Deep Learning IV

김연지
kimyeonji3@gmail.com

텍스트 학습/분석 …

- 말뭉치 (corpus)
- 자연언어 연구를 위해 특정한 목적을 가지고 언어의 표본을 추출한 집합
  - https://corpus.korean.go.kr/
- 안녕하세요. 좋은 아침입니다.
- 1. 문장단위 : 안녕하세요. / 좋은 아침입니다 > 2
- 2. 단어단위: 안녕/하세요./

  좋은/아침/입니다. >5

  품사단위로 쪼개서 분석을 시키는게 가장 정확도가 높다

- 3. 음절단위: 안/녕/하/세/요/./좋/은/아/…… > 14

형태소 : 뜻을 가진 가장 작은 말의 단위.

음절 : 하 / 느 / 리 / 참 / 놉 / 꼬 / 푸 / 르 / 다 → 9개
형태소 : 하늘 / 이 / 참 / 높 / 고 / 푸르 / 다 → 7개
단어 : 하늘 / 이 / 참 / 높고 / 푸르다 → 5개
어절 : 하늘이 / 참 / 높고 / 푸르다 → 4개

[네이버 지식백과] 형태소 (3일만에 끝내는 국어문법교과서, 2014. 1. 15., 강승임)

Tokenizer : 입력문장을 일정한 단위로 분할

Char Tokenizer

장점

• 모든 문장을 적은 수의 Vocabulary로 표현할 수 없음
• Vocabulary에 글자가 없어서 '[UNK]'로 표현해야 하는 OOV(Out of Vocabulary) 문제가 발생할 가능성이 낮음

• 단점

• 글자 단위로 분할하기 때문에 token 수가 많아짐
token 수가 많으면 연산이 많아지고 학습속도가 늦어짐
• 각 글자 하나하나를 벡터로 표현할 경우 단어의 의미를 표현한다고 할 수 없음

예)
• 이런 문제를 해결하기 위해서 char 기반의 Neural Network은 많은 layer를 필요로 함

Tokenizer : 입력문장을 일정한 단위로 분할

Word Tokenizer

- 장점
  띄어쓰기 단위로 분할하기 때문에 token 수가 적음

- 단점
  어미변화로 인해 유사 단어들이 많아짐

‘책’, ‘책을’, ‘책에다’, ‘책에서’, ‘책이’ 등
‘play’, ‘plays’, ‘playing’, ‘played’ 등

- 어미변화로 인한 단어를 표현하는 벡터들이 다른 의미를 가질 수 있음
- Vocabulary 개수가 매우 많아짐

- 메모리 사용량 증가
- 연산량 또한 증가

Tokenizer : 입력문장을 일정한 단위로 분할

Morph Tokenizer
형태소 단위로 분할하는 방법

- 장점

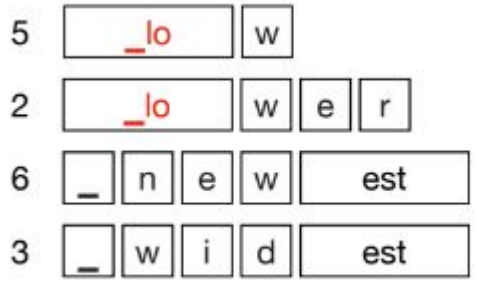
- 형태소 단위로 분할하기 때문에 token 들이 적당한 의미를 가짐
- Word tokenizer와 char tokenizer의 중간쯤의 token 개수를 가짐

- 단점

- 형태소 분석기의 발전 속도가 언어의 발전 속도에 비해서 느림
- 형태소 분석기들이 어느 정도의 오류를 가지고 있음
- Word에 비해서는 vocabulary 개수가 많이 줄어들지만 여전히 많음

Tokenizer : 입력문장을 일정한 단위로 분할

- BPE(Byte Pair Encoding)
  - 압축 알고리즘을 이용하여 띄어쓰기 단위의 단어를 subword 단위로 분할
  - 빈도수가 가장 많은 subword pair를 새로운 subword로 변환



• 장점
  • 말뭉치가 있다면 비교적 간단하게 만들 수 있음
  • 적은 수의 vocabulary를 가지고 OOV(Out of Vocabulary)를 최소화

• 단점
  • Subword의 분할이 의미 기준이 아닐 수 있음
  • '수원에'라는 문장을 분할할 때 ['_수원', '에']가 아닌 ['_수', '원에']로 분할 될 수 있음

'대한민국을', '대한민국은', '대한민국으로' 등의 빈도수가 많은 단어들은
['대한민국', '을', '은', '으로'] 형태가 아닌
['대한민국을', '대한민국은', '대한민국으로'] 그대로 분류되기도 함

Sentencepiece

Google에서 제공하는 Tokenizer tool

- https://arxiv.org/abs/1808.06226
- https://github.com/google/sentencepiece
- Char, word, BPE, Unigram 등 다양한 Tokenizer 제공
- https://github.com/google/sentencepiece/blob/master/python/

Sentencepiece_python_module_example.ipynb

Sentencepiece with Morph

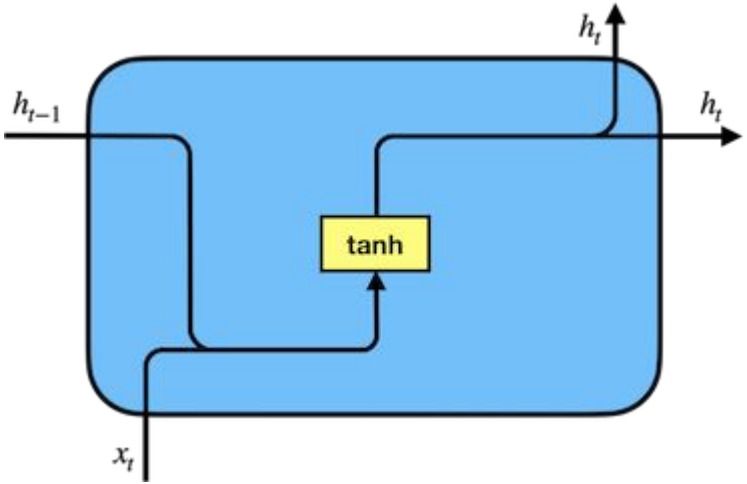
형태소 분석기와 sentencepiece를 동시에 사용

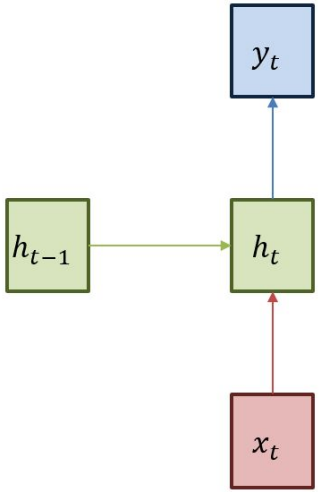
- 형태소 분석기로 문장을 우선 tokenize
- sentencepiece를 이용해 다시한번 tokenize

- 성능이 sentencepiece를 그냥 사용하는 것보다 성능이 좋다고 알려져 있음

RNN(Recurrent Neural Network)

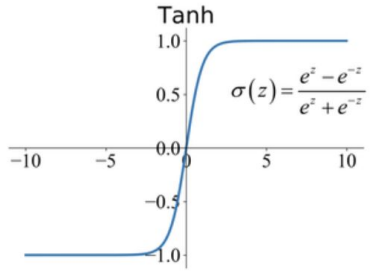
- 순서가 있는 데이터를 처리하기 위한 Neural Network
- 순서가 있는 데이터는 음성, 언어, 주가 등 발생 순서가 중요한 데이터를 의미
- 예시. 문장에서 이전에 발생한 단어를 보고 다음 단어를 예측하는 경우



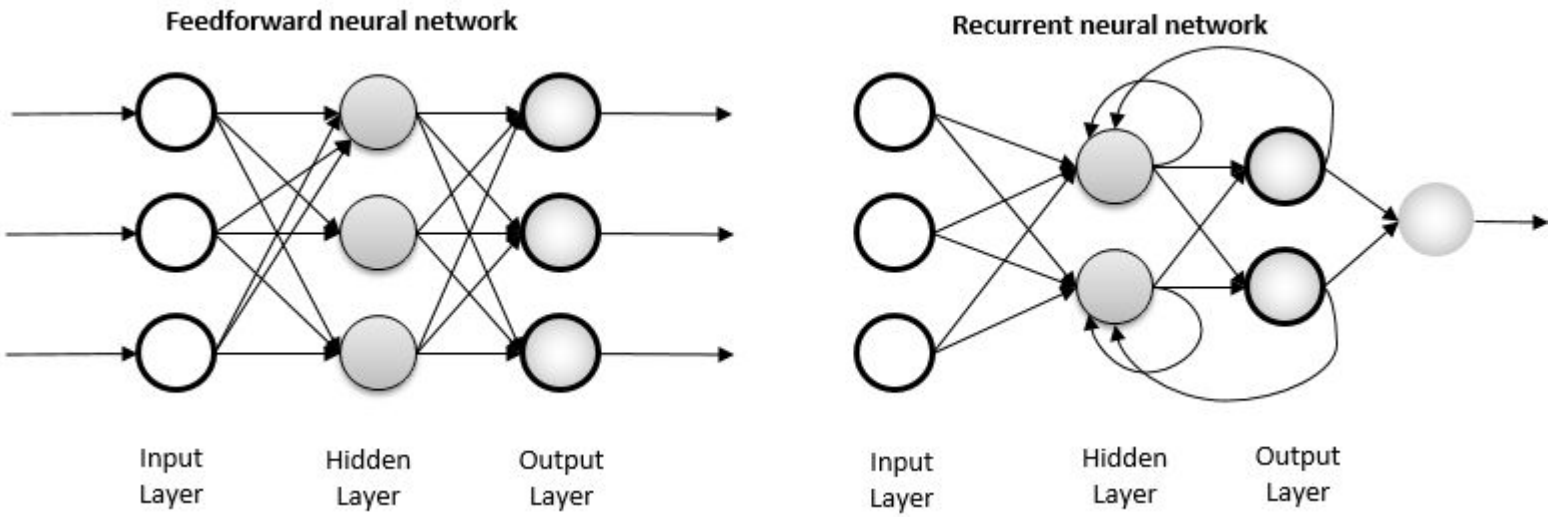


$$y_t = W_{hy}h_t + b_y$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



RNN(Recurrent Neural Network, 순환신경망)



https://www.researchgate.net/figure/Feed-forward-and-recurrent-ANN-architecture_fig1_315111480

RNN(Recurrent Neural Network, 순환신경망)

Feed Forward Network vs Recurrent Network

Feed Forward Network

일반적인 구조의 신경망

입력 → 은닉 → 출력층 으로 이어지는 단방향 구조

이전 스텝의 출력의 영향을 받지 않음

Recurrent Network

이전 층(Layer), 또는 스텝의 출력이 다시 입력으로 연결되는 신경망 구조

각 스텝마다 이전 상태를 기억 시스템(Memory System)

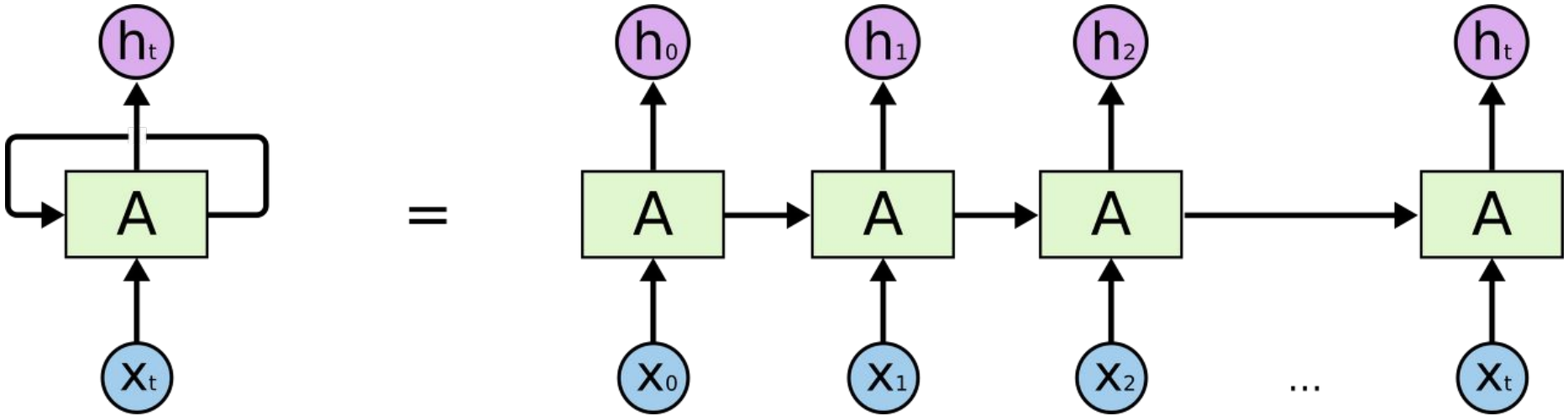
현재 상태가 이전 상태에 종속

RNN(Recurrent Neural Network, 순환신경망)

활성함수로 비선형 함수인 tanh를 쓰는 이유

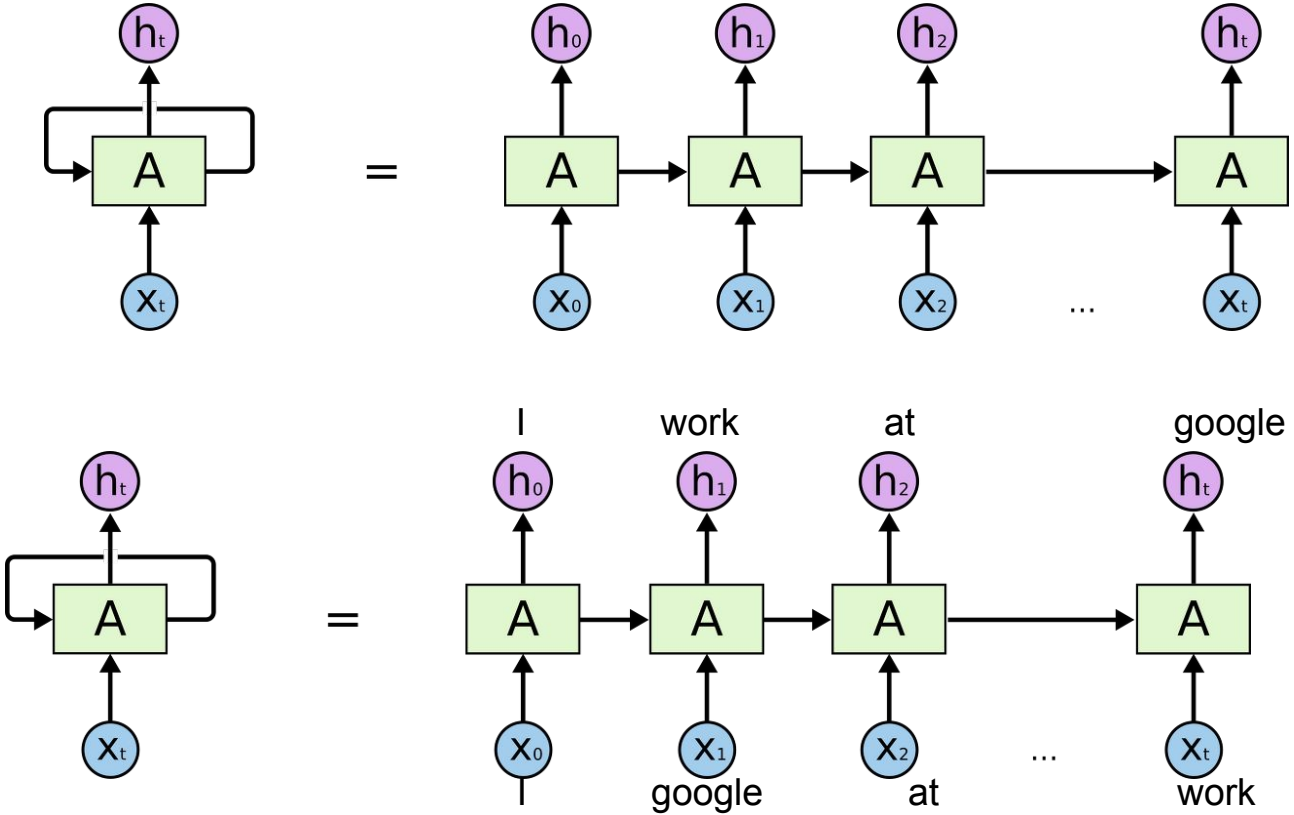
“선형 함수인 h(x)=cx를 활성 함수로 사용한 3층 네트워크를 떠올려 보세요.
이를 식으로 나타내면 y(x)=h(h(h(x)))가 됩니다.
이 계산은 y(x)=c*c*c*x처럼 세번의 곱셈을 수행하지만 실은 y(x)=ax와 똑같은 식입니다. a=c3이라고만 하면 끝이죠.
즉 히든레이어가 없는 네트워크로 표현할 수 있습니다.
그래서 층을 쌓는 혜택을 얻고 싶다면 활성함수로는
반드시 비선형함수를 사용해야 합니다.”

RNN(Recurrent Neural Network, 순환신경망)

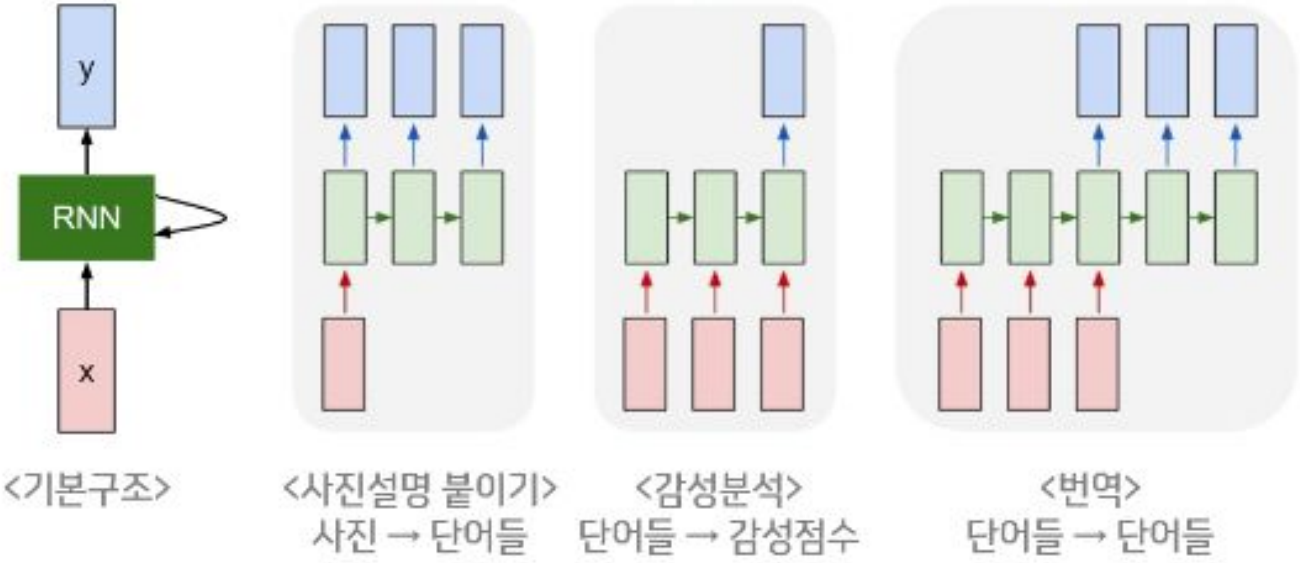


https://www.researchgate.net/figure/Feed-forward-and-recurrent-ANN-architecture_fig1_315111480

RNN(Recurrent Neural Network, 순환신경망)



RNN(Recurrent Neural Network, 순환신경망)



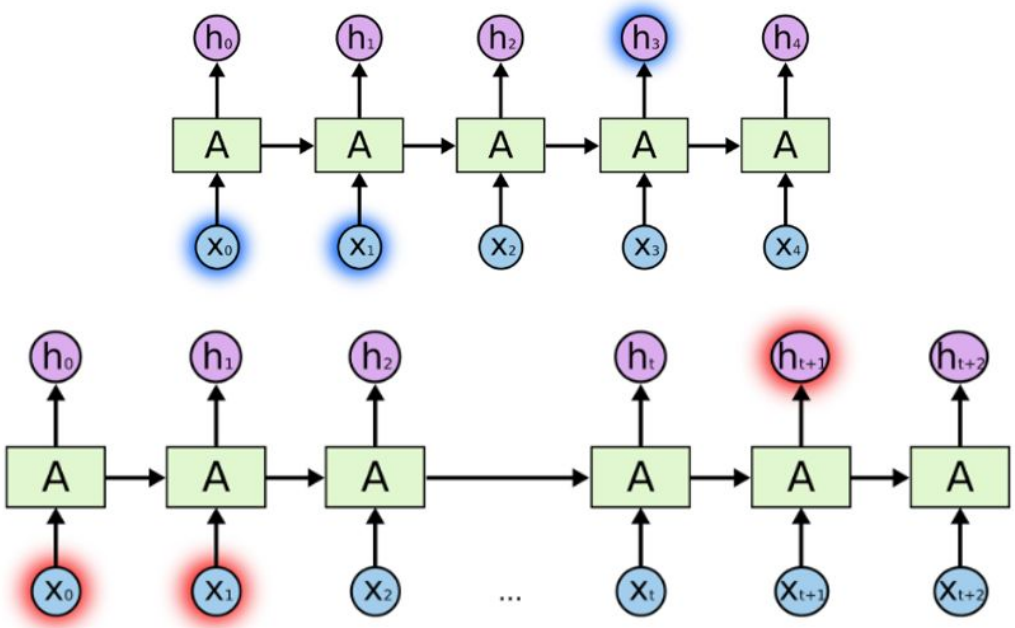
<기본구조>

<사진설명 붙이기>
사진 → 단어들

<감성분석>
단어들 → 감성점수

<번역>
단어들 → 단어들

https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/



RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하

RNN(Recurrent Neural Network, 순환신경망)

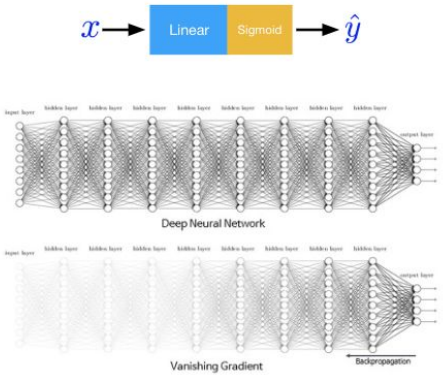
- SimpleRNN은 실전에 사용하기엔 너무 단순
- SimpleRNN은 이론적으로 시간 t 에서 이전의 모든 타임스텝의 정보를 유지할 수 있지만, 실제로는 긴 시간에 걸친 의존성은 학습할 수 없음
- 그래디언트 소실 문제(vanishing gradient problem)
- 이를 방지하기 위해 LSTM 같은 레이어 등장

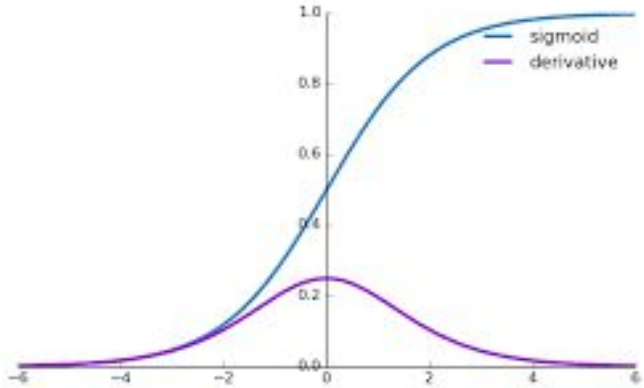
Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Sigmoid: Vanishing Gradient Problem

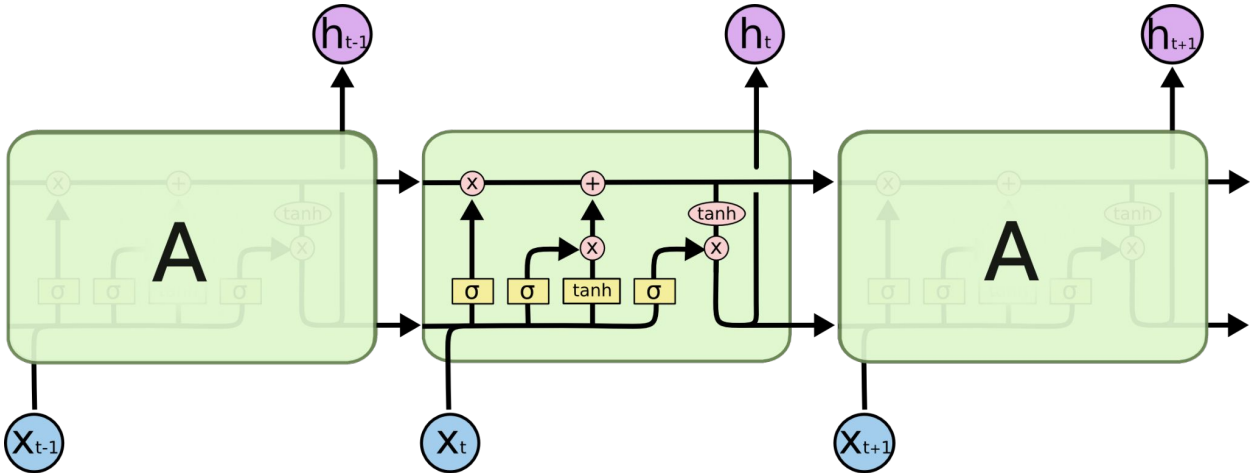




https://www.youtube.com/watch?v=NgxMUHTJYmU

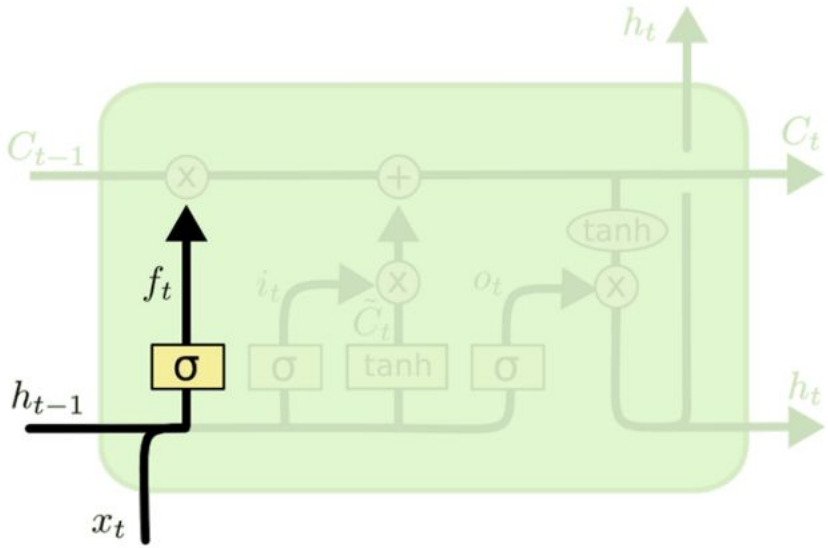
LSTM(Long Short-Term Memory)

- 장단기 기억 알고리즘
- LSTM은 RNN의 히든 state에 cell-state를 추가한 구조
- 나중을 위한 정보를 꾸준히 저장함으로써 오래된 시그널이 점차 소실되는 것을 막아줌



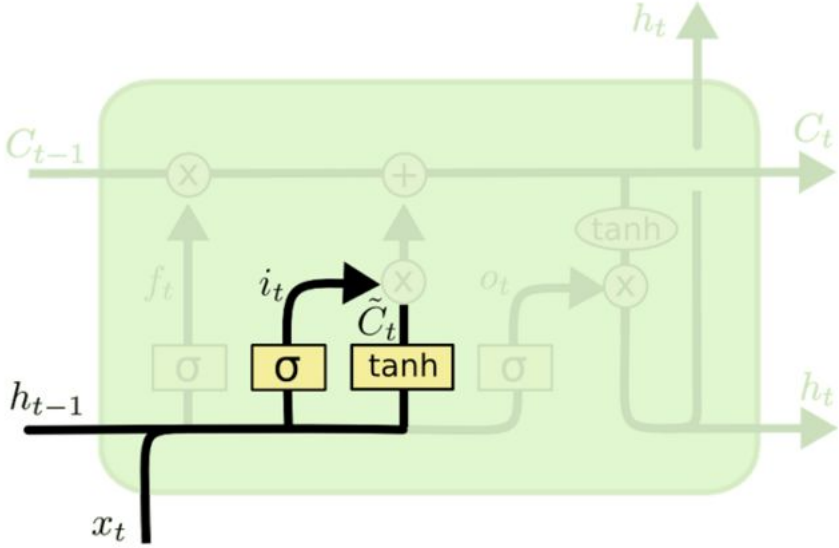
https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

LSTM(Long Short-Term Memory)



<forget gate>

forget gate ft는 '과거 정보 잊기'를 위한 게이트
출력값은 ht−1과 xt를 받아 시그모이드를 취해준 값
시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에
그 값이 0이라면 이전 상태의 정보는 잊고,
1이라면 이전 상태의 정보를 온전히 기억



<input gate>

input gate it⊙gt는 '현재 정보를 기억하기' 위한 게이트ht−1과
xt를 받아 시그모이드를 취하고, 또 같은 입력으로
하이퍼볼릭탄젠트를 취해준 다음 Hadamard product 연산을 한
값이 바로 input gate가 내보내는 값
it의 범위는 0~1, gt의 범위는 -1~1이기 때문에
강도와 방향을 나타냄