RNN & LSTM

김균엽

Word embedding

- **Problem of Natural Language in machine learning**
  - 기존의 키, 나이와 같은 숫자형태의 feature나 image의 경우 데이터가 숫자로 이루어져있다.
  - 그렇기에 별다른 변환없이 머신러닝이나 딥러닝의 입력으로 사용할 수 있었다.

  - 하지만 “I am going to school“와 같은 자연어 문장은 숫자형태의 데이터로 표현할 수 없다.
  - 그렇기에 자연어를 머신러닝이나 딥러닝에 사용하기 위해서는 자연어 데이터를 숫자 형태로 변환해주는것이 중요하다.

Word embedding

- **One-hot embedding**
  - 가장 간단한 방법은 문장을 각 알파벳 또는 문자 단위의 토큰으로 분리한 뒤, 각 토큰에 번호를 붙여주는 것이다

  - 예를들어 “I am going to school”의 경우 [“I”, “am”, “going”, “to”, school”]으로 분리한다
  - 이 과정을 tokenization이라 한다.

  - 또한 사전에 준비한 각 단어와 index를 매칭시켜주는 사전을 통해 각 token을 index 숫자로 변환한다.
  - Vocab 예시 Ex)

    I : 0
    you: 1
    am: 2
    are: 3
    ....
    School: 2301

    [“I”, “am”, “going”, “to”, school”] → [0, 2, 156, 32, 2301]

Word embedding

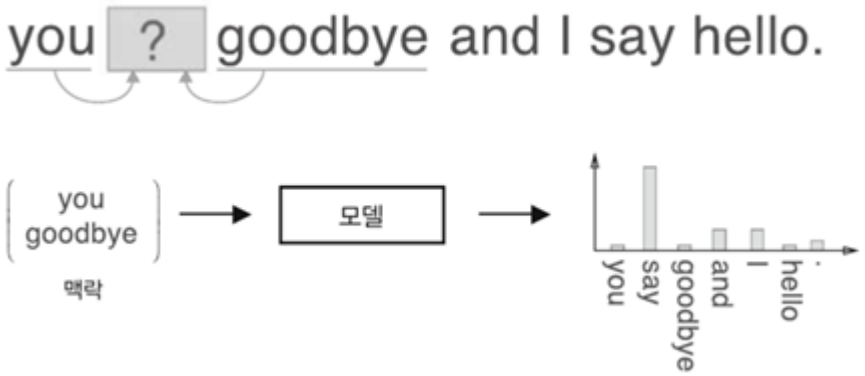
- **One-hot embedding**
  - 인덱스만을 사용했을 때는 유사한 숫자를 가진 데이터는 유사한 의미를 가져야 한다.
  - 유사한 숫자가 입력되면 유사한 정답이 나올 확률이 높기 때문에 전혀다른 의미의 token의 index가 비슷하다면 성능의 저하가 일어날 수 있다.

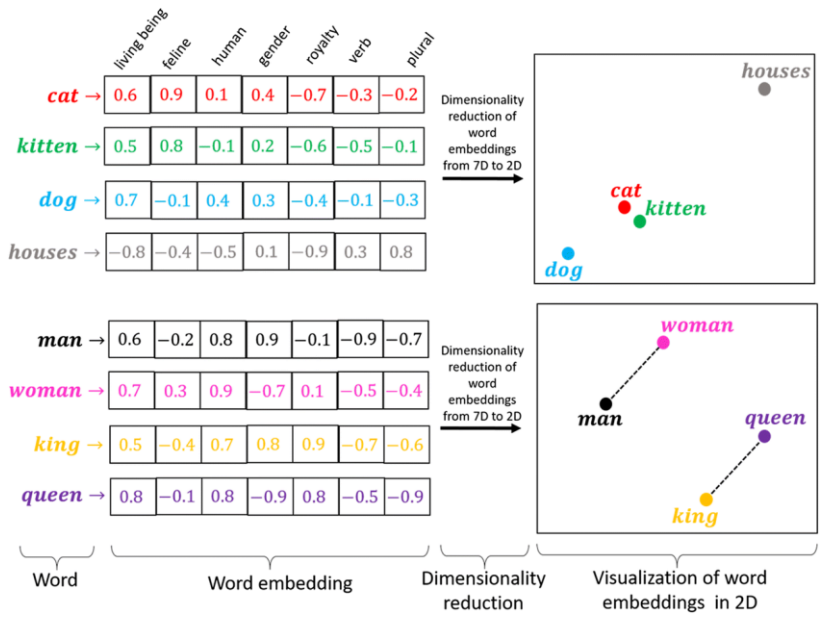
  - 그렇기에 각 index를 one-hot vector로 변환하여 모든 단어 벡터의 거리가 동일하게 변환하여 사용한다.
  - 이렇게 숫자 데이터로 이루어진 텐서의 형태로 변환하여 데이터로 사용된다.

[0, 2, 156, 32, 2301] → [[1,0,0,0,...,0,0]
[0,0,0,1,...,0,0]
[0,0,...,0,1,0,...,0,0]
[0,0,...,1,0,0,...,0,0]
[0,0,0,0,...,1,0]]

Word embedding

- **word embedding**
  - 하지만 자연어처리에서는 성능 향상을 위해 각 단어 벡터가 단어의 의미를 내포하게 하는 방법을 사용한다.
  - Word2vec이라는 딥러닝을 기반으로 주변단어를 통해 현재 단어를 예측하는 기법을 통해 단어 벡터에 의미가 포함되도록 학습하였다.



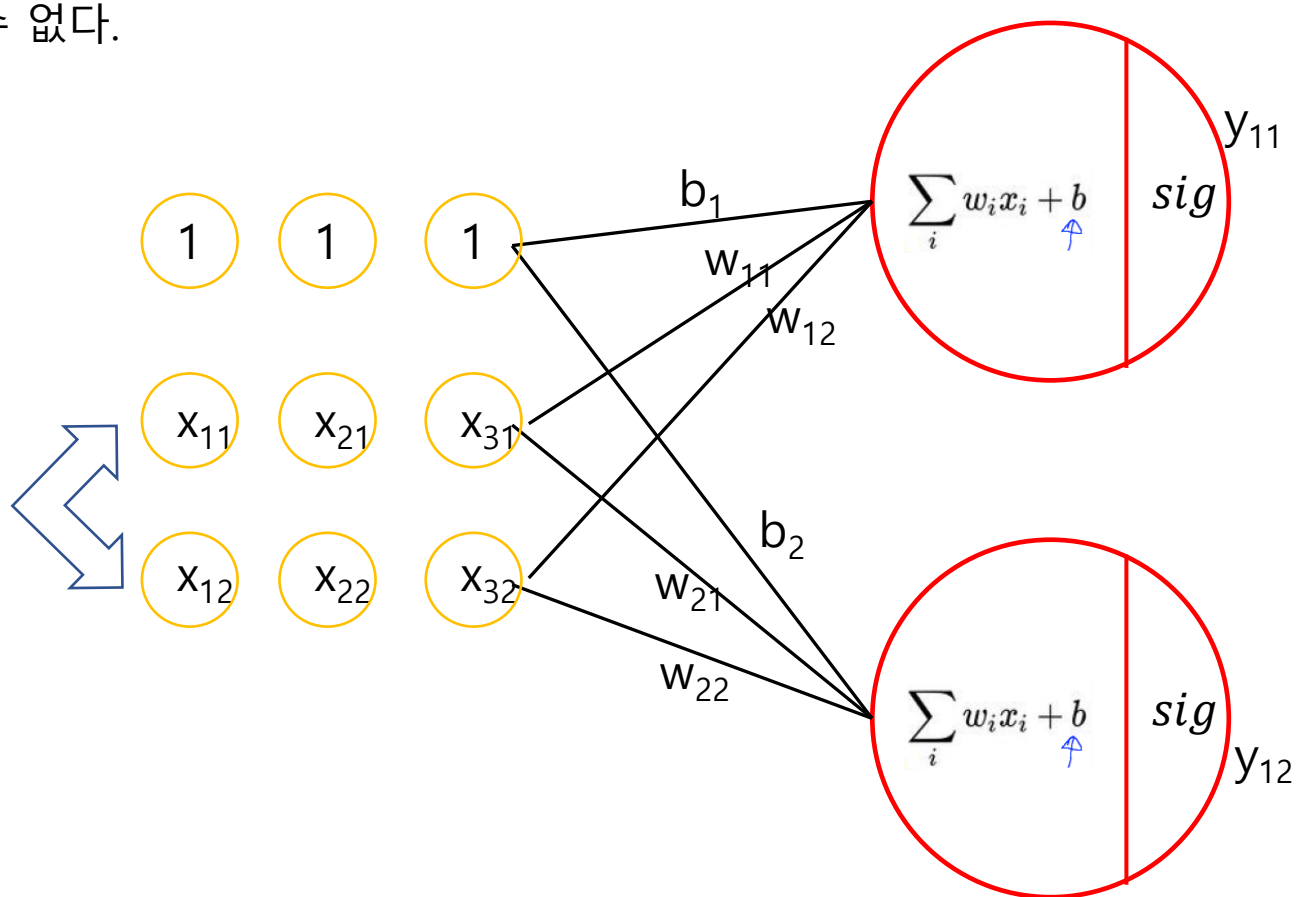


RNN

- **Problem of MLP**
  - MLP은 공간정보를 이해할 수 없다.
  - 모든 data를 1차원으로 변환하여 학습하기 때문에 2차원, 3차원 공간에 대한 학습이 어렵다.
    - 2차원 그림에 X가 존재하는지 여부 판별
    - → CNN
  - 또한 ANN은 순서가 바뀌어도 다른데이터라 인지할 수 없다
    - 데이터의 순서가 키,나이, 성적 순으로 입력되는 것과 나이, 키, 성적 순으로 입력되는 것을 다른데이터라 판별할 수 없다.
    - → RNN

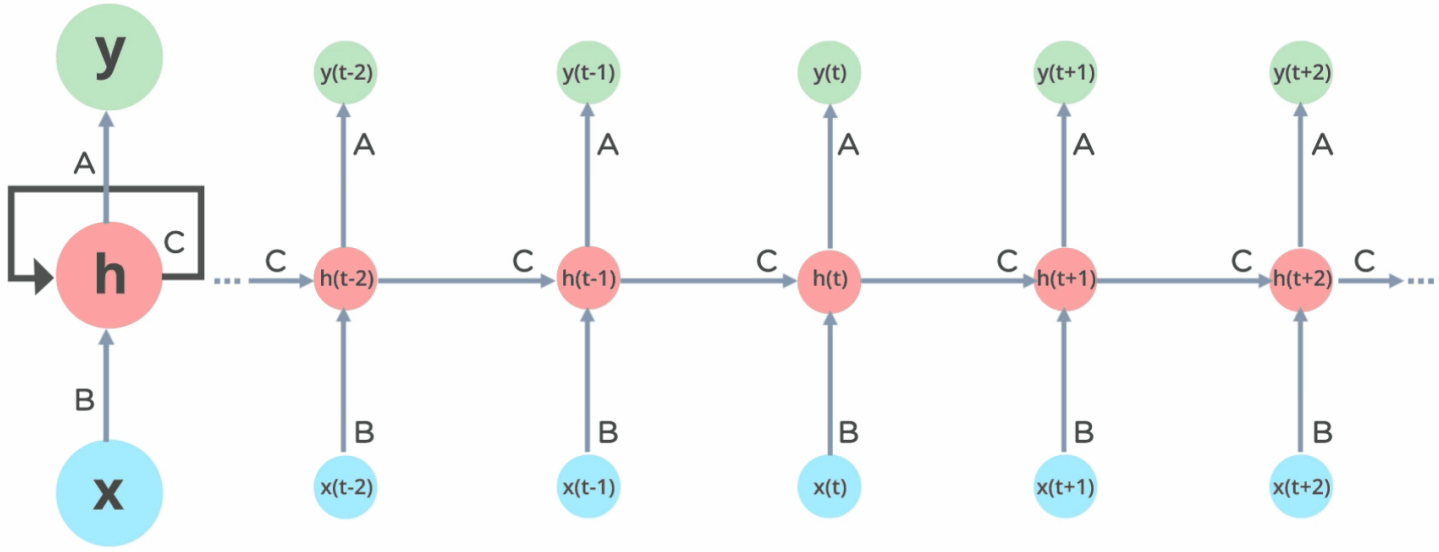
RNN

- **Problem of MLP**
  - 또한 ANN은 순서가 바뀌어도 다른데이터라 인지할 수 없다
    - 데이터의 순서가 키,나이, 성적 순으로 입력되는 것과 나이, 키, 성적 순으로 입력되는 것을 다른데이터라 판별할 수 없다.



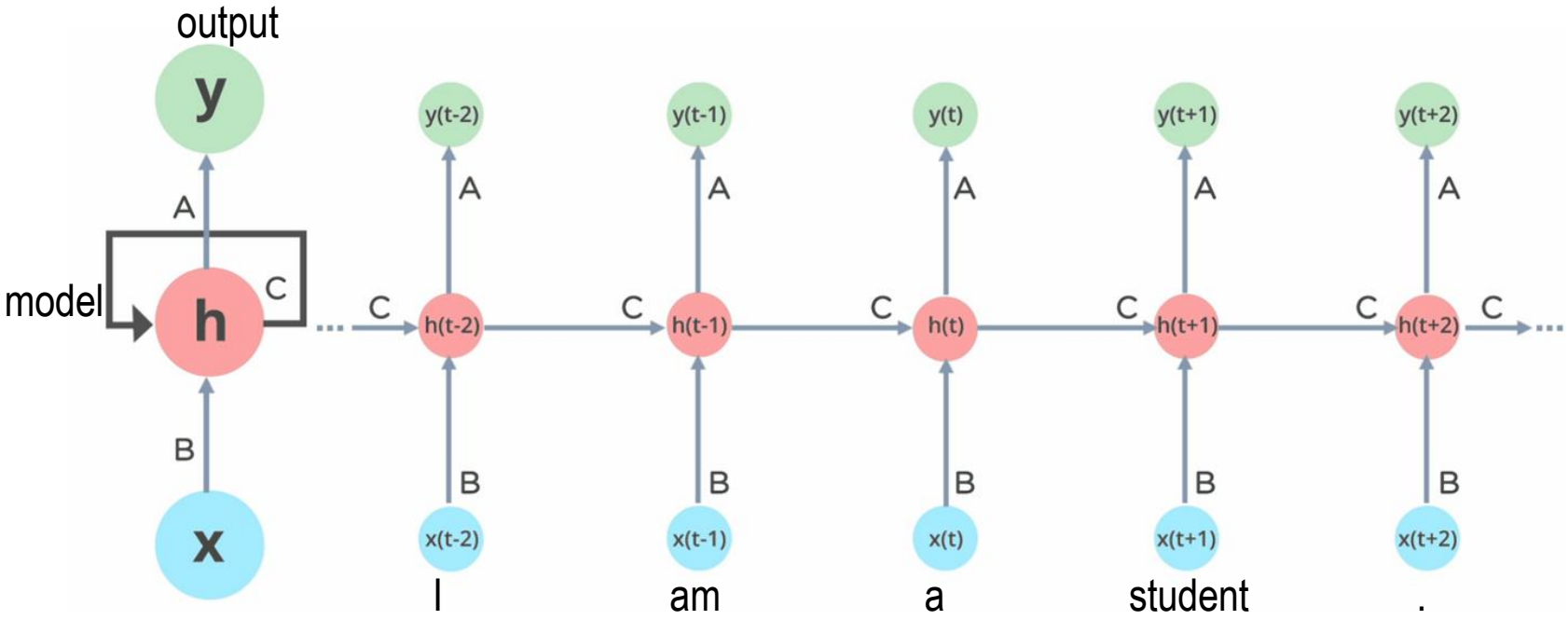
RNN

- **RNN**
  - RNN은 각 단어의 순서(time) 정보를 학습하기 위한 방법이다.
  - 이전 time에 대한 정보를 현재 time에 대한 학습에 사용함으로써 순서(time)정보를 학습하였다.
  - 이 방법은 문장에서 출현하는 단어의 순서가 중요한 자연어처리에서 높은 성능을 보였다.
  - 현재 time의 연산을 위해 현재 time에 대한 input과 이전 time에 대한 output을 사용한다.



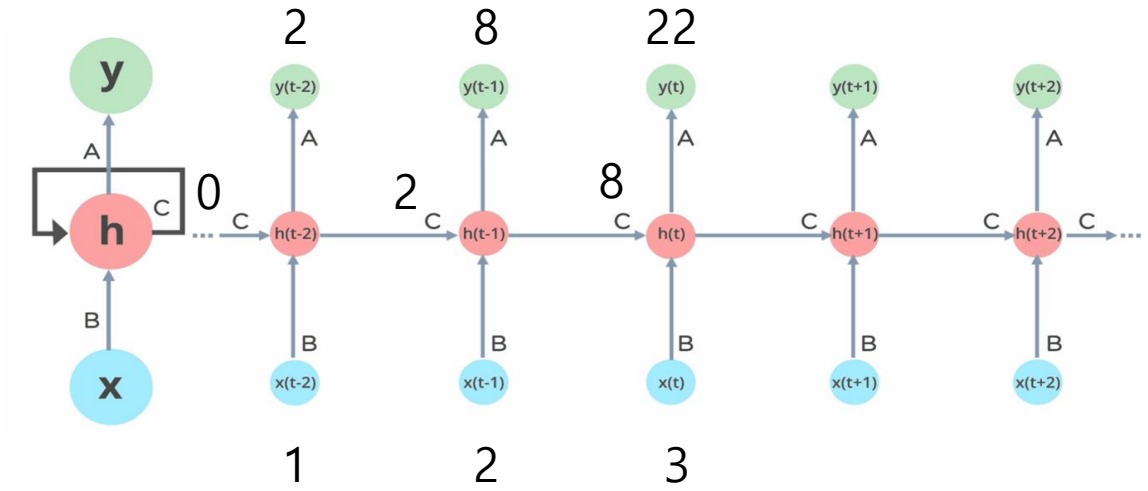
RNN

- **RNN**
  - RNN은 각 단어의 순서(time) 정보를 학습하기 위한 방법이다.
  - 이전 time에 대한 정보를 현재 time에 대한 학습에 사용함으로써 순서(time)정보를 학습하였다.
  - 이 방법은 문장에서 출현하는 단어의 순서가 중요한 자연어처리에서 높은 성능을 보였다.
  - 현재 time의 연산을 위해 현재 time에 대한 input과 이전 time에 대한 output을 사용한다.



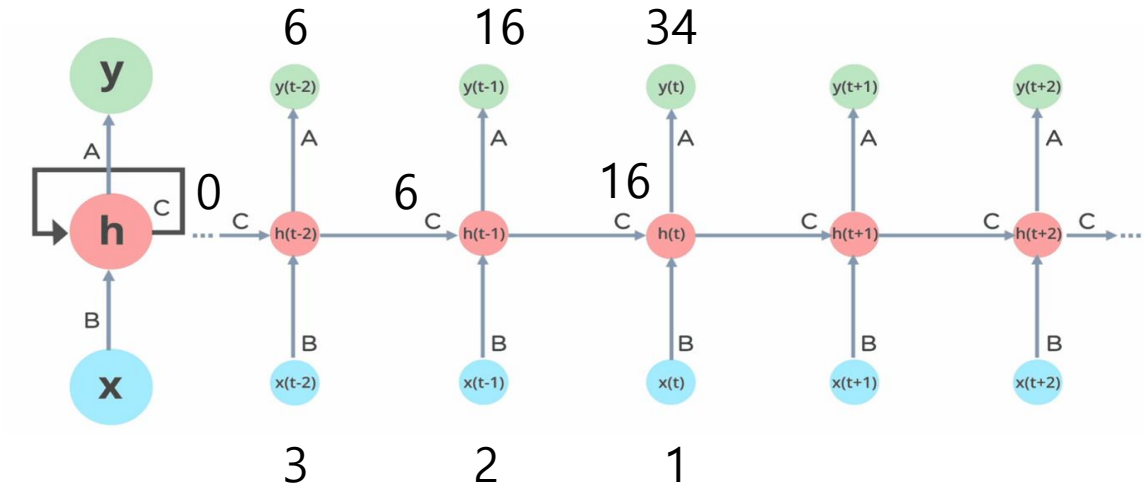
RNN

- **RNN**
  - RNN은 단어가 한단어씩 입력되기 때문에 만약 문장의 단어순서가 바뀌면 다른 결과를 도출함
  - 예를 들어 연산 cell이 이전 time의 input과 현재 time의 input을 더하고 2를 곱한다고 가정하자
    - $Y(x) = (y_{t-1} + x_t)*2$
  - 1,2,3이 입력되었을 때와 3,2,1이 입력될 때 서로 다른 결과를 보인다



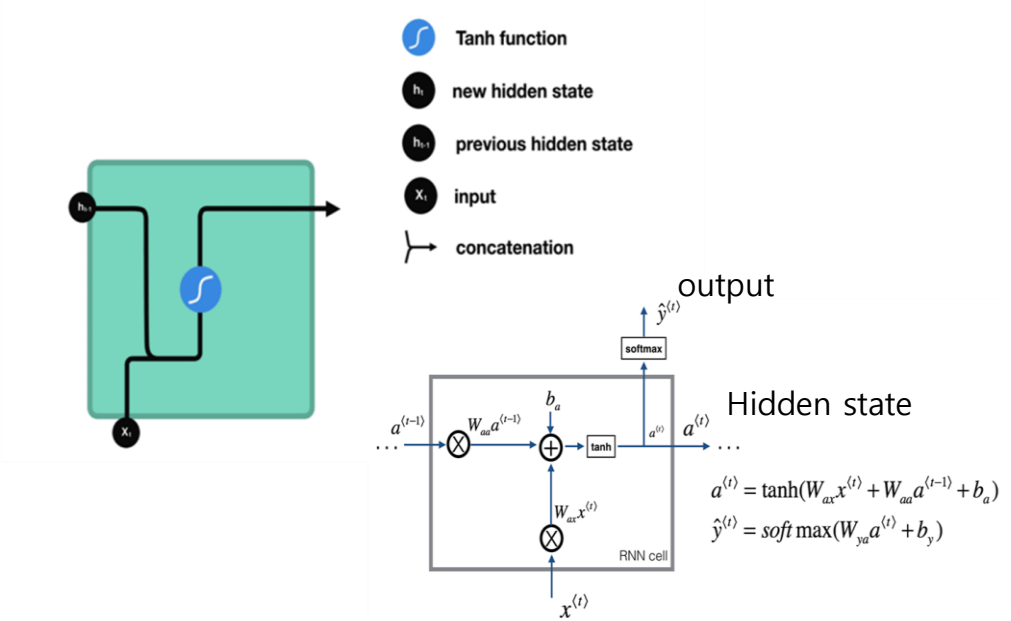
RNN

- **RNN**
  - RNN은 단어가 한단어씩 입력되기 때문에 만약 문장의 단어순서가 바뀌면 다른 결과를 도출함
  - 예를 들어 연산 cell이 이전 time의 input과 현재 time의 input을 더하고 2를 곱한다고 가정하자
    - $Y(x) = (y_{t-1} + x_t)*2$
  - 1,2,3이 입력되었을 때와 3,2,1이 입력될 때 서로 다른 결과를 보인다



RNN

- **RNN**
  - 현재의 hidden state(h)를 구하기 위해 이전time(t-1)의 hidden state와 current(t)의 input을 이용하여 hidden state을 계산
  - current hidden state(h)에 DNN을 통해서 현재 cell에 대한 output(y) 계산
  - h는 다음 cell에서의 t+1의 hidden state를 구하기 위해 다음 cell에 전달



RNN

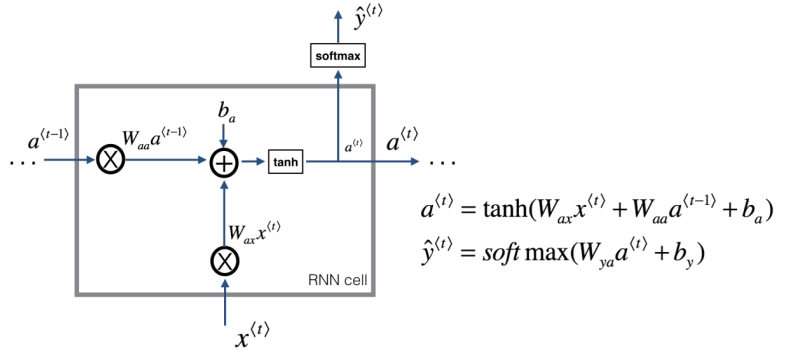
RNN cell을 MLP로 표현한식

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$
$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

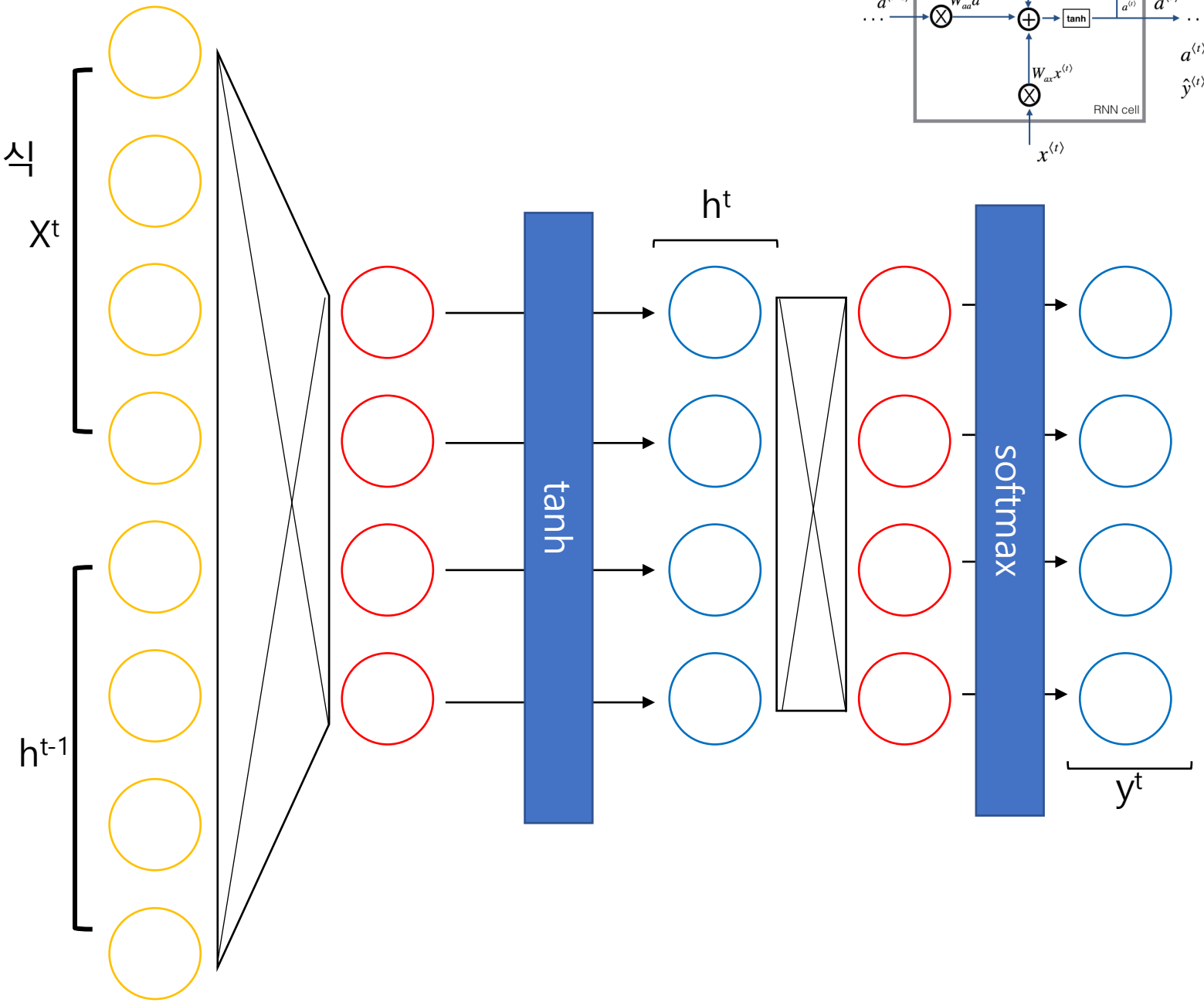
||
||

$$h^t = \tanh([xt; ht^{-1}]\begin{bmatrix} W_x \\ W_h \end{bmatrix})$$

$$y^t = \mathrm{softmax}(Wyht + by)$$

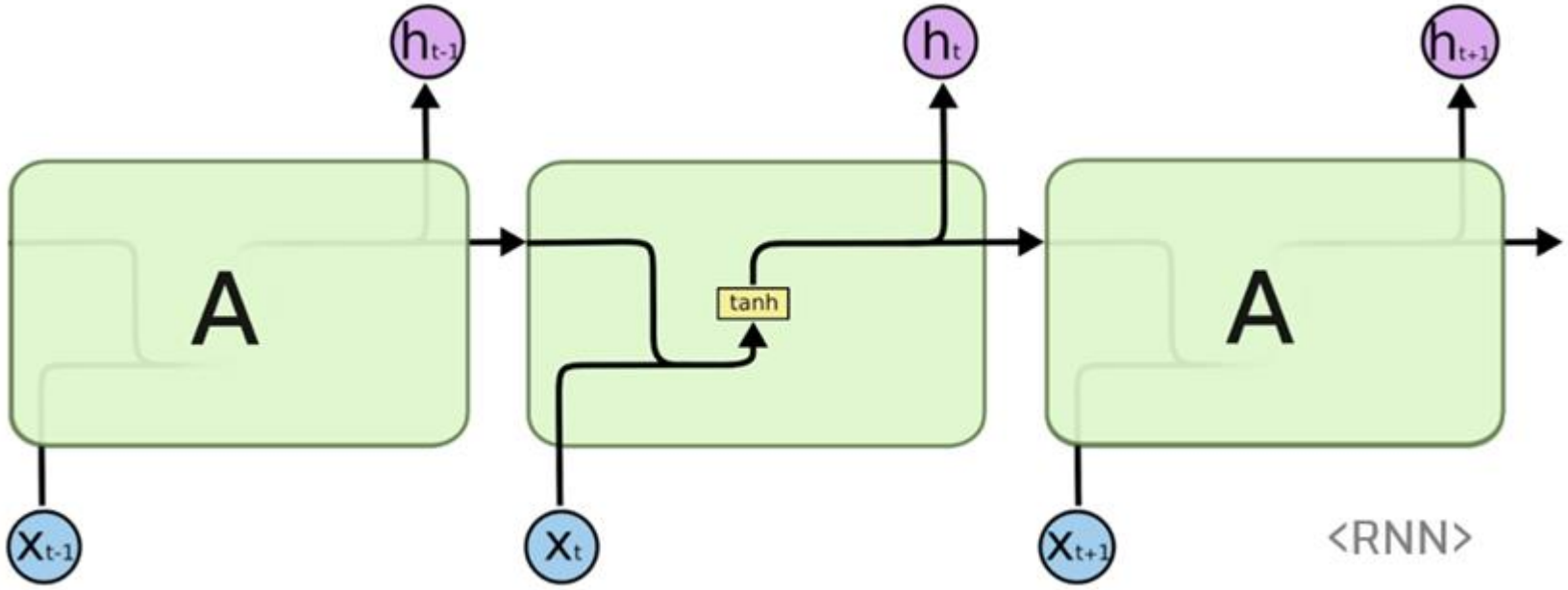


$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$
$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$



RNN

- **RNN**
  - 하나의 RNN cell을 통해서 W를 학습
  - Word1과 h0을 통해 h1을 추출하면 해당 셀을 재사용하여 word2와 h1을 통해서 h2추출

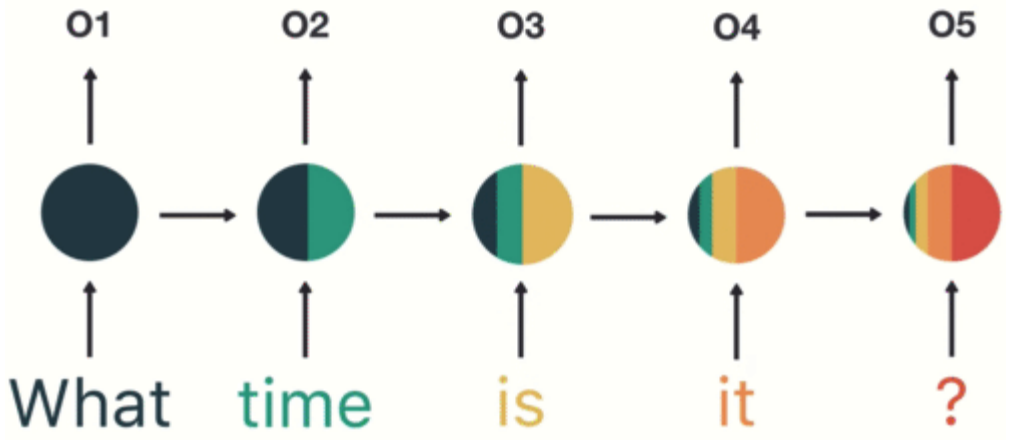


RNN Problem

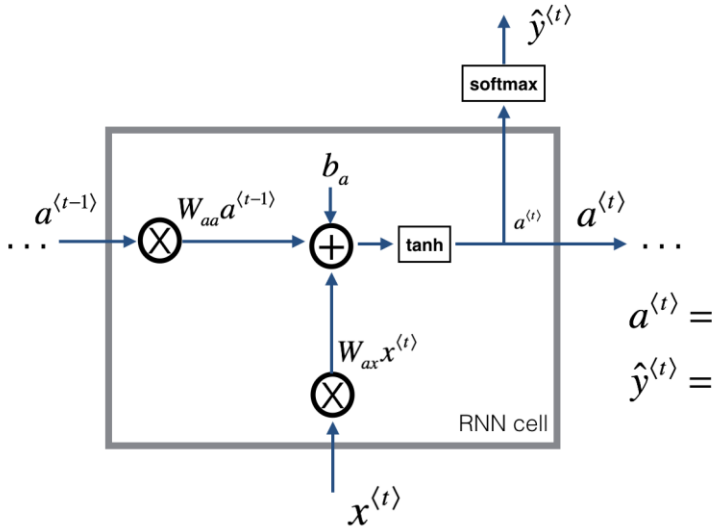
vanishing gradient problem
시간이 지날수록 단어의 정보가 소실되는 문제

처음 cell에서는 50%의 정보를 보유한다면
다음셀에서는 25% 그다음은 12.5%와 같이
매 time마다 정보가 반으로 손실됨

h^{t-1}과 w^{t}를 1대1 비율로 더해서 결과를 얻어내기
때문에 정보가 반씩 줄음



What 100%	What 50% time 50%	What 25% Time 25% Is 50%	What 12.5% Time 12.5% Is 25% It 50%	What 6.25% Time 6.25% Is 12.5% It 25% ? 50%



$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

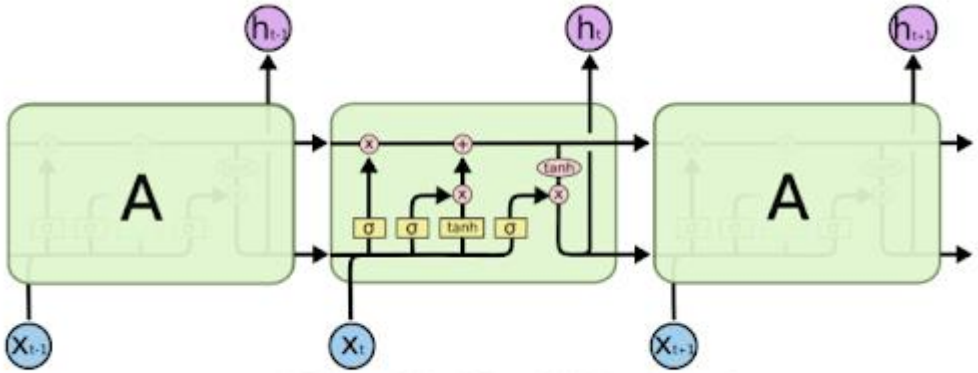
$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

LSTM

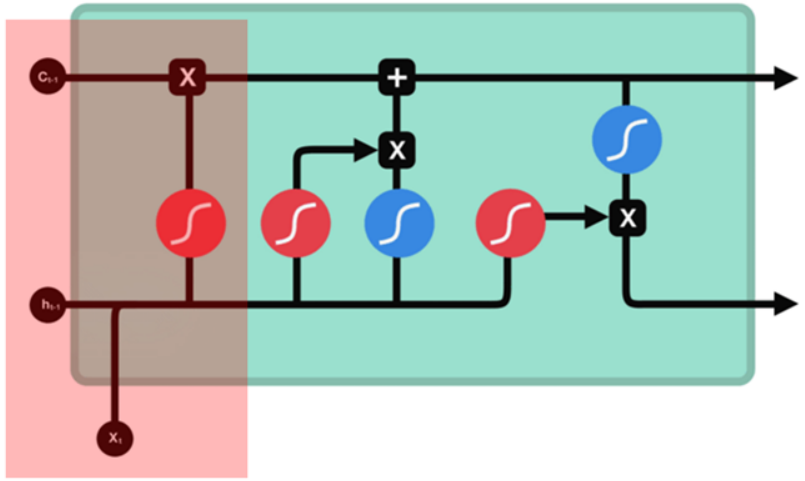
- **RNN**
  - RNN의 vanishing gradient problem을 해결하기 위해 고안된 모델
  - 기존의 RNN에서 이전 정보를 위주로 가지게되는 vector C를 사용함으로써 이전 time에 대한 정보를 더 담으려고 함
  - RNN과 동일하게 현재의 input과 이전 time의 hidden state를 perceptron을 통해 합성

LSTM

RNN의 vanishing gradient problem을 해결하기 위해
이전의 정보를 가지고있는 새로운 hidden state C를 선언



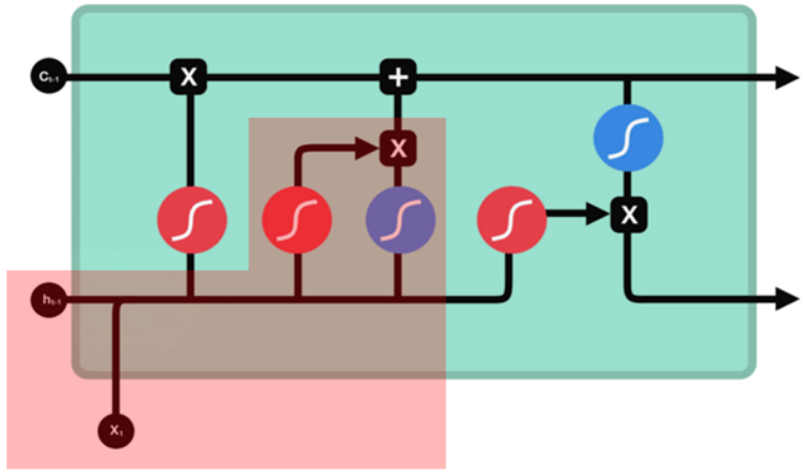
LSTM에 들어있는 4개의 상호작용하는 레이어가 있는 반복되는 모듈



$$f_t = \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f})$$
$$i_t = \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i})$$
$$o_t = \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o})$$
$$g_t = \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

LSTM

RNN의 vanishing gradient problem을 해결하기 위해
이전의 정보를 가지고있는 새로운 hidden state C를 선언



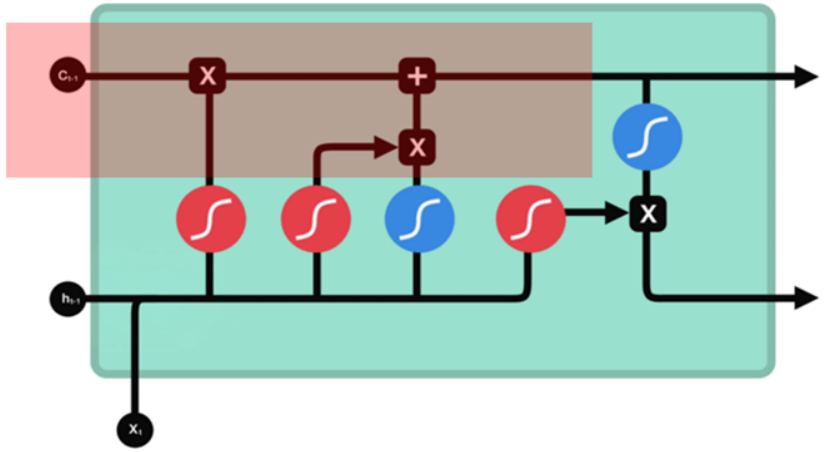
previous cell state

forget gate output

$$f_t = \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f})$$
$$i_t = \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i})$$
$$o_t = \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o})$$
$$g_t = \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

LSTM

RNN의 vanishing gradient problem을 해결하기 위해
이전의 정보를 가지고있는 새로운 hidden state C를 선언



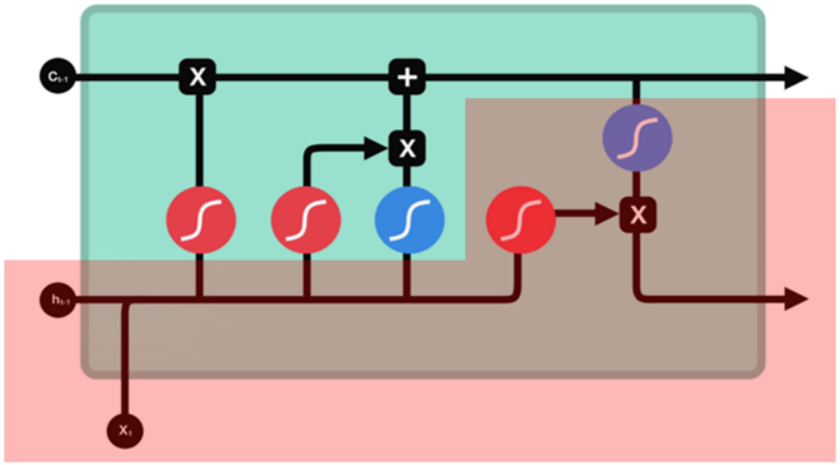
previous cell state

forget gate output

$$f_t = \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f})$$
$$i_t = \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i})$$
$$o_t = \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o})$$
$$g_t = \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

LSTM

RNN의 vanishing gradient problem을 해결하기 위해
이전의 정보를 가지고있는 새로운 hidden state C를 선언



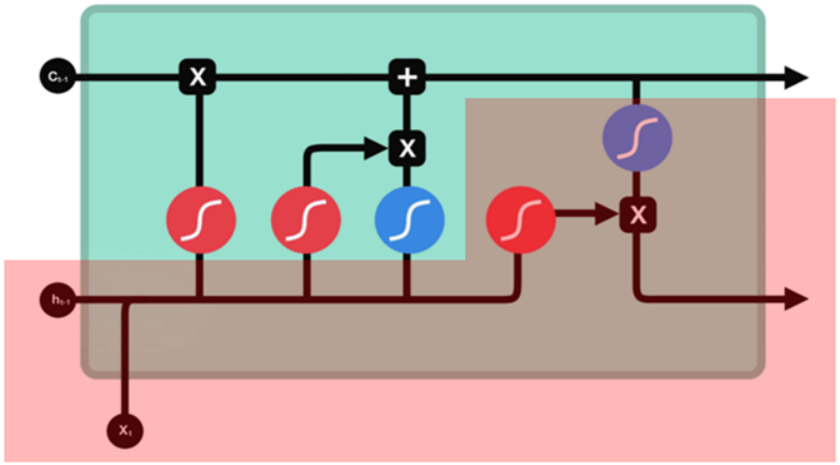
previous cell state

forget gate output

$$f_t = \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f})$$
$$i_t = \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i})$$
$$o_t = \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o})$$
$$g_t = \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

LSTM

RNN의 vanishing gradient problem을 해결하기 위해
이전의 정보를 가지고있는 새로운 hidden state C를 선언



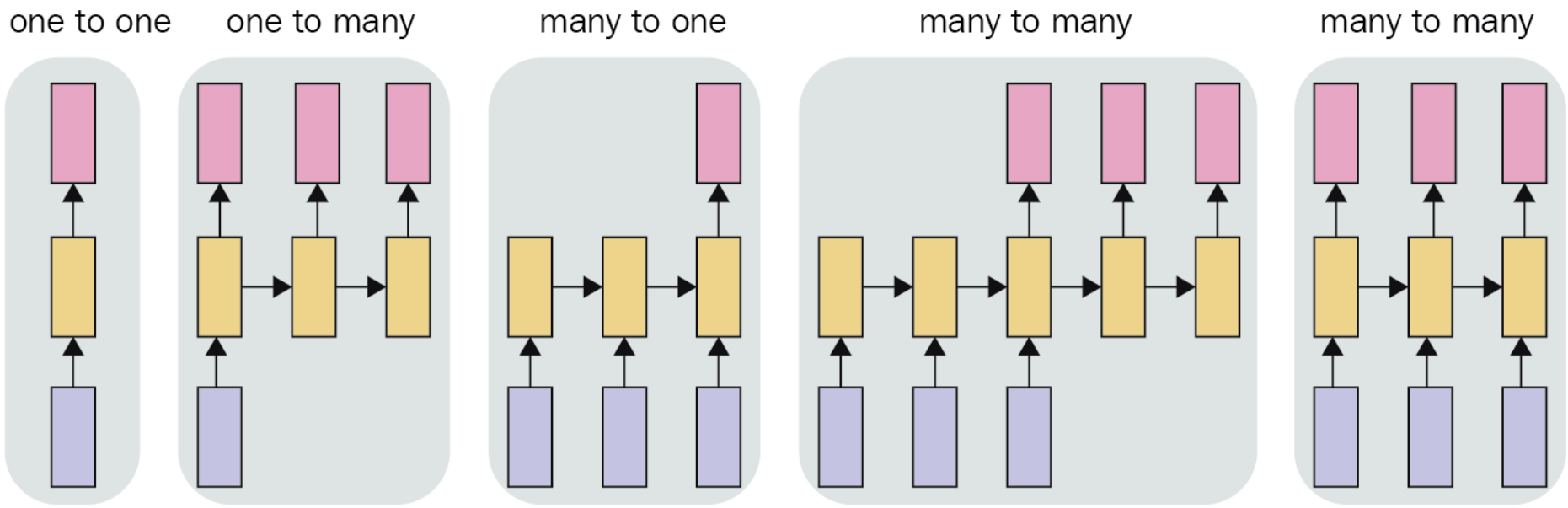
previous cell state

forget gate output

$$f_t = \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f})$$
$$i_t = \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i})$$
$$o_t = \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o})$$
$$g_t = \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

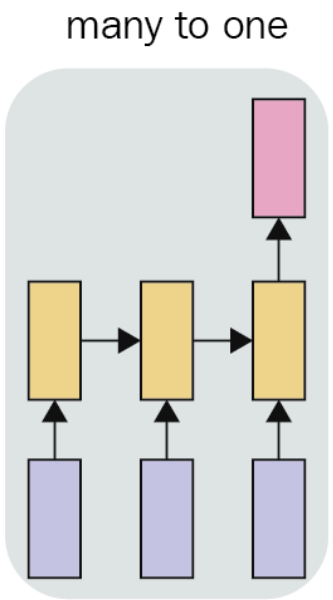
NLP

- **LSTM in NLP**
  - RNN이나 LSTM은 시간의 정보를 포함하며 각 시간마다 1개의 결과가 나온다.
  - 즉, 입력으로 10개의 단어가 입력되면 결과도 10개의 결과가 나온다.
  - 그렇기에 각 task에 many-to-one, many-to-many의 형태로 학습한다.



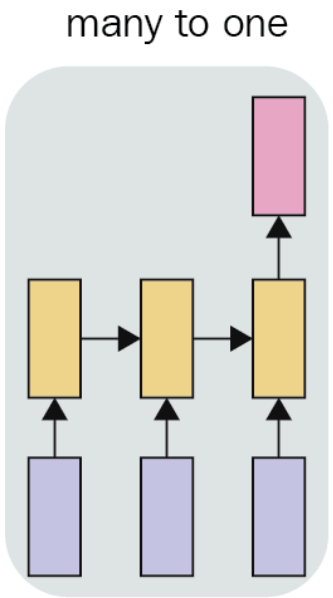
NLP

- **Many to one**
  - 입력의 값이 문장이고 결과값으로 한 개의 값이 필요한 task에서 많이 사용한다.
  - 문장 분류, 감성 분석



NLP

- **Many to one**
  - 입력 값이 문장이고 결과값이 각 단어에 대한 정보가 필요할 경우에 많이 사용된다.
  - NER, 품사 태깅 등
  - 입력의 값이 문장이고 결과값이 문장인 task에서 많이 사용된다.
  - 번역, 챗봇 등

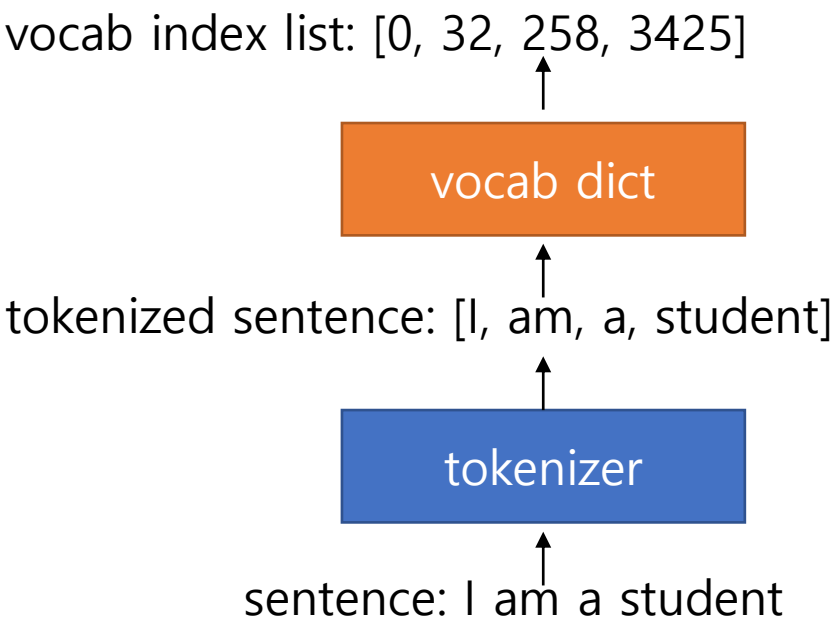


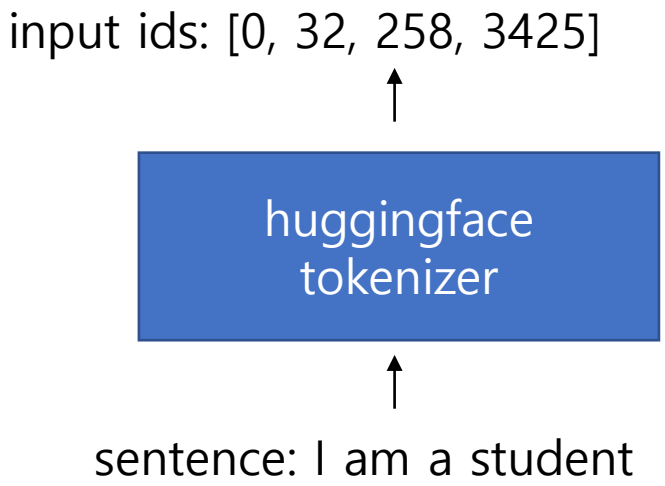
RNN & LSTM in pytorch

김균엽

NLP

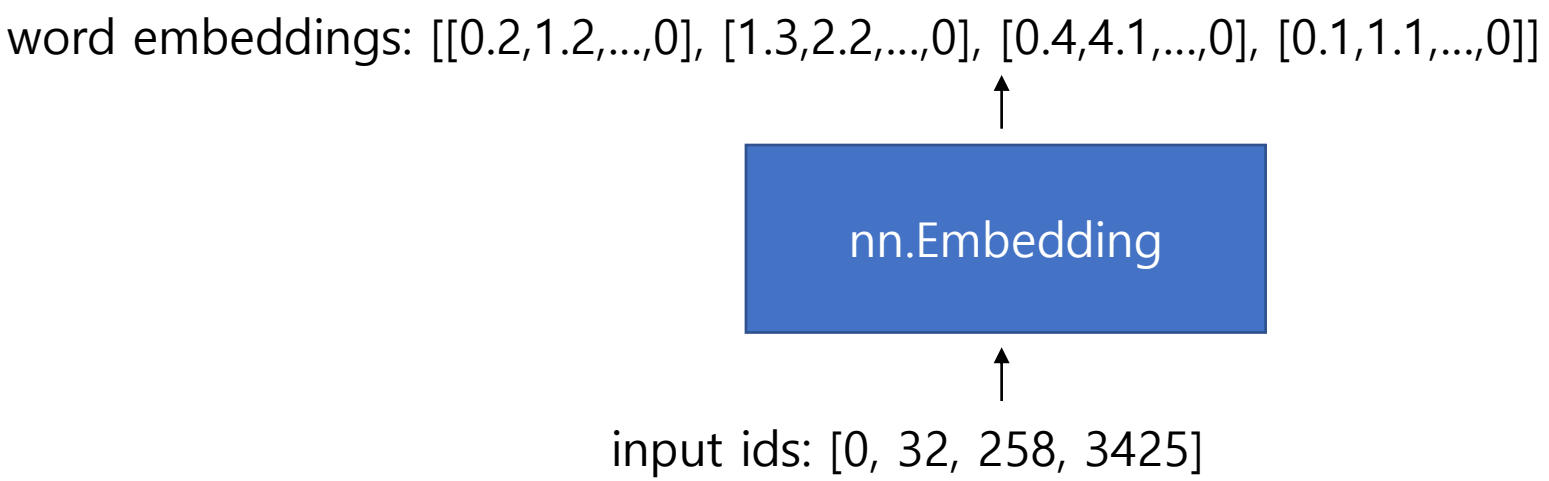
- **Tokenization**
  - torchtext의 모듈을 통해 tokenziation을 진행할 수 있다.
    - torchtext.utils.get_tokenizer("basic_english")를 통해 영어기반의 tokenizer를 사용한다.
    - torchtest.vocab.build_vocab_form_iterator(tokenized data list)를 통해 train data에서 출현한 모든 data에 대해서 vocab dictionary를 구축한다.
  - 하지만 일반적으로는 huggingface에서 제공되는 pre-trained tokenzier를 통해 진행된다.





NLP

- **nn.Embedding**
  - 각 vocab의 index가 입력되면 해당 vocab에 대응하는 word embedding을 반환하는 딥러닝 모듈
  - nn.Embedding(num_words, embedding_dim)로 선언한다.
  - 입력으로는 모든 vocab의 index들이 입력되어 (batch, sequence length)의 shape가 입력
  - 출력으로는 모든 vocab에 대한 word embedding이 출력되므로 (batch, seq length, embedding_dim)의 형태로 out된다.



NLP

- **nn.LSTM/RNN**
  - LSTM/RNN을 수행하는 딥러닝 모듈
  - nn.LSTM(input dim, output dim)의 형태로 선언된다.
  - 입력은 각 LSTM의 cell에 대한 입력을 넣기 때문에 (batch, seq len, input dim)로 입력된다.
  - 출력 또한 각 LSTM에대한 hidden을 결과로 받기 때문에 (batch, seq len, output dim)의 결과를 도출하면 추가로 마지막 cell의 h와 c를 반환한다.