순환 신경망 이해하기 - LSTM

학습 내용

- LSTM에 대해 알아보기

LSTM 층으로 모델을 구성하고 IMDB 데이터에서 훈련

- 케라스에서는 SimpleRNN외에 다른 순환 층도 있다. LSTM과 GRU 2개입니다.
- LSTM 층은 출력 차원만 지정하고 다른 (많은) 매개변수는 케라스의 기본값
- 케라스는 좋은 기본값을 가지고 있어서 직접 매개변수를 튜닝하는 데 시간을 쓰지 않고도 거의 항상 어느정도 작동하는 모델 획득 가능

In [1]:

```
from keras.layers import LSTM
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN
from keras.layers import Dense
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
```

In [2]:

```
%%time

max_features = 10000  # 특성으로 사용할 단어의 수
maxlen = 500  # 사용할 텍스트의 길이(가장 빈번한 max_features 개의 단어만 사용합니다)
batch_size = 32

(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
print(len(input_train), '훈련 시퀀스')
print(len(input_test), '테스트 시퀀스')

print('시퀀스 패딩 (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train 크기:', input_train.shape)
print('input_test 크기:', input_test.shape)
```

```
<__array_function__ internals>:5: VisibleDeprecationWarning: Creating an ndarray fro
m ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays w
ith different lengths or shapes) is deprecated. If you meant to do this, you must sp
ecify 'dtype=object' when creating the ndarray
c:\Users\Toto\Anaconda3\Envs\Tf20\Lib\site-packages\Tensorflow\Python\Keras\Datasets
\Imdb.py:159: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequ
ences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different length
s or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object'
when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
c:\Users\Toto\Anaconda3\Envs\Tf20\Lib\site-packages\Tensorflow\Python\Keras\Datasets
\Imdb.py:160: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequ
ences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different length
s or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object'
when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])

25000 훈련 시퀀스
25000 테스트 시퀀스
시퀀스 패딩 (samples x time)
input_train 크기: (25000, 500)
input_test 크기: (25000, 500)
Wall time: 6.04 s
```

LSTM과 GRU층 이해하기

- simpleRNN은 이론적으로 시간 t에서 이전의 모든 타임스텝의 정보를 유지할 수 있다.
- 실제로 긴 시간에 걸친 의존성은 학습할 수 없는 것이 문제.
  - 층이 많은 일반 네트워크(피드포워드 네트워크)에서 나타나는 것과 비슷한 현상인 그래디언트 소실 문제(vanishing gradient problem)때문.
  - 1990년대 초 호크라이터(Hochreiter), 슈미트후버(Schmidhuber), 벤지오(Bengio)가 이런 현상에 대한 원인을 연구.
    - 이 문제를 해결하기 위해 고안된 것은 LSTM과 GRU층이다.
- 이것은 SimpleRNN의 한 변종이다. 정보를 여러 타임스텝에 걸쳐 나르는 방법이 추가.

In [3]:

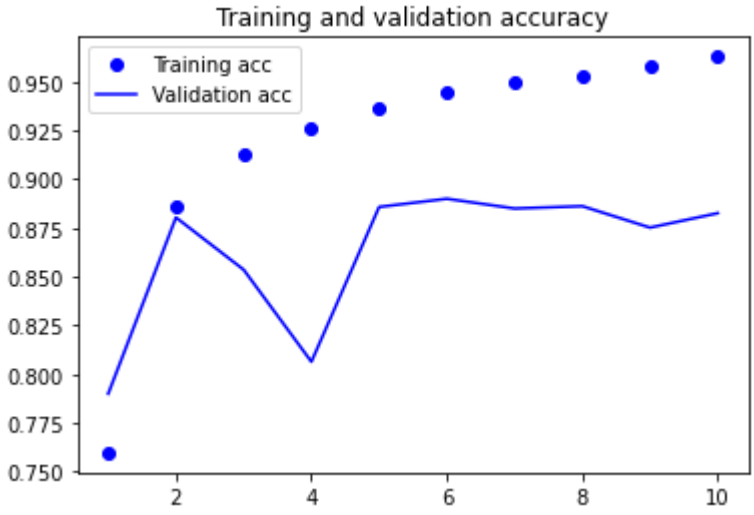
```
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

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

history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

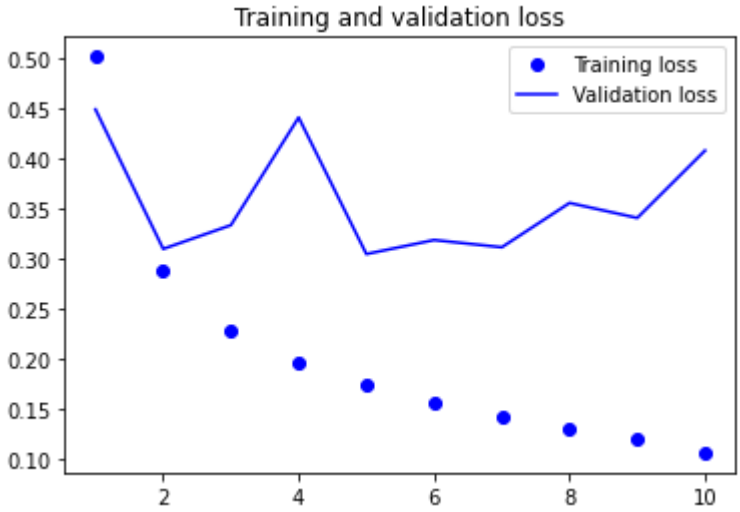
```
Epoch 1/10
157/157 [==============================] - 53s 322ms/step - loss: 0.6065 - acc: 0.66
29 - val_loss: 0.4496 - val_acc: 0.7900
Epoch 2/10
157/157 [==============================] - 47s 299ms/step - loss: 0.3077 - acc: 0.87
86 - val_loss: 0.3103 - val_acc: 0.8804
Epoch 3/10
157/157 [==============================] - 47s 301ms/step - loss: 0.2309 - acc: 0.91
32 - val_loss: 0.3341 - val_acc: 0.8536
Epoch 4/10
157/157 [==============================] - 48s 304ms/step - loss: 0.1967 - acc: 0.92
64 - val_loss: 0.4416 - val_acc: 0.8064
Epoch 5/10
157/157 [==============================] - 48s 306ms/step - loss: 0.1667 - acc: 0.93
84 - val_loss: 0.3052 - val_acc: 0.8858
Epoch 6/10
157/157 [==============================] - 48s 303ms/step - loss: 0.1537 - acc: 0.94
80 - val_loss: 0.3192 - val_acc: 0.8900
Epoch 7/10
157/157 [==============================] - 47s 301ms/step - loss: 0.1361 - acc: 0.95
35 - val_loss: 0.3120 - val_acc: 0.8850
Epoch 8/10
157/157 [==============================] - 48s 306ms/step - loss: 0.1196 - acc: 0.95
94 - val_loss: 0.3563 - val_acc: 0.8862
Epoch 9/10
157/157 [==============================] - 48s 304ms/step - loss: 0.1192 - acc: 0.95
85 - val_loss: 0.3413 - val_acc: 0.8752
Epoch 10/10
157/157 [==============================] - 48s 304ms/step - loss: 0.1026 - acc: 0.96
51 - val_loss: 0.4085 - val_acc: 0.8826
```

검증의 손실과 정확도

In [4]:

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

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

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





결과 확인

- 좀 더 높은 검증 정확도 달성.
- SimpleRNN 네트워크보다 더 낫다. LSTM이 그래디언트 소실 문제로부터 덜 영향을 받기 때문.
- 그렇다고 반드시 LSTM이 모든 분야에 나은 것은 아니다.
  - LSTM이 더 높은 성능을 보이는 곳은 자연어 처리문제이다.
    - 질문-응답(question-answering)과 기계 번역(machine translation)

- (실습) 평가지표 (loss, AI 등)