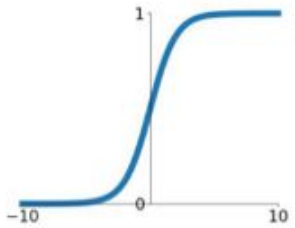
순환 신경망(RNN)

1 활성화 함수(Activation Function)

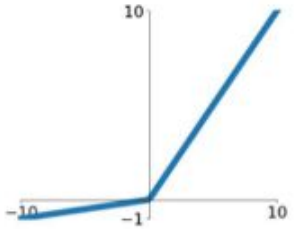
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



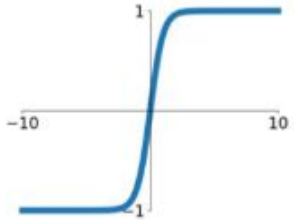
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

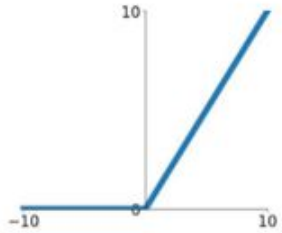


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

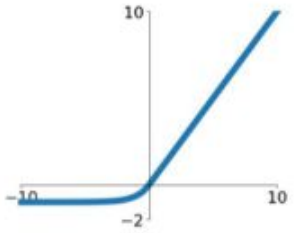
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

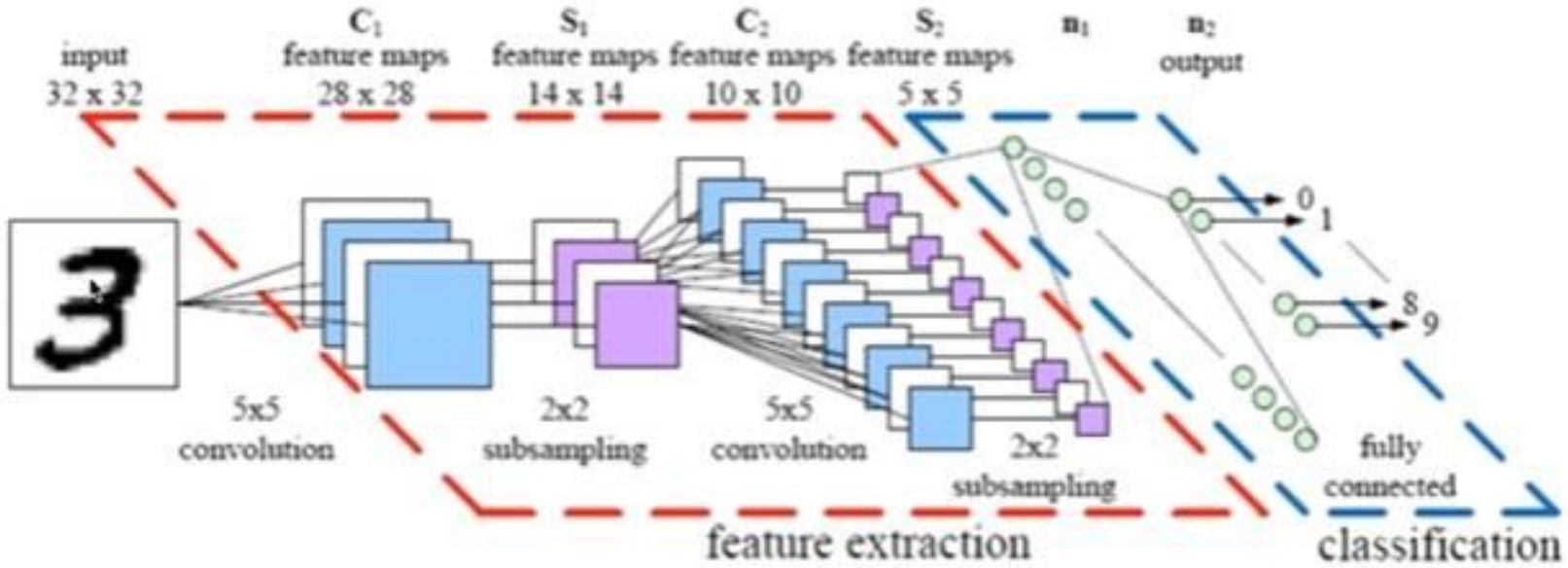


- 신경망에서는 미분 가능한 'smooth'한 S자 모양 함수 적용
- 최근 딥러닝 모델에서는 ReLU를 선호 및 많이 사용함
- 순환신경망 내부에서는 Sigmoid와 Tanh 함수를 사용함

2 Input-Output Mapping

- 정적인 입출력 사상(Static input-output mapping)
  - 시간을 고려하지 않음
  - 입력과 출력만을 고려하는 모델
  - Example : CNN

- CNN(Convolutional Neural Network)
  - Convolution 연산(특징 추출)과 Pooling(정보 압축)을 사용

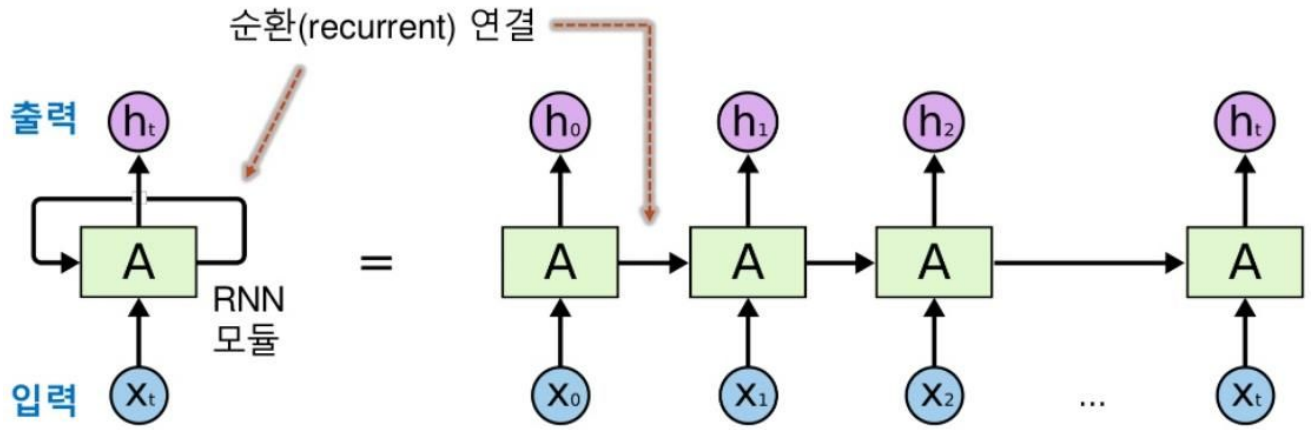


MNIST dataset을 이용한 CNN Classification

2 Input-Output Mapping

- 동적인 입출력 사상(dynamic input-output mapping)
  - 시간을 고려함
  - 모든 시간의 입력을 고려하는 모델
  - Example : RNN

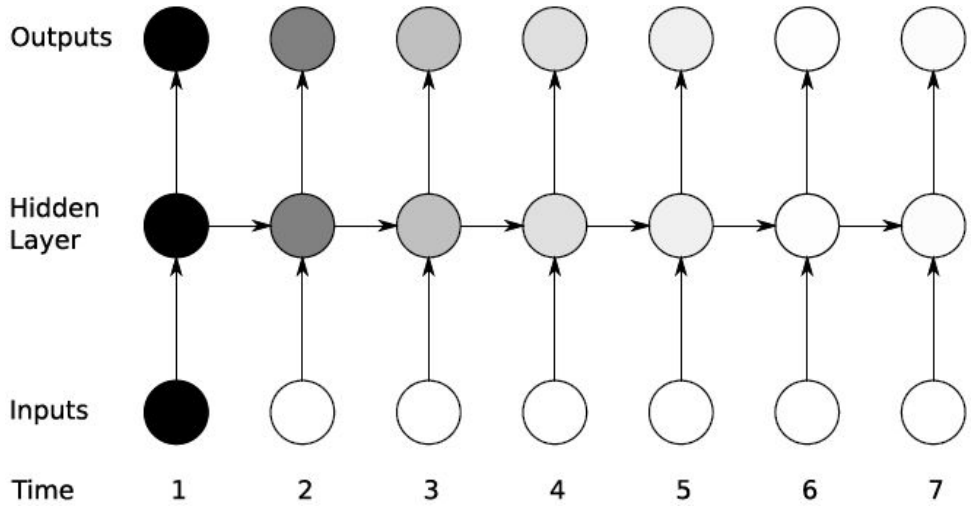
- RNN(Recurrent Neural Network)
  - 순차적인 데이터를 학습, 각 layer의 parameter들을 공유



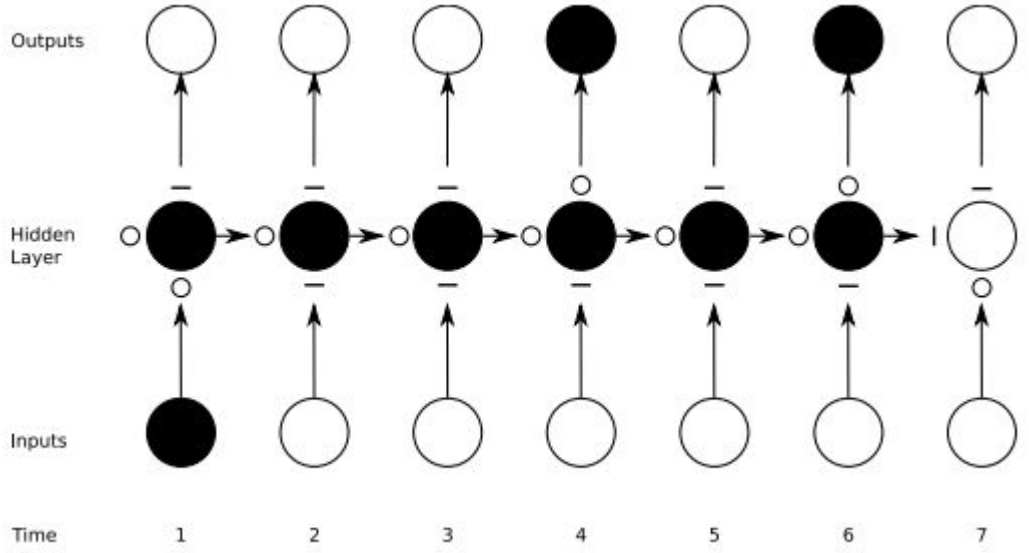
RNN(Recurrent Neural Network) 구조

1 RNN 개요

- RNN, LSTM 비교



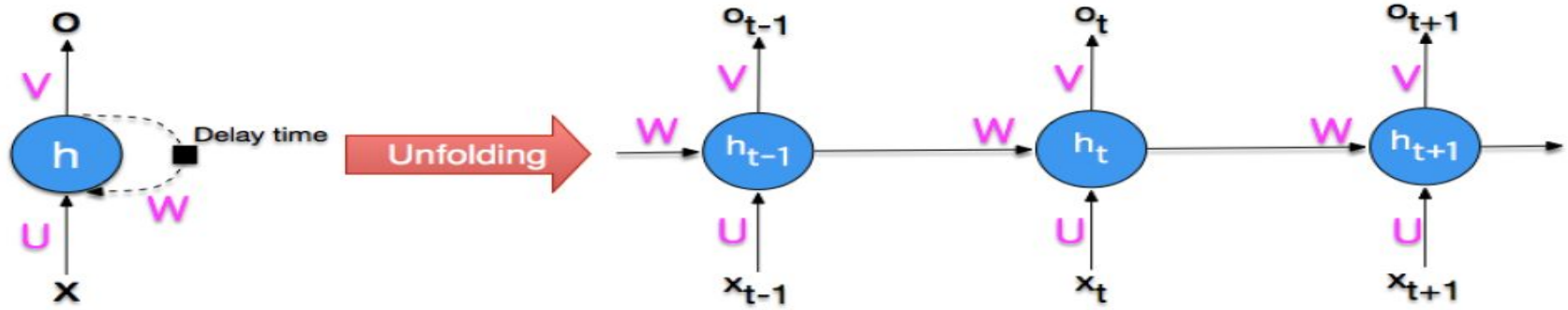
RNN: 시간이 지나면 이전 입력값을 잊어버림.



LSTM:이전 입력값의 정보가
계속 다음 상태 메모리에 반영
시간이 지나도 이전 입력값을 잊어버리지 않음.

1 RNN 개요

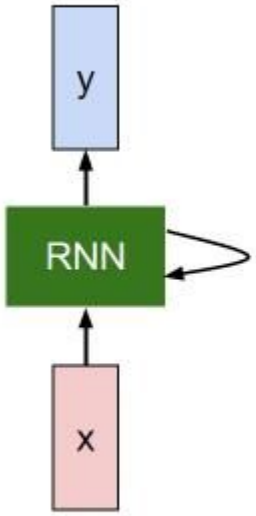
- RNN(Recurrent Neural Network)
  - 순차적인 데이터(Sequence Data)를 학습하여 Classification 또는 Prediction을 수행
  - RNN은 각 layer마다 Parameter들을 공유함(기존 DNN의 경우 각 layer의 Parameter들이 독립적)
  - 현재의 출력 결과는 이전의 Time Step의 결과에 영향을 받음, Hidden layer는 일종의 메모리 역할!



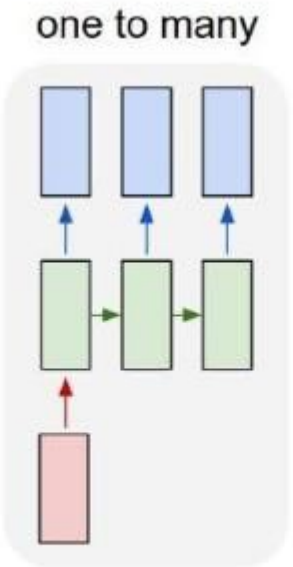
<RNN Architecture>

1 RNN 개요

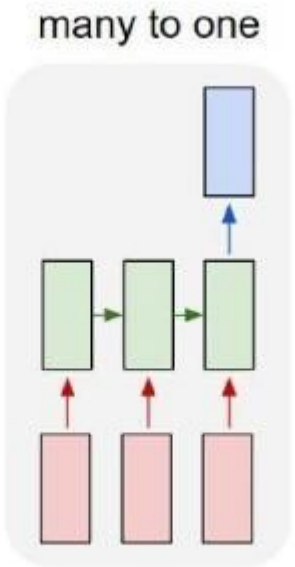
RNN의 종류



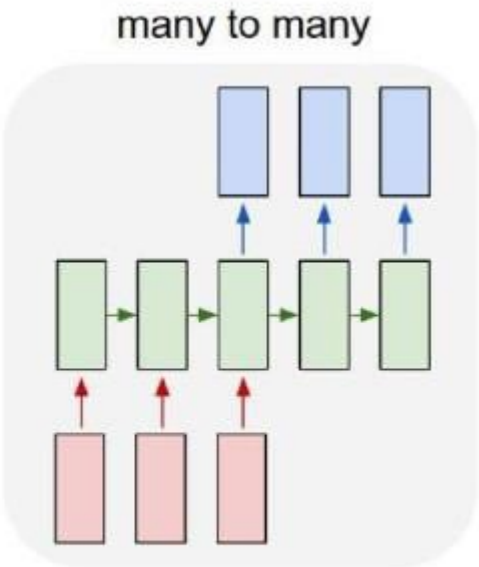
<기본구조>



<사진설명 붙이기>
사진 → 단어들

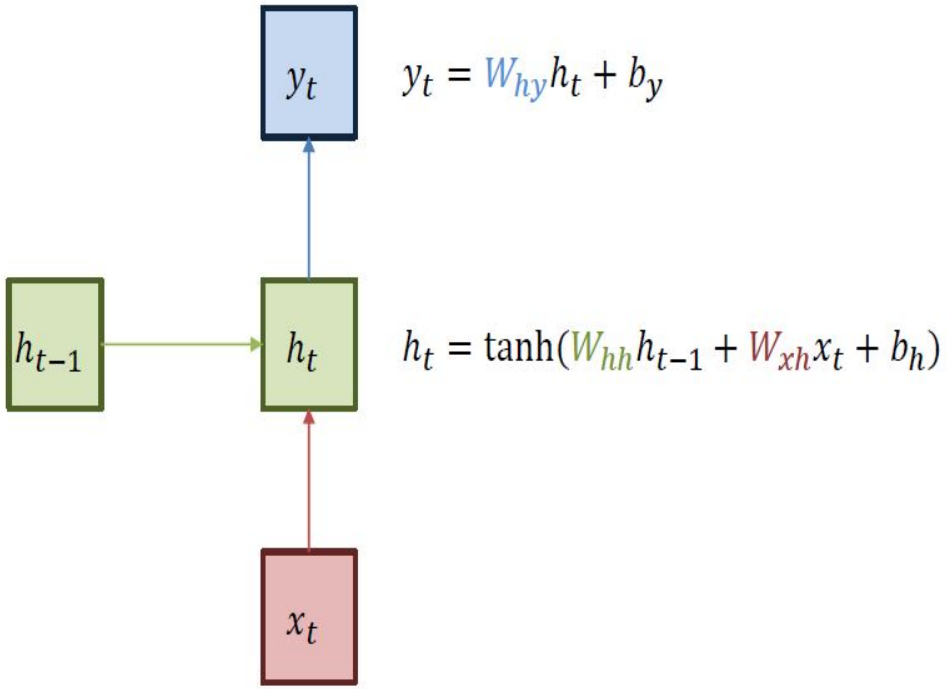


<감성분석>
단어들 → 감성점수

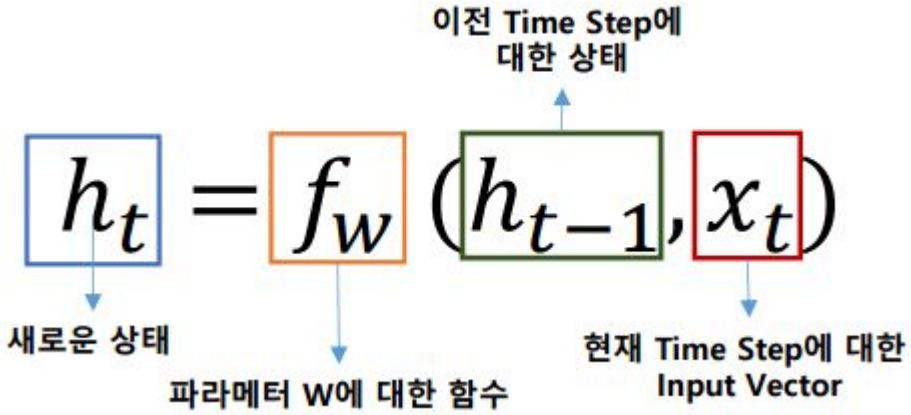


<번역>
단어들 → 단어들

1 RNN 개요

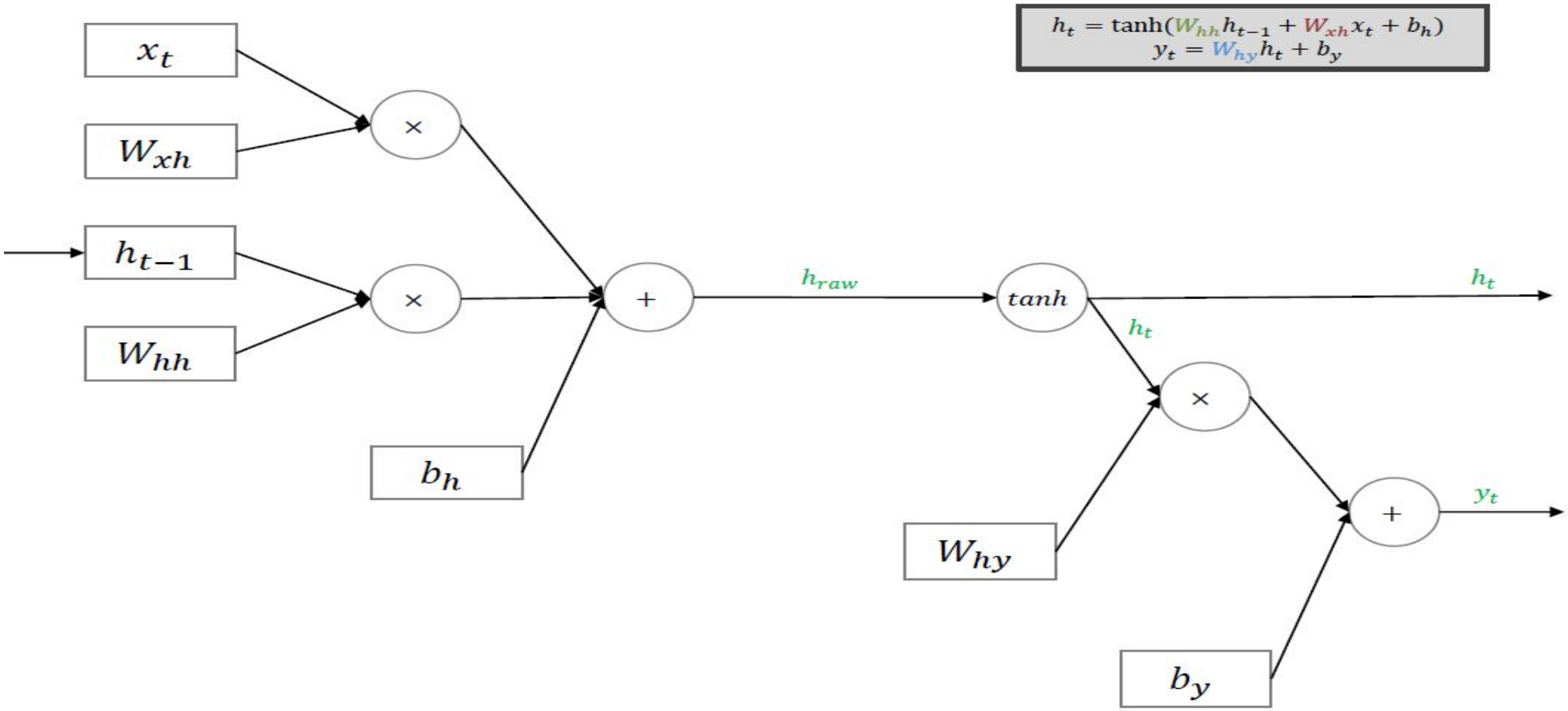


RNN의 기본 동작



RNN은 외부 입력과 자신의 이전 상태를 입력 받아 자신의 상태를 갱신한다.

1 RNN 개요


$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$
$y_t = W_{hy}h_t + b_y$
x_t
W_{xh}
h_{t-1}
W_{hh}
b_h
×
×
+
h_{raw}
tanh
h_t
h_t
×
W_{hy}
+
y_t
b_y

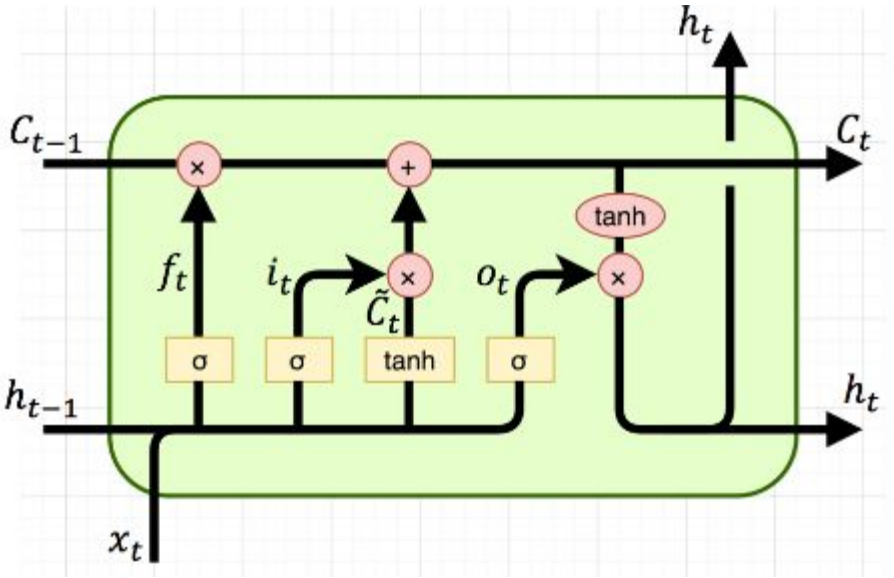
1 RNN 개요

- Vanishing Gradient Problem 발생
  - Weight가 업데이트 되는 과정에서 gradient가
  - 1보다 작은 값이 계속 곱해져, gradient가 사라지는 현상

- 해결 방안
  - LSTM(Long Short Term Memory)
  - GRU(Gated Recurrent Unit)

1 RNN 개요

- LSTM
  - LSTM은 Long Short Term Memory
  - 입력, 기억, 출력의 정도를 조절하는 세 개의 게이트(Gate)로 구성
  - 기본 RNN과 달리, 내부에 Cell이 추가되어 있음
  - Input Gate
    - 현재 메모리에 새로운 정보를 반영할지 결정
  - Forget Gate
    - 이전 상태 정보를 현재 메모리에 반영할지 결정
  - Output Gate
    - 갱신된 메모리의 출력값 제어

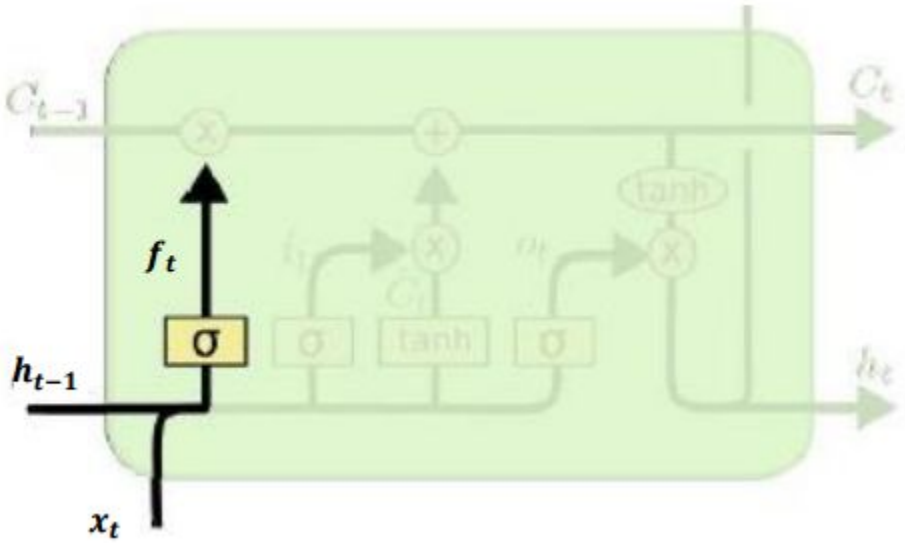


1 RNN 개요

- LSTM
  - Forget gate(reset, keep gate)
    - 이전 상태 정보를 현재 메모리에 반영할지 결정
    - 계산한 값이 1이면 바로 전 time의 memory를 유지
    - 계산한 값이 0이면 reset

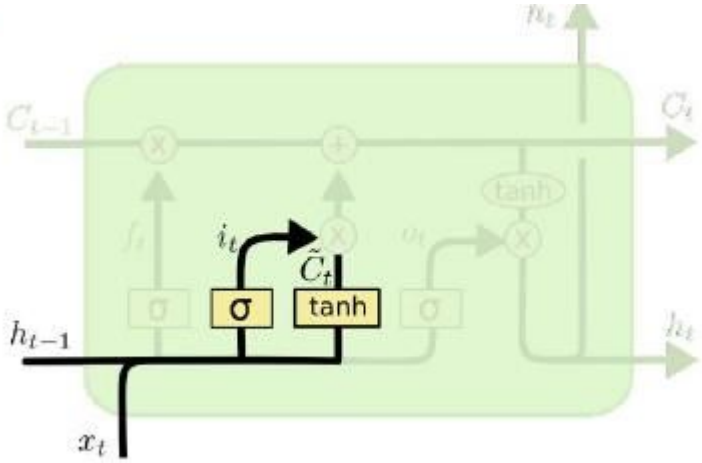
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f)$$



1 RNN 개요

- LSTM
  - Input gate(write)
    - 현재 메모리에 새로운 정보를 반영할지 결정
    - 계산한 값이 1이면 입력 xt가 들어올 수 있도록 허용(open)
    - 계산한 값이 0이면 block(close)



Input gate layer

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_f)$$

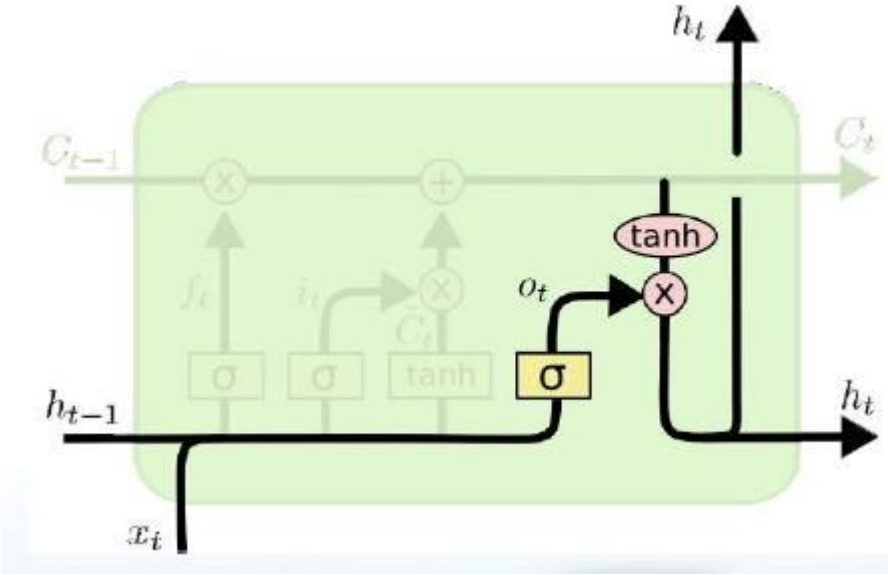
New contribution to cell state

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$\tilde{C}_t = tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_f)$$

1 RNN 개요

- LSTM
  - Output gate(read)
    - 갱신된 메모리의 출력값 제어
    - 계산한 값이 1이면 의미 있는 결과로 최종 output(open)
    - 계산한 값이 0이면 해당 연산 출력을 안함(close)



Output gate layer

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

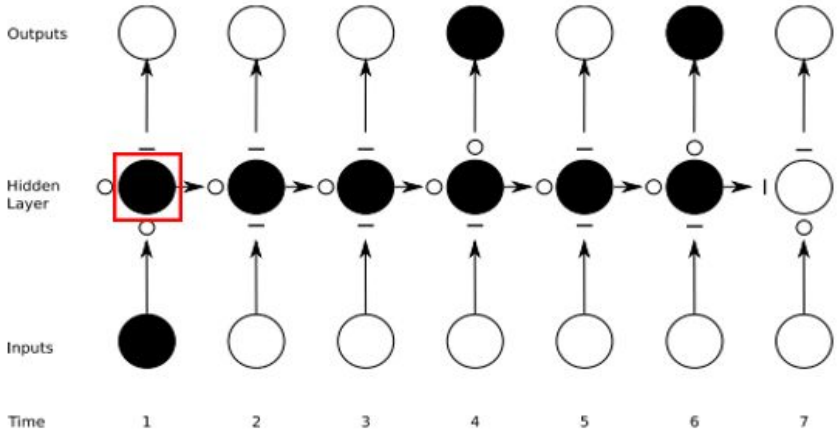
$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o)$$

Output to next layer

$$h_t = o_t * tanh(c_t)$$

1 RNN 개요

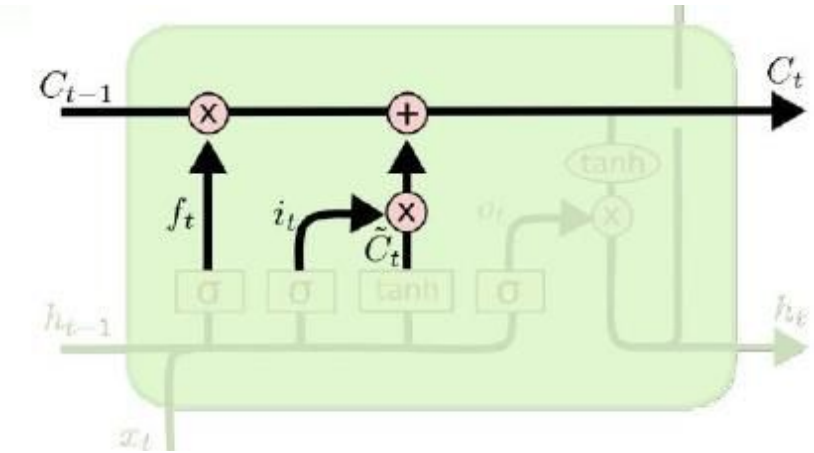
- LSTM
  - Cell
    - time step에 대한 hidden node를 메모리 셀이라고 함
    - Cell의 값을 반영하기 위해 “sum”을 사용
    - (vanishing/exploding 문제 해결에 도움!)
    - Update cell state
      - Forget, input gate와 이전 time step cell 정보를 계산하여 현재 time step의 cell state 업데이트



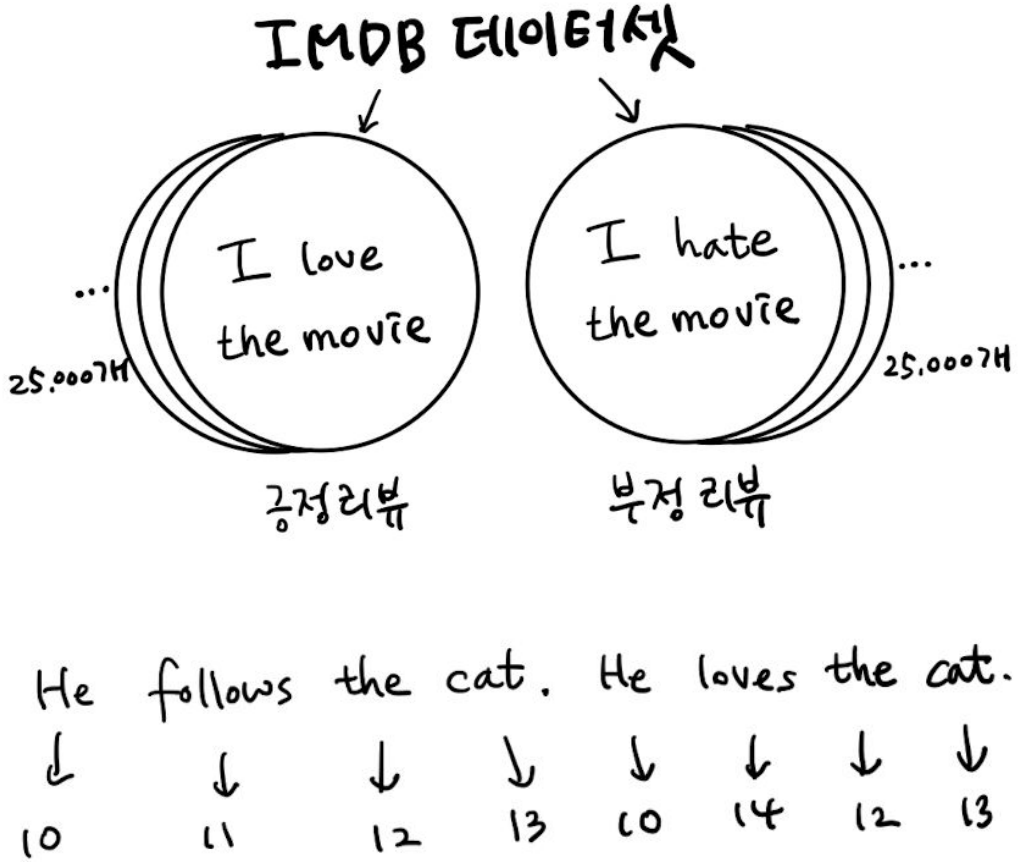
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Forget gate 정보와
이전 time step의 cell 정보 곱

Input gate 정보와
현재 time step의 cell 정보 곱



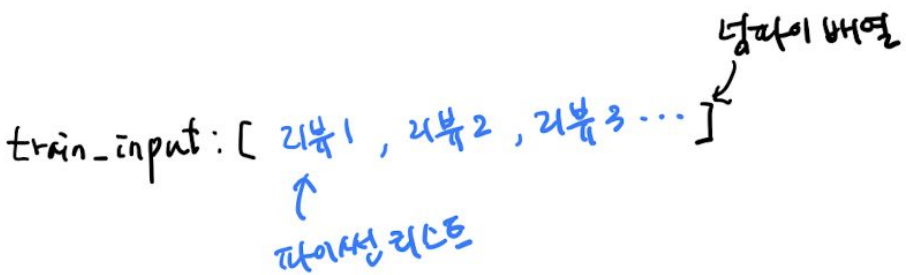
IMDB 리뷰 데이터셋



- NLP
- 말뭉치
- 토큰
- 어휘 사전

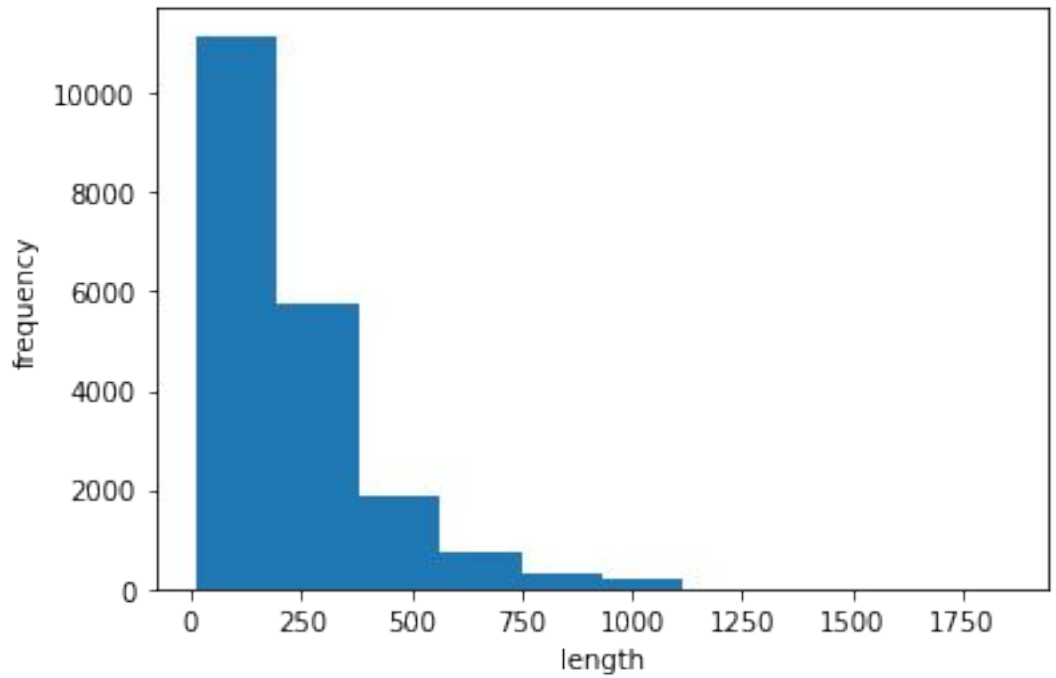
케라스로 IMDB 데이터 불러오기

```
from tensorflow.keras.datasets import imdb

(train_input, train_target), (test_input, test_target) = imdb.load_data(
    num_words=500)
print(train_input.shape, test_input.shape)
(25000,) (25000,)

print(train_input[0])
[1, 14, 22, 16, 43, 2, 2, 2, 2, 65, ...

print(train_target[:20])
[1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1]
```



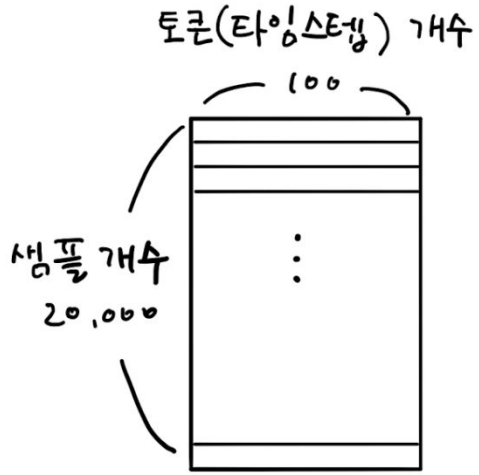
훈련 세트 준비

```
train_input, val_input, train_target, val_target = train_test_split(
    train_input, train_target, test_size=0.2, random_state=42)

lengths = np.array([len(x) for x in train_input])
print(np.mean(lengths), np.median(lengths))
239.00925 178.0

plt.hist(lengths)
plt.xlabel('length')
plt.ylabel('frequency')
plt.show()
```



시퀀스 패딩

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

train_seq = pad_sequences(train_input, maxlen=100)

print(train_seq.shape)
(20000, 100)

print(train_seq[0])
[  10    4   20    9    2  364  352    5   45    6    2    2   33  269    8    2  142    2
    5    2   17   73   17  204    5    2   19   55    2    2   92   66  104   14   20   93
   76    2  151   33    4   58   12  188    2  151   12  215   69  224  142   73  237    6
    2    7    2    2  188    2  103   14   31   10   10  451    7    2    5    2   80   91
    2   30    2   34   14   20  151   50   26  131   49    2   84   46   50   37   80   79
    6    2   46    7   14   20   10   10  470  158]

print(train_input[0][-10:])
[6, 2, 46, 7, 14, 20, 10, 10, 470, 158]

print(train_seq[5])
[  0   0   0   0   1   2 195  19  49   2   2 190   4   2 352   2 183  10
  10  13  82  79   4   2  36  71 269   8   2  25  19  49   7   4   2   2
   2   2   2  10  10  48  25  40   2  11   2   2  40   2   2   5   4   2
   2  95  14 238  56 129   2  10  10  21   2  94 364 352   2   2  11 190
  24 484   2   7  94 205 405  10  10  87   2  34  49   2   7   2   2   2
   2   2 290   2  46  48  64  18   4   2]

val_seq = pad_sequences(val_input, maxlen=100)
```



순환 신경망 모델 만들기

```
from tensorflow import keras

model = keras.Sequential()

model.add(keras.layers.SimpleRNN(8, input_shape=(100, 500)))
model.add(keras.layers.Dense(1, activation='sigmoid'))
```

원-핫 인코딩

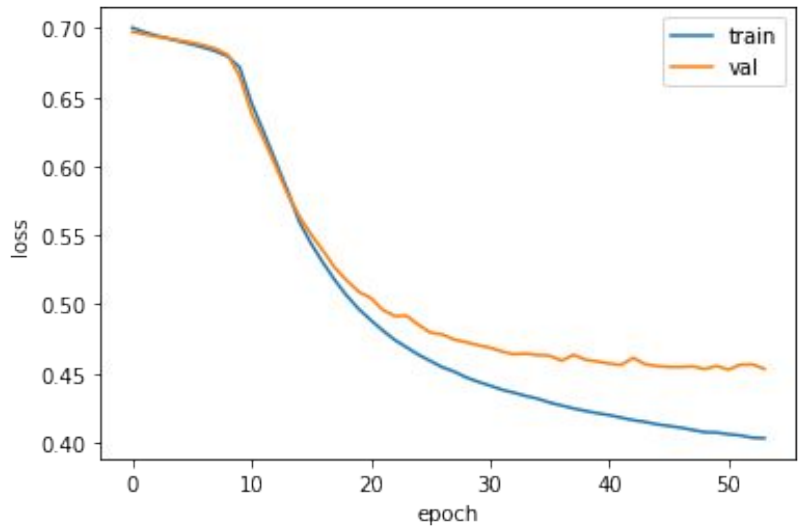

```
train_oh = keras.utils.to_categorical(train_seq)

print(train_oh.shape)
(20000, 100, 500)

print(train_oh[0][0][:12])
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]

print(np.sum(train_oh[0][0]))
1.0

val_oh = keras.utils.to_categorical(val_seq)
```

모델 구조 확인

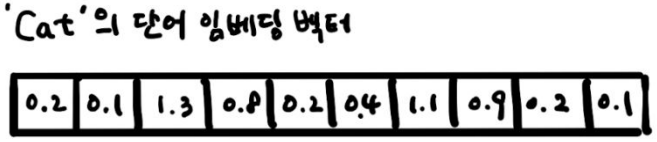
```
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 8)                 4072
_________________________________________________________________
dense (Dense)                (None, 1)                 9
=================================================================
Total params: 4,081
Trainable params: 4,081
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련

```
rmsprop = keras.optimizers.RMSprop(learning_rate=1e-4)
model.compile(optimizer=rmsprop, loss='binary_crossentropy',
              metrics=['accuracy'])

checkpoint_cb = keras.callbacks.ModelCheckpoint('best-simplernn-model.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3,
                                                  restore_best_weights=True)

history = model.fit(train_oh, train_target, epochs=100, batch_size=64,
                    validation_data=(val_oh, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```



임베딩




```
model2 = keras.Sequential()

model2.add(keras.layers.Embedding(500, 16, input_length=100))
model2.add(keras.layers.SimpleRNN(8))
model2.add(keras.layers.Dense(1, activation='sigmoid'))

model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 16)           8000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 8)                 200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 8,209
Trainable params: 8,209
Non-trainable params: 0
_________________________________________________________________
```