순환 신경망 이해하기

학습 내용

- 데이터 셋 : IMDB 영화 리뷰 분류 문제 적용

목차

01. SimpleRNN 기본 이해

목차로 이동하기

In [1]:

```
import keras
from keras.layers import SimpleRNN

print( keras.__version__ )
```

2.9.0

- SimpleRNN은 하나의 시퀀스가 아니라 시퀀스 배치를 처리한다는 것.
  - (timesteps, input_features) -> (batch_size, timesteps, input_features) 입력받음.
- SimpleRNN은 두 가지 실행 모드가 가능(return_sequences 매개변수로 선택 가능)
  - return_sequences= True
    - 각 타임스텝 출력을 모은 전체 시퀀스 반환 - (batch_size, timesteps, output_features) - 3D 텐서
  - return_sequences= False
    - 입력 시퀀스에 대한 마지막 출력 반환 - (batch_size, output_features) - 2D 텐서

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 [2]:

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

Embedding층의 입력

- (samples, 임베딩차원, sequence_length)
  - samples : 샘플수
  - 임베딩 차원 : 출력 임베딩 차원수
  - 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 [3]:

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

SimpleRNN 파라미터 개수

- Wh * Wh + Wx * dim + dim
- (32 * 32) + (32 * 32) + 32 * 1 = 2080

return_sequences = True

In [4]:

```
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_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
_________________________________________________________________
```

은닉 상태 출력

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
_________________________________________________________________
```

02. IMDB의 데이터 셋을 활용한 실습

목차로 이동하기

In [6]:

```
from tensorflow.keras.datasets import imdb
from tensorflow.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...
25000 훈련 시퀀스
25000 테스트 시퀀스
```

In [7]:

```
# 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 [8]:

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

 simple_rnn_6 (SimpleRNN)    (None, 32)                2080

 dense (Dense)               (None, 1)                 33

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

In [9]:

```
%%time

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

```
Epoch 1/10
157/157 [==============================] - 14s 86ms/step - loss: 0.5730 - acc: 0.696
3 - val_loss: 0.4303 - val_acc: 0.8126
Epoch 2/10
157/157 [==============================] - 14s 86ms/step - loss: 0.3595 - acc: 0.850
4 - val_loss: 0.3528 - val_acc: 0.8576
Epoch 3/10
157/157 [==============================] - 14s 90ms/step - loss: 0.2868 - acc: 0.885
2 - val_loss: 0.3337 - val_acc: 0.8648
Epoch 4/10
157/157 [==============================] - 14s 92ms/step - loss: 0.2306 - acc: 0.911
6 - val_loss: 0.4230 - val_acc: 0.8186
Epoch 5/10
157/157 [==============================] - 14s 91ms/step - loss: 0.1933 - acc: 0.930
5 - val_loss: 0.3624 - val_acc: 0.8604
Epoch 6/10
157/157 [==============================] - 14s 91ms/step - loss: 0.1641 - acc: 0.942
4 - val_loss: 0.3716 - val_acc: 0.8590
Epoch 7/10
157/157 [==============================] - 15s 93ms/step - loss: 0.1194 - acc: 0.957
8 - val_loss: 0.3875 - val_acc: 0.8574
Epoch 8/10
157/157 [==============================] - 14s 89ms/step - loss: 0.0928 - acc: 0.969
6 - val_loss: 0.5483 - val_acc: 0.8148
Epoch 9/10
157/157 [==============================] - 14s 89ms/step - loss: 0.0660 - acc: 0.977
3 - val_loss: 0.6824 - val_acc: 0.7710
Epoch 10/10
157/157 [==============================] - 17s 109ms/step - loss: 0.0538 - acc: 0.98
27 - val_loss: 0.4944 - val_acc: 0.8372
CPU times: total: 10min 49s
Wall time: 2min 24s
```

In [13]:

```
model.evaluate(input_test, y_test)
```

```
782/782 [==============================] - 14s 17ms/step - loss: 0.5044 - acc: 0.838
2
```

Out[13]:

```
[0.5043584704399109, 0.8381999731063843]
```

- 검증 정확도 : 83%

학습 후, 정확도 및 손실 그래프 확인

In [10]:

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

In [11]:

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

In [12]:

```
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()
```