순환 신경망 이해하기

학습 내용

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

In [1]:

```
import keras
from keras.layers import SimpleRNN

print( keras.__version__ )
```

2.7.0

- SimpleRNN이 하나의 시퀀스가 아니다.
- 다른 케라스 층과 마찬가지로 시퀀스 배치를 처리
  - (timesteps, input_features) 크기 아니다.
  - (batch_size(샘플크기), timesteps, input_features) 크기의 입력

SimpleRNN 두가지 실행 모드

- return_sequences : 기본값(False)
  - False : 마지막 상태만 출력(Many-to-One)
  - True : 모든 지점의 은닉 상태 출력 (Many-to-Many)

In [3]:

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

Embedding층의 입력

- (samples, 임베딩차원, sequence_length)
  - samples : 샘플수
  - sequence_length : 시퀀스 길이 (단순히 길이)
  - 정수 텐서를 입력으로 받음. 2D텐서
- 여기서 sequence_length가 작은 길이의 시퀀스는 **0으로 패딩**되고, **긴 시퀀스는 잘리게** 됩니다.

Embedding층의 출력

- (samples, sqeuence_length, 임베딩 차원)
  - samples : 샘플수
  - sequence_length : 시퀀스 길이
  - embedding_dimensionality : 임베딩 차원
  - 출력은 3D 텐서가 된다.

return_sequences = False

In [4]:

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, None, 32)          320000

 simple_rnn_1 (SimpleRNN)     (None, 32)                2080

=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

파라미터 개수

- 32 * (32 + 32 + 1) = 2080

return_sequences = True

In [5]:

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

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

=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

은닉 상태 출력

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_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_3 (Embedding)      (None, None, 32)          320000

 simple_rnn_3 (SimpleRNN)     (None, None, 32)          2080

 simple_rnn_4 (SimpleRNN)     (None, None, 32)          2080

 simple_rnn_5 (SimpleRNN)     (None, None, 32)          2080

 simple_rnn_6 (SimpleRNN)     (None, 32)                2080

=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
_________________________________________________________________
```

실습 : IMDB 영화 리뷰 분류 문제 풀기

In [8]:

```
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')
```

```
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 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
25000 훈련 시퀀스
25000 테스트 시퀀스
```

In [9]:

```
# 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)
```

```
시퀀스 패딩 (samples x time)
input_train 크기: (25000, 500)
input_test 크기: (25000, 500)
```

모델 구축

In [11]:

```
from keras.layers import Dense

model = Sequential()
model.add(Embedding(max_features, 32)) # max_features = 10000
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

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

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, None, 32)          320000

 simple_rnn_8 (SimpleRNN)    (None, 32)                2080

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

```
%%time

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

```
Epoch 1/10
157/157 [==============================] - 27s 164ms/step - loss: 0.6057 - acc: 0.66
29 - val_loss: 0.5027 - val_acc: 0.7732
Epoch 2/10
157/157 [==============================] - 25s 162ms/step - loss: 0.3806 - acc: 0.84
12 - val_loss: 0.3812 - val_acc: 0.8382
Epoch 3/10
157/157 [==============================] - 26s 163ms/step - loss: 0.2923 - acc: 0.88
53 - val_loss: 0.3400 - val_acc: 0.8602
Epoch 4/10
157/157 [==============================] - 26s 165ms/step - loss: 0.2210 - acc: 0.91
71 - val_loss: 0.4086 - val_acc: 0.8176
Epoch 5/10
157/157 [==============================] - 26s 164ms/step - loss: 0.1739 - acc: 0.93
79 - val_loss: 0.5124 - val_acc: 0.8146
Epoch 6/10
157/157 [==============================] - 26s 164ms/step - loss: 0.1272 - acc: 0.95
54 - val_loss: 0.4040 - val_acc: 0.8628
Epoch 7/10
157/157 [==============================] - 26s 164ms/step - loss: 0.0919 - acc: 0.96
93 - val_loss: 0.4030 - val_acc: 0.8576
Epoch 8/10
157/157 [==============================] - 26s 164ms/step - loss: 0.0635 - acc: 0.98
00 - val_loss: 0.4503 - val_acc: 0.8466
Epoch 9/10
157/157 [==============================] - 26s 164ms/step - loss: 0.0430 - acc: 0.98
76 - val_loss: 0.4926 - val_acc: 0.8440
Epoch 10/10
157/157 [==============================] - 26s 165ms/step - loss: 0.0303 - acc: 0.99
09 - val_loss: 0.5428 - val_acc: 0.8532
CPU times: user 7min 16s, sys: 23.4 s, total: 7min 39s
Wall time: 4min 23s
```

- 검증 정확도 : 85%

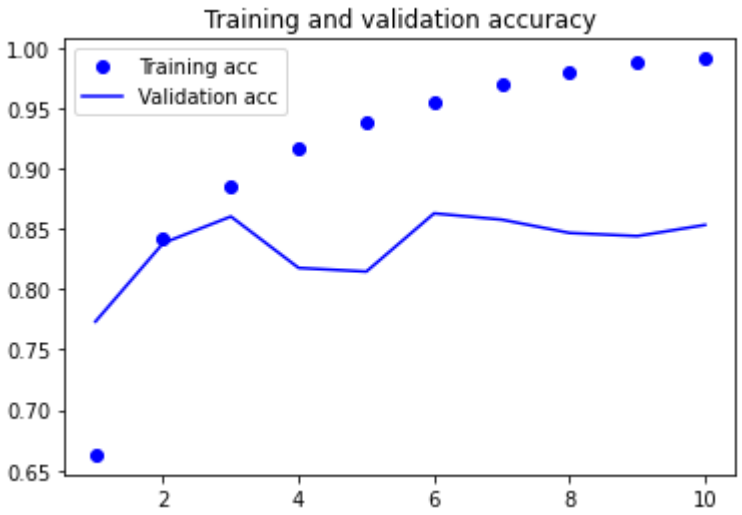
훈련, 검증의 손실과 정확도 그래프 확인

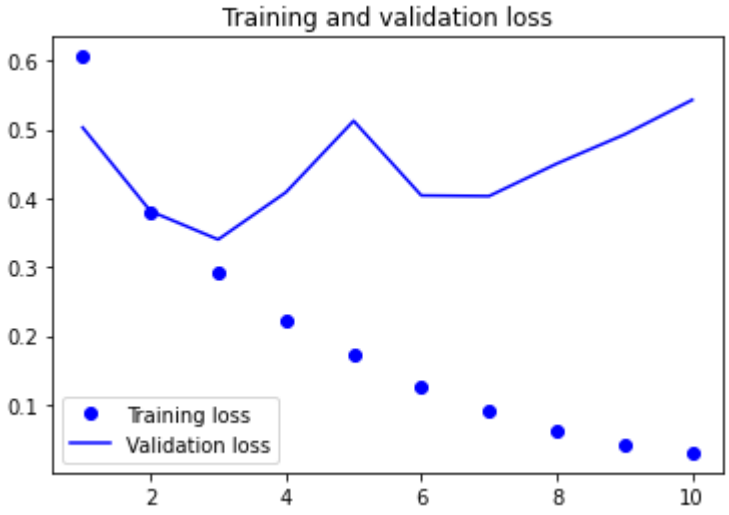
In [15]:

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

In [16]:

```
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)
  - 예상되는 원인 : 처음 500개의 단어만 입력에 사용했기 때문.