영화 리뷰 분류 : 이진 분류 예제

가장 널리 적용된 머신 러닝 문제이다.

리뷰 텍스트를 기반으로 영화 리뷰를 긍정과 부정으로 분류하는 방법을 배워보자.

IMDB 데이터셋

5만개의 IMDB 데이터셋을 사용, 2만5000개는 훈련데이터, 2만5000개는 테스트 데이터. 50%은 긍정, 50%은 부정

IMDB 데이터셋 로드하기

```
from keras.datasets import imdb

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

num_words=10000는 훈련데이터에서 가장 자주 나타나는 단어 1만개만 사용하겠다는 의미이다. 드물게 나타나는 단어는 무시한다. 변수 train_data와 test_data는 리뷰의 목록이다. 각 리뷰는 단어 인덱스의 리스트이다. train_labels와 test_labels는 부정을 나타내는 0과 긍정을 나타내는 1의 리스트이다.

데이터 준비

신경망에 숫자 리스트를 주입할 수는 없다. 리스트를 텐서로 바꾸어야 한다. 여기서는 리스트를 원-핫 인코딩을 하여 0과 1의 벡터로 변환해보자. 예를 들어 시퀀스[3, 5]를 인덱스 3과 5의 위치는 1이고 그 외는 모두 0인 10,000 차원의 벡터로 각각 변환한다. 그 다음 부동 소수 벡터 데이터를 다룰 수 있는 Dense 층을 신경망의 첫번째 층으로 사용한다.

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
  results = np.zeros((len(sequences), dimension)) #크기가 len(sequences), dimension 이고 모든 원소가 0인 행렬을 만든다.
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1. #results[i]에서 특정 인덱스의 위치를 1로 만든다.
  return results

x_train = vectorize_sequences(train_data) #훈련데이터를 벡터로 변환
x_test = vectorize_sequences(test_data) #테스트 데이터를 벡터로 변환
```

신경망 모델 만들기

입력 데이터가 벡터고, 레이블은 스칼라이다. 이런 문제에 잘 작동하는 네트워크 종류는 relu활성화 함수를 사용한 완전 연결 층을 그냥 쌓는 것이다.

Dense 층에 전달한 매개변수(16)은 은닉 유닛의 개수이다. 하나의 은닉 유닛은 층이 나타내는 표현 공간에서 하나의 차원이 된다.

16개의 은닉 유닛이 있다는 것은 가중치 행렬 W의 크기가 (input_dimension, 16)이라는 뜻이다. 입력 데이터와 W를 점곱하면 입력 데이터가 16차원으로 표현된 공간으로 투영된다. 표현 공간의 차원을 '신경망이 내재된 표현을 학습할 때 가질 수 있는 자유도'로 이해할 수 있다.

Dense층을 쌓을 때 두 가지 중요한 구조상의 결정이 필요하다.

- 얼마나 많은 층을 사용할 것인가?
- 각 층에 얼마나 많은 은닉 유닛을 둘 것인가?

이번에는

- 16개의 은닉 유닛을 가진 2개의 은닉 층
- 현재 리뷰의 감정을 스칼라 값의 예측으로 출력하는 세 번째 층

으로 구조를 만들 것이다.

모델정의하기

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
          input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

마지막으로는 손실 함수와 옵티마이저를 선택해야 한다. 이진 분류 문제고 신경망의 출력이 확률이기 때문에 binary_crossentropy 손실이 적합하다.

확률을 출력하는 모델을 사용할 때는 크로스엔트로피가 최선의 선택이다.

크로스엔트로피는 정보 이론 분야에서 온 개념으로 확률 분포 간의 차이를 측정한다. 여기서는 원본 분포와 예측 분포 사이를 측정한다

모델 컴파일하기

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

옵티마이저 설정하기

```
from keras import optimizers
```

```
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

손실과 측정을 함수 객체로 지정하기

```
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

훈련 검증

훈련하는 동안 처음 본 데이터에 대한 모델의 정확도를 측정하기 위해서는 원본 훈련 데이터에서 10,000의 샘플을 떼어 검증 세트를 만들어야 한다.

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

이제 모델을 512개의 샘플씩 미니 배치를 만들어 20번의 에포크 동안 훈련시킨다. 동시에 따로 떼어 놓은 1만개의 샘플에서 손실과 정확도를 측정한다. 이렇게 하기 위해서는 validation_data매개변수에 검증 데이터를 전달해야 한다.

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

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

model.fit() 메서드는 History 객체를 반환한다. 이 객체는 훈련하는 동안 발생한 모든 정보를 담고 있는 딕셔너리인 history속성을 가지고 있다.

>>> history_dict = history.history

>>> history_dict.keys()

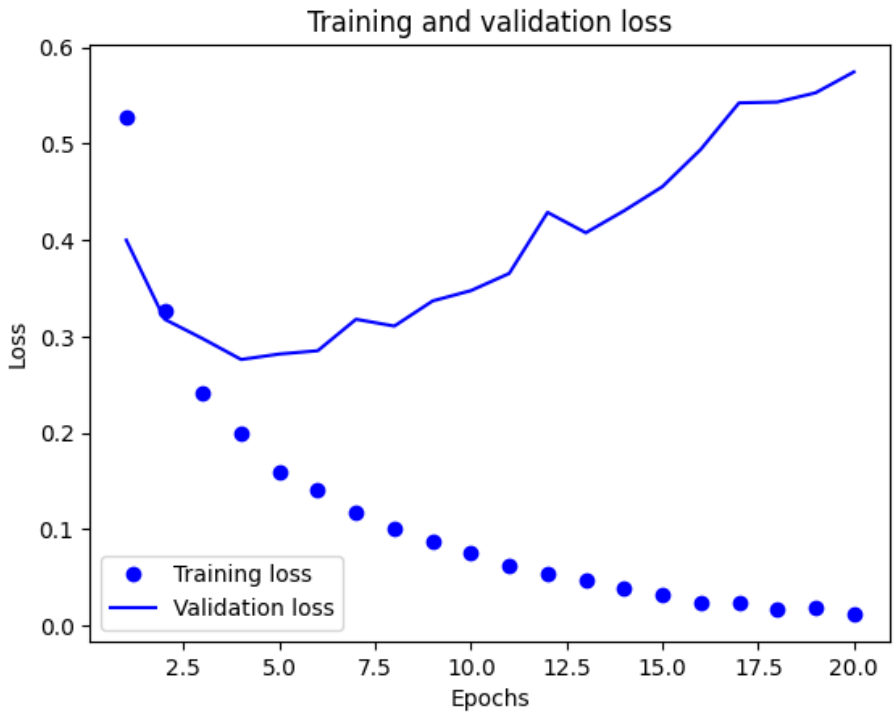
[u'acc', u'loss', u'val_acc', u'val_loss']

이 딕셔너리는 훈련과 검증하는 동안 모니터링할 측정 지표당 하나씩 모두 4개의 항목을 담고 있다.

훈련과 검증 손실 그리기

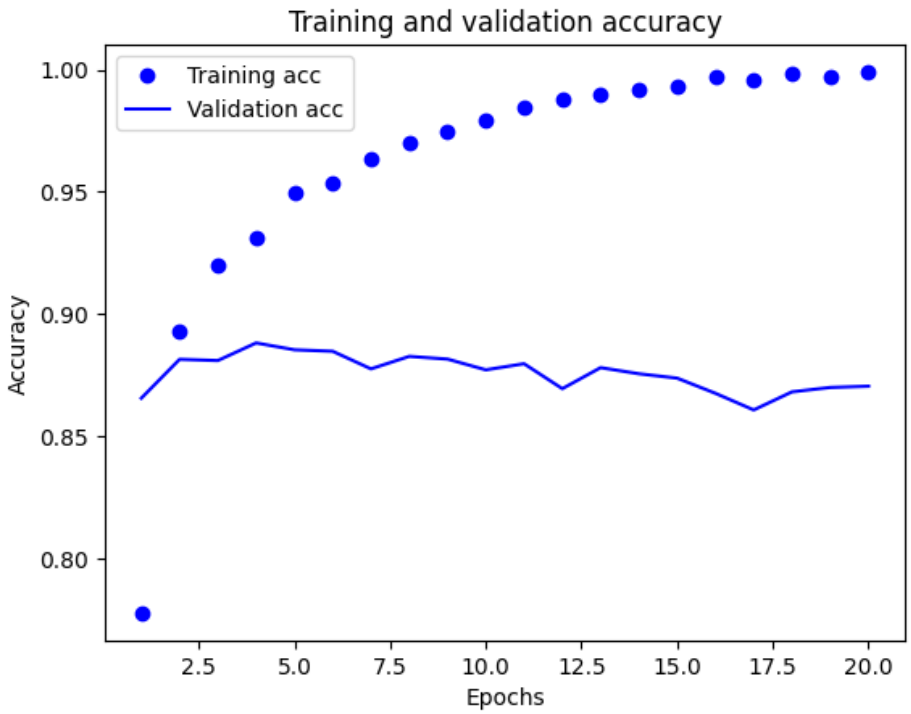
```
import matplotlib.pyplot as plt

history_dict = history.history
loss = history_dict['loss']
```

```
val_loss = history_dict['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss') #'bo'는 파란색 점을 의미한다.
plt.plot(epochs, val_loss, 'b', label='Validation loss') #'b'는 파란색 실선을 의미한다.
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



훈련과 검증 정확도 그리기

```
plt.clf() #그래프 초기화
acc = history_dict['acc']
val_acc = history_dict['val_acc']

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()
```



훈련 손실이 에포크마다 감소하고 훈련 정확도는 에포크마다 증가한다. 네 번째 에포크에서 그래프가 역전되는 것 같다. 이것은 과대적합되었다. 두 번째 에포크 이후부터 훈련데이터에 과도하게 최적화되어 훈련 데이터에 특화된 표현을 학습하므로 훈련 세트 이외의 데이터에는 일반화되지 못한다.

모델을 처음부터 다시 훈련하기

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
                       input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(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)
results = model.evaluate(x_test, y_test)

>>>result
[[0.2853509485721588, 0.8863599896430969]]
```

훈련된 모델로 새로운 데이터에 대해 예측하기

predict 메서드를 사용해서 어떤 리뷰가 긍정일 확률을 예측할 수 있다.

```
>>> model.predict(x_test)
array([[0.1814169 ],
       [0.99974245],
       [0.66858053],
```

```
       ...,
       [0.09067097],
       [0.06671336],
       [0.48276353]], dtype=float32)
```

여기처럼 이 모델은 어떤 샘플에 대해 확신을 가지고 있지만, 어떤 샘플에 대해서는 확신이 부족하다.

정리

- 원본 데이터를 신경망에 텐서로 주입하기 위해서는 꽤 많은 전처리가 필요하다. 단어 시퀀스는 이진 벡터로 인코딩될 수 있고 다른 인코딩 방식도 있다.
- relu 활성화 함수와 함께 Dense 층을 쌓은 네트워크는 여러 종류의 문제에 적용할 수 있어 앞으로 자주 사용하게 될 것이다.
- 이진 분류 문제에서 네트워크는 하나의 유닛과 sigmoid 활성화 함수를 가진 Dense 층으로 끝나야 한다. 신경망의 출력은 확률을 나타내는 0과 1 사이의 스칼라 값이다.
- 이진 분류 문제에서 이런 스칼라 시그모이드 출력에 대해 사용할 손실 함수는 binary_crossentropy이다.
- rmsprop 옵티마이저는 문제에 상관없이 일반적으로 충분히 좋은 선택이다. 걱정할 거리가 하나 줄은 셈이다.
- 훈련 데이터에 대해 성능이 향상됨에 따라 신경망은 과대적합되기 시작하고 이전에 본적 없는 데이터에서는 결과가 점점 나빠지게 된다. 항상 훈련 세트 이외의 데이터에서 성능을 모니터링해야한다.