순환 신경망 이해하기

학습 내용

- RNN에 대해 실습을 통해 알아본다.
    - 데이터 셋 : IMDB 영화 리뷰 분류 문제 적용

In [2]:

```
import keras
keras.__version__
```

Out[2]:

```
'2.4.3'
```

케라스의 순환 층

In [3]:

```
from keras.layers import SimpleRNN
```

- SimpleRNN이 한 가지 다른 점은 넘파이 예제처럼 하나의 시퀀스가 아니다.
- 다른 케라스 층과 마찬가지로 시퀀스 배치를 처리
- (timesteps, input_features) 크기 아니다.
- (batch_size, timesteps, input_features) 크기의 입력

SimpleRNN은 두가지 모드로 실행

- (batch_size, timesteps, output_features)인 3D 텐서
    - 이 모드는 return_sequences 매개변수 선택(True)
- (batch_size, output_features)인 2D 텐서
    - 이 모드는 return_sequences 매개변수 선택(False)

In [4]:

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN

model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          320000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, None, 32)          2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

In [6]:

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32)) # 맨 위 층만 마지막 출력을 반환합니다.
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_4 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
_________________________________________________________________
```

IMDB 영화 리뷰 분류 문제 적용

In [7]:

```
from keras.datasets import imdb
from keras.preprocessing import sequence

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

print('데이터 로딩...')
(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')

# 문장에서 maxlen 이후의 있는 단어들을 pad_sequences()함수로 잘라낸다.
# 문장 길이가 maxlen보다 작으면 부족한 부분을 0으로 채웁니다.
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)
```

```
데이터 로딩...
```

```
<__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)
```

- Embedding 층과 SimpleRNN 층을 사용해 간단한 순환 네트워크를 훈련

In [8]:

```
%%time

from keras.layers import Dense

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(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 [==============================] - 22s 133ms/step - loss: 0.6518 - acc: 0.59
91 - val_loss: 0.4440 - val_acc: 0.8054
Epoch 2/10
157/157 [==============================] - 20s 125ms/step - loss: 0.3903 - acc: 0.83
95 - val_loss: 0.3797 - val_acc: 0.8318
Epoch 3/10
157/157 [==============================] - 20s 126ms/step - loss: 0.2895 - acc: 0.88
69 - val_loss: 0.3588 - val_acc: 0.8624
Epoch 4/10
157/157 [==============================] - 19s 123ms/step - loss: 0.2454 - acc: 0.90
90 - val_loss: 0.4394 - val_acc: 0.8284
Epoch 5/10
157/157 [==============================] - 20s 127ms/step - loss: 0.2062 - acc: 0.92
39 - val_loss: 0.3867 - val_acc: 0.8444
Epoch 6/10
157/157 [==============================] - 20s 126ms/step - loss: 0.1626 - acc: 0.94
20 - val_loss: 0.3876 - val_acc: 0.8664
Epoch 7/10
157/157 [==============================] - 19s 124ms/step - loss: 0.1263 - acc: 0.95
51 - val_loss: 0.4124 - val_acc: 0.8596
Epoch 8/10
157/157 [==============================] - 21s 132ms/step - loss: 0.1006 - acc: 0.96
74 - val_loss: 0.4559 - val_acc: 0.8370
Epoch 9/10
157/157 [==============================] - 20s 130ms/step - loss: 0.0767 - acc: 0.97
43 - val_loss: 0.4995 - val_acc: 0.8304
Epoch 10/10
157/157 [==============================] - 20s 128ms/step - loss: 0.0537 - acc: 0.98
32 - val_loss: 0.5145 - val_acc: 0.8448
Wall time: 3min 21s
```

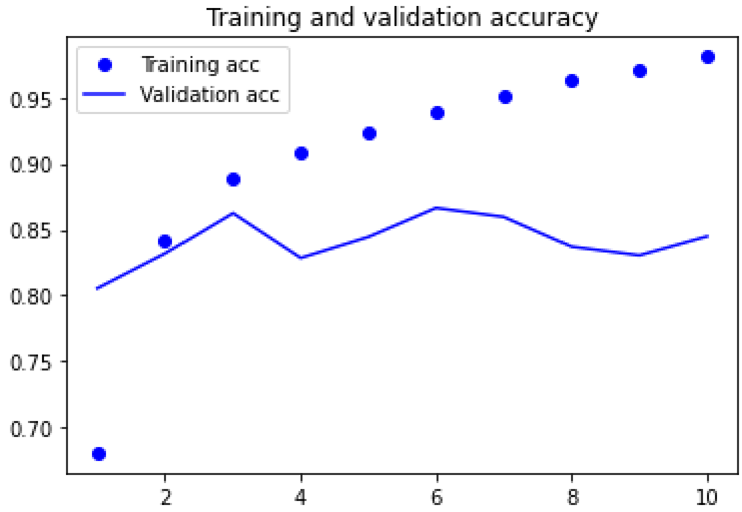
- 훈련과 검증의 손실과 정확도를 그래프

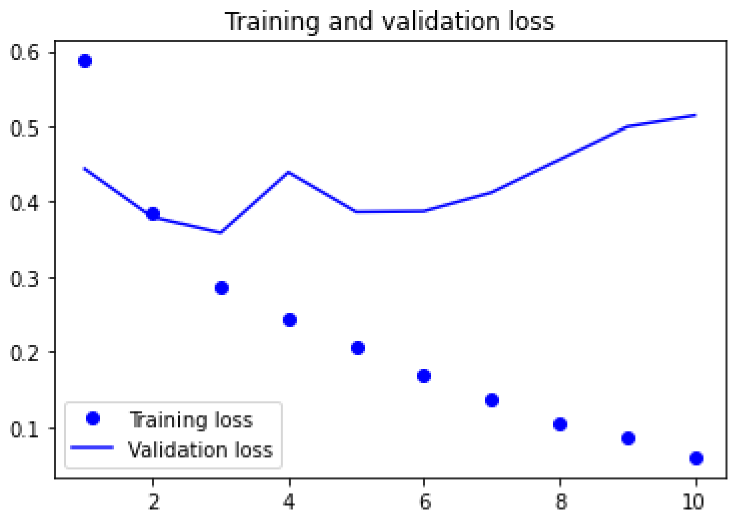
In [9]:

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

In [10]:

```
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()
```





- 간단한 순환 네트워크는 이 기준 모델보다 성능이 높지 않다.(85%의 정도의 검증 정확도를 얻음)
  - 예상 원인 : 처음 500개의 단어만 입력에 사용했기 때문.