12. 순환신경망

12.1. 순차적 데이터

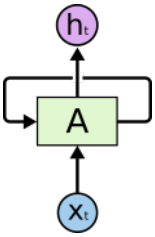
주가나 날씨처럼 시간에 따라 변화하거나, 텍스트나 음악처럼 글자나 음이 순서대로 나타난 정보들이 있다. 이러한 데이터를 **순차적 데이터**(sequential data)라고 한다.

순차적 데이터에서는 대개 순서상 앞이나 뒤에 있는 정보가 서로 영향을 주기도 하고, 주기성이나 경향성을 띄기도 한다. 날씨의 경우 1년을 주기로 추웠다 더워지기를 반복하고, 텍스트의 경우 앞에 나온 말을 보면 뒤에 나올말을 짐작할 수 있다.

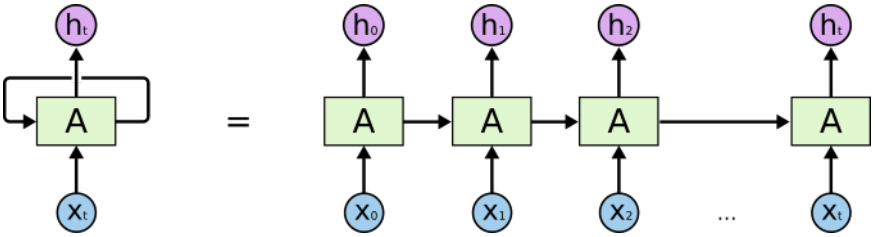
12.2. 순환신경망

순차적 데이터를 다루는 전통적인 방법으로는 자기회귀(autoregression)나 마코브 연쇄(Markov chain)와 같은 방법들이 있다. 딥러닝에서 이러한 특성을 분석할 수 있는 방법으로는 순환신경망(Recurrent Neural Network, 이하 RNN)가 있다.

RNN에는 여러가지 형태가 있으나 가장 대표적인 것은 아래 그림과 같은 형태이다.



입력층 X에서 은닉층 A를 거쳐 출력층 h로 신호가 전달된다는 점에서 RNN은 앞먹임 신경망과 동일한 구조를 가진다. 한 가지 차이는 은닉층에서 자기 자신으로 돌아오는 고리가 있다는 것이다. 즉 첫번째 입력 X_1이 한 번 신경망을 거쳐 나가고 나면, 두번째 입력 X_2가 처리될 때는 X_1의 은닉층 상태가 X_2의 입력과 함께 은닉층으로 들어오게 된다. 이러한 과정을 통해서 앞의 입력이 뒤의 입력에 미치는 영향을 파악할 수 있다. 따라서 RNN을 펼치면 아래와 같은 형태의 네트워크가 된다.



12.3. 장기 의존성의 문제

RNN에서 은닉층에서 은닉층으로 가는 연결은 같은 신호를 반복해서 전달한다. 즉, 다음과 같은 형태의 식으로 표현된다.

$$A_t = f(A_{t-1}, X_t) = \sigma(wA_{t-1} + vX_t + b)$$

논의를 간단히 하기 위해 입력층과 절편은 제외하고 생각해보자.

$$A_t = \sigma(wA_{t-1})$$

그러면 은닉층의 이전 상태에 대한 다음 상태의 미분은 아래와 같다.

$$\frac{\partial A_t}{\partial A_{t-1}} = w\sigma'(wA_{t-1})$$

만약 은닉층을 여러 층 거칠 경우에 그 미분은 아래와 같은 식이 된다.

$$\begin{aligned}\frac{\partial A_{t+n}}{\partial A_t} &= \frac{\partial A_{t+n}}{\partial A_{t+n-1}} \cdots \frac{A_{t+1}}{A_t} \\ &= \prod_{k=0}^{n-1} \frac{A_{t+k+1}}{A_{t+k}} \\ &= \prod_{k=0}^{n-1} w\sigma'(wA_{t+k}) \\ &= w^n \prod_{k=0}^{n-1} \sigma'(wA_{t+k})\end{aligned}$$

여기서 보면 w^n 때문에 n이 커지면 $w < 1$인 경우 미분이 0으로 수렴하고, $w > 1$인 경우 발산하게 된다. 전자는 사라지는 경사, 후자는 폭발하는 경사(exploding gradient)라고 한다.

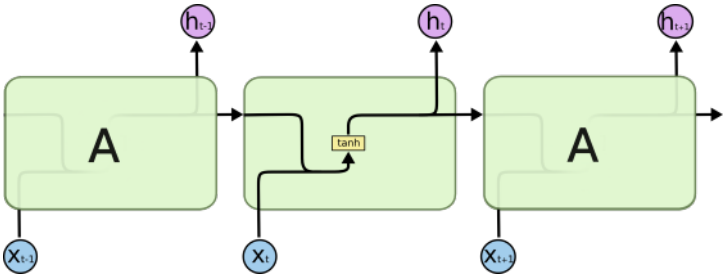
사라지는/폭발하는 경사 자체는 앞먹임 신경망에서도 동일한 문제이나 RNN에서는 더 심각한 문제가 된다. 앞먹임 신경망에서는 모형의 깊이가 깊어지면서 생기는 문제지만, RNN에서는 데이터가 길어지기만해도 생기는 문제이기 때문이다.

두 입력 사이에 거리가 멀리 떨어진 경우($n \gg 0$)에 존재하는 영향을 장기 의존성(long-term dependency)이라고 한다. 한국어의 텍스트의 경우 주어는 문장의 맨 처음에, 동사는 문장의 맨 끝에 나오므로 문장이 길어지면 주어와 동사 사이에 장기 의존성이 생기게 된다. 그런데 RNN은 두 입력 사이의 거리가 멀면 경사하

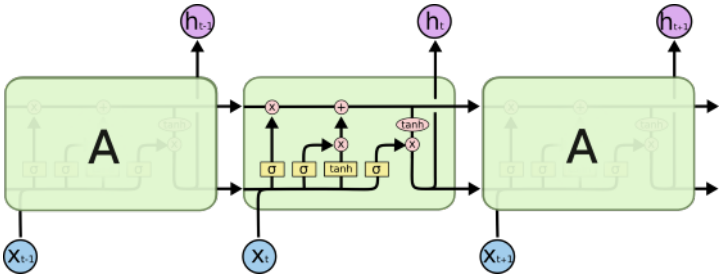
강법이 잘 작동하지 않아, 장기의존성을 학습하기가 어렵다.

12.4. LSTM

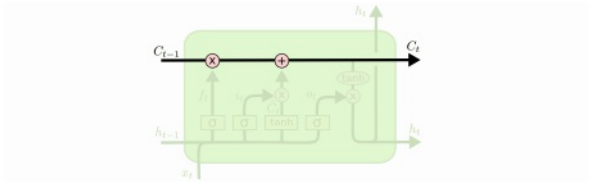
단순한 RNN을 좀 더 자세히 보면 아래와 같은 구조를 가지고 있다. 입력이 이전 은닉층의 상태와 합쳐져 활성화 함수로 들어가고 그 출력이 출력층과 다음 은닉층으로 넘어가는 것이다.



RNN이 장기 의존성을 학습하지 못하는 이유는 사라지는/폭발하는 경사 때문이고, 사라지는/폭발하는 경사는 활성화 함수를 여러 번 반복해서 거치기 때문이다. 그렇다면 은닉층에서 은닉층으로 바로 신호가 전달할 수 있게 하면 어떨까? 이 아이디어에 바탕을 둔 것이 LSTM(Long Short-Term Memory)이다.



매우 복잡하게 보이지만 위의 그림에서 핵심 아이디어는 아래 부분이다.



일단 LSTM에서는 은닉층에서 은닉층으로 전달되는 신호 C와 은닉층에서 출력층으로 전달되는 신호 h가 분리되었다. 그리고 C는 별다른 활성화 함수를 거치지 않고 바로 다음 은닉층으로 전달된다. 따라서 사라지는/폭발하는 경사 문제에서 자유롭게 된다. 대신 망각 게이트(forgetting gate, 위의 그림에서 분홍색 원 안의 X)를 두어 신호를 차단하거나, 입력 게이트(input gate, 분홍색 원 안의 +)를 통해 새로운 신호를 추가한다.

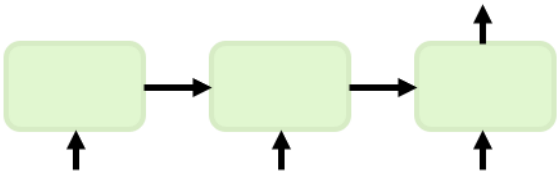
LSTM은 최근에 특히 텍스트, 음성 분석 등에서 각광을 받고 있다. (LSTM 자체는 1997년에 발표된 모형이다.) LSTM은 다음과 같은 문제들에 탁월한 성과를 보여주고 있다.

- 손글씨 인식
- 음성 인식
- 기계 번역
- 이미지 설명 생성
- 문법 분석

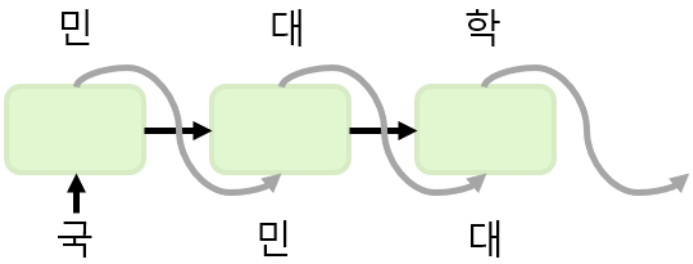
12.5. 순환신경망의 여러 구조

순환신경망의 세부적인 구조는 문제에 따라 달라진다. 우선은 입력과 출력이 다른 경우를 보자. 예를 들어 입력에는 거시경제 변수를, 출력에는 코스피 지수를 넣을 수 있다. 이럴 때는 입력과 출력의 갯수가 똑같아야 한다.

그런데 입력은 여러 개지만 출력은 1개만 있을 때도 있다. 예를 들면 텍스트를 분석해서 하나의 점수를 주는 경우가 있다. 이때는 아래 그림과 같이 마지막 노드만 출력을 하도록 한다.



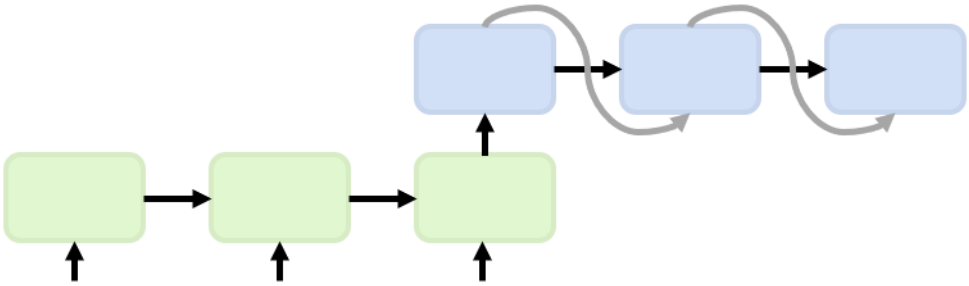
입력과 출력이 같은 경우도 있다. 스마트폰의 키보드에는 다음에 입력할 글자나 단어를 추천해주는 경우가 있다. 이럴 경우 입력을 앞 글자, 출력을 뒷 글자로 하고 앞의 출력을 다시 뒤의 입력으로 넣어주는 구조로 만든다. 아래 그림은 "국민대학"이라는 4글자를 이런 구조의 순환신경망으로 학습/예측시키는 예이다.



이렇게 출력을 다시 입력으로 넣는 구조에서는 학습시킬 때와 예측할 때가 달라진다. 예측을 할 때는 위의 그림처럼 바로 출력을 입력으로 넣으면 되지만, 학습을 시킬 때는 뒤로 갈 수록 실제 데이터와 입력이 달라져버리는 경우가 생긴다. 예를 들어 2번째 노드가 "민" 다음에 "의"를 예측하면 3번째 노드는 입력이 "의"가 되고 출력이 "학"이 되어 버린다. 이런 문제를 막기 위해 학습 시킬 때는 실제 데이터를 입력에 넣어주는 방법을 쓴다. 이런 기법을 teacher forcing이라고 한다.

12.5.1. Seq2Seq

번역의 경우에는 입력된 단어의 갯수와 출력된 단어의 갯수가 서로 다르다. 이럴 때는 앞에서 배운 구조들을 합쳐서 Seq2Seq라는 구조를 만든다.



seq2seq는 2개의 네트워크로 이뤄지는데 번역의 경우 원래 언어를 처리하는 인코더(encoder)와 번역될 언어를 생성하는 디코더(decoder)로 나뉘어진다. 인코더는 원래의 언어의 의미를 하나의 벡터로 디코더에 전달하고, 디코더는 이를 다시 번역될 언어의 순서로 생성하는 것이다.

seq2seq는 번역만이 아니라 문서 요약 등에도 사용할 수 있다. 다만 seq2seq는 긴 문장의 의미를 하나의 벡터로 압축해야한다는 어려움이 있는데, 최근에는 원 문장의 특정 부분에만 주의(attention)를 주는 방법이 해결책으로 제안되고 있다.

12.6. 셀레늄이란

셀레늄은 웹 브라우저를 제어해주는 라이브러리이다. 셀레늄의 원래 목적은 웹 사이트를 테스트하는 것이다.

웹에 있는 버튼이 작동을 하는지 확인하거나, 버튼을 100번씩 반복해서 누르는 테스트 등을 일일이 사람이 클릭하여 테스트하면 번거롭기 때문에, 웹 브라우저를 제어하여 테스트하도록 도와준다.

크롤링이 어려운 사이트의 경우, 셀레늄을 사용하면 실제 웹 브라우저를 제어하여 사람처럼 접속할 수 있기 때문에 쉽게 크롤링을 할 수 있다.

다만, 실제 웹 브라우저를 띄우기 때문에 `requests`를 쓰는 경우보다 속도가 느리다는 단점이 있다.

12.6.1. 셀레늄 설치

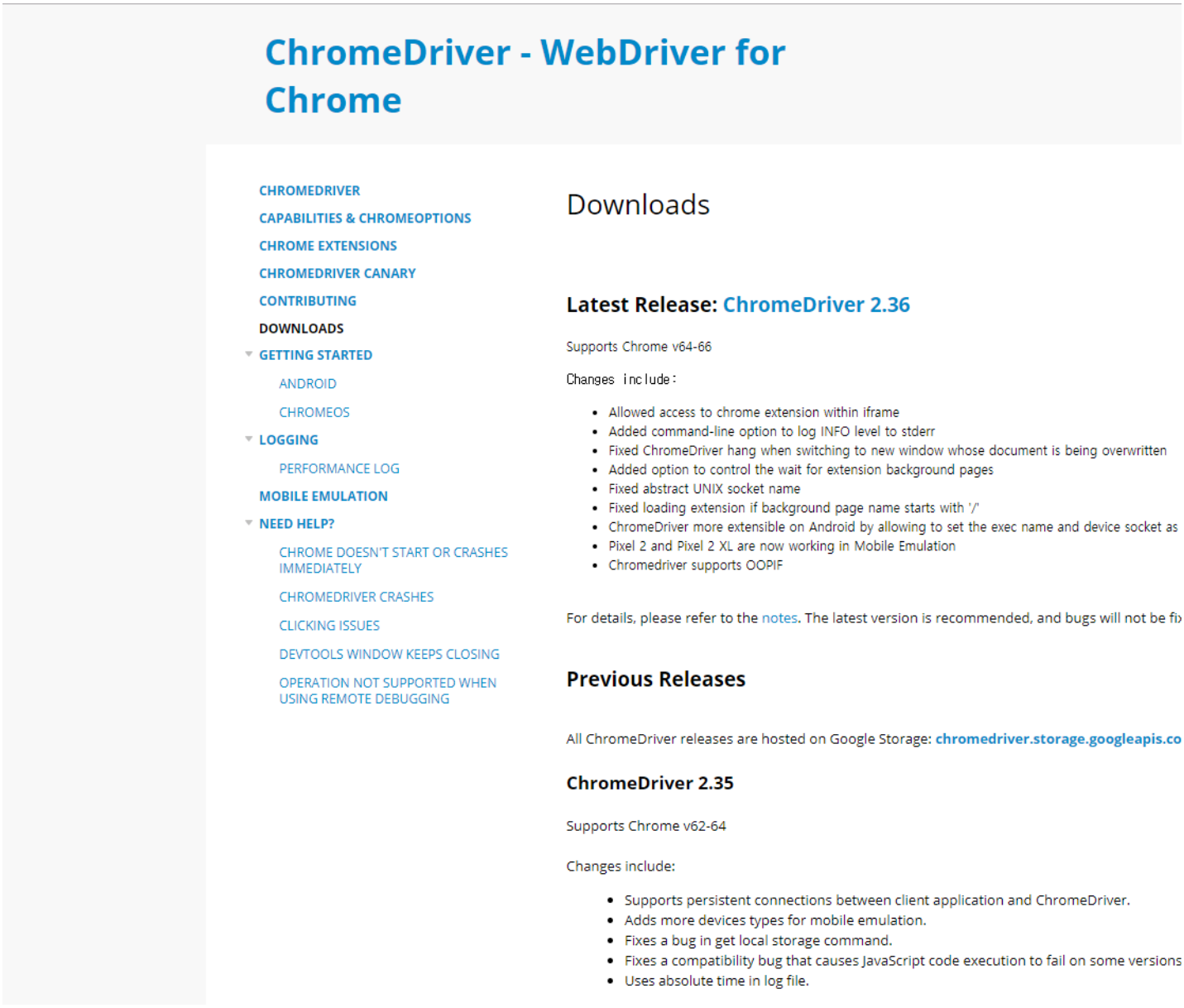
우선 셀레늄을 설치하자. `pip install` 로 설치하면 된다.

```
pip install selenium
```

셀레늄에서는 웹드라이버라는 모듈을 통해 "파이어폭스", "크롬", "인터넷 익스플로러" 등 여러 웹 브라우저를 제어할 수 있다. 이번 예제에서는 크롬 브라우저를 사용하도록 한다. 크롬을 사용하기 위해서는 최신 크롬 드라이버를 받아야한다.

셀레늄 브라우저별 웹드라이버 링크는 셀레늄의 다운로드 페이지에서 찾을 수 있다.

- 셀레늄 다운로드 페이지: https://www.seleniumhq.org/download
- 크롬 웹드라이버: https://sites.google.com/a/chromium.org/chromedriver/downloads



사이트에 들어가서 최신 크롬 드라이버를 클릭한다. 2018년 5월 현재 최신 버전은 **2.38** 이다. 접속 후에, 본인의 운영체제(OS)에 맞는 버전을 다운 받는다.

윈도의 경우 다운 받은 파일의 압축을 풀면 `chromedriver.exe` 하나가 들어있다. 다운로드한 실행 파일은 현재의 **작업 디렉토리**에 저장해주면 된다.

현재의 작업 디렉토리는 아래 코드로 알 수 있다.

```
import os
os.getcwd()    # 현재 작업 디렉토리를 확인한다.
```

12.7. 셀레늄으로 트립어드바이저 스크랩

```
from selenium.webdriver import Chrome
```

크롬 브라우저를 연다. `FileNotFoundError`가 발생한다면 크롬 웹드라이버를 작업 디렉토리에 복사한다.

```
browser = Chrome()
```

원하는 페이지의 주소로 이동한다.

```
url = 'https://www.tripadvisor.co.kr/Restaurant_Review-g294197-d1371740-Reviews-Mugyodong_Bugeokukjib-Seoul.html'
```

```
browser.get(url)
```

`find_elements_by_css_selector`로 '더보기' 링크를 찾는다. 단수형 element가 아닌 복수형 elements임에 주의. 이 메소드는 `requests`의 `cssselect`에 해당한다.

```
more_links = browser.find_elements_by_css_selector('.ulBlueLinks')
```

모든 링크를 클릭한다. 중간에 예외가 발생할 경우 넘어간다.

```
for link in more_links:
    try:
        link.click()
    except:
        pass
```

리뷰들을 찾는다

```
reviews = browser.find_elements_by_css_selector('.review-container')
```

```
s = len('ui_bubble_rating bubble_')
```

```
review_list = []
for review in reviews:
    # 별점을 찾는다. 하나만 찾기 때문에 단수형 element를 사용
    rating = review.find_element_by_css_selector('.ui_bubble_rating')

    # 별점에 지정된 클래스를 가져온다 (requests의 .attrib에 해당)
    cls = rating.get_attribute('class')

    # 앞부분을 떼어내고 정수로 해석한다
    score = int(cls[s:])

    # 평을 찾는다.
    comment = review.find_element_by_css_selector('.partial_entry')

    # 점수와 텍스트를 리스트에 추가한다
    review_list.append((score, comment.text))
```

데이터 프레임으로 바꾼다.

```
import pandas

df = pandas.DataFrame(review_list, columns=['score', 'text'])

df
```

	score	text
0	50	왜 미리 몰랐을까 라고 후회 했던 집 정말 맛보시길 추천 합니다 일요일 오...더 보기
1	50	"숙취에 최고의 음식일 듯, 유명한 만큼 맛았고 친절하고 회전이 빠른 식당"\n"직...
2	50	변함없는 속풀이. 깔끔하고 회전 빠르고 가격도 주변에 비해 착함. 포장도 맛있음. ...
3	40	엄청난 맛집은 아니지만 나쁘지 않음으로 꾸준히 유지가 되는 집. 너무 큰 기대를 가...
4	50	단일 품목으로 오랫동안 손님들의 사랑을 듬뿍 받는 이곳, 5 년만에 다시 들렀어요....
5	40	첫느낌은 노포에서 맛볼 수 있는 북어국이 아닐까 했는데,들어가보니\n전혀 반대입니다...
6	50	이집은 정말 대한민국에서 최고의 해장 북엉국집이라 인정해요\n\n어제 과음으로 힘뜰...
7	40	짭자름한 북엇국물에 밥 한 공기면, 전날 숙취를 깔끔히 떨어낼 수 있다.\n가격도 ...
8	30	명동에 명동교자가 있다면 무교동엔 북어국집이 있습니다..\n오래된 곳이라는 느낌이지...
9	50	담백하고 부드러운 국물맛이 일품\n술먹은 다음날 최고의 선택\n점심시간에는 대기줄이...

csv 파일로 저장한다.

```
df.to_csv('tripadvisor.csv', encoding='utf8')
```

12.8. RNN 실습

keras를 불러들인다.

```
import keras
```

```
Using TensorFlow backend.
```

12.8.1. imdb 데이터

imdb 영화평 데이터를 불러온다. 해당 데이터는 케라스에 내장되어 있다.

```
from keras.datasets import imdb

data = imdb.load_data()
```

훈련용 데이터와 테스트용 데이터를 불러온다. x는 영화평, y는 영화평의 긍/부정 여부다.

```
(x_train, y_train), (x_test, y_test) = data
```

x는 단어 번호로 이뤄져 있다.

```
x_train[0][:10]
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
```

```
y_train[0]
```

```
1
```

keras의 RNN은 항상 펼쳐진 상태로만 사용가능하다. 따라서 최대 길이를 미리 지정해줘야 한다.

```
MAXLEN = 20
```

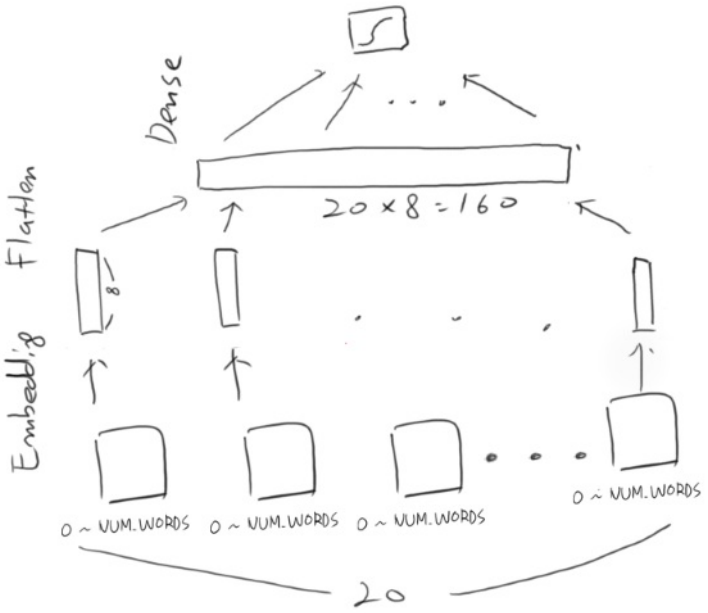
pad_sequence는 최대 길이보다 짧으면 0으로 채우고, 길면 잘라낸다.

```
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

x_train_seq = pad_sequences(x_train, MAXLEN)
x_test_seq = pad_sequences(x_test, MAXLEN)
```

가장 큰 단어 번호에 1을 더해 단어의 개수를 계산한다.

```
NUM_WORDS = x_train_seq.max() + 1
```

12.8.2. 앞먹임 신경망



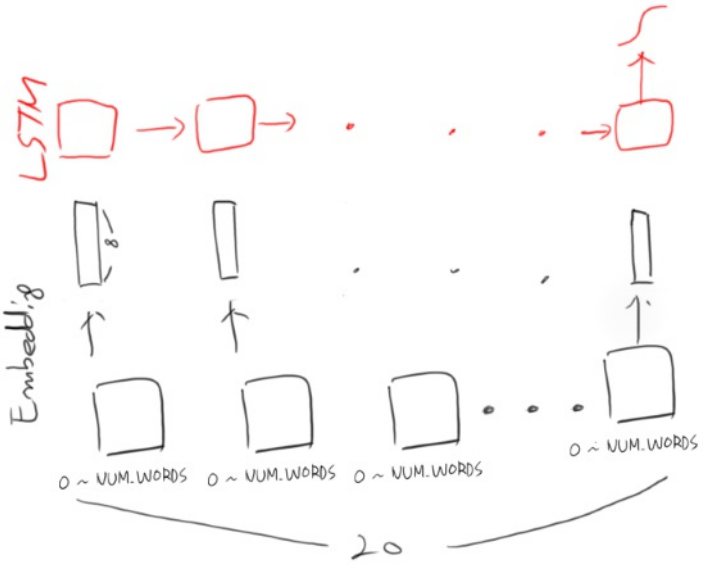

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, Flatten

ff = Sequential()
ff.add(Embedding(input_dim=NUM_WORDS, output_dim=8, input_length=MAXLEN))
ff.add(Flatten())
ff.add(Dense(1, activation='sigmoid'))

ff.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 20, 8)             708632
_________________________________________________________________
flatten_3 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 161
=================================================================
Total params: 708,793
Trainable params: 708,793
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.optimizers import RMSprop

ff.compile(optimizer=RMSprop(), loss='binary_crossentropy', metrics=['acc'])

from keras.callbacks import EarlyStopping

history = ff.fit(
  x_train_seq,
    y_train,
    epochs=30,
    batch_size=128,
    validation_split=0.2,
    callbacks=[EarlyStopping(monitor='val_acc')])
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/30
20000/20000 [==============================] - 1s 51us/step - loss: 0.6858 - acc: 0.5822 - val_loss: 0.6709 - val_acc: 0.6672
Epoch 2/30
20000/20000 [==============================] - 1s 43us/step - loss: 0.6321 - acc: 0.7402 - val_loss: 0.6084 - val_acc: 0.7094
Epoch 3/30
20000/20000 [==============================] - 1s 53us/step - loss: 0.5502 - acc: 0.7728 - val_loss: 0.5509 - val_acc: 0.7266
Epoch 4/30
20000/20000 [==============================] - 1s 57us/step - loss: 0.4817 - acc: 0.7974 - val_loss: 0.5163 - val_acc: 0.7394
Epoch 5/30
20000/20000 [==============================] - 1s 57us/step - loss: 0.4326 - acc: 0.8194 - val_loss: 0.4987 - val_acc: 0.7488
Epoch 6/30
20000/20000 [==============================] - 1s 56us/step - loss: 0.3954 - acc: 0.8383 - val_loss: 0.4895 - val_acc: 0.7562
Epoch 7/30
20000/20000 [==============================] - 1s 57us/step - loss: 0.3647 - acc: 0.8515 - val_loss: 0.4860 - val_acc: 0.7568
Epoch 8/30
13824/20000 [===================>.....] - ETA: 0s - loss: 0.3374 - acc: 0.868320000/20000 [==============================] - 1s 58us/step
Epoch 9/30
20000/20000 [==============================] - 1s 59us/step - loss: 0.3135 - acc: 0.8778 - val_loss: 0.4859 - val_acc: 0.7600
Epoch 10/30
20000/20000 [==============================] - 1s 57us/step - loss: 0.2909 - acc: 0.8903 - val_loss: 0.4891 - val_acc: 0.7590
```

12.8.3. LSTM




```
rnn = Sequential()

from keras.layers import LSTM

rnn.add(Embedding(input_dim=NUM_WORDS, output_dim=8, input_length=MAXLEN))
rnn.add(LSTM(1, activation='sigmoid', return_sequences=False))

rnn.summary()
```

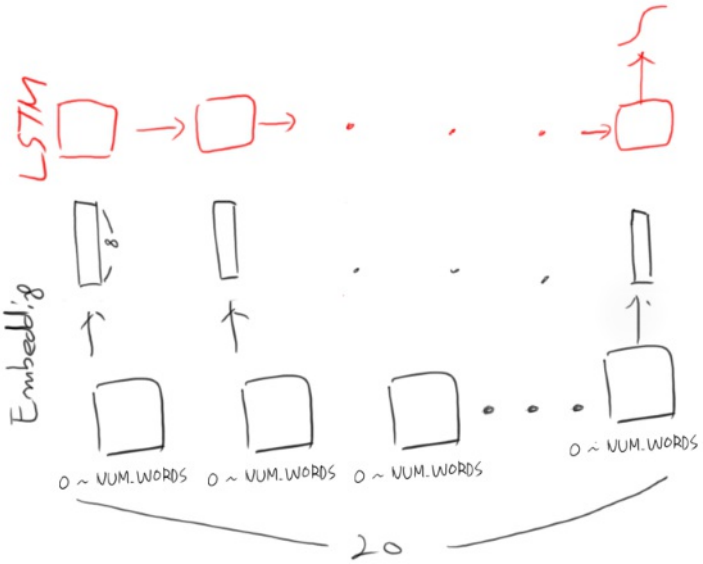
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 20, 8)             708632
_________________________________________________________________
lstm_1 (LSTM)                (None, 1)                 40
=================================================================
Total params: 708,672
Trainable params: 708,672
Non-trainable params: 0
_________________________________________________________________
```

```
rnn.compile(optimizer=RMSprop(), loss='binary_crossentropy', metrics=['accuracy'])

rnn.fit(
    x_train_seq,
    y_train,
    epochs=30,
    batch_size=128,
    validation_split=0.2,
    callbacks=[EarlyStopping(monitor='val_acc')])
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/30
20000/20000 [==============================] - 4s 224us/step - loss: 0.7062 - acc: 0.5050 - val_loss: 0.6787 - val_acc: 0.5482
Epoch 2/30
20000/20000 [==============================] - 4s 201us/step - loss: 0.6631 - acc: 0.6223 - val_loss: 0.6553 - val_acc: 0.6420
Epoch 3/30
20000/20000 [==============================] - 4s 200us/step - loss: 0.6331 - acc: 0.6949 - val_loss: 0.6328 - val_acc: 0.6612
Epoch 4/30
20000/20000 [==============================] - 4s 196us/step - loss: 0.6028 - acc: 0.7232 - val_loss: 0.6132 - val_acc: 0.6720
Epoch 5/30
 4736/20000 [======>.......................] - ETA: 2s - loss: 0.5860 - acc: 0.740320000/20000 [==============================] - 4s 193us/step
Epoch 6/30
20000/20000 [==============================] - 4s 193us/step - loss: 0.5533 - acc: 0.7536 - val_loss: 0.5915 - val_acc: 0.6940
Epoch 7/30
20000/20000 [==============================] - 4s 200us/step - loss: 0.5381 - acc: 0.7625 - val_loss: 0.5917 - val_acc: 0.6970
Epoch 8/30
20000/20000 [==============================] - 4s 199us/step - loss: 0.5220 - acc: 0.7719 - val_loss: 0.5911 - val_acc: 0.7050
Epoch 9/30
13184/20000 [==================>...........] - ETA: 1s - loss: 0.5087 - acc: 0.780520000/20000 [==============================] - 4s 204us/step
Epoch 10/30
20000/20000 [==============================] - 4s 200us/step - loss: 0.4956 - acc: 0.7880 - val_loss: 0.5857 - val_acc: 0.7138
Epoch 11/30
20000/20000 [==============================] - 4s 198us/step - loss: 0.4843 - acc: 0.7988 - val_loss: 0.5796 - val_acc: 0.7204
Epoch 12/30
20000/20000 [==============================] - 4s 201us/step - loss: 0.4741 - acc: 0.8054 - val_loss: 0.5803 - val_acc: 0.7238
Epoch 13/30
13440/20000 [===================>..........] - ETA: 1s - loss: 0.4669 - acc: 0.810520000/20000 [==============================] - 4s 203us/step
Epoch 14/30
20000/20000 [==============================] - 4s 198us/step - loss: 0.4532 - acc: 0.8149 - val_loss: 0.5778 - val_acc: 0.7268
Epoch 15/30
20000/20000 [==============================] - 4s 195us/step - loss: 0.4449 - acc: 0.8196 - val_loss: 0.5784 - val_acc: 0.7288
Epoch 16/30
20000/20000 [==============================] - 4s 196us/step - loss: 0.4355 - acc: 0.8238 - val_loss: 0.5797 - val_acc: 0.7288
```

```
<keras.callbacks.History at 0x7f5763ab8828>
```

12.8.4. RNN 뒤에 Dense 레이어 붙이기

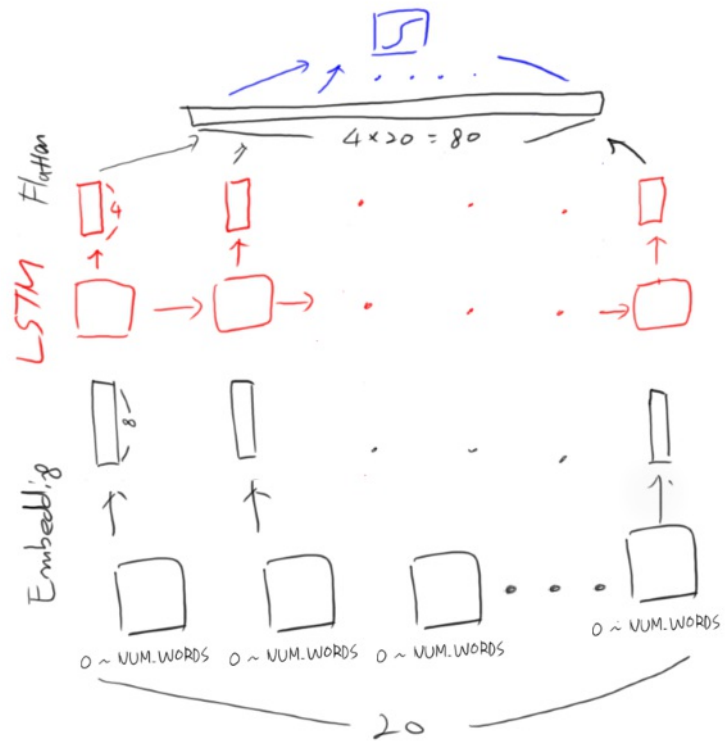



```
rnn2 = Sequential()
rnn2.add(Embedding(input_dim=NUM_WORDS, output_dim=8, input_length=MAXLEN))
rnn2.add(LSTM(32, activation='tanh', return_sequences=False))
rnn2.add(Dense(1, activation='sigmoid'))

rnn2.compile(optimizer=RMSprop(), loss='binary_crossentropy', metrics=['accuracy'])

rnn2.fit(
    x_train_seq,
    y_train,
    epochs=30,
    batch_size=128,
    validation_split=0.2,
    callbacks=[EarlyStopping(monitor='val_acc')])
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/30
20000/20000 [==============================] - 6s 277us/step - loss: 0.6232 - acc: 0.6435 - val_loss: 0.5592 - val_acc: 0.7134
Epoch 2/30
20000/20000 [==============================] - 5s 236us/step - loss: 0.4734 - acc: 0.7753 - val_loss: 0.5151 - val_acc: 0.7384
Epoch 3/30
20000/20000 [==============================] - 5s 235us/step - loss: 0.4021 - acc: 0.8175 - val_loss: 0.5005 - val_acc: 0.7474
Epoch 4/30
15232/20000 [=====================>........] - ETA: 1s - loss: 0.3554 - acc: 0.848520000/20000 [==============================] - 5s 235us/step
Epoch 5/30
20000/20000 [==============================] - 5s 237us/step - loss: 0.3248 - acc: 0.8640 - val_loss: 0.5190 - val_acc: 0.7544
Epoch 6/30
20000/20000 [==============================] - 5s 238us/step - loss: 0.3016 - acc: 0.8744 - val_loss: 0.5085 - val_acc: 0.7556
Epoch 7/30
20000/20000 [==============================] - 5s 240us/step - loss: 0.2813 - acc: 0.8858 - val_loss: 0.5300 - val_acc: 0.7486
```

```
<keras.callbacks.History at 0x7f5763af1f98>
```

12.8.5. RNN의 모든 노드로부터 출력을 받아 Dense 레이어로 합치기



```
rnn3 = Sequential()
rnn3.add(Embedding(input_dim=NUM_WORDS, output_dim=8, input_length=MAXLEN))
rnn3.add(LSTM(4, activation='tanh', return_sequences = True))
rnn3.add(Flatten())
rnn3.add(Dense(1, activation='sigmoid'))

rnn3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 20, 8)             708632
_________________________________________________________________
lstm_4 (LSTM)                (None, 20, 4)             208
_________________________________________________________________
flatten_4 (Flatten)          (None, 80)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 81
=================================================================
Total params: 708,921
Trainable params: 708,921
Non-trainable params: 0
_________________________________________________________________
```

```
rnn3.compile(optimizer=RMSprop(), loss='binary_crossentropy', metrics=['accuracy'])

rnn3.fit(
    x_train_seq,
    y_train,
    epochs=30,
    batch_size=128,
    validation_split=0.2,
    callbacks=[EarlyStopping(monitor='val_acc')])
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/30
20000/20000 [==============================] - 5s 250us/step - loss: 0.6585 - acc: 0.6235 - val_loss: 0.6023 - val_acc: 0.6630
Epoch 2/30
20000/20000 [==============================] - 4s 215us/step - loss: 0.5287 - acc: 0.7389 - val_loss: 0.5528 - val_acc: 0.7108
Epoch 3/30
20000/20000 [==============================] - 4s 213us/step - loss: 0.4497 - acc: 0.7915 - val_loss: 0.5350 - val_acc: 0.7258
Epoch 4/30
17664/20000 [=========================>....] - ETA: 0s - loss: 0.3937 - acc: 0.826220000/20000 [==============================] - 4s 211us/step
```