms/step - loss: 1.1227 - accuracy: 0.7
137 - val_loss: 1.3274 - val_accuracy: 0.6760
Epoch 19/20
16/16 [==============================] - 0s 21ms/step - loss: 1.1090 - accuracy: 0.7
182 - val_loss: 1.3085 - val_accuracy: 0.6790
Epoch 20/20
16/16 [==============================] - 0s 18ms/step - loss: 1.0654 - accuracy: 0.7
221 - val_loss: 1.2902 - val_accuracy: 0.6850
```

검증 정확도가 감소

- 검증 정확도의 약간 감소

- 추가 실험해보기
  - 더 크거나 작은 층을 사용해 보세요: 32개 유닛, 128개 유닛 등
  - 여기에서 두 개의 은닉층을 사용했습니다. 한 개의 은닉층이나 세 개의 은닉층을 사용해 보세요.

Summary

- 단일 레이블, 다중 분류 문제에서는 N개의 클래스에 대한 확률 분포를 출력하기 위해 softmax 활성화 함수를 사용
- 항상 범주형 크로스엔트로피를 사용
  - 이 함수는 모델이 출력한 확률 분포와 타깃 분포 사이의 거리를 최소화
- 다중 분류에서 레이블을 다루는 두가지 방법
  - 레이블을 범주형 인코딩(또는 원-핫 인코딩)으로 인코딩하고 categorical_crossentropy 손실 함수를 사용
  - 레이블을 정수로 인코딩하고 sparse_categorical_crossentropy 손실 함수를 사용