

KUBIG 24-S
여름방학 BASIC STUDY SESSION

NLP SESSION

WEEK3

01 Announcement, 복습과제 우수 코드 review

02 RNN

03 LSTM

04 GRU

05 ELMo

06 예습과제 우수 코드 review, Announcement

01 Announcement, 우수 복습과제 Review

오늘의 공지



KUBIG Contest Team Build

18기 강동현

19기 강지윤 심승현 이동주 이소희 이승준 정종락 최지우 하진우

20기 강민정 김채원 박준희 윤시호 이세은 이유진

금일 세션이 종료된 후, 관심 분야 투표 공지 예정.
구글 폼으로 수요 조사 후 3인씩 총 5개 팀으로 분할

팀 생성 후

- 주 1회 이상(권장) 회의 진행한 후 회의록 작성
- 6주차에 중간 진행 과정 보고
- 8월 29일 KUBIG Contest에서 결과 발표

19기 이동주님

2주차
복습과제1

IMDB 텍스트 감성분석

20기 강민정님

2주차
복습과제2

FastText vs Word2Vec

20기 이세은님

2주차
복습과제3

Word2Vec CBOW 구현

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

02 RNN

Sequence data를 처리하는 순환신경망

2-1. What is Sequential Data

Sequential Data

데이터 집합 내의 객체들이
특정 순서를 따르는 데이터.
그 순서가 변경될 경우 고유의 특성이 변질됨

문장도 Sequential data!
단어들이 순서를 이루는 집합 데이터

2-1. What is Sequential Data

Sequential Data :

US oil prices turn negative

Price per barrel of WTI



Source: Bloomberg, 20 April 2020, 20:15 GMT

BBC

Time Series



강화학습

Sequential Data :

- Text Data
- Time-Series Data
- Reinforcement Learning / Planning
- Image Frames in Video
- Music
- Dialogue
- Object Tracking
- ...

인접한 정보들이 Highly Correlated

2-1. What is Sequential Data

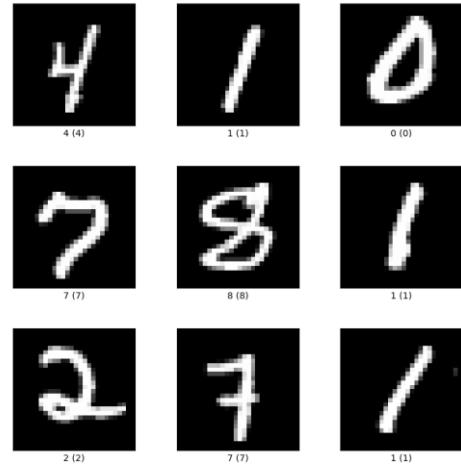


Image Data

- 공간적 특성에 focus



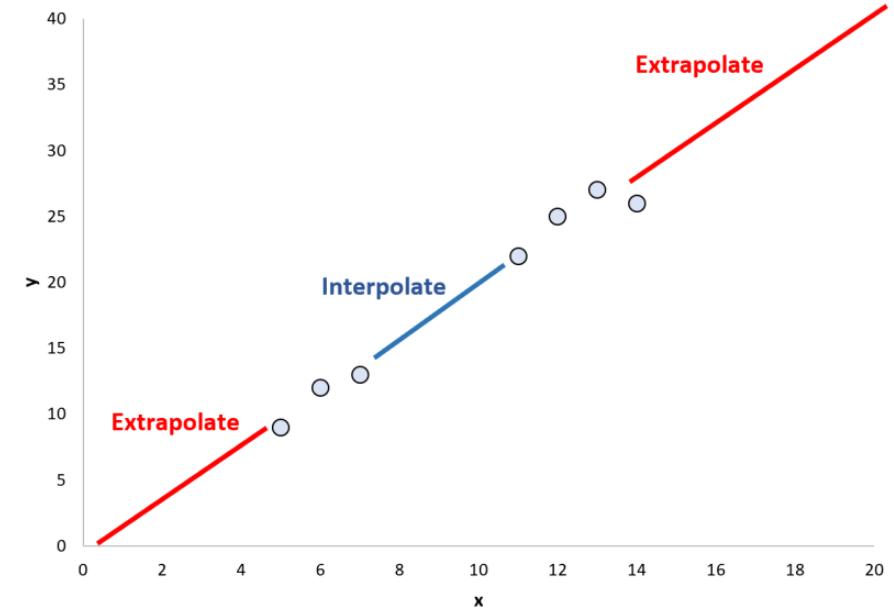
Graphical Data

- Vertex와 edge의 연결이 중요

2-1. What is Sequential Data

Why Sequential patterns are difficult?

- Sequence length NOT Fixed
- Extrapolation is more difficult than interpolation
- Sequential data are NOT i.i.d.
 - Dog bites man vs Man bites dog



- **Sequentiality**
 - Inputs should be used sequentially to deal with **non-iid** data

$$P(X_1, \dots, X_n) = \prod_{k=1} P(X_k | X_1, \dots, X_{k-1})$$

- If independent,(iid)

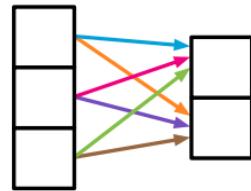
$P(\text{man bites dog}) \neq P(\text{dog bites man})$

$P(X_1, X_2, X_3) \neq P(X_1)P(X_2)P(X_3)$

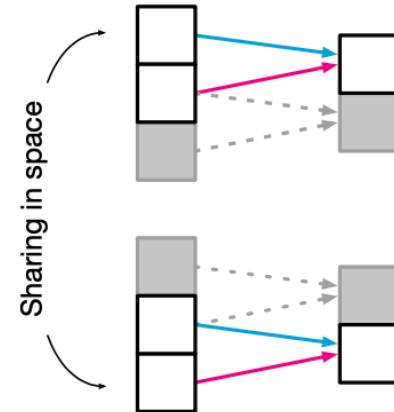
2-2. Invariance In Sequential Data

- **Temporal Invariance**
 - Translation variance in sequence order

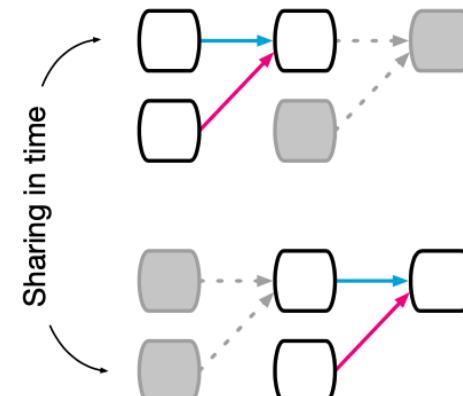
$$\begin{aligned} P(X_{1+j}, \dots, X_{n+j}) &= \prod_{k=1} P(X_k | X_1, \dots, X_{k-1}) = \prod_{k=1} P(X_{k+j} | X_{1+j} = \mathbf{x}_1, \dots, X_{k-1+j} = \mathbf{x}_{k-1}) \\ &= f_\theta(\mathbf{x}_1, \dots, \mathbf{x}_{k-1}) \end{aligned}$$



(a) Fully connected

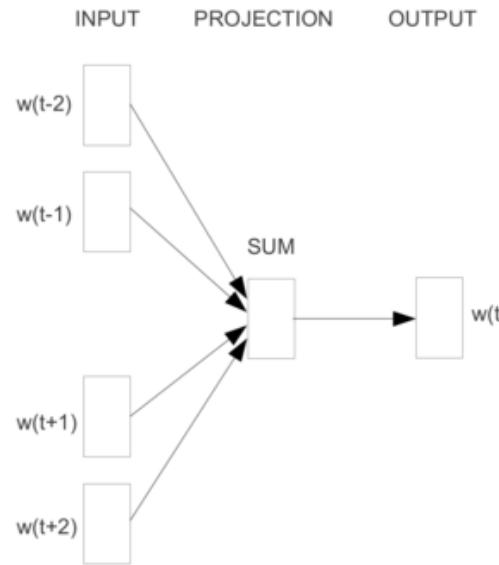


(b) Convolutional



(c) Recurrent

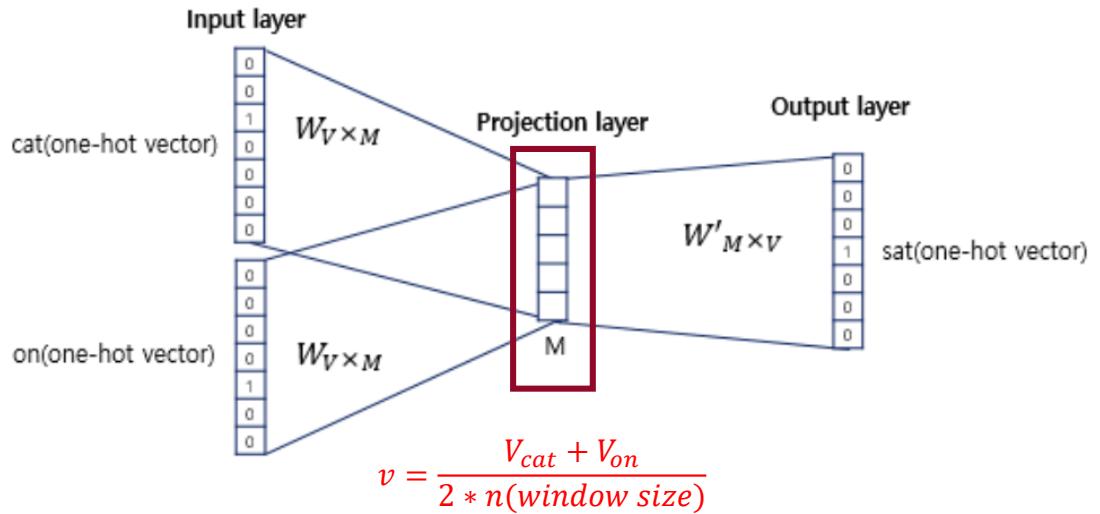
2-3. Limitation of CBOW(NNLM)



단어의 유사성을 계산할 수 있게 되어
주변 문맥에 따라 예측하는 task가 가능해짐
하지만 여전히 window size 내에서의 문맥만 반영하고
단어 전체의 sequence는 고려하지 못함

Window size를
무한정 늘리면 되지 않나?
=> 학습시킬 가중치 행렬 W 가
과하게 많아져 비효율적

예문: I am _____ good enough!

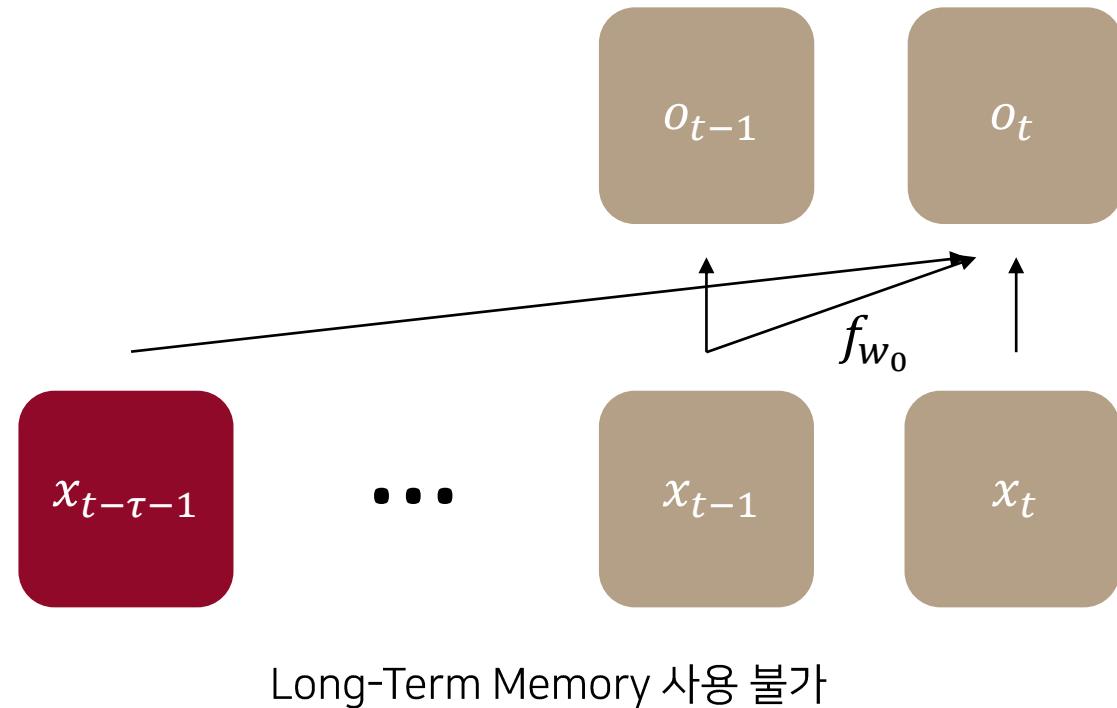


문맥을 반영하기는 하나,
Projection layer에서 벡터들이 합해지기 때문에
앞, 뒤 순서 맥락을 잊어버림

2-4. Build RNN - Autoregressive Model

- Autoregressive Model with fixed length

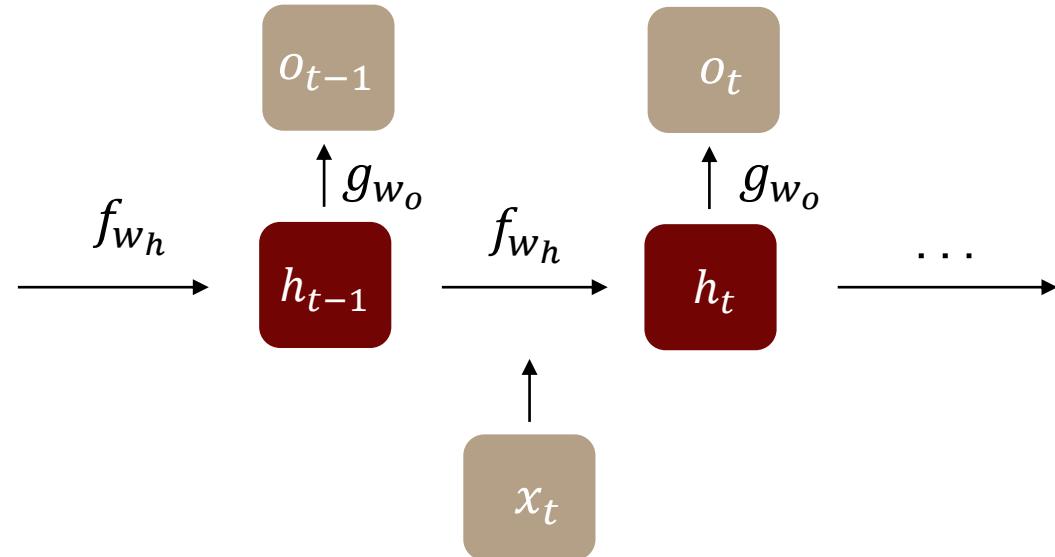
$$o_t = f_{w_0}(\mathbf{x}_t, \mathbf{x}_{t-1}, \dots, \mathbf{x}_{t-\tau})$$



- Latent Autoregressive Model

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

$$\mathbf{o}_t = g_{w_o}(\mathbf{x}_t, \mathbf{x}_{t-1}, \dots, \mathbf{x}_{t-\tau})$$



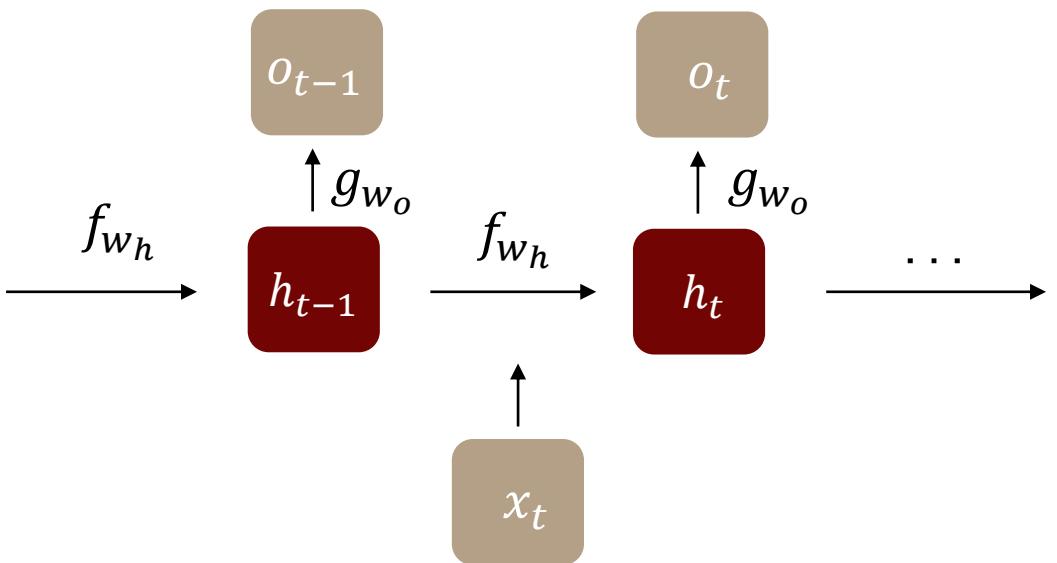
2-4. Build RNN – Naïve Recurrent Neural Network

- Latent Autoregressive Model

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

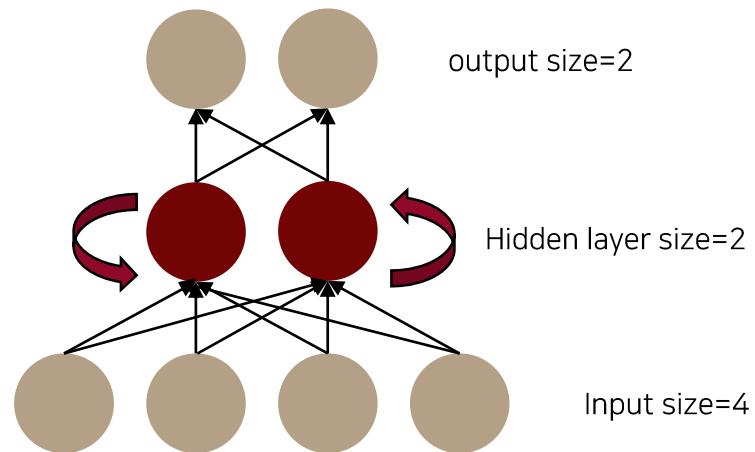
$$\mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

$$\begin{aligned}\mathbf{h}_t &= \phi(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + b_h) \\ \mathbf{o}_t &= W_{oh}\mathbf{h}_t + b_o\end{aligned}$$

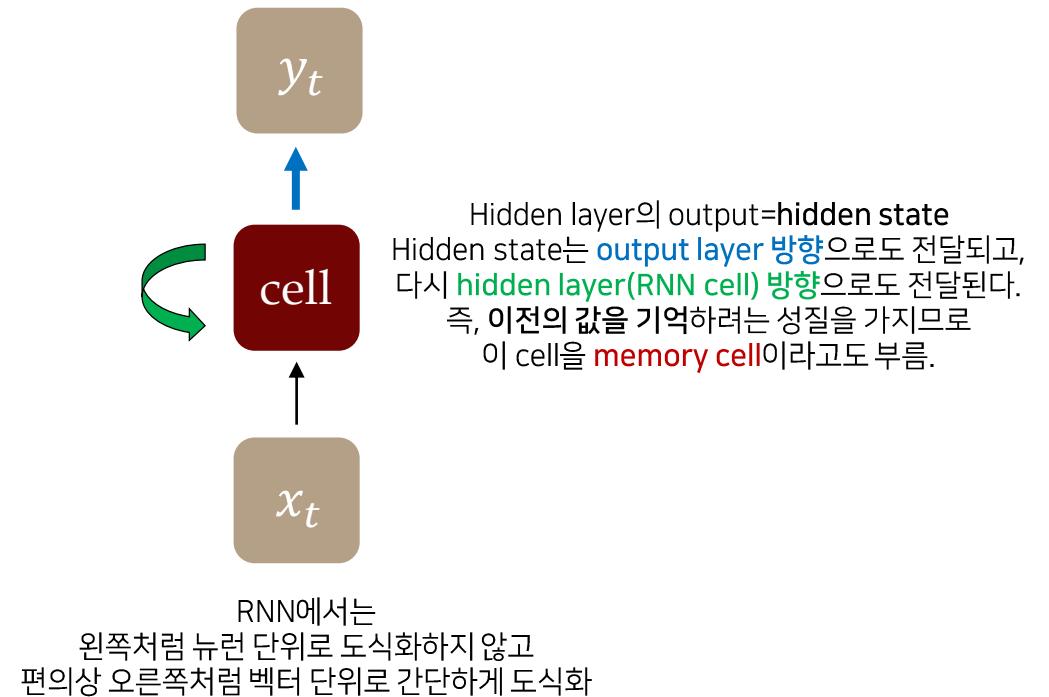


2-5. RNN(Recurrent Neural Net)

문장 순서를 잘 보존하는 모델로, RNN 제안
그렇다면 어떻게 문장의 순서를 고려할 수 있을까?



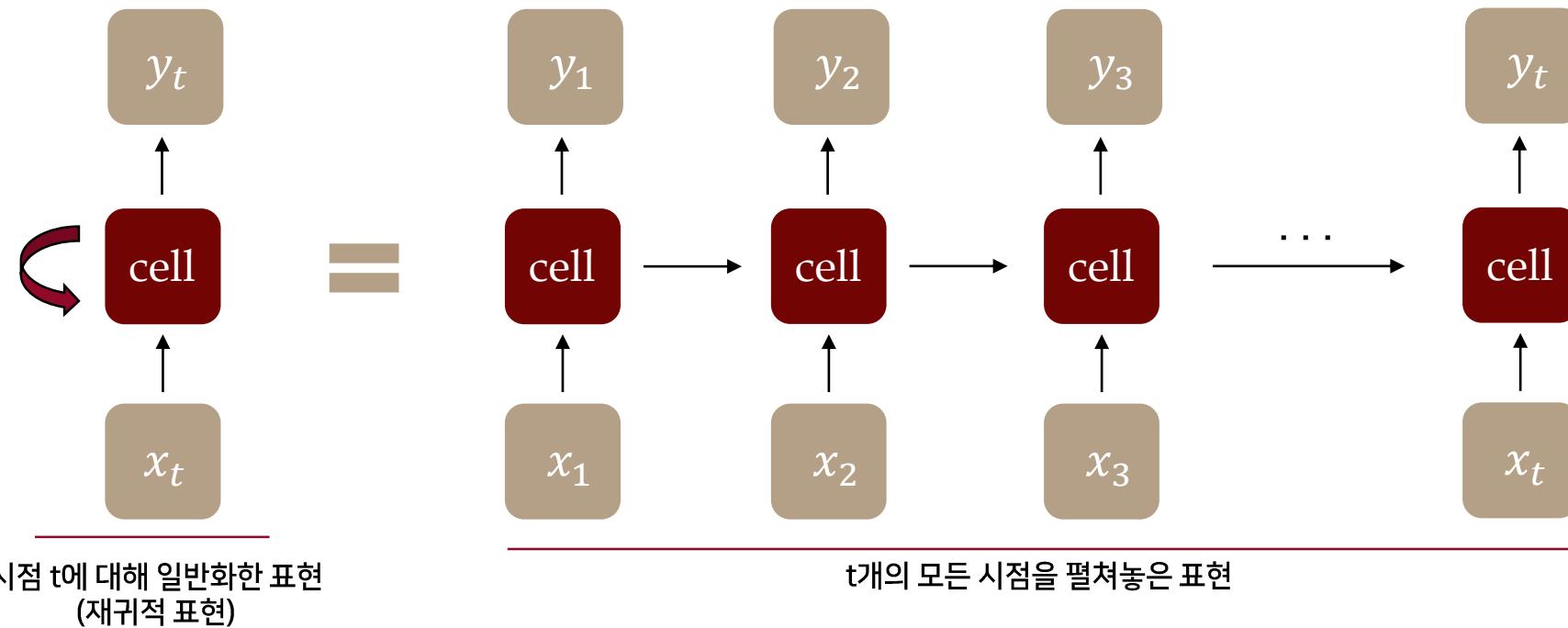
NNLM은 앞으로만 향하는 feed forward 방식이었다면,
RNN은 output을 다시 hidden layer로 순환시키는
Recurrent 방식!



RNN에서는
왼쪽처럼 뉴런 단위로 도식화하지 않고
편의상 오른쪽처럼 벡터 단위로 간단하게 도식화

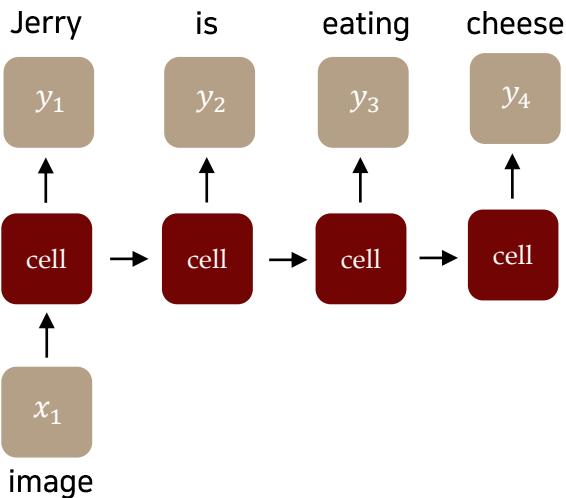
2-5. RNN(Recurrent Neural Net)

문장 순서를 잘 보존하는 모델로, RNN 제안
그렇다면 어떻게 문장의 순서를 고려할 수 있을까?

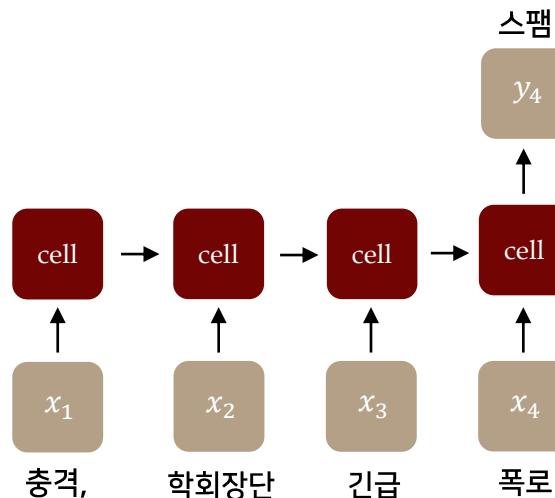


2-5. RNN(Recurrent Neural Net)

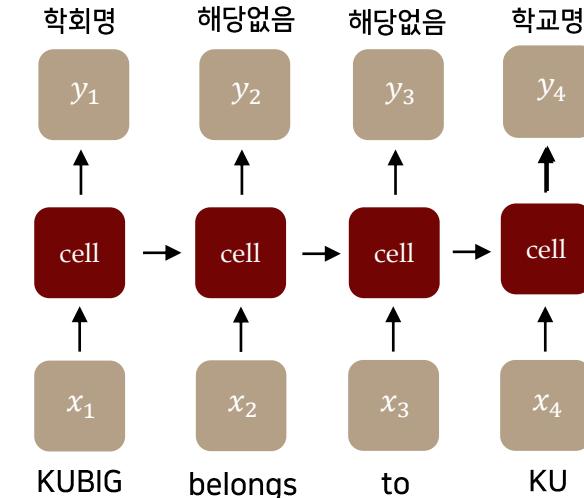
RNN can be applied to various task in NLP



one-to-many
Ex) image captioning



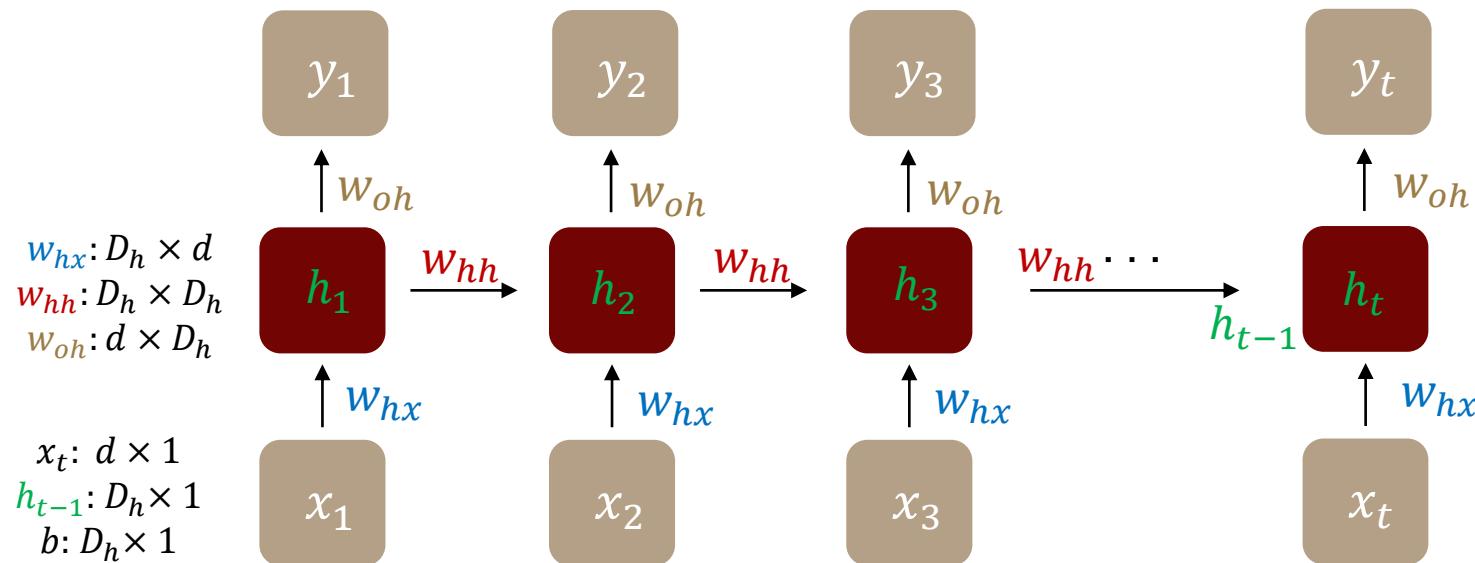
many-to-one
Ex) spam classification
Sentiment analysis



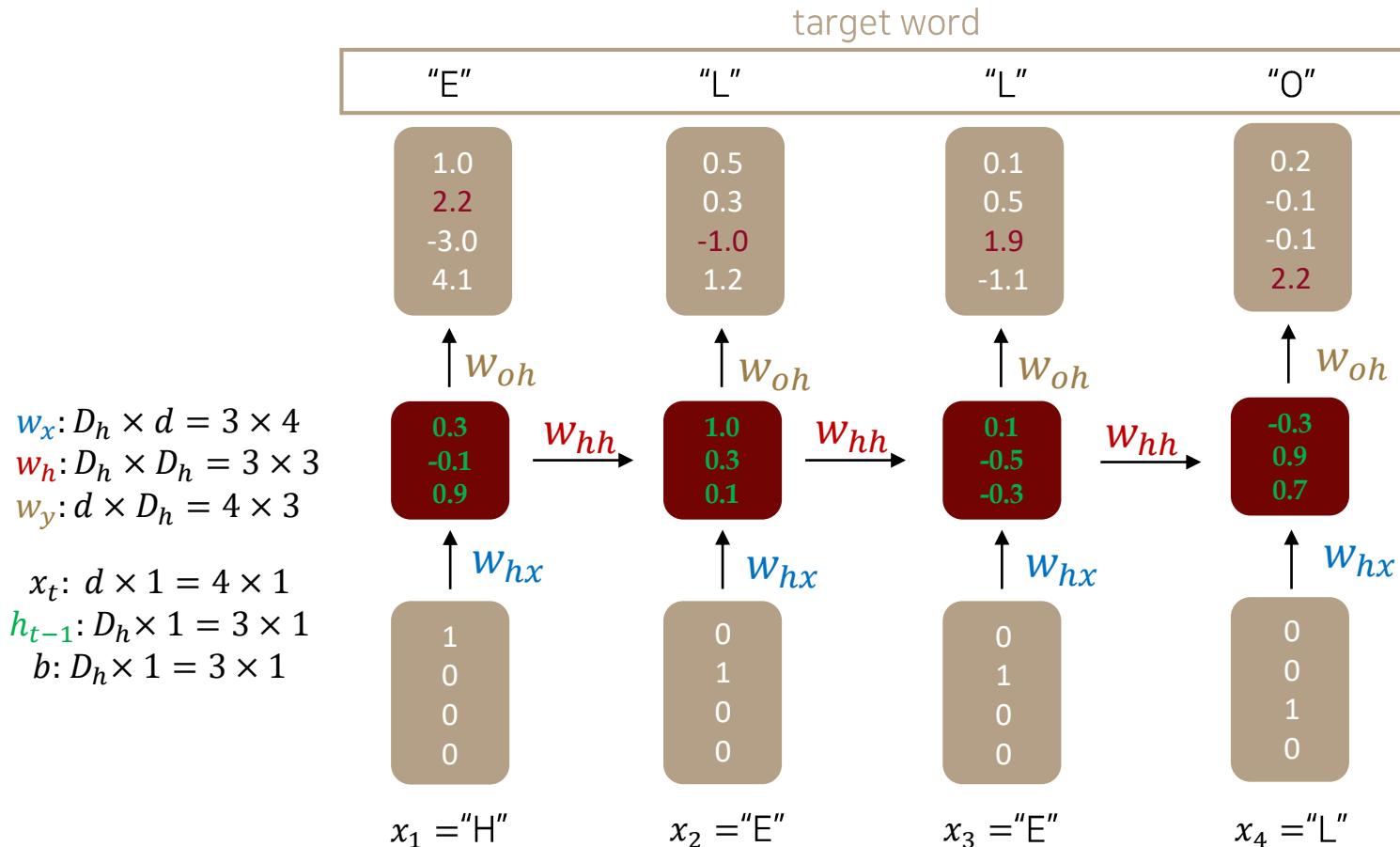
many-to-many
Ex) Entity recognition(개체명 인식)
POS tagging(품사 태깅)

2-5. RNN(Recurrent Neural Net)

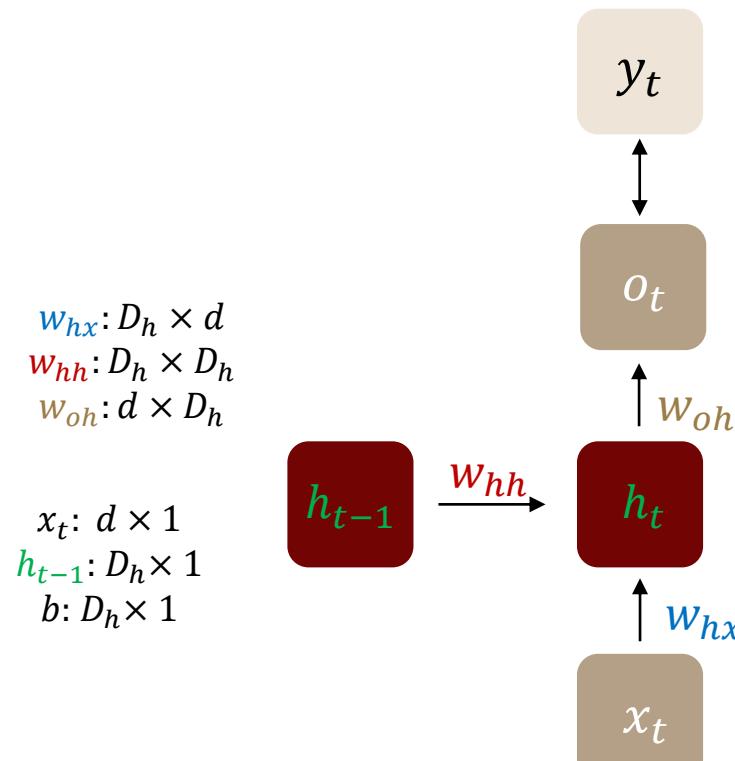
- 가중치 행렬 w_{xh}, w_{hh}, w_{hy} 가 매 time step마다 동일함(shared weight)
- 업데이트 해야 할 가중치(파라미터)가 input 길이에 상관없이 고정됨
- time step이 거듭될 때마다 context information이 누적됨



2-5. RNN(Recurrent Neural Net)



2-6. forward, backward of RNN



loss function = cross entropy(y_t, o_t)

output layer: $o_t = \text{softmax}(w_{oh}h_t + b)$

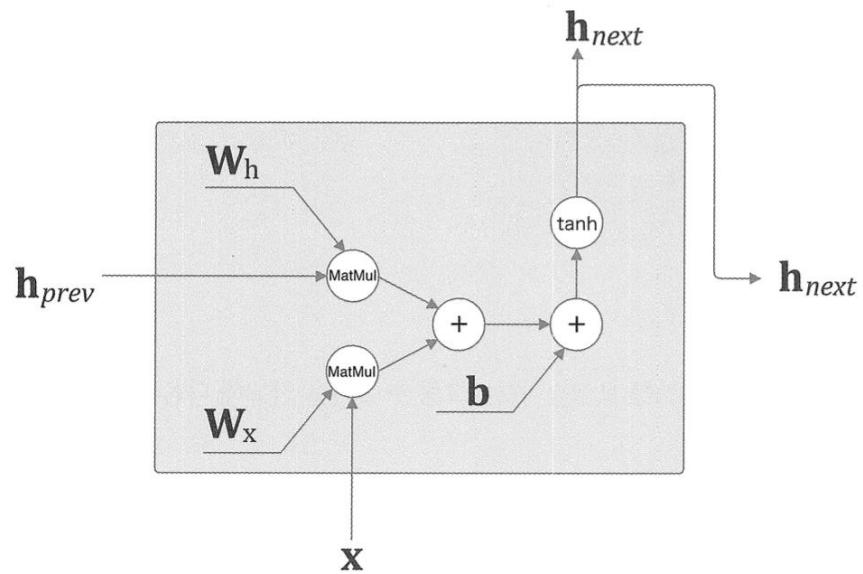
hidden layer: $h_t = \tanh(w_{hx}x_t + w_{hh}h_{t-1} + b)$

$$\tanh\left(\begin{array}{|c|c|c|c|} \hline \textcolor{blue}{B} & \textcolor{blue}{B} & \textcolor{blue}{B} & \textcolor{blue}{B} \\ \hline \end{array} \right) \times \begin{array}{|c|c|c|} \hline \textcolor{gray}{A} \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline \textcolor{red}{C} & \textcolor{red}{C} & \textcolor{red}{C} \\ \hline \end{array} \times \begin{array}{|c|c|} \hline \textcolor{green}{D} \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline \textcolor{gray}{E} \\ \hline \end{array}$$

w_{hx} x_t w_{hh} h_{t-1} b

2-6. forward, backward of RNN

forward



```
class RNN:  
    def __init__(self, Wx, Wh, b):  
        self.params = [Wx, Wh, b]  
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]  
        self.cache = None  
  
    def forward(self, x, h_prev):  
        Wx, Wh, b = self.params  
        t = np.matmul(h_prev, Wh) + np.matmul(x, Wx) + b  
        h_next = np.tanh(t)
```

Given $\mathbf{h}_t = \tanh(\mathbf{x}_t, \mathbf{h}_{t-1})$ and $\mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$

Loss Function

$$L(x, y, w_h, w_o) = \sum_{t=1}^T l(y_t, o_t)$$

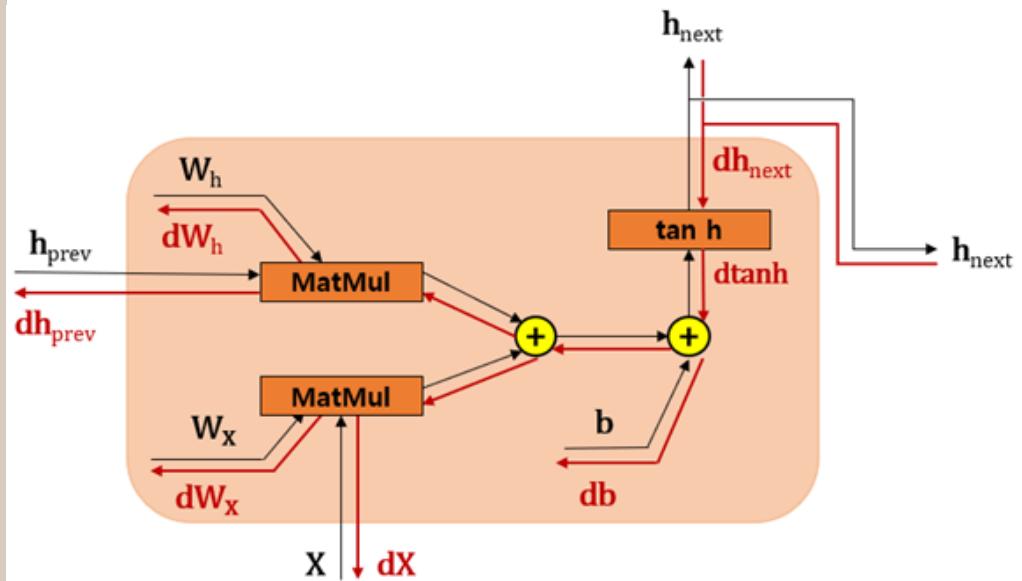
$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^T \partial_{w_h} l(y_t, o_t) = \sum_{t=1}^T \partial_{o_t} l(y_t, o_t) \partial_{h_t} g(h_t, w_h) [\partial_{w_h} h_t]$$

$$\partial_{w_h} h_t = (\partial_h f_{w_h})(x_t, h_{t-1}) \partial_{w_h} h_{t-1} + (\partial_{w_h} f_{w_h})(x_t, h_{t-1})$$

$$\partial_{w_h} h_t = \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \partial_{h_{j-1}} f_{w_h}(x_j, h_{j-1}) \right) \partial_{w_h} f_{w_h}(x_t, h_{t-1}) + \partial_{w_h} f_{w_h}(x_t, h_{t-1})$$

2-6. forward, backward of RNN

Given $\mathbf{h}_t = \tanh(w_{hx}\mathbf{x}_t + w_{hh}\mathbf{h}_{t-1} + b)$ and $\mathbf{o}_t = \text{softmax}(w_{oh}\mathbf{h}_t + b)$



덧셈 역전파: 상위 node에서 흐른 gradient를 그대로 보냄
곱셈 역전파: 상위 node의 forward 때의 값을 서로 바꾼다

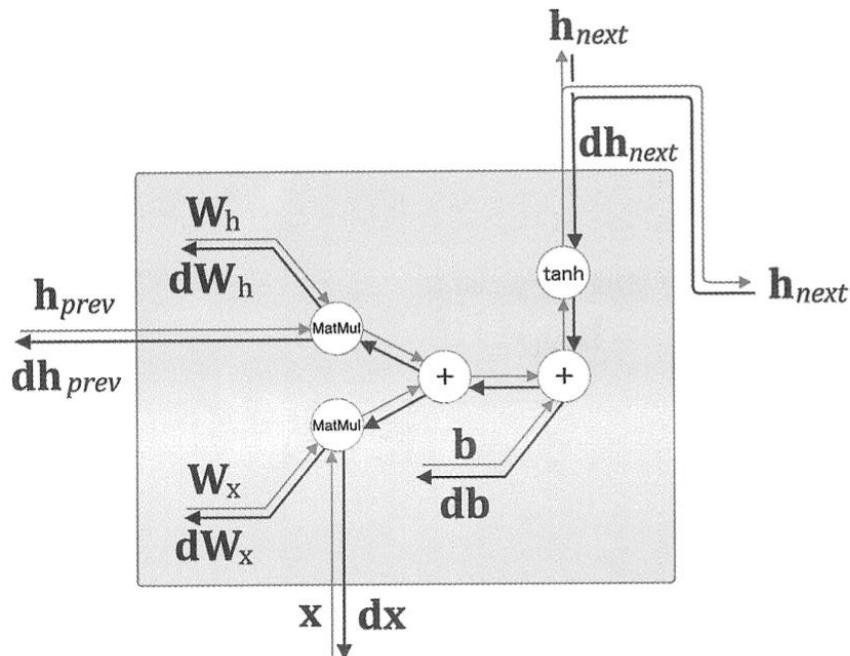
dh_{next} : 상위 node에서 그대로 받아서 시작
 $dtanh = dh_{\text{next}} * (\tanh)' = dh_{\text{next}} * (1 - \tanh^2)$

db : 데이터의 0번째 column의 합

$$dW_h = dtanh * h_{\text{prev}} = dh_{\text{next}} * (1 - \tanh^2) * h_{\text{prev}}$$
$$dh_{\text{prev}} = dtanh * W_h = dh_{\text{next}} * (1 - \tanh^2) * W_h$$

2-6. forward, backward of RNN

backward



```
def backward(self, dh_next):  
    Wx, Wh, b = self.params  
    x, h_prev, h_next = self.cache  
    dt = dh_next * (1 - h_next ** 2)  
    db = np.sum(dt, axis=0)  
    dWh = np.matmul(h_prev.T, dt)  
    dh_prev = np.matmul(dt, Wh.T)  
    dWx = np.matmul(x.T, dt)  
    dx = np.matmul(dt, Wx.T)  
    self.grads[0][...] = dWx  
    self.grads[1][...] = dWh  
    self.grads[2][...] = db  
    return dx, dh_prev
```

Given $\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1})$ and $\mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$

Loss Function

$$L(x, y, w_h, w_o) = \sum_{t=1}^T l(y_t, o_t)$$

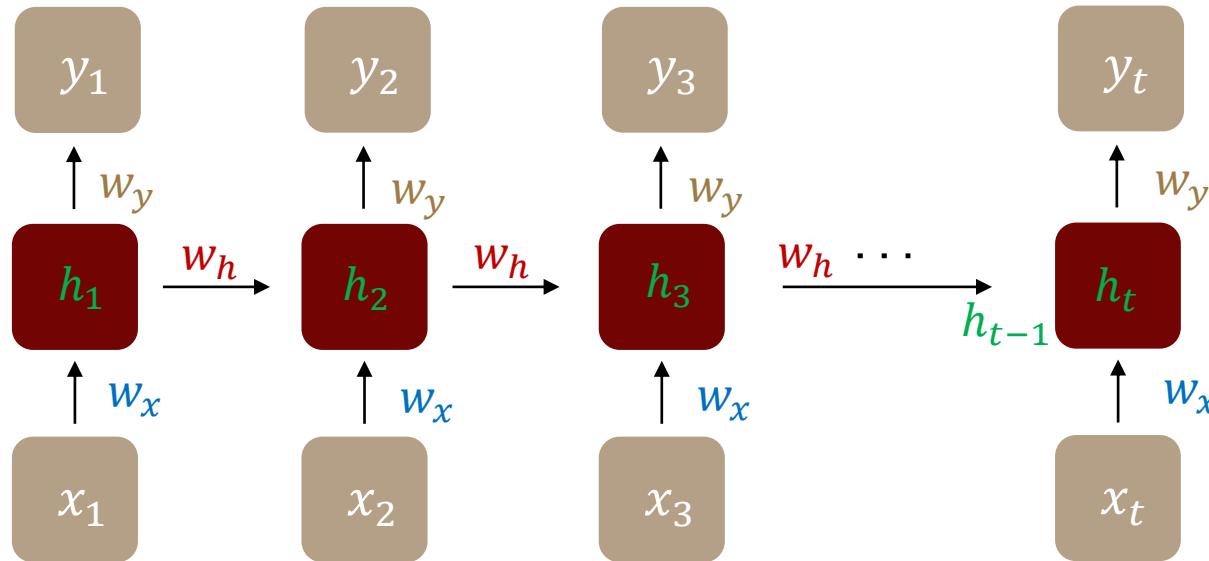
$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^T \partial_{w_h} l(y_t, o_t) = \sum_{t=1}^T \partial_{o_t} l(y_t, o_t) \partial_{h_t} g(h_t, w_h) [\partial_{w_h} h_t]$$

$$\partial_{w_h} h_t = (\partial_h f_{w_h})(x_t, h_{t-1}) \partial_{w_h} h_{t-1} + (\partial_{w_h} f_{w_h})(x_t, h_{t-1})$$

$$\partial_{w_h} h_t = \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \partial_{h_{j-1}} f_{w_h}(x_j, h_{j-1}) \right) \partial_{w_h} f_{w_h}(x_t, h_{t-1}) + \partial_{w_h} f_{w_h}(x_t, h_{t-1})$$

What if $1 < \|\partial_{h_{j-1}} f\|$ occur?

BPTT(Backpropagation Through Time)

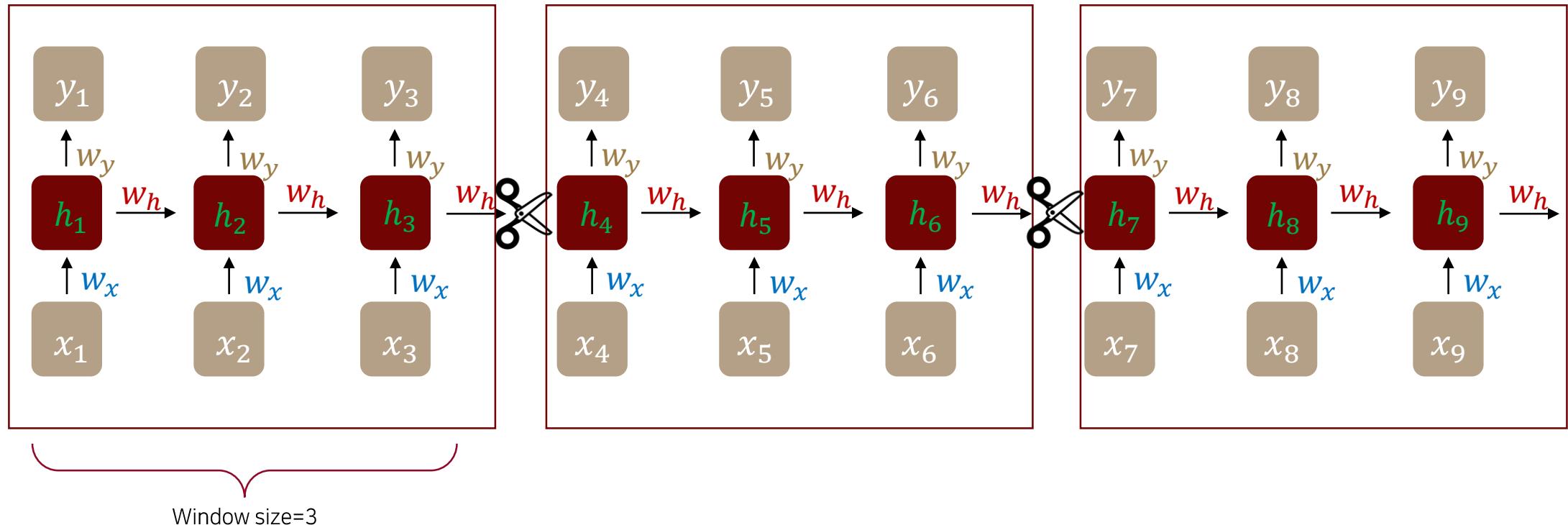


time step(=문장길이)가 길어짐에 따라

- BPTT에 소모되는 computing power가 커짐
- 메모리 사용량 커짐
- Gradient vanishing or exploding problem

Sol) **Truncated BPTT**

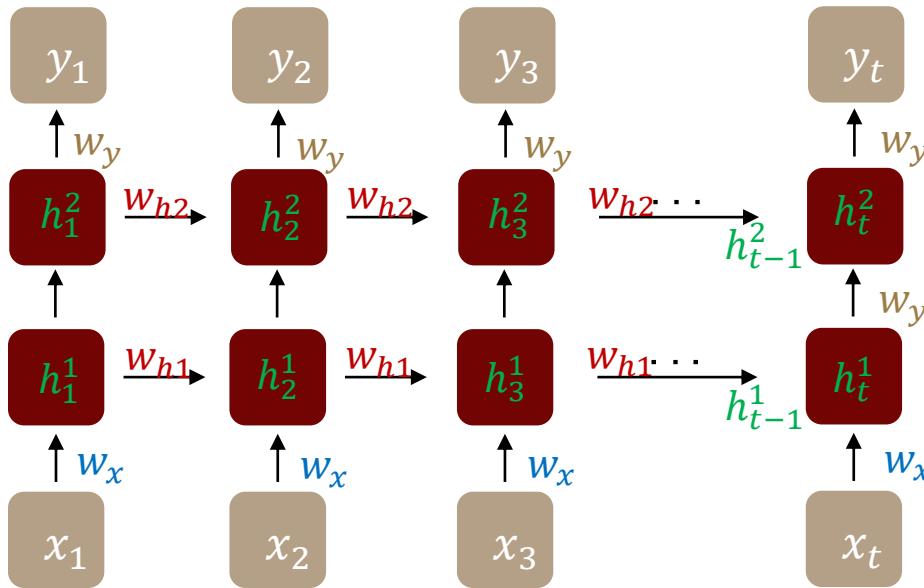
Truncated BPTT



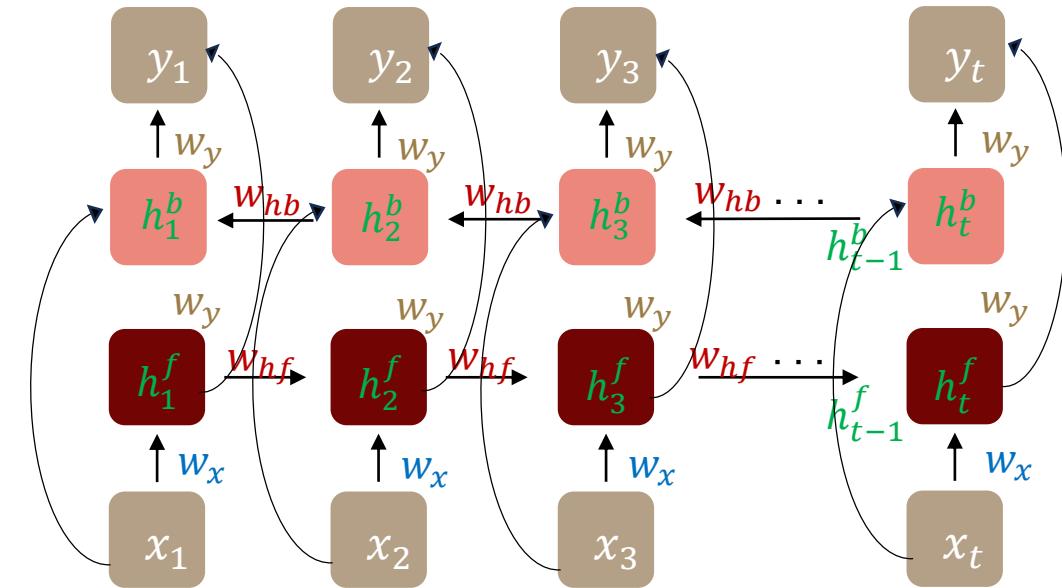
기울기 소실, 메모리 공간 낭비, 연산량 문제를 해결하기 위해
 Backpropagation할 시에 적당한 길이로 끊음
 forward propagation에선 끊어지지 않음!

2-8. Deep, Bidirectional RNN

Deep RNN



Bidirectional RNN



- Hidden layer를 1개가 아닌 여러 개 쌓은 형태
- Hidden layer 개수만큼 가중치도 늘어남

- 앞 시점의 hidden state와 뒤 시점의 hidden state를 사용하는 양방향식

예문: 민아는 교환학생을 다녀온 뒤로 얼굴에 _____ 가득해서 참 다행이야
1) 화색이 2) 슬픔이 3) 정색이

2-8. Deep, Bidirectional RNN

Vanilla RNN

```
import torch
import torch.nn as nn

input_size=5
hidden_size=8

inputs = torch.Tensor(1,10,5) #batch size, time step, input size

# batch_firse=True => inputs의 첫번째 차원(=1)이 batch size임을 명시함
cell = nn.RNN(input_size, hidden_size, batch_firse=True)

# outputs = 각 time step에서 output layer에 들어가기 직전 hidden state
# _status = 마지막 time step의 hidden state(layer 수에 따라 여러개일 수 있음)
outputs, _status = cell(inputs)
print(outputs.shape)
print(_status.shape)

torch.Size([1, 10, 8])
torch.Size([1, 1, 8])
```

Deep RNN

```
input_size=5
hidden_size=8

inputs = torch.Tensor(1,10,5) #batch size, time step, input size

# num_layers = hidden layer의 개수
cell = nn.RNN(input_size, hidden_size, num_layers=2, batch_firse=True)

outputs, _status = cell(inputs)
print(outputs.shape)
print(_status.shape)

torch.Size([1, 10, 8])
torch.Size([2, 1, 8])
```

Bidirectional RNN

```
input_size=5
hidden_size=8

inputs = torch.Tensor(1,10,5) #batch size, time step, input size

# num_layers = hidden layer의 개수
cell = nn.RNN(input_size, hidden_size, num_layers=2,
              batch_firse=True, bidirectional=True)

outputs, _status = cell(inputs)
print(outputs.shape)
print(_status.shape)

torch.Size([1, 10, 16])
torch.Size([4, 1, 8])
```

03 LSTM

보다 효율적으로 기억하려면?

3-1. Vanishing or Exploding Gradient

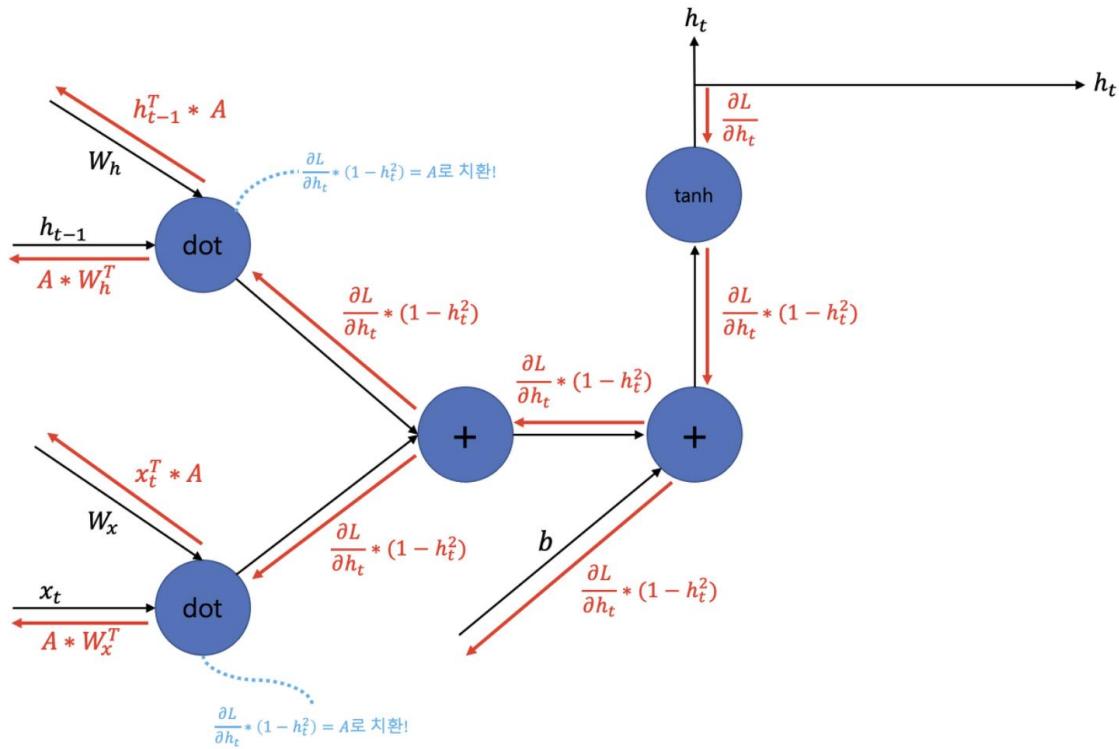
승현이는 시계열 스터디를 하러 스룸에 왔다.
역시 부지런하다.
오자마자 짐을 풀고 아메리카노 한잔을 했다.
뒤이어 동주도 스터디를 위해 두번째로 스룸에 도착했다.
승현이는 _____에게 인사했다.

🤔 문장이 길어져도 빈칸에 들어갈 사람이 '서연'이라고
RNN은 기억할 수 있을까?

장기 의존성 문제(the problem of Long-Term Dependencies)

문장이 길어짐(time step이 많아짐)에 따라
기울기가 소실(vanishing)되거나 폭등(exploding)

3-1. Vanishing or Exploding Gradient



tanh 연산에서 backprop하면 기울기 소실 위험

그림 6-6 $y = \tanh(x)$ 의 그래프(점선은 미분)

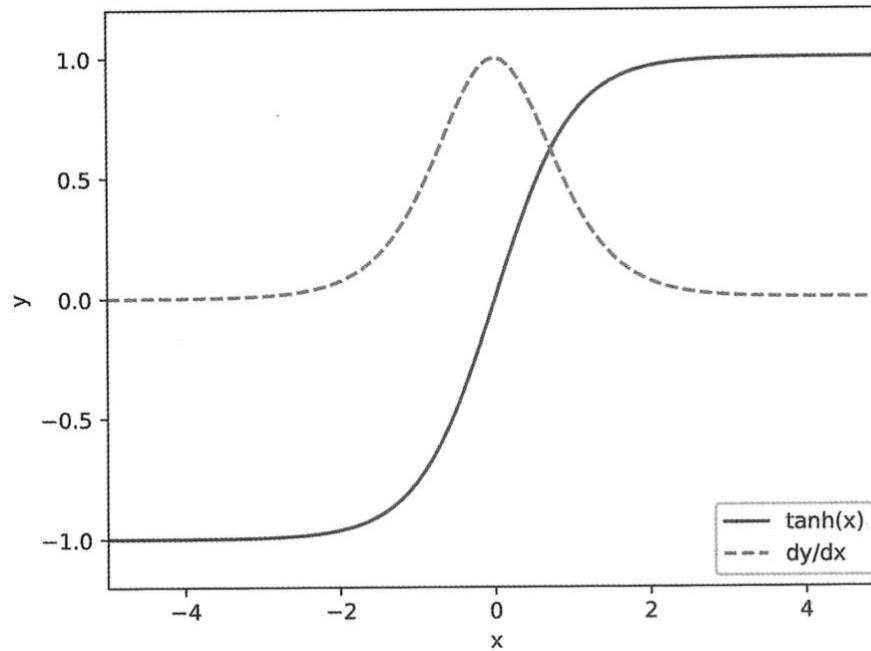


그림: 밑바닥부터 시작하는 딥러닝2

3-1. Vanishing or Exploding Gradient

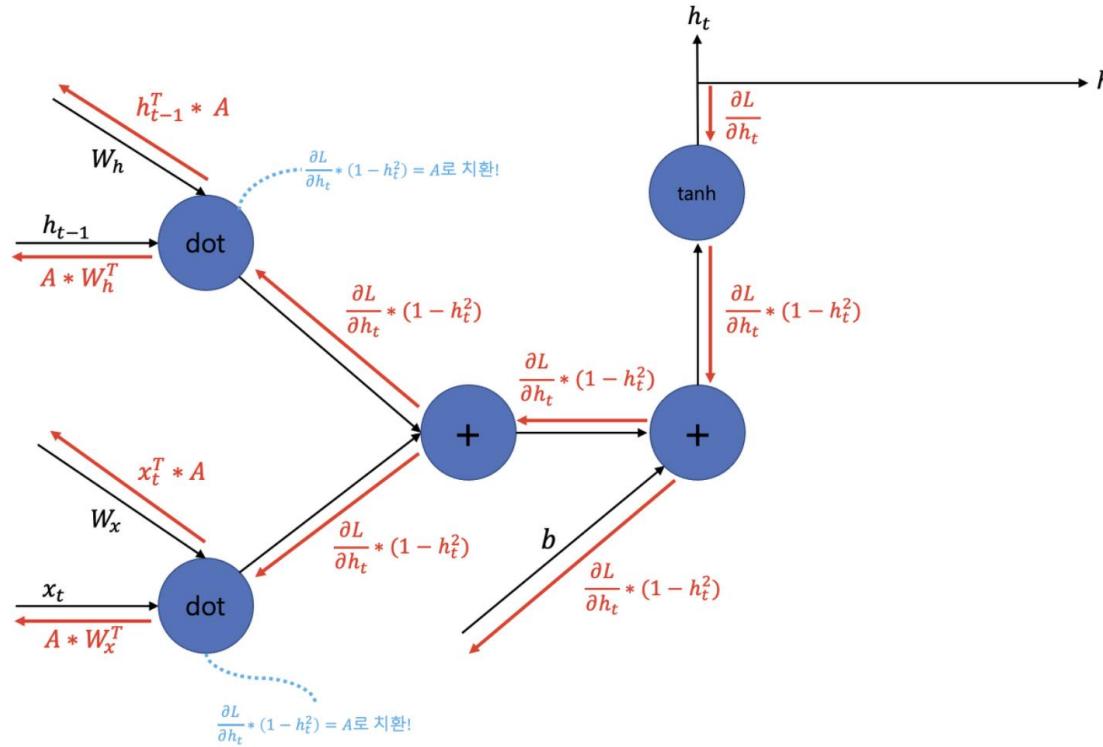


그림: 밑바닥부터 시작하는 딥러닝2

matmul 연산에서 backprop하면 기울기 폭등 위험

그림 6-7 RNN 계층의 행렬 곱에만 주목했을 때의 역전파의 기울기

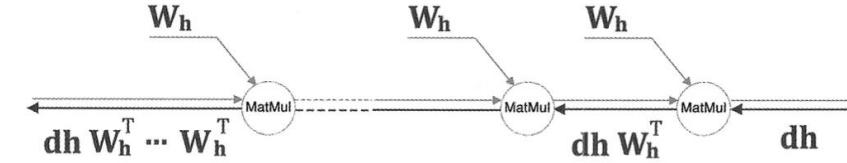
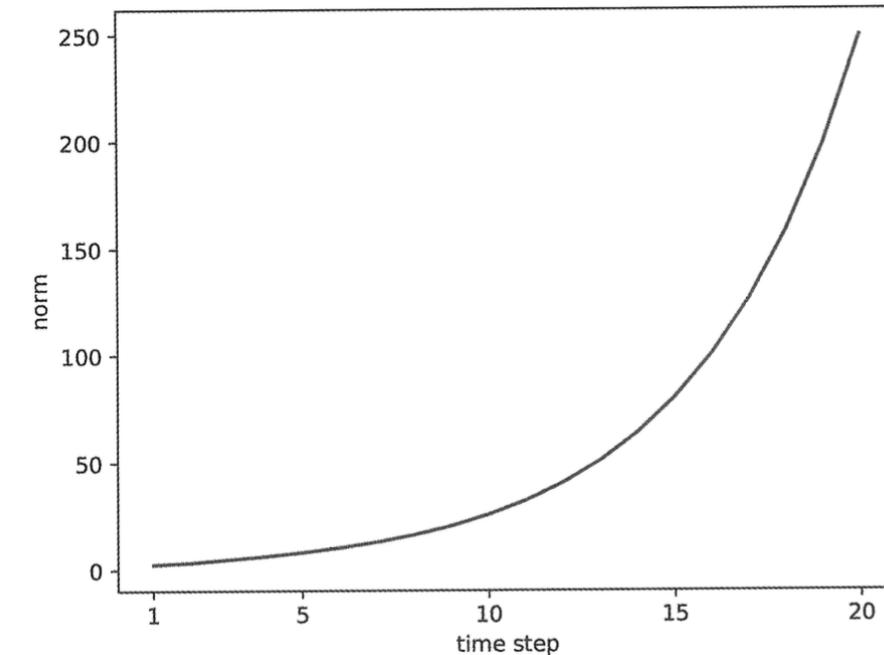


그림 6-8 기울기 dh는 시간 크기에 비례하여 지수적으로 증가한다.



3-1. Vanishing or Exploding Gradient

Solution to exploding gradient

Clipping

기울기가 exploding하려 할 때
인위적으로 그 기울기값에 특정 조치를 취해주는 기법

if $\|\hat{g}\| \geq threshold :$

$$\hat{g} = \frac{threshold}{\|\hat{g}\|} \hat{g}$$

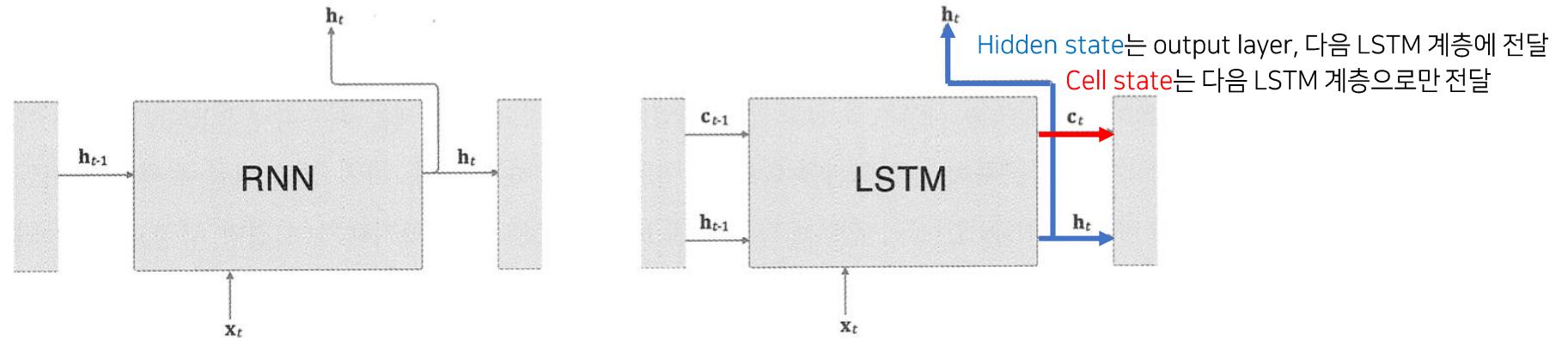
Solution to vanishing gradient

LSTM

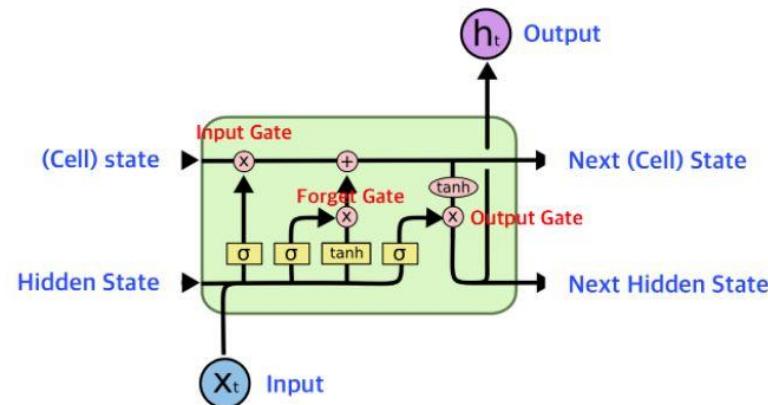
기존 RNN architecture에 Gate를 추가함으로써
기억력을 개선하는 기법
Long Short Term Memory

Then, how?

그림 6-11 RNN 계층과 LSTM 계층 비교



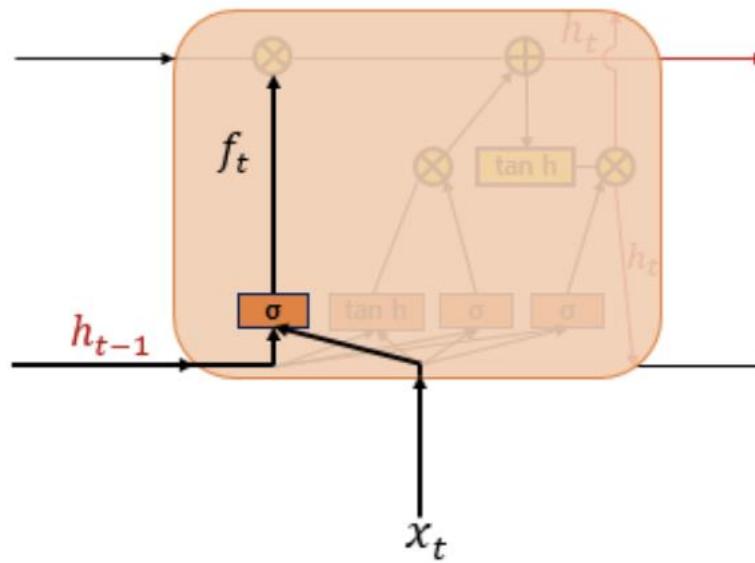
- Hidden state만 존재하던 RNN과 달리 Cell state가 추가됨.
- c_t 에는 t 시점까지의 기억 정보들이 누적되어있음.
- 이 때 정보들을 얼마나 기억할지를 정할 수 있음.



- LSTM은 아래 세 가지 gate로 얼마나 기억하고, 잊을지를 정한다!
- Input gate
 - Forget gate
 - Output gate

그림: 밑바닥부터 시작하는 딥러닝2

forget gate



이전 기억을 얼마나 삭제할 것인지 정하는 gate

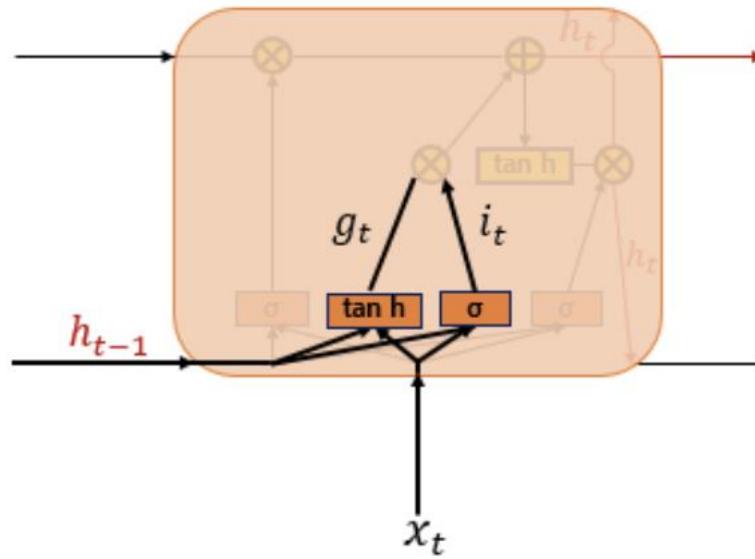
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \Rightarrow 0\text{에서 }1\text{의 값 출력}$$

- output f_t 는 직전 시점의 cell state(c_{t-1})와 곱해짐.
- 0에 가까울수록 기억이 많이 삭제된 것이고, 1에 가까울수록 온전히 기억한 것임
- 얼마나 기억할 것인지를 LSTM이 가중치를 업데이트하며 스스로 학습함.

그림: 딥러닝을 이용한 자연어처리

3-3. Three Gates of LSTM

Input gate



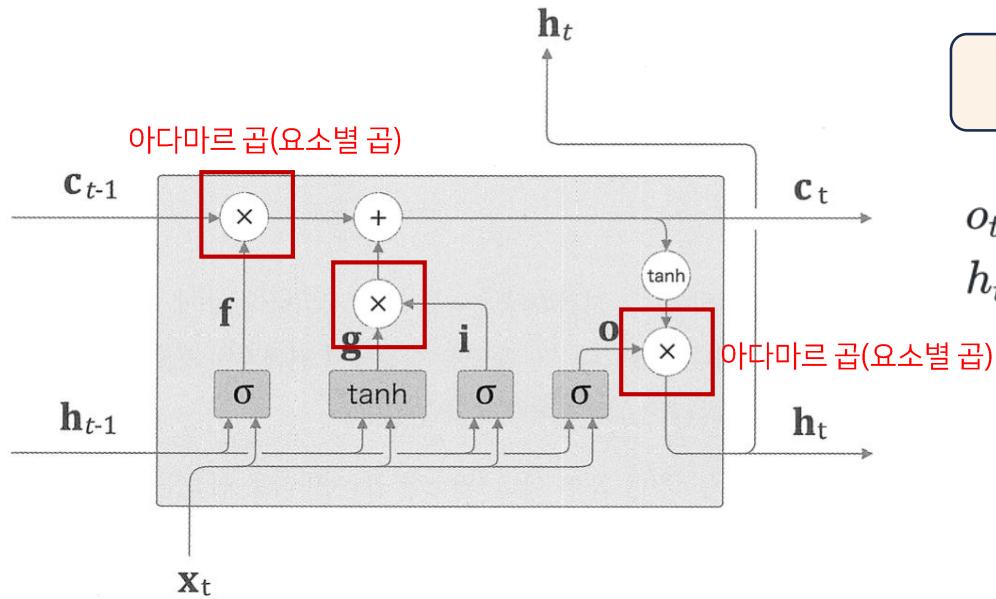
새로운 정보를 얼마나 누적할 것인지 정하는 gate

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \Rightarrow 0\text{에서 }1\text{의 값 출력}$$
$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \Rightarrow -1\text{에서 }1\text{의 값 출력}$$

그림: 딥러닝을 이용한 자연어처리

3-3. Three Gates of LSTM

output gate



Input, forget gate의 값들을 다음 단계로 전달해주는 gate

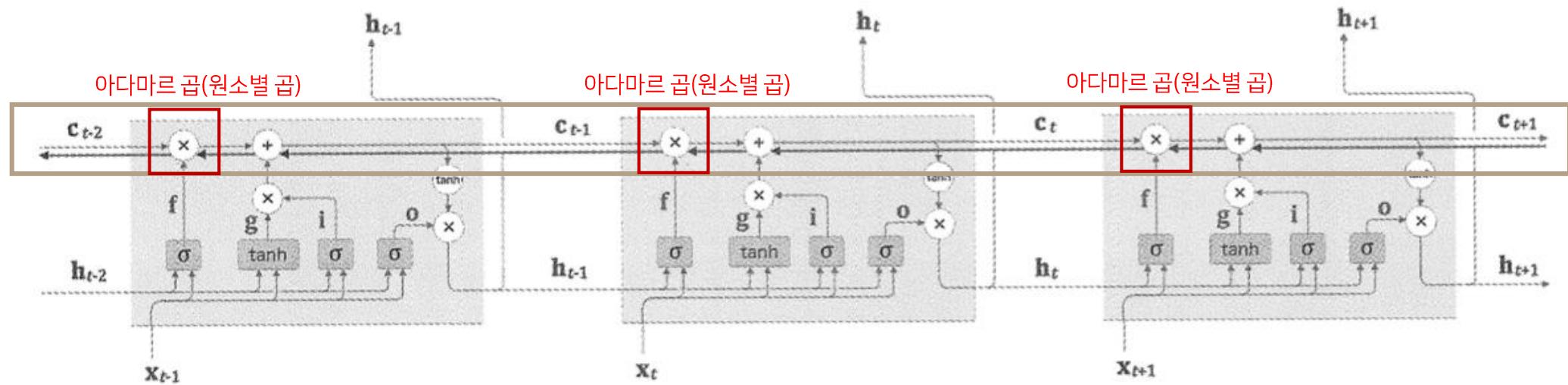
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \Rightarrow 0\text{에서 }1\text{의 값 출력}$$
$$h_t = o_t \circ \tanh(c_t)$$

그림: 밑바닥부터 시작하는 딥러닝2

3-3. Three Gates of LSTM

Cell state

$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$



🤔 이렇게 gate를 만들어주는 것이 어떤 원리로 Vanishing gradient를 막아줄 수 있는 건가?

그림: 밑바닥부터 시작하는 딥러닝2

04 GRU

LSTM보다도 적은 계산량으로

GRU(Gated Recurrent Unit): gate를 2개로 줄임으로써 성능은 LSTM과 유사하면서 속도를 개선시킨 모델

reset gate

과거의 정보를 적당히 reset해주는 gate

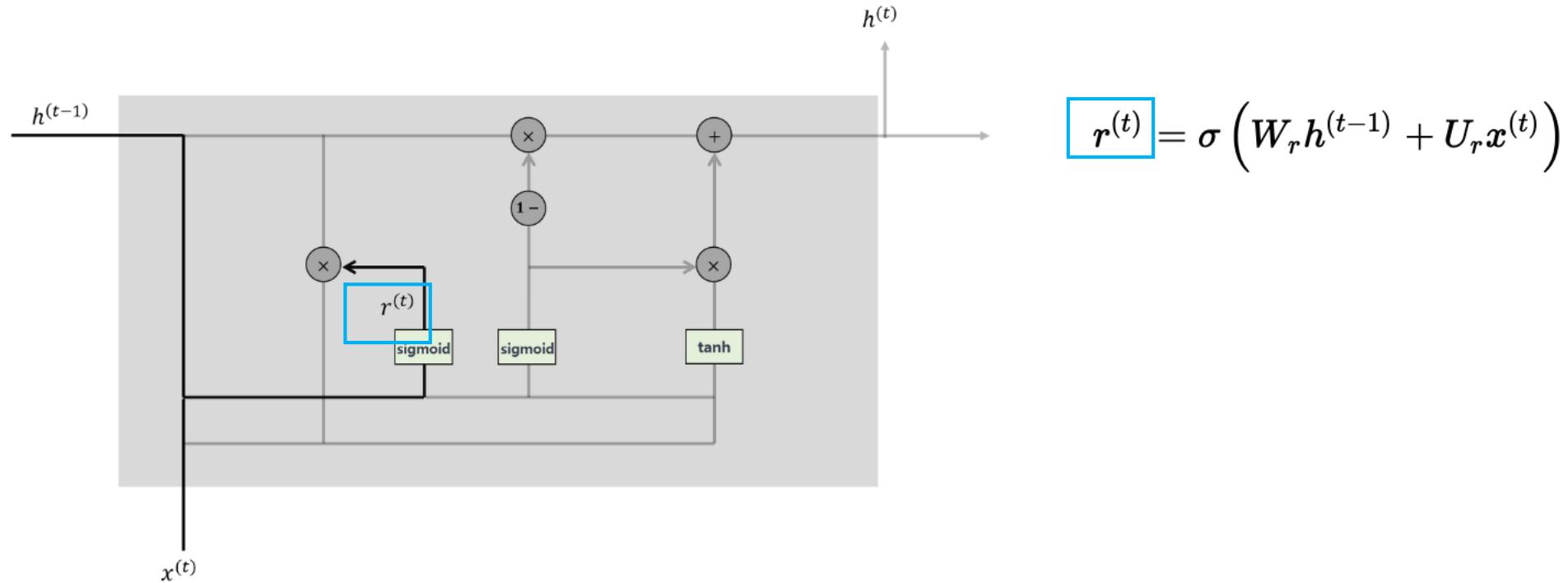


그림: <https://yjjo.tistory.com/18>

GRU(Gated Recurrent Unit): gate를 2개로 줄임으로써 성능은 LSTM과 유사하면서 속도를 개선시킨 모델

update gate

과거 정보와 현재 정보의 업데이트 비율을 결정하는 gate

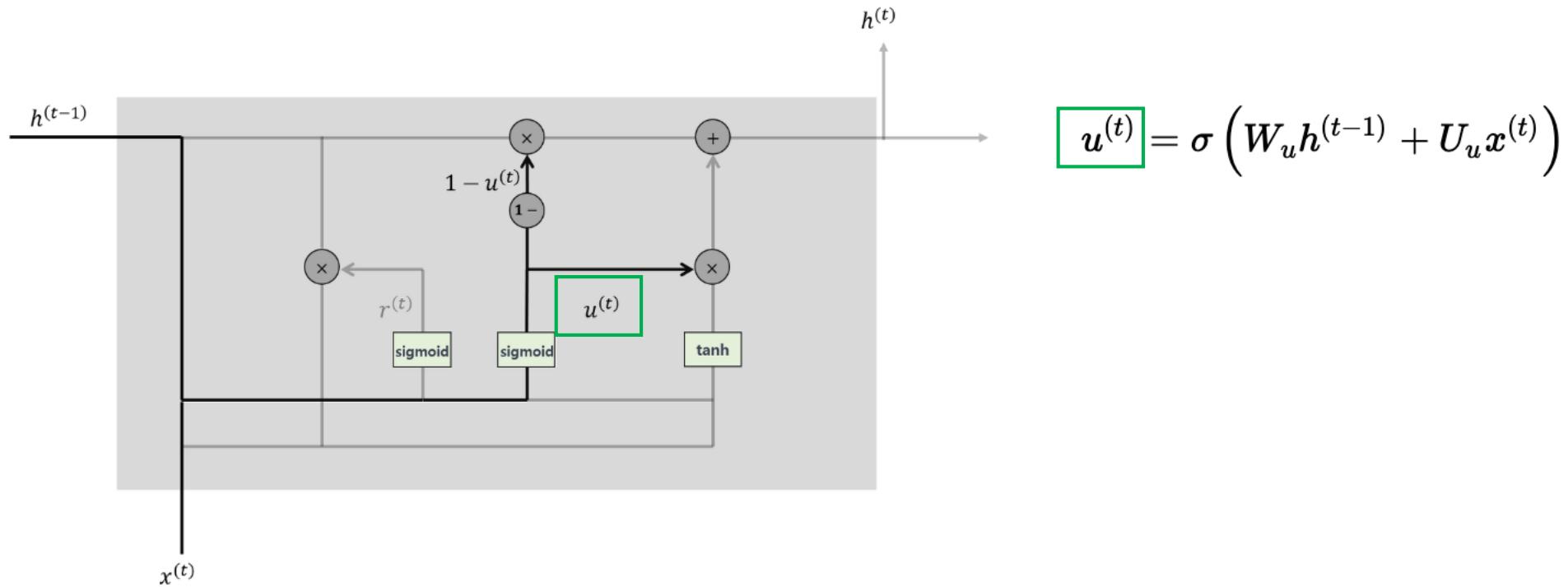
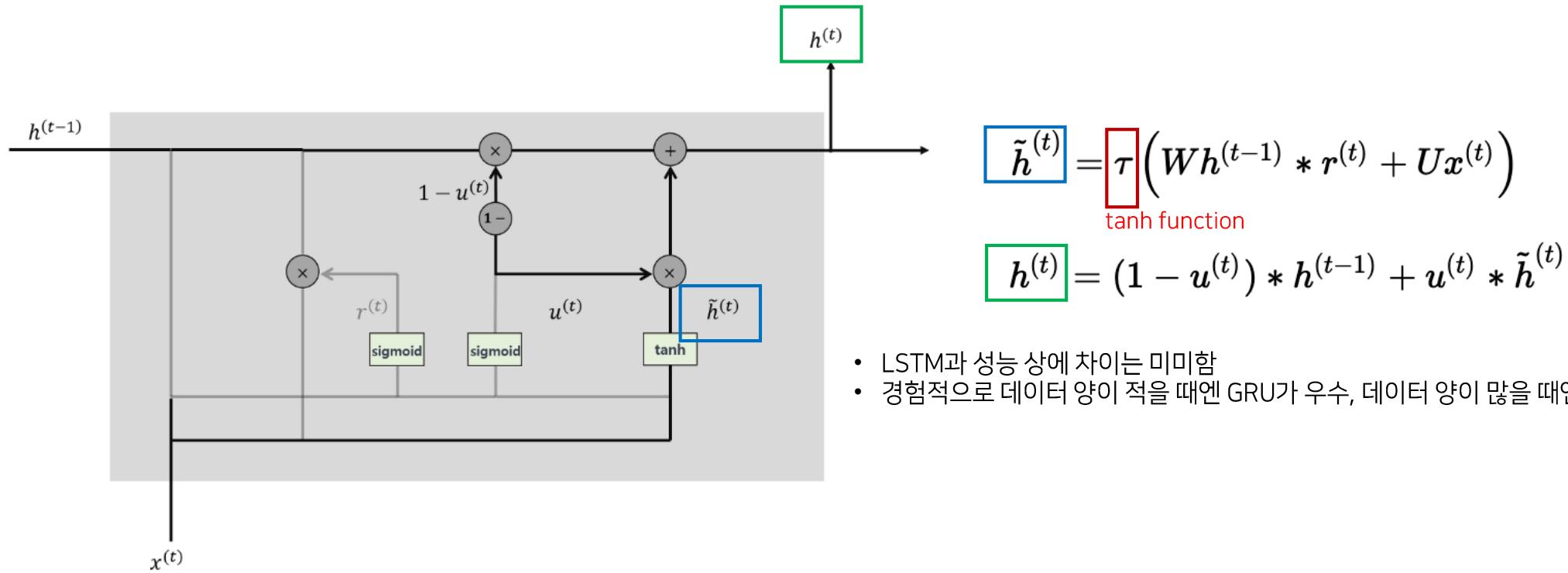


그림: <https://yjjo.tistory.com/18>

GRU(Gated Recurrent Unit): gate를 2개로 줄임으로써 성능은 LSTM과 유사하면서 속도를 개선시킨 모델

Candidate Hidden State & Final Hidden State



- LSTM과 성능 상에 차이는 미미함
- 경험적으로 데이터 양이 적을 때엔 GRU가 우수, 데이터 양이 많을 때엔 LSTM이 우수

그림: <https://yjjo.tistory.com/18>

05 ELMo

문맥을 반영한 워드임베딩

5-1. What is ELMo?



삼성, 사상 첫 통합우승 4연패(종합2보)

word2vec

0.45	0.02	...	0.89
------	------	-----	------

"연패" embedding vector



'18연패' 한화 이글스 무한 추락, KBO리그도 우울

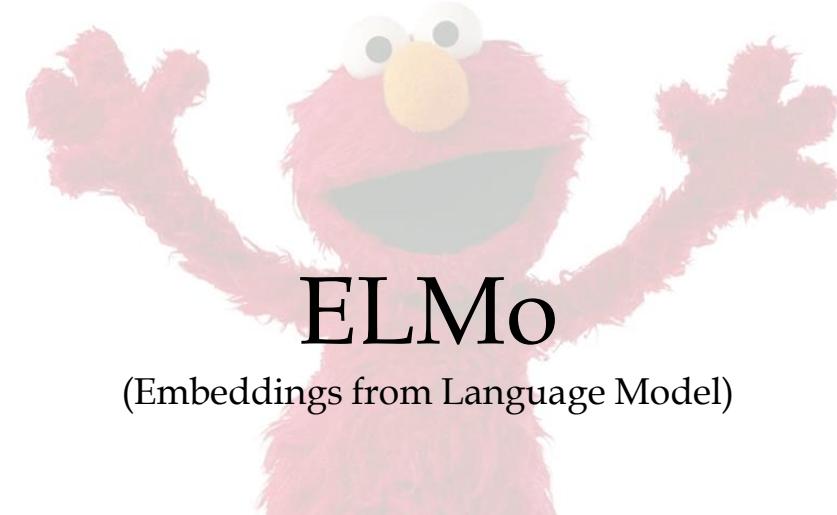
word2vec

0.45	0.02	...	0.89
------	------	-----	------

"연패" embedding vector



다른 의미임에도 같은 벡터로 임베딩
문맥을 고려한 워드임베딩이 필요할 것 같은데…



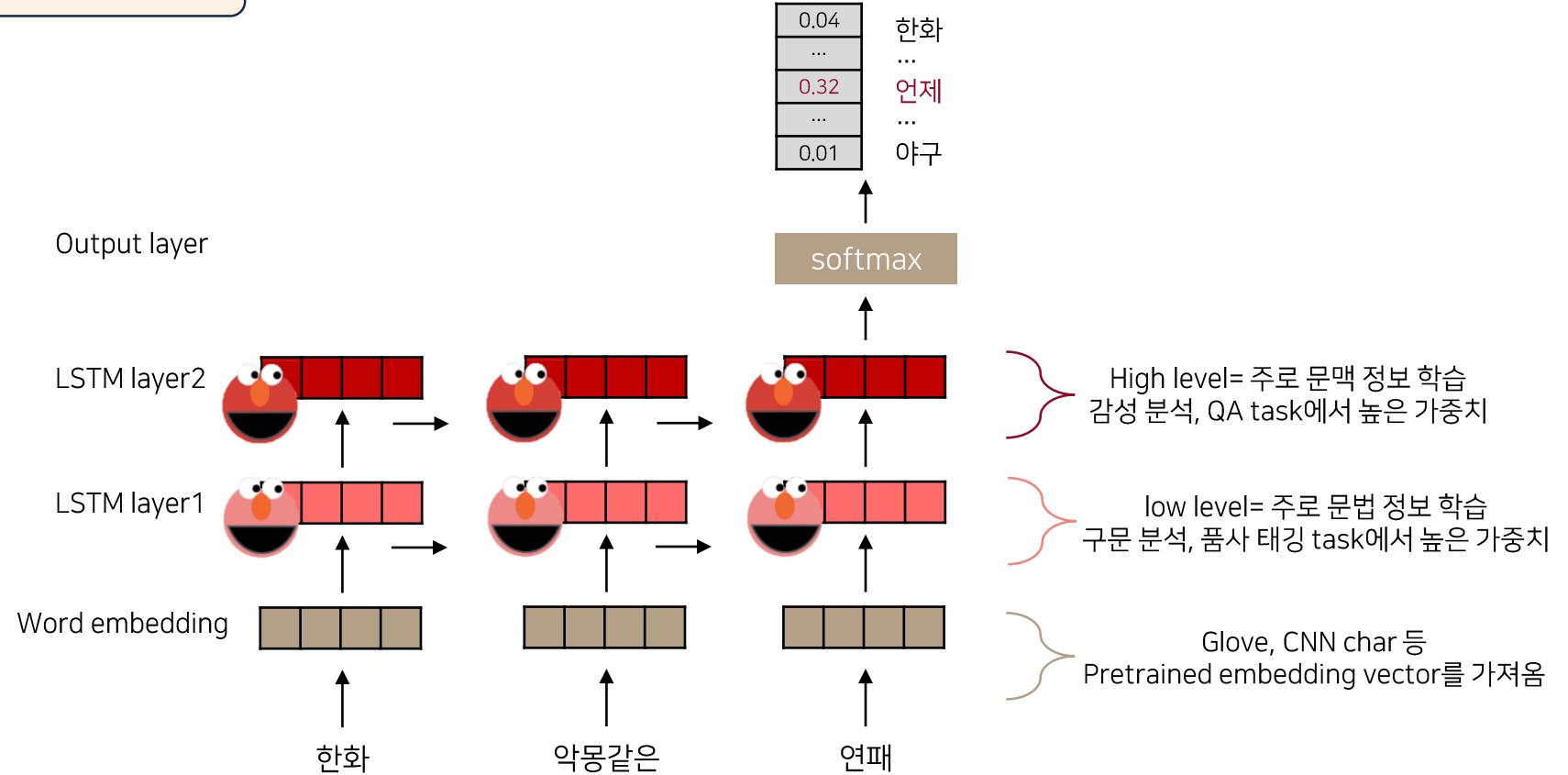
ELMo
(Embeddings from Language Model)



BERT
See you soon

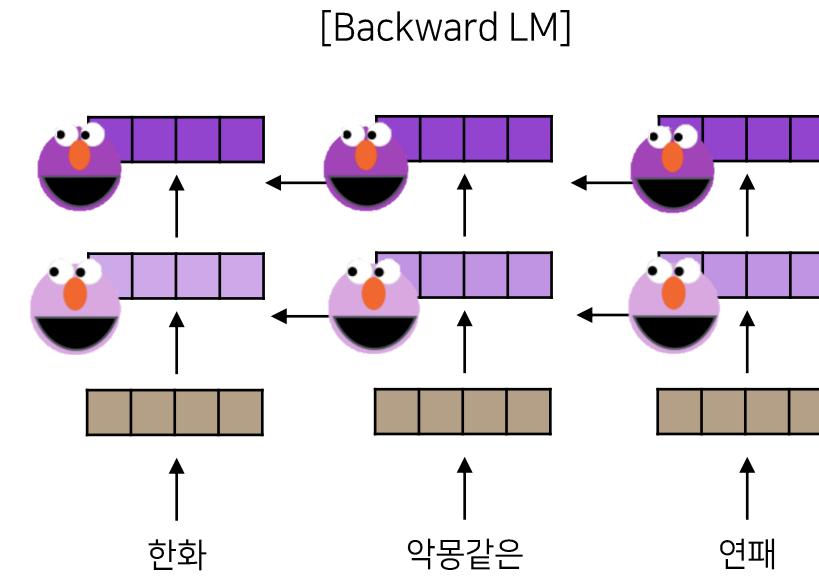
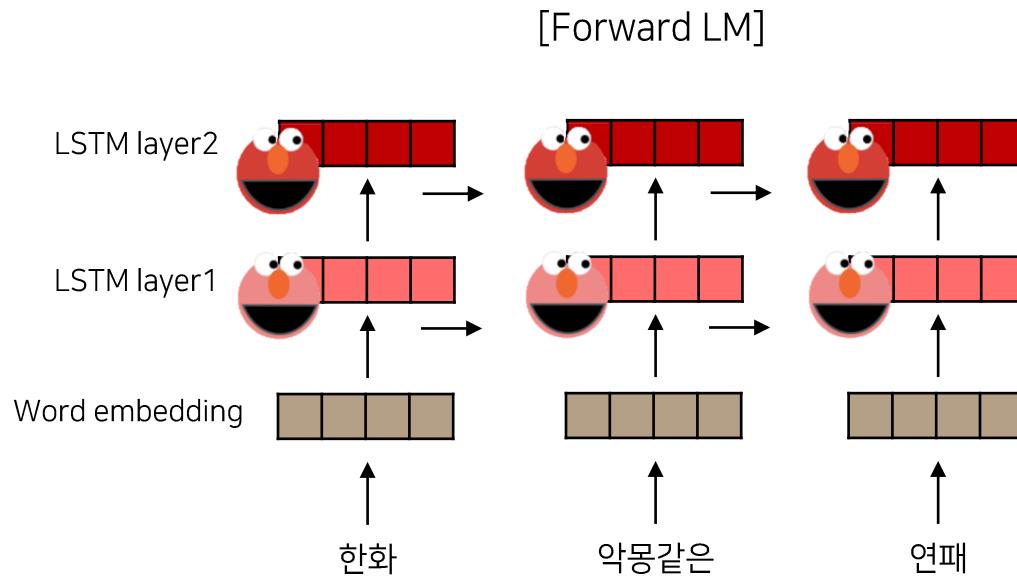
5-2. ELMo with biLSTM

예문: 한화, 악몽같은 연패 언제 탈출하나

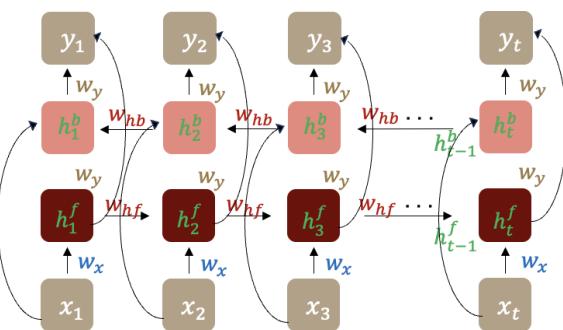


5-2. ELMo with biLSTM

예문: 한화, 악몽같은 연파 언제 탈출하나



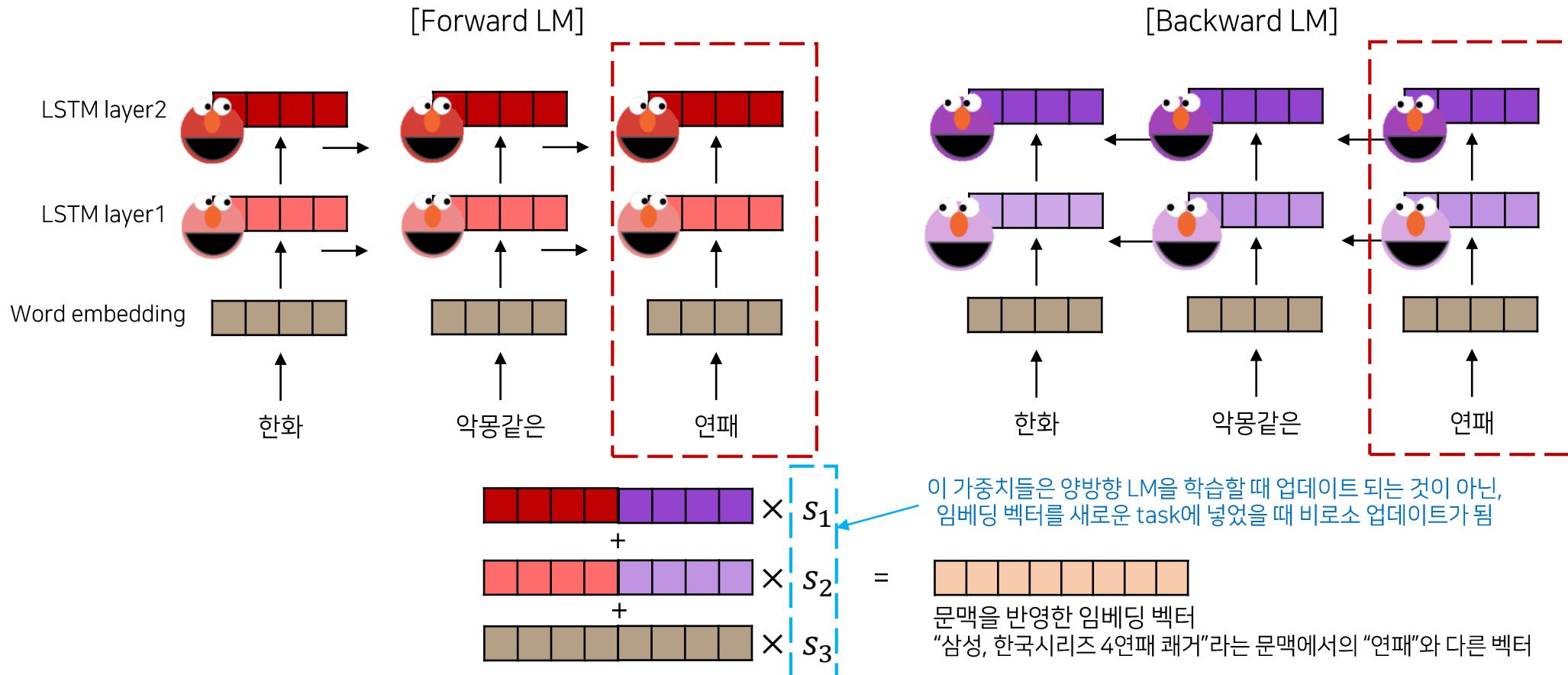
Bidirectional RNN이 hidden state만 앞뒤로 공유하는 반면,
위 경우 두 개를 각각 다른 Language Model로 보고 개별적으로 학습



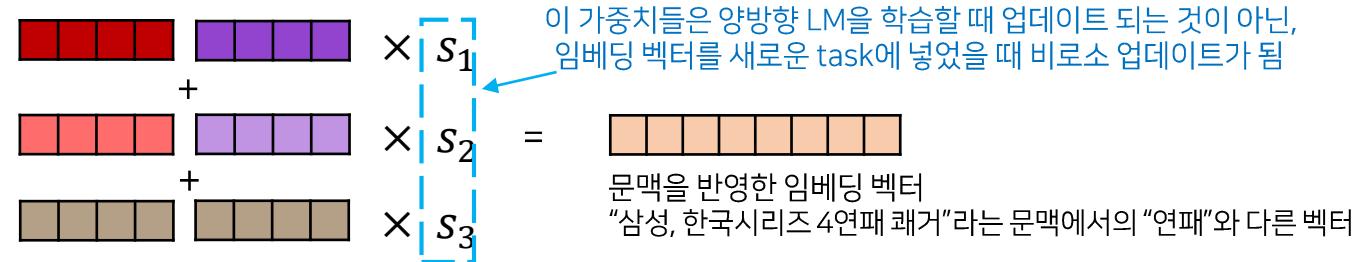
5-2. ELMo with biLSTM

예문: 한화, 악몽같은 연패 언제 탈출하나

GOAL: 예문의 문맥 안에서 “연패”의 임베딩 벡터 구하기



5-2. task specific ELMo



ELMo: Embeddings from Language Models

Peters et. al (2018)

- ELMo for downstream task

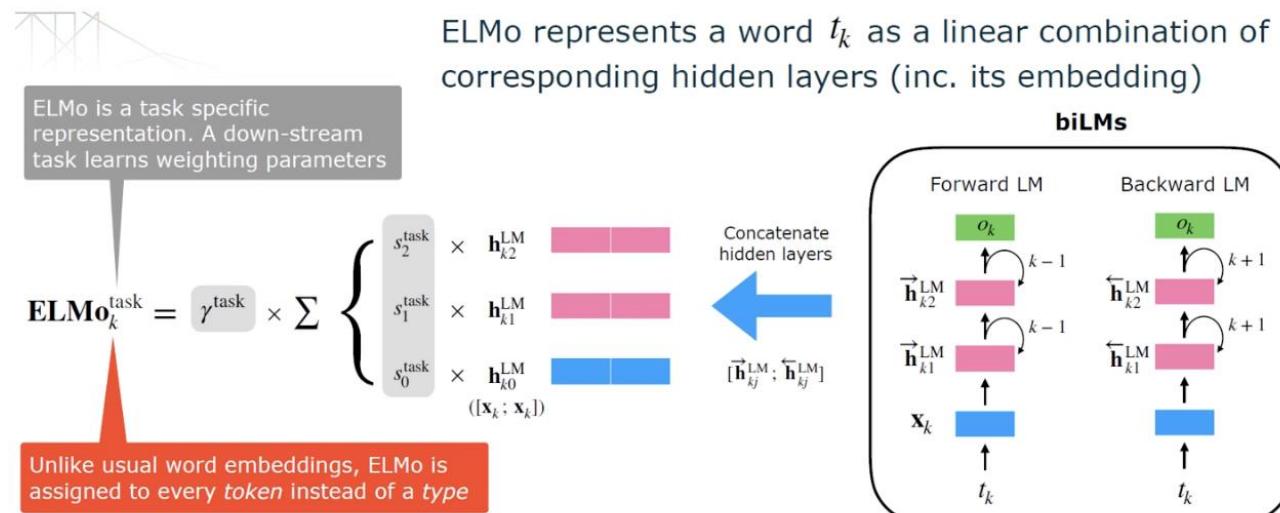


그림: 고려대학교 DSBA <https://www.youtube.com/watch?v=zV8klUwH32M&t=1053s>

06 Announcement

Week2 예습과제 Review, week3 예복습 과제 안내, week4 진도 안내

6-1. 우수 예습과제 Review

19기 심승현님

2주차
예습과제1

RNN vs LSTM

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

6-2. Week3 예, 복습과제 안내, Week4 진도 안내



코드과제의 파일형식은 ipynb로, KUBIG 24-1 Github repo에 업로드 될 예정입니다!
Colab 환경에서 제작된 과제들이므로 [google colab](#)에서 실행하시는 것을 권장드립니다.

WEEK3 복습과제1

RNN, LSTM layer 구현

WEEK3 복습과제2

네이버 쇼핑 리뷰 감성분석
with RNN, LSTM, GRU

WEEK3 예습과제1

Attention 기계번역

WEEK4 진도

- Attention
- Transformer

WEEK4 진도 해당 범위(읽어오시길 권장 드립니다!)

- [딥러닝을 이용한 자연어 처리 입문]
- Ch15. 어텐션 매커니즘
 - Ch16. 트랜스포머

- [밑바닥부터 시작하는 딥러닝2]
- Ch8. 어텐션

E.O.D
수고하셨습니다!