순환 신경망 이해하기

학습 내용

- RNN에 대해 실습을 통해 알아본다.

In [1]:

```
import keras
keras.__version__
```

Out[1]:

```
'2.4.3'
```

케라스의 순환 층

In [2]:

```
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 [3]:

```
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 [4]:

```
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 [5]:

```
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 [6]:

```
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...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/im
db.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz)
17465344/17464789 [==============================] - 1s 0us/step
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 [7]:

```
%%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 [==============================] - 73s 467ms/step - loss: 0.6267 - acc: 0.63
40 - val_loss: 0.4717 - val_acc: 0.7978
Epoch 2/10
157/157 [==============================] - 73s 464ms/step - loss: 0.4335 - acc: 0.82
02 - val_loss: 0.3731 - val_acc: 0.8440
Epoch 3/10
157/157 [==============================] - 73s 464ms/step - loss: 0.3084 - acc: 0.87
70 - val_loss: 0.3658 - val_acc: 0.8436
Epoch 4/10
157/157 [==============================] - 73s 465ms/step - loss: 0.2441 - acc: 0.90
64 - val_loss: 0.4101 - val_acc: 0.8208
Epoch 5/10
157/157 [==============================] - 72s 461ms/step - loss: 0.2015 - acc: 0.92
50 - val_loss: 0.3465 - val_acc: 0.8616
Epoch 6/10
157/157 [==============================] - 73s 463ms/step - loss: 0.1622 - acc: 0.94
07 - val_loss: 0.3709 - val_acc: 0.8498
Epoch 7/10
157/157 [==============================] - 73s 464ms/step - loss: 0.1246 - acc: 0.95
60 - val_loss: 0.4679 - val_acc: 0.7950
Epoch 8/10
157/157 [==============================] - 74s 468ms/step - loss: 0.0859 - acc: 0.97
22 - val_loss: 0.4657 - val_acc: 0.8330
Epoch 9/10
157/157 [==============================] - 73s 464ms/step - loss: 0.0568 - acc: 0.98
11 - val_loss: 0.6012 - val_acc: 0.7798
Epoch 10/10
157/157 [==============================] - 73s 465ms/step - loss: 0.0439 - acc: 0.98
66 - val_loss: 0.5320 - val_acc: 0.8342
CPU times: user 18min 1s, sys: 2min 3s, total: 20min 4s
Wall time: 12min 16s
```

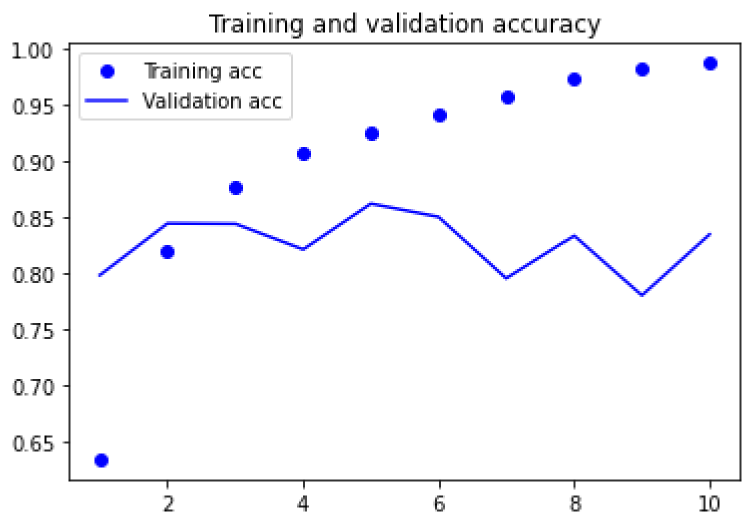
- 훈련과 검증의 손실과 정확도를 그래프

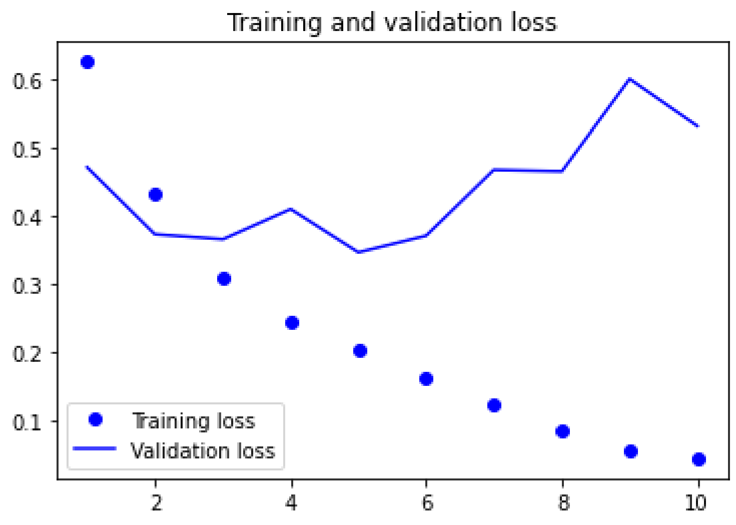
In [8]:

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

In [9]:

```
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()
```





In [ ]:

```

```