Recurrent Neural Networks

https://youtu.be/D9WEP6tvyxs

Contents

Recurrent Neural Network

vanilla RNN

LSTM

implementation using keras (RNN,LSTM)

CryptoCraft LAB

Sequential patterns

- **Text**
  - text to text
  - 구, 절 단위로 번역(단어 단위가 아님)
- **Speech**
  - speech to text
  - Siri, Bixby …
- **Audio, Video**
- **Physical processes**
- 시간에 따라 달라지는 시계열 데이터의 과거와 현재의 종속 관계를 분석하여 예측 : 앞뒤 문맥 파악하여 번역하는 느낌
- Simple RNN, LSTM, GRU 등의 모델
- 주가 분석, 기상 예측, 번역 등에 사용

Sequence modeling

- one to one
  - 한 시점의 데이터만 입력 받아 하나의 출력을 생성
  - fixed size input, output
  - image classification
- one to many
  - 한 시점의 데이터만 입력 받아 sequence output 생성
  - image captioning*
- many to one
  - sequence input을 받아 하나의 출력을 생성
  - sentiment analysis*
- many to many
  - machine translation
    - sequence input, sequence output (sequence of words → sequence of words)
    - 한국어 → 영어 번역
  - syncing video image
    - synced sequence input, synced sequence output
    - video classification on frame level*

*video classification on frame level

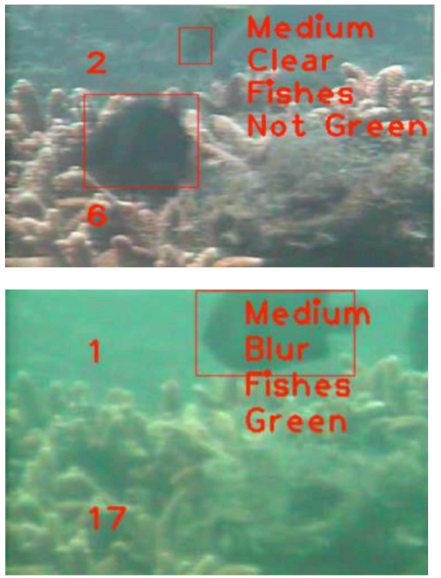


*image captioning

: 텍스트로 이미지를 설명하는 것

한 이미지 안의 여러 객체를 인식

길에 서있는 닭 두마리

*sentiment analysis(감성 분석)

: text등의 sequence of vectors

→ sentiment class

: 영화 리뷰, 감정 파악

data 예시

- **배열을 사용하여 벡터 형식으로 입력**

각 단어를 정수로 변환하여 나열

```
[1, 27595, 28842, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186, 90, 67, 7, 89, 5, 19, 102, 6, 19, 12
4, 15, 90, 67, 84, 22, 482, 26, 7, 48, 4, 49, 8, 864, 39, 209, 154, 6, 151, 6, 83, 11, 15, 22, 155, 11, 15, 7, 48,
9, 4579, 1005, 504, 6, 258, 6, 272, 11, 15, 22, 134, 44, 11, 15, 16, 8, 197, 1245, 90, 67, 52, 29, 209, 30, 32, 132,
6, 109, 15, 17, 12]
3
```

aptsellindex_gangnamgu

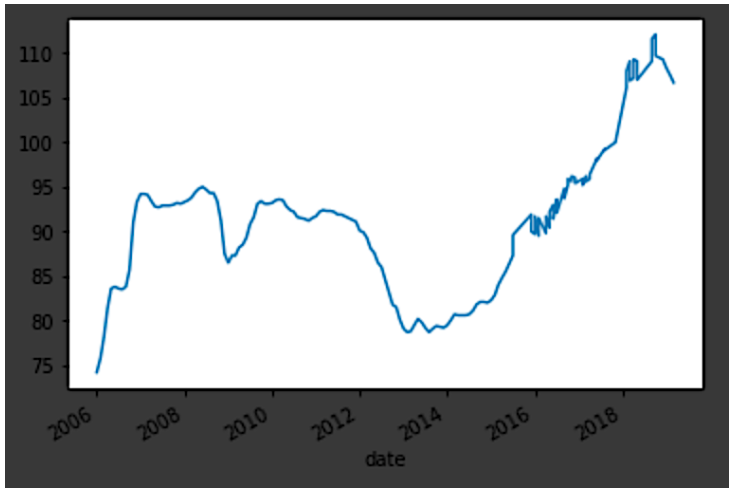
date	trade_price_idx_value
2006-01-01	74.200000000000003
2006-02-01	75.799999999999997
2006-03-01	78.099999999999994
2006-04-01	81.400000000000006
2006-05-01	83.599999999999994
2006-06-01	83.799999999999997
2006-07-01	83.599999999999994
2006-08-01	83.5
2006-09-01	83.799999999999997
2006-10-01	85.700000000000003
2006-11-01	91
2006-12-01	93.299999999999997
2007-01-01	94.200000000000003
2007-02-01	94.200000000000003
2007-03-01	94.099999999999994
2007-04-01	93.400000000000006

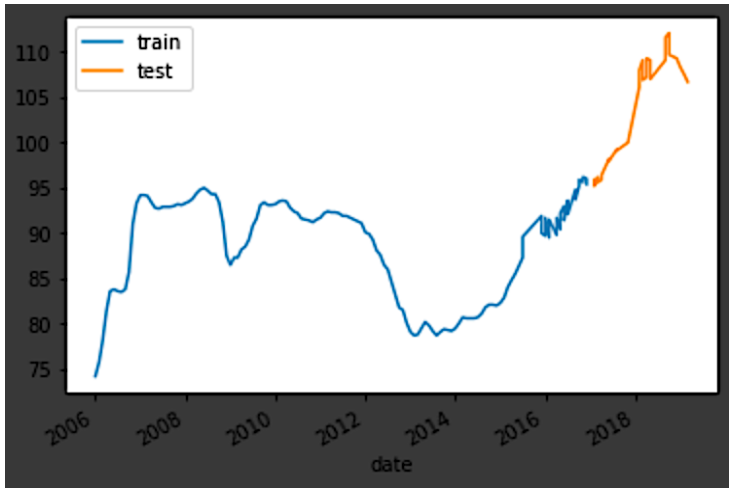
csv파일

pandas를 통해 특정 시점을 기준으로 train, test 데이터 분할 가능

```
import pandas as pd
split_date = pd.Timestamp('01-01-2017')
```

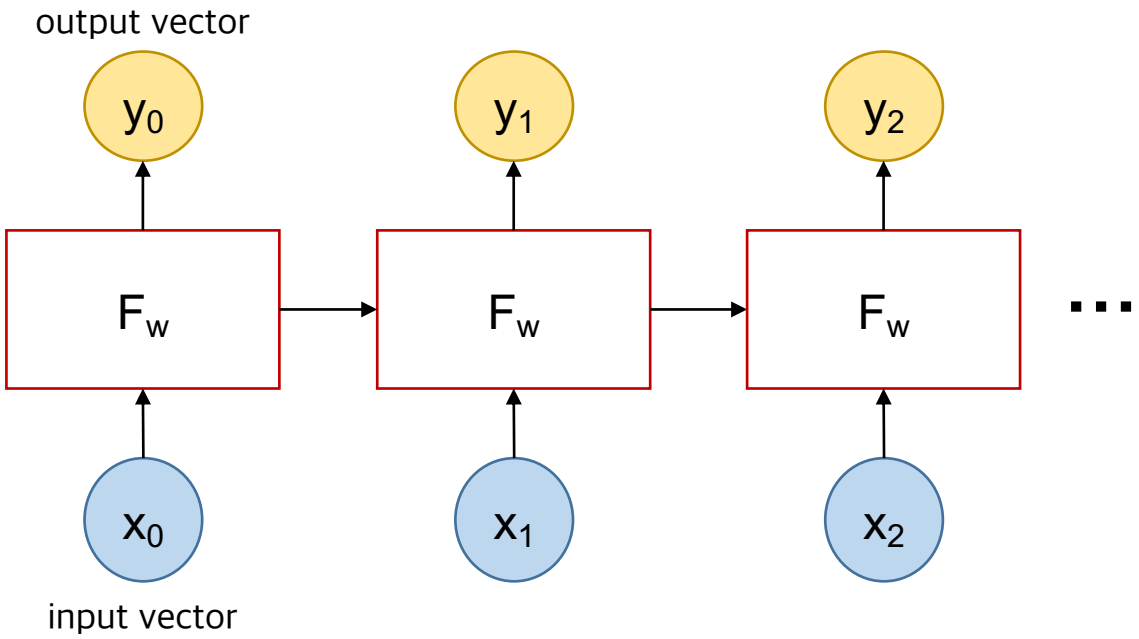
시간에 따른 아파트 거래가격 지수



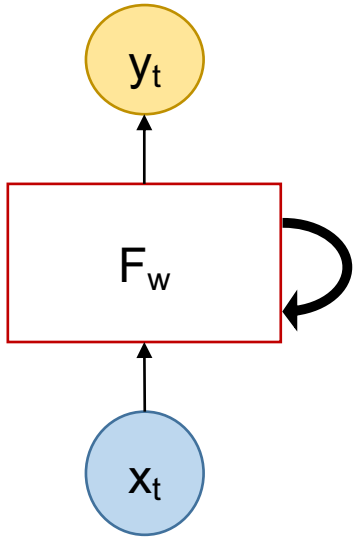


100x100크기의 이미지의 경우
100개로 나누어서
100차원의 벡터 100개로 변형
하여 RNN에 입력 가능

basis form of RNN



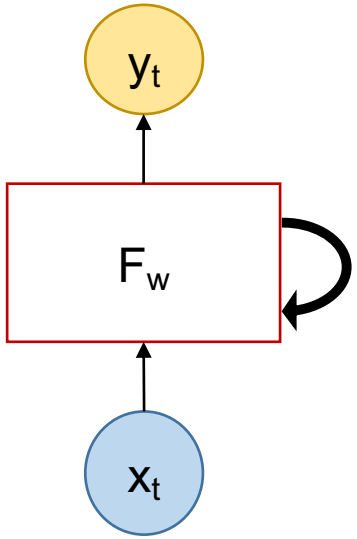
- 이전 함수(F)의 모든 요소가 다음 함수에 영향을 줌
- y_0은 x_0에 영향 받음, y_1은 x_0, x_1에 영향 받음 …
- y는 다음 함수로 들어감
- y(output)는 전체 x(input)과 y(output)에 영향 받음



$$y_t = Fw(xt, y_{t-1})$$

Recurrent Neural Networks

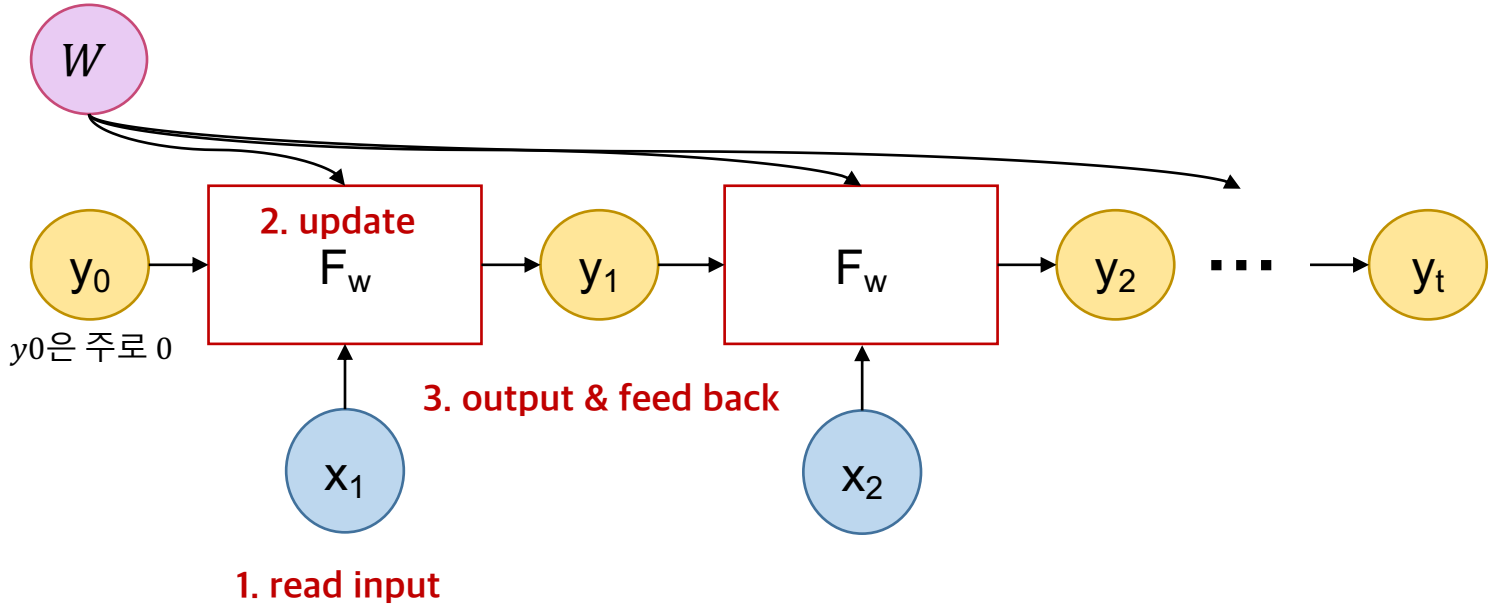
plain/vanilla RNN



$$y_t = Fw(xt, y_{t-1})$$

$$y_t = tanh(w_{xy}x_t + w_{yy}y_{t-1})$$

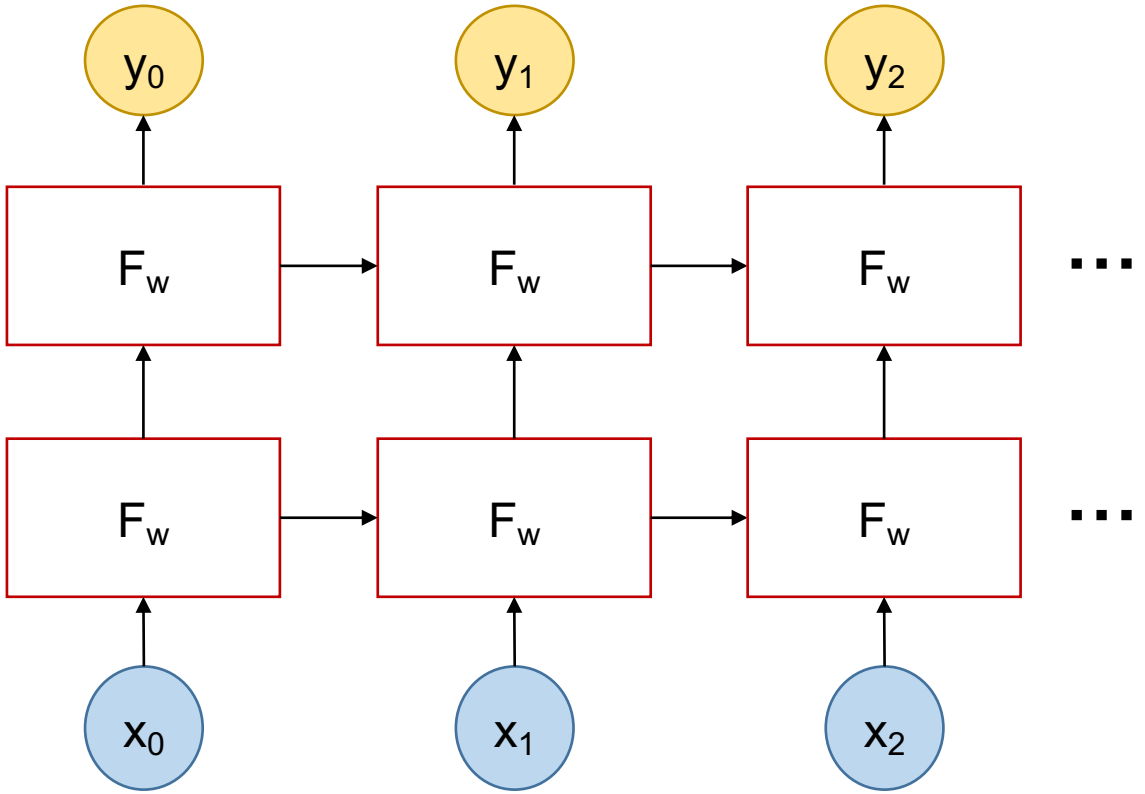
*모든 F에 동일한 weight 적용



➢ 매 단계마다 이러한 recurrence formula 적용

stacked RNN

- 여러 층으로 구성하여 다중 RNN 설계 가능



activation function (tanh)

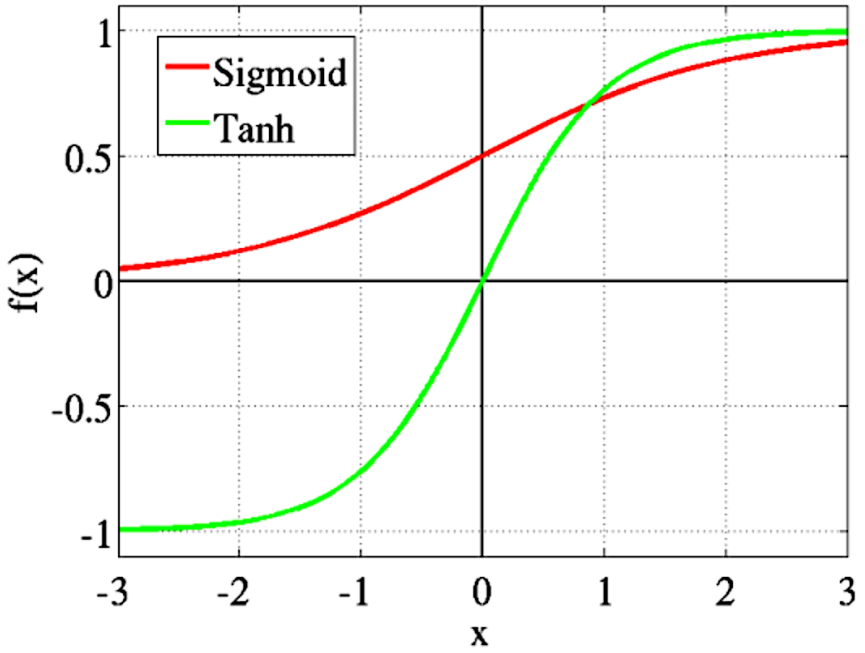
- activation function
  - 입력신호가 일정 기준 이상이면 다음 뉴런으로 보내는데 그 신호를 결정 해주는 것
- sigmoid를 보완하기 위한 활성화 함수 (성능 향상)
- 입력신호를 -1 ~ 1 사이의 값으로 normalization
  - sigmoid는 0 ~ 1 사이의 값 반환
- binary classification에 주로 사용
- x = 0에서 기울기 1로 최대값 (sigmoid는 0.25)
- Gradient Vanishing, Exploding(기울기 소실,폭발)의 문제점 존재
  - 각 layer의 값을 미분하여 input layer까지 값을 전달 (역전파 방식으로 학습)

    → layer을 거쳐갈 때마다 곱셈 연산 (최대값이 1인 기울기들을 계속해서 곱하면 0에 수렴)

    → layer가 많아짐에 따라 기울기가 사라져 버리는 것

    → 그 반대로 기울기가 1이상인 경우가 계속해서 곱해지면 발산

$$tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



RNN의 문제점

- 레이어를 거슬러 올라가며 기울기 소실, 폭발이 발생하면 장기 종속성을 확보할 수 없어 정보 연결이 어려워짐 (제대로 학습되지 않음)

1. 오늘은 날씨가 맑아서 하늘이 (파랗다).

2. 나는 한국인이지만 태어났을 때부터 20년간 영국에 살아서 (한국어)를 잘 못한다.

→ 괄호를 예측한다고 할 때,

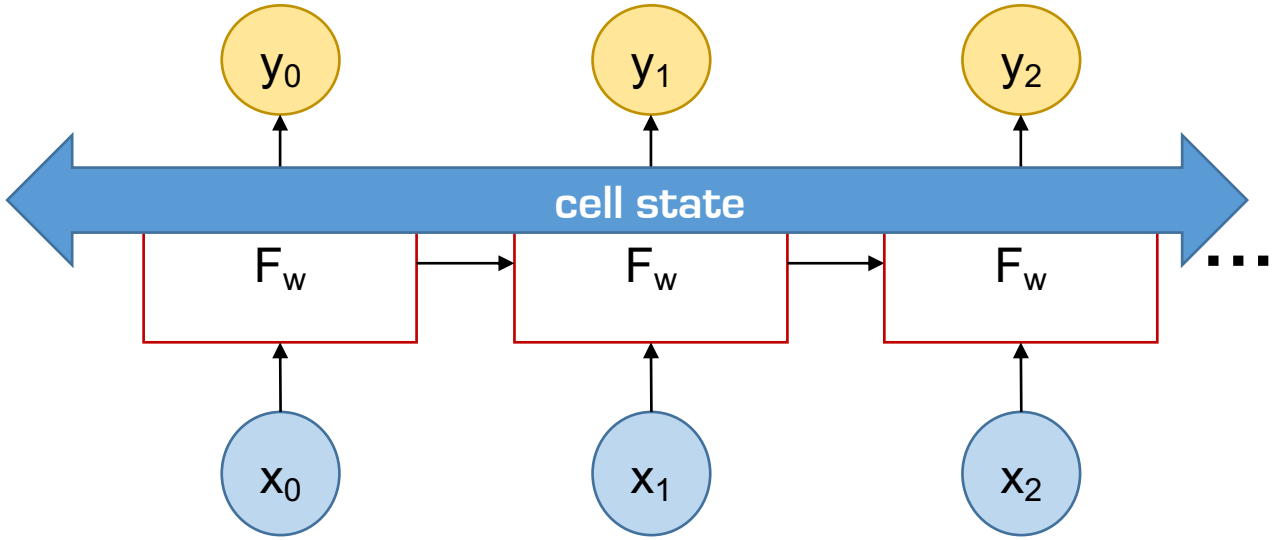
1번의 맑아서 하늘이 파랗다는 단어의 간격 짧아서 비교적 문맥 파악이 쉬움

2번의 한국인과 한국어는 장기 종속성이 확보되지 않으면 정보 연결이 되지 않아 예측이 어려워짐

LSTM

- RNN에서 발생하는 vanishing/exploding gradient 문제 해결 위해 고안
- RNN과 다르게 LSTM에는 cell state라는 추가적인 context pipeline 존재

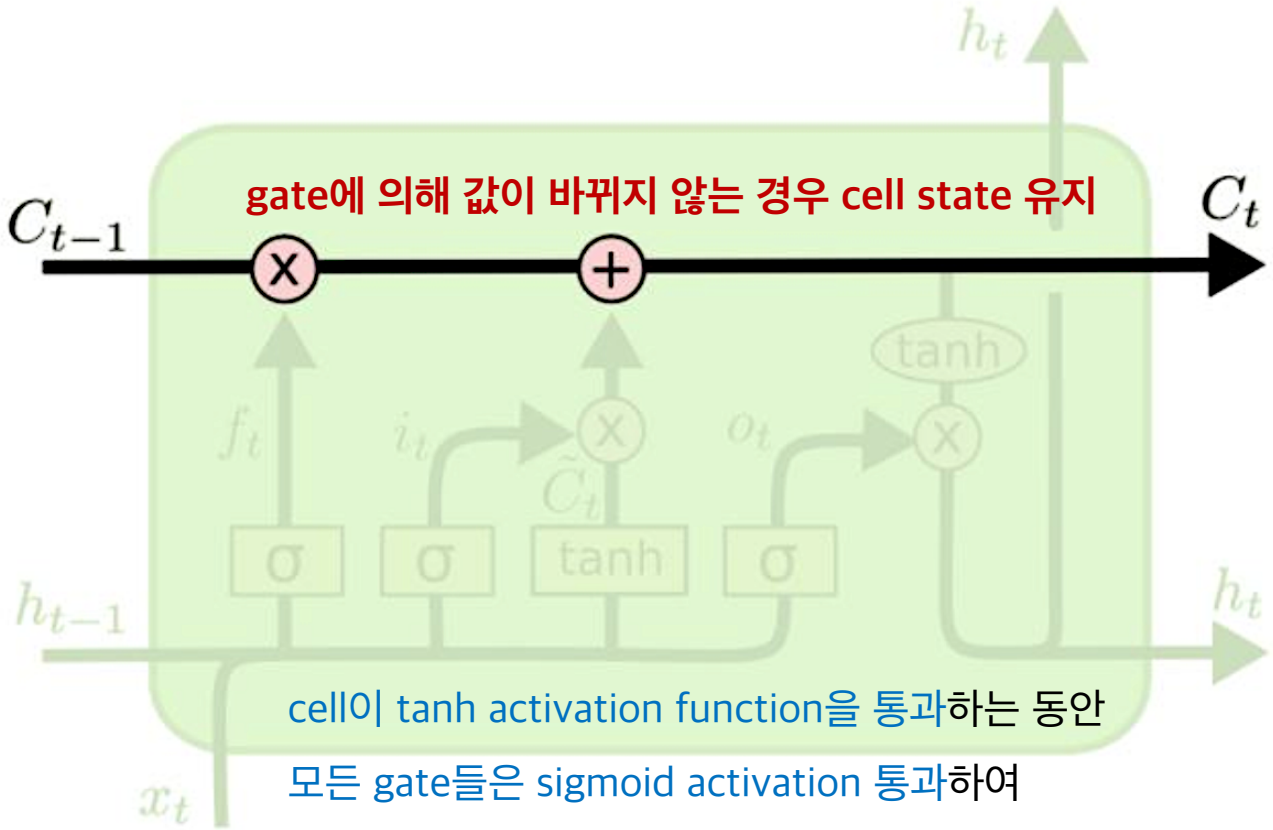
  → context 유지 가능하여 장기 종속성 확보
- 하나의 LSTM 유닛은 여러 개의 gate들이 연결되어있는 cell들로 구성
- gate들은 0에서 1사이의 값
  - Forget Gate
  - Input gate
  - Output Gate
  - Cell state
- 데이터가 적은 경우 언더피팅이 존재한다고 함



LSTM

C_{t-1} : 이전 LSTM unit의 memory

h_{t-1} : 이전 LSTM unit의 output



cell이 tanh activation function을 통과하는 동안

모든 gate들은 sigmoid activation 통과하여

0~1 사이의 값으로 결정

LSTM Equations

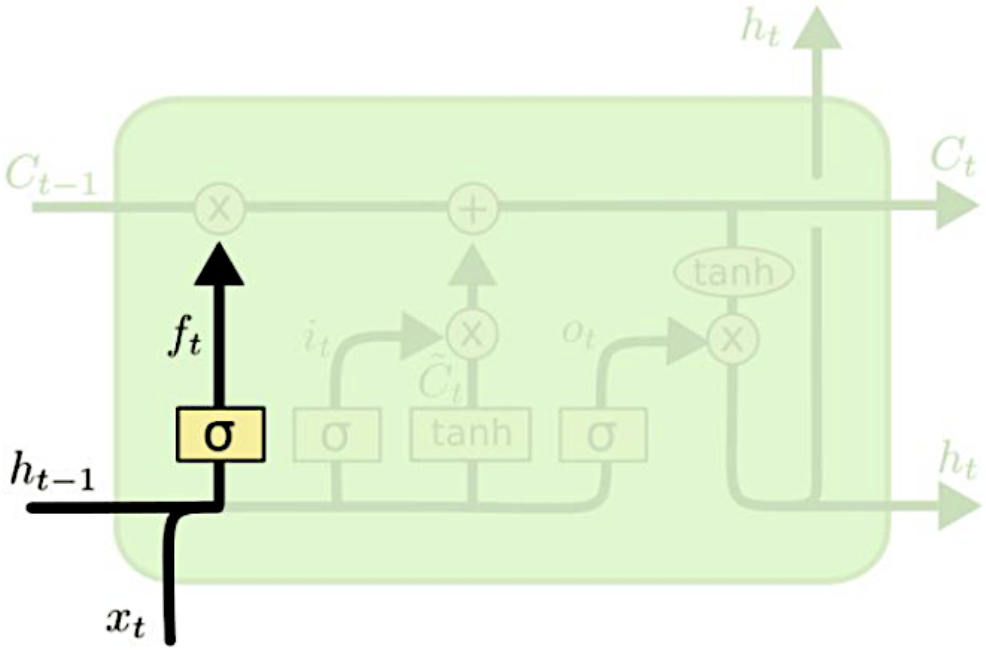
gate

input에 가중치 적용한 후 활성화 함수 통과

$$\begin{pmatrix} input \\ forget \\ output \\ cell \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} w \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + bias$$

*모든 gate와 cell은 own weight, bias가 존재하며, 요소별로 곱셈 수행

LSTM



forget gate

입력 값을 sigmoid activation 통과시켜 0 ~ 1 사이의 값으로 결정

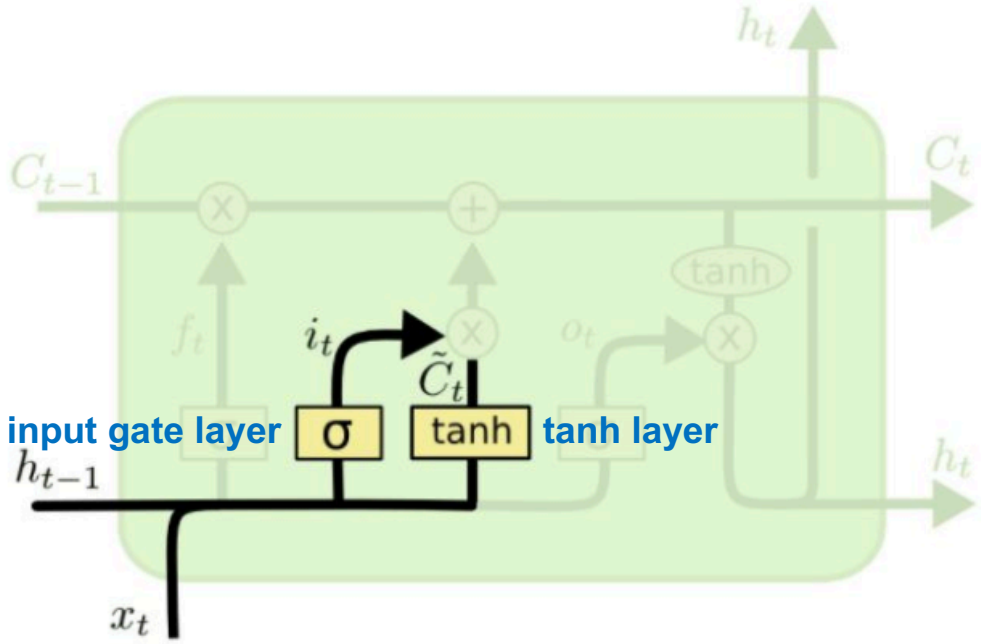
0 : cell state 잊음

1 : cell state 유지

ex) an apple, apples 중, 앞의 문맥을 기억하여 예측

기억할 필요가 없는 경우 forget

LSTM



input gate

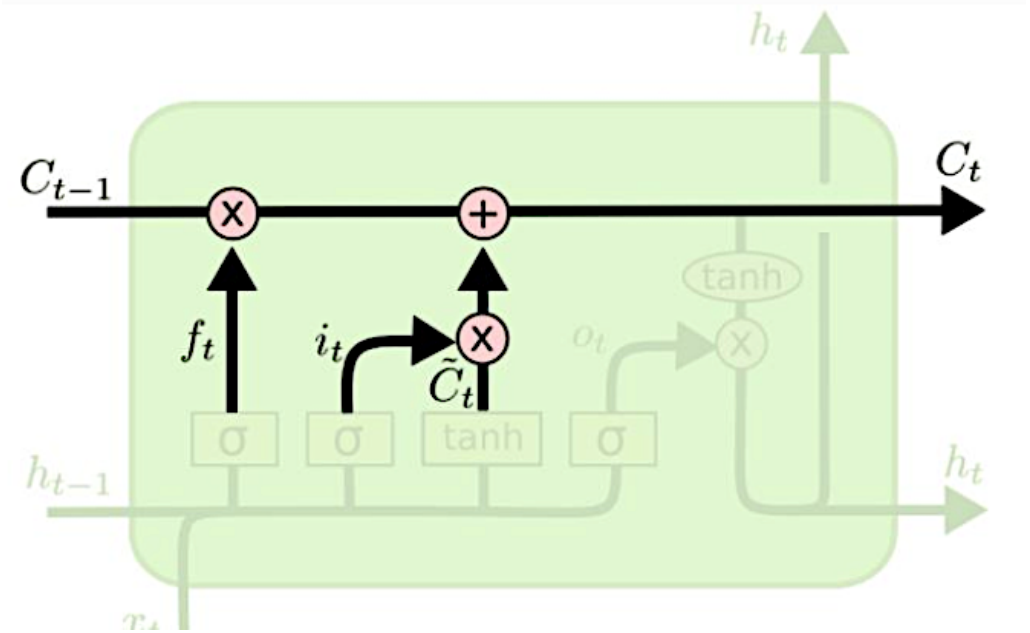
어떤 정보를 cell state에 담을 것인지 결정

sigmoid activation에 의해 결정

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

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$ *cell state 후보 값들

LSTM



input gate

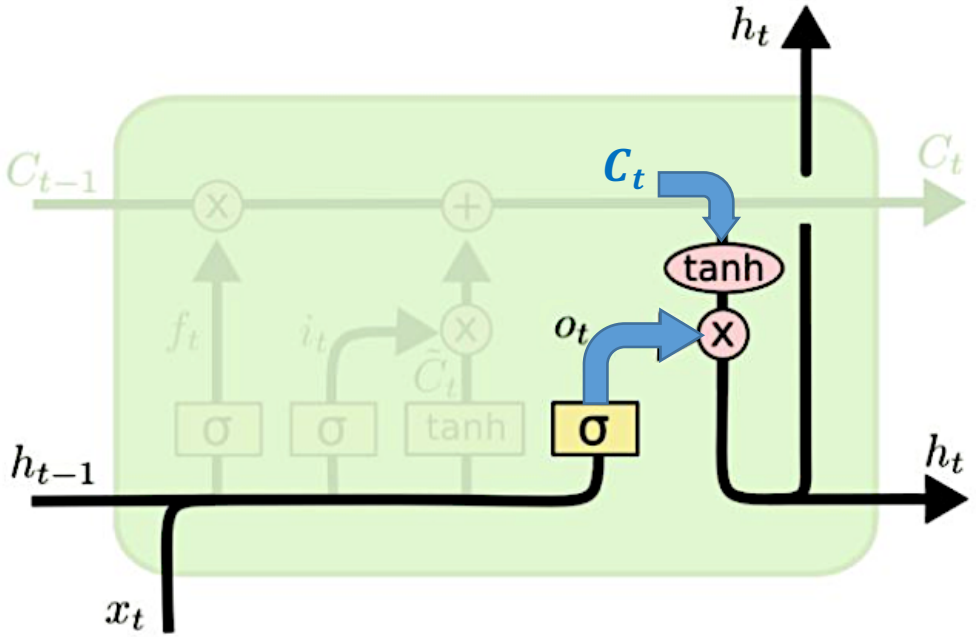
앞의 두 값을 곱하여 업데이트할 값 결정

forget gate에서 결정한 값을 더하여 반영

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

forget gate의 값 cell state 후보 값과 input gate 값

LSTM



output gate

출력 값을 결정하는 단계

입력 값에 sigmoid activation을 적용시켜 값 결정(o_t)

cell state(C_t)에 tanh activation을 적용시켜 -1~1사이의 값으로 반환

두 값을 곱하여 출력 값 결정 → 다음 LSTM의 입력 값으로 들어감

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

$$h_t = o_t * \tanh\left(C_t\right)$$

vanilla RNN (keras)

```
def vanilla_rnn():
    model = Sequential()
    model.add(SimpleRNN(50, input_shape = (49,1), return_sequences = False ))
    model.add(Dense(46))
    model.add(Activation('softmax'))

    model.compile(loss = 'categorical_crossentropy', optimizer = Adam(lr = 0.001), metrics = ['accuracy'])

    return model
```

RNN의 기본 activation function은 tanh

False : 마지막 timestep에서만 output 출력
True : 모든 timestep에서 output 출력

다음 LSTM unit이 없을 경우 False (반대는 True)

CNN에서 compile했던 방법과 동일

```
def vanilla_rnn():
    model = Sequential()
    model.add(Bidirectional SimpleRNN(50, input_shape = (49,1), return_sequences = False)))
    model.add(Dense(46))
    model.add(Activation('softmax'))

    model.compile(loss = 'categorical_crossentropy', optimizer = Adam(lr = 0.001), metrics = ['accuracy'])

    return model
```

이전의 값 뿐만 아니라 이후의 값으로도 예측 가능한 양방향 순환 신경망(Bidirectional RNN)

stacked vanilla RNN (keras)

```
def stacked_vanilla_rnn():
    model = Sequential()
    model.add(SimpleRNN(50, input_shape = (49,1), return_sequences = True))
    model.add(SimpleRNN(50, return_sequences = False))
    model.add(Dense(46))
    model.add(Activation('softmax'))

    model.compile(loss = 'categorical_crossentropy', optimizer = Adam(lr = 0.001), metrics = ['accuracy'])

    return model
```

simple RNN을 다층으로 구성

ex) 100개의 입력데이터가 (49,1)의 input shape을 가짐
(100,49,1) → (49,1)의 shape을 가진 input이 100개이고 하나의 sequence가 됨

LSTM (keras)

```
from keras.layers import LSTM

def lstm():
    model = Sequential() LSTM 또한 simple RNN처럼 import하여 사용    return_sequence 동일한 방식
    model.add LSTM 50, input_shape = (49,1), return_sequences = False))
    model.add(Dense(46))
    model.add(Activation('softmax'))

    adam = optimizers.Adam(lr = 0.001)
    model.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics = ['accuracy'])

    return model
```

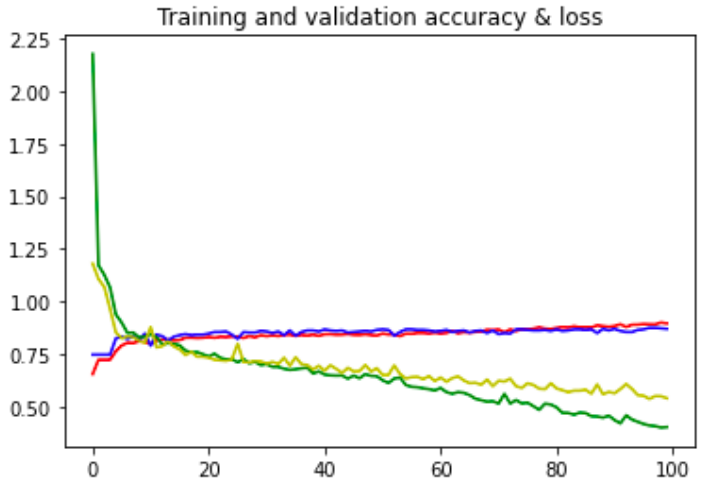
stacked LSTM (keras)

```
def stacked_lstm():
    model = Sequential() 여러층으로 쌓을 수 있음
    model.add(LSTM(50, input_shape = (49,1), return_sequences = True))
    model.add(LSTM(50, return_sequences = False))
    model.add(Dense(46))
    model.add(Activation('softmax'))

    adam = optimizers.Adam(lr = 0.001)
    model.compile(loss = 'categorical_crossentropy', optimizer = adam, metrics = ['accuracy'])

    return model
```

training



학습 방법은 동일

model.fit을 통해 학습 가능

```
model = stacked_vanilla_rnn()
history = model.fit X_train, y_train, epochs=200, batch_size = 50, validation_data=(X_val, y_val), verbose=2)
```

```
Epoch 172/200
 - 1s - loss: 0.4333 - accuracy: 0.8623 - val_loss: 0.7482 - val_accuracy: 0.8246
Epoch 173/200
 - 1s - loss: 0.4360 - accuracy: 0.8651 - val_loss: 0.8023 - val_accuracy: 0.7945
Epoch 174/200
 - 1s - loss: 0.4649 - accuracy: 0.8484 - val_loss: 0.8162 - val_accuracy: 0.8195
Epoch 175/200
 - 1s - loss: 0.4389 - accuracy: 0.8545 - val_loss: 0.7652 - val_accuracy: 0.8271
```

test

***keras에서 제공하는 로이터 뉴스 분류 데이터 셋으로 실험**

46개의 토픽을 갖고 있으며 각 뉴스기사마다 하나의 토픽 가짐, 각 뉴스를 토픽으로 분류

model.predict을 통해 예측 가능

```
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})

y_predtest = model.predict(X_test)
print("actual label\n{test}".format(test=y_test))
print("\npredicted label\n{pred}".format(pred=(y_predtest)))
#임계값추가가능 print(Y_pred>0.5)
```

```
import numpy as np
rounded_test=np.argmax(y_test, axis=1)
print("\nactual label\n{test}".format(test=rounded_test))

rounded_pred=np.argmax(y_predtest, axis=1)
print("\npredicted label\n{pred}".format(pred=rounded_pred))
```

```
[f1_score(macro)]
0.36212524291994497

[f1_score(micro)]
0.87

[confusion_matrix]
[[  2   0   0   1   0   0   0   0   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0   0   0   0   0   0]
 [  0   0 146   1   0   0   0   0   0   0   0   0   2]
 [  1   1   1  20   2   0   0   0   0   1   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   0]
 [  1   0   0   0   0   1   0   0   0   0   0   0   0]
 [  0   1   0   0   0   0   1   0   0   0   0   0   0]
 [  1   0   0   1   0   0   0   0   0   0   0   0   0]
 [  0   0   2   1   0   0   0   0   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0   0   3   0   0   0]
 [  0   0   0   1   0   0   0   0   0   1   1   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   0]
 [  0   0   2   1   0   0   0   0   0   0   0   0   0]]

[precision_score]
0.4360410830999066

[recall_score]
0.3460843493109884

[precision_score(micro)]
0.87

[recall_score(micro)]
0.87
```

실제 label과 예측 label

```
actual label
[ 3  4  4  3  3  4  3  3  3  3  3  4  3  3  3  6 20  8  3  3  3  4  3  3
 19  3  3  3  8  3  3  3  3  3 16  4  3  3  3  3  4 20  3 19 13  4 24
  3  3  3  3  4  3  3  3  3  3  3 24  3  3  4  3  3  3  3  4  3  1  3  3
  3  3  3  3  4  3  3  3  4  4  3  3  3  4  3  3  3  3  3 10  4  3  1  4
 20  3  1  3  3  3  3  3  3  3  4  3  3  3  3  3  3  3  3  3  3  3  3  3
  3  3  3  3  3  3 19 13  3  3  3  3 16  4  3  3  3  3  3  3  3  3  3  3
  3  3  3  3  3  3  3  4  4  3  3  3  3  3  3  4  3  3  3  3  3  3  3  3
  3  3  3  4 23  3  3  3  3  4  3  3  3  3  3 24  3  3  4  4  3 19  3  3
 10 16  3  3  2  3  3  3]

predicted label
[ 3  4  4  3  3  4  3  3  3  3  3  6  3  3  3  4 19  1  3  3  3  4  3  3
 19  3  3 24  8  3  3  3  3  3  4  3  3  3  3  3  3  4 20  3 19  1  1  4
  3  3  3  3  4  3  3  3  3  3  3  3  3  3  6  3  3  3  3  4  3  1  3  3
  3  3  3  3  4  4  3  3  4  4  3  3  3  4  3  3  3  3  3 10 19  3  1  4
  4  3  4  3  3  3  3  3  3  3  4  3  3  3  3  3  3  3  3  3  3  3  3  3
  3  3  3  3  3  3 19  4  3  3  3  3  4  3  3  3  3  3  3  3  3  3  3  3
  3  3  3  3  3  3  3  4  4  3  3  3  3  3  3  4 24  3  3  3  3  3  3  3
  3  3  3  4  4  3  3  3  3  2  3  3  3  3  3  3  3  3  4  4  3  3  3  3
  2  3  3  3  3  3  3  3]
```

Q & A