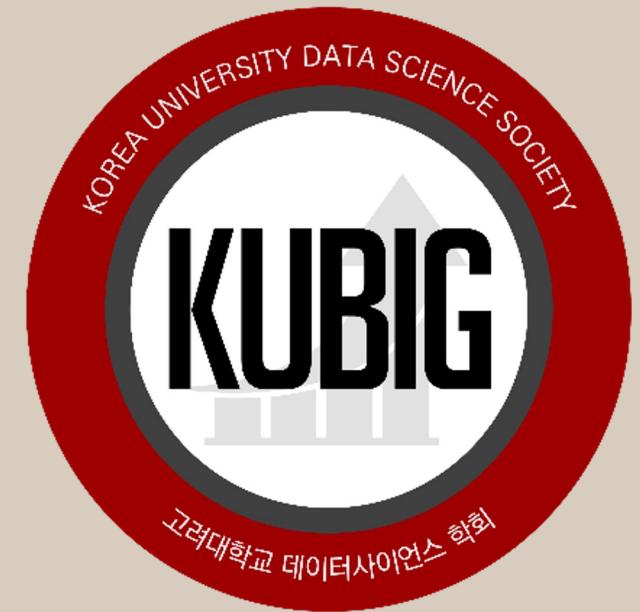


KUBIG 24-W  
겨울방학 BASIC STUDY SESSION

# NLP SESSION WEEK3



Today's Session Leader: 17기 김희준

---

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



분반원 총 13명

16기 임정준

17기 서지민 안영지 황민아

18기 김나연 김송성 이수민 장원준 정해원 진서연 최유민

19기 이승준 최주희

오늘 세션이 종료된 후, 희망 분야 투표 공지 예정.

구글폼으로 희망 분야를 받아 4/3/3/3으로 총 네 팀 빌딩 예정.

문장생성/문장요약/검색/model evaluation 등의 분야 중 복수선택  
혹은 원하는 분야 단답형으로 응답

영지님

2주차  
복습과제1

IMDB 텍스트 감성분석

유민님

2주차  
복습과제2

FastText vs Word2Vec

주희님

2주차  
복습과제3

Word2Vec CBOW 구현

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

## 02 RNN

Sequence data를 처리하는 순환신경망

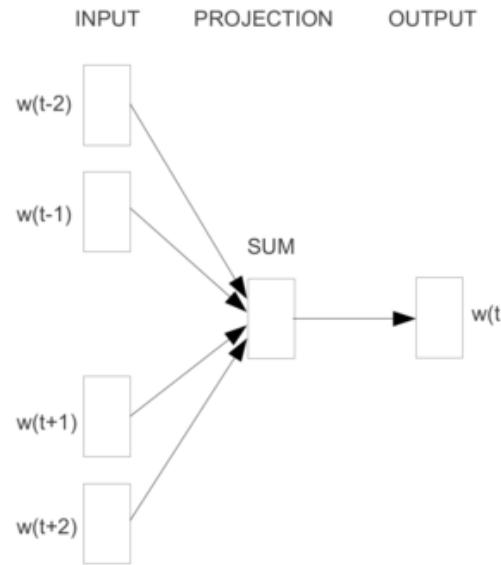
## 2-1. What is Sequential Data

### 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

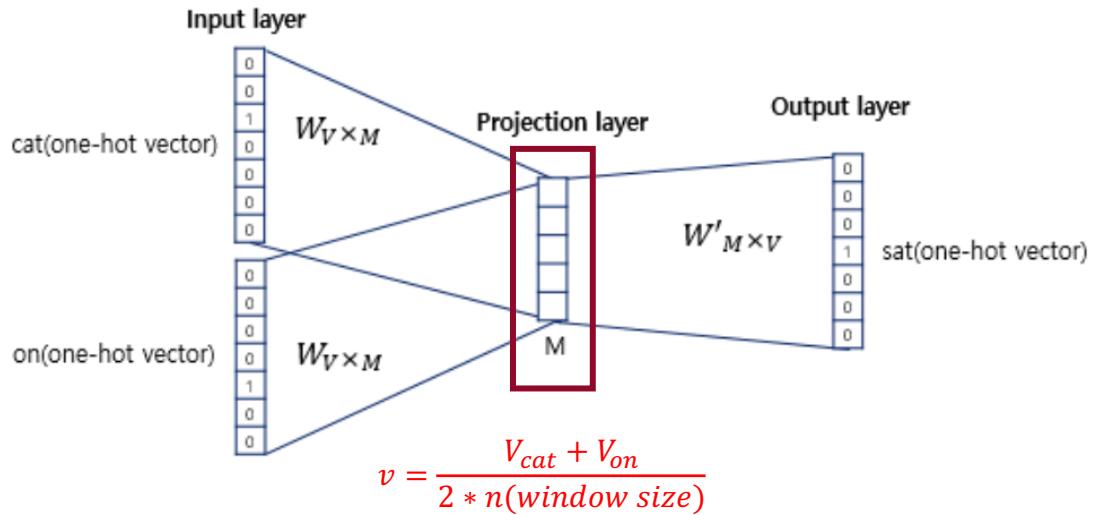
## 2-2. Limitation of CBOW(NNLM)



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

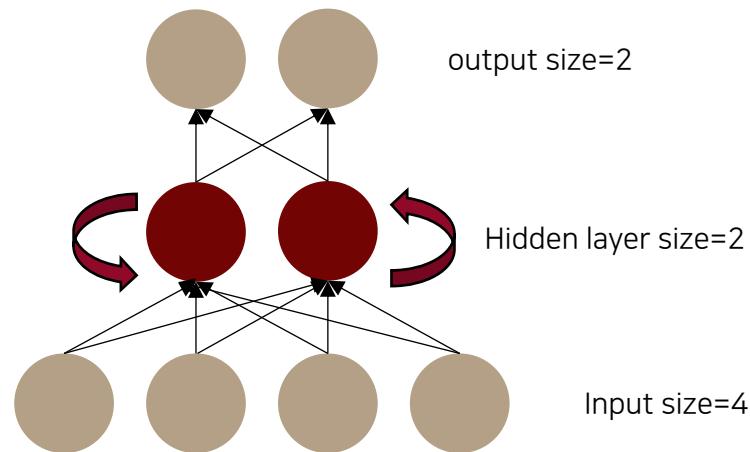
예문: 실패하면 반역 \_\_\_\_\_ 혁명 아닙니까?



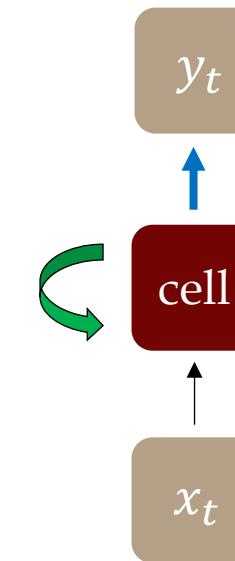
문맥을 반영하기는 하나,  
Projection layer에서 벡터들이 합해지기 때문에  
앞, 뒤 순서 맥락을 잃어버림

## 2-3. 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!

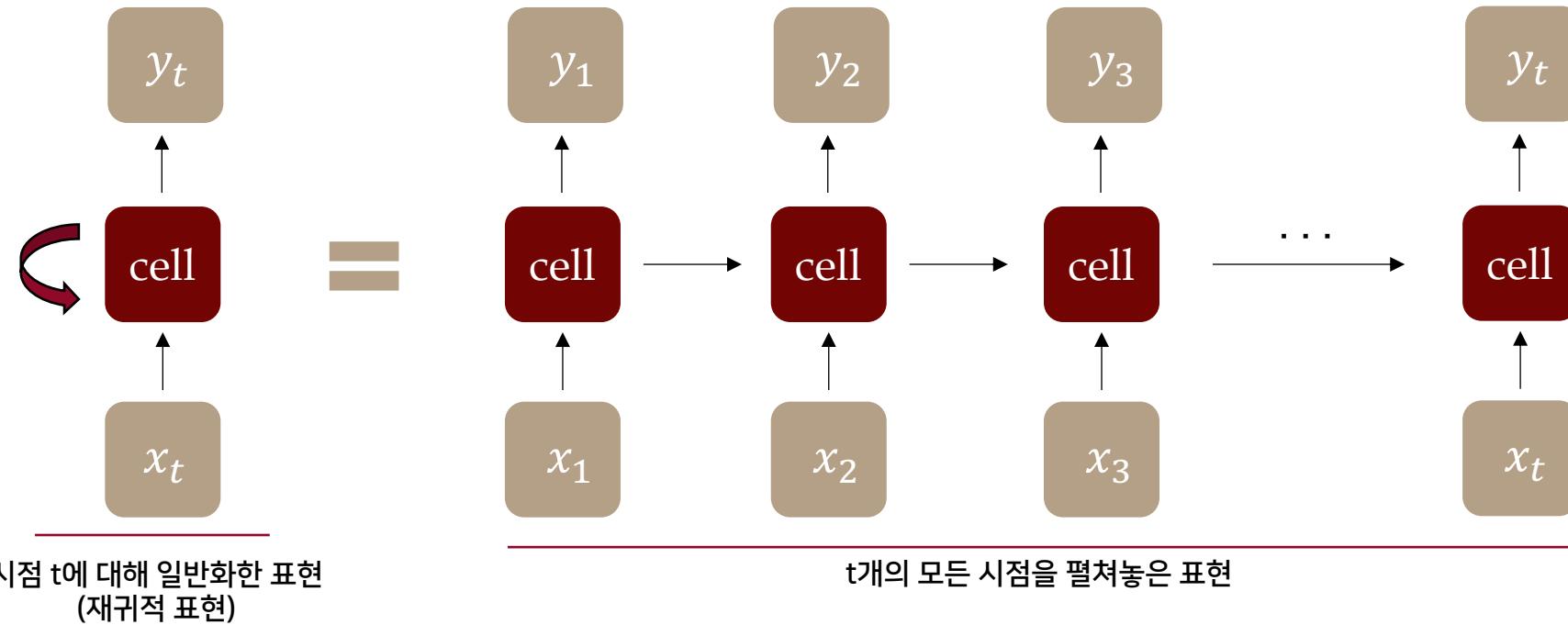


Hidden layer의 output=hidden state  
Hidden state는 output layer 방향으로도 전달되고,  
다시 hidden layer(RNN cell) 방향으로도 전달된다.  
즉, 이전의 값을 기억하려는 성질을 가지므로  
이 cell을 memory cell이라고도 부름.

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

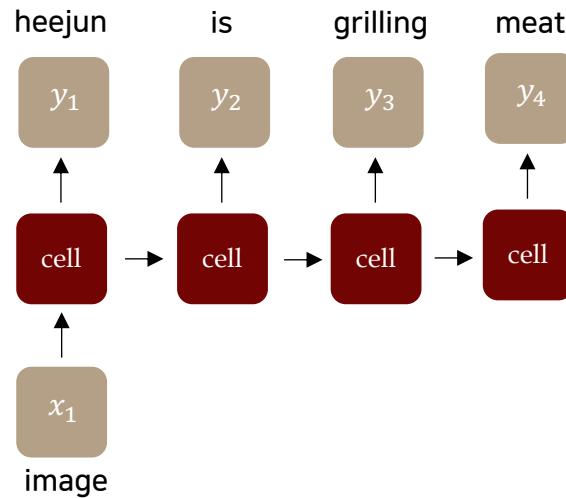
## 2-3. RNN(Recurrent Neural Net)

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

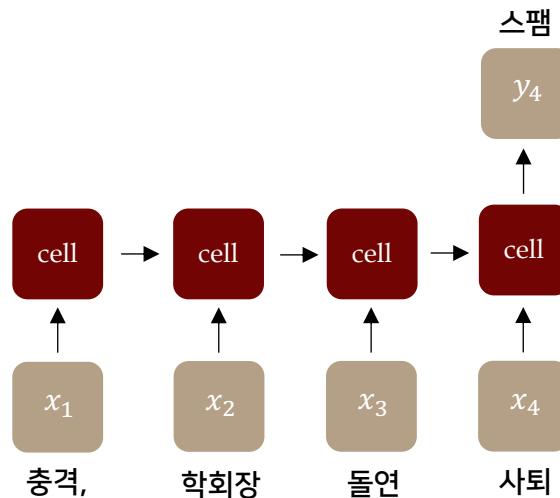


## 2-3. RNN(Recurrent Neural Net)

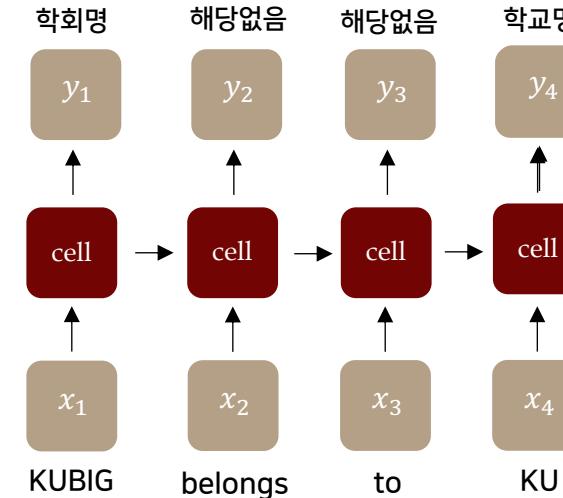
RNN can be applied to various task in NLP



one-to-many  
Ex) image captioning



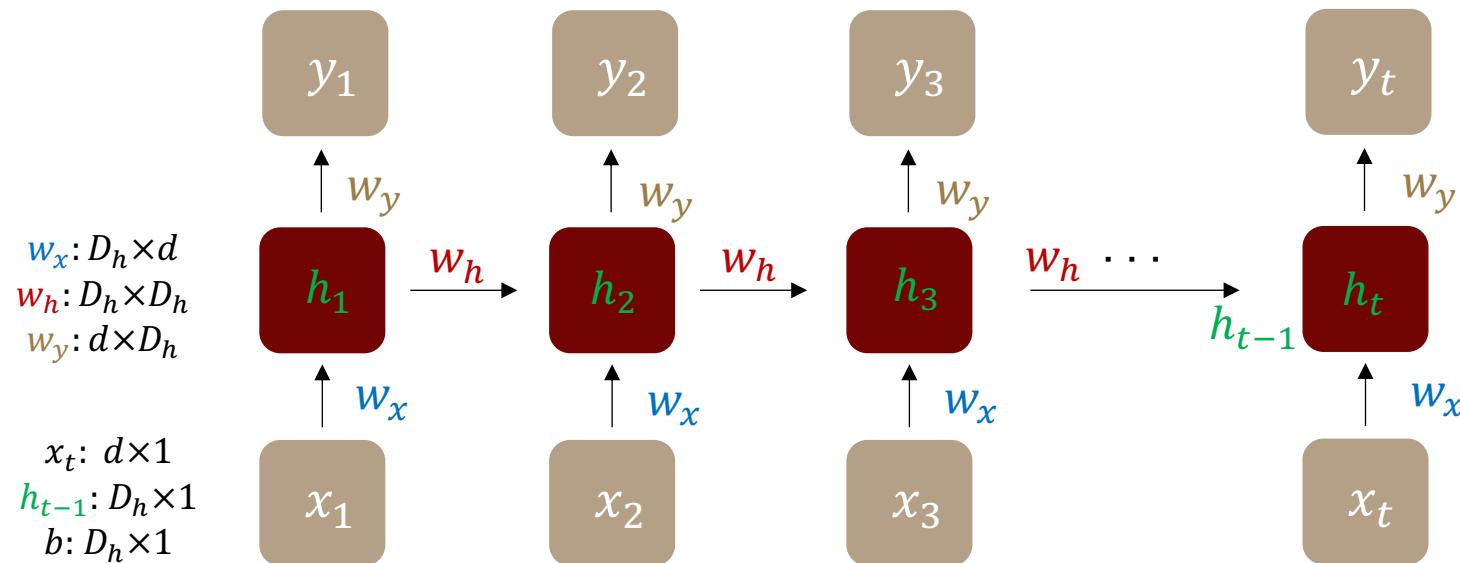
many-to-one  
Ex) spam classification  
Sentiment analysis



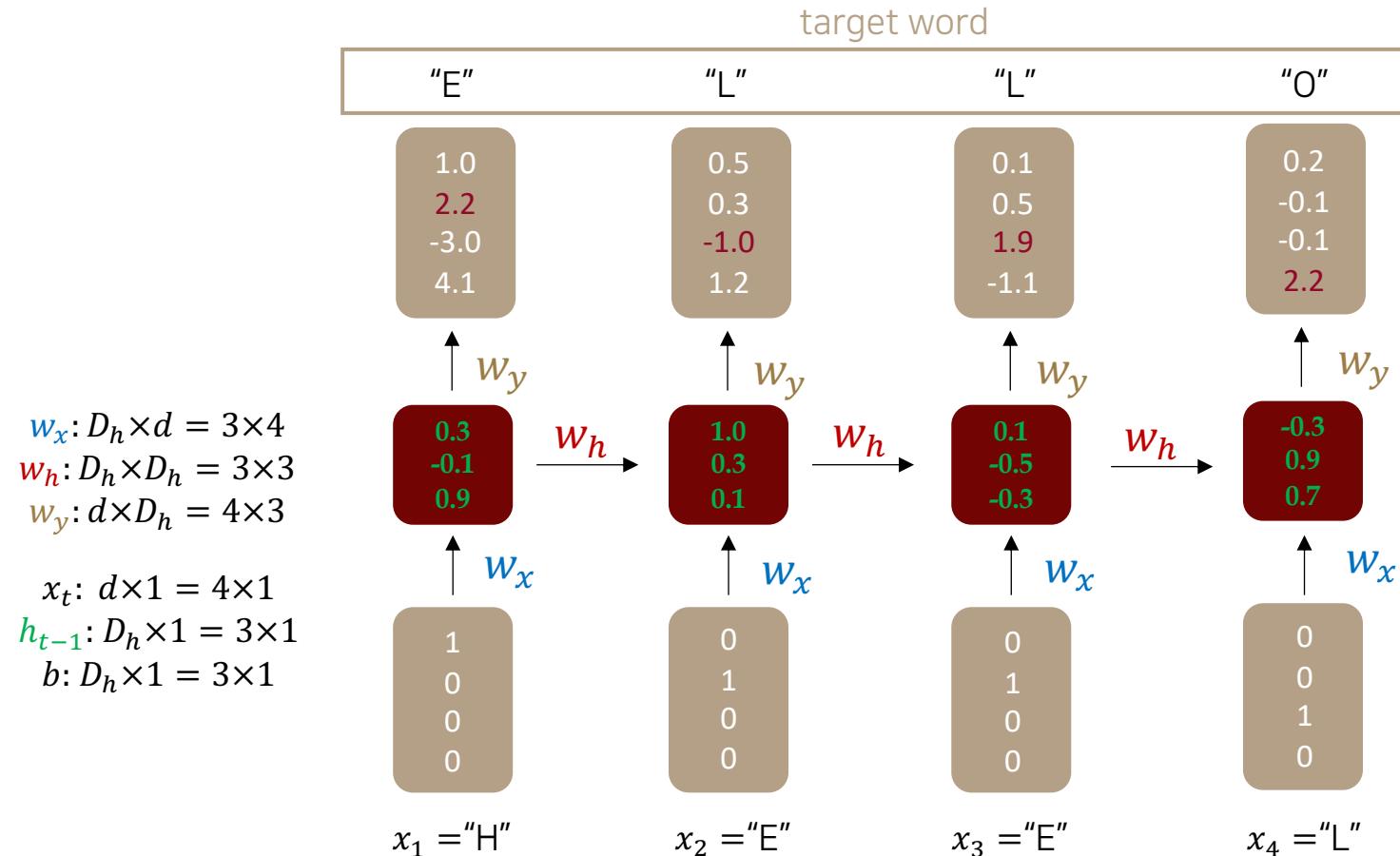
many-to-many  
Ex) Entity recognition(개체명 인식)  
POS tagging(품사 태깅)

## 2-3. RNN(Recurrent Neural Net)

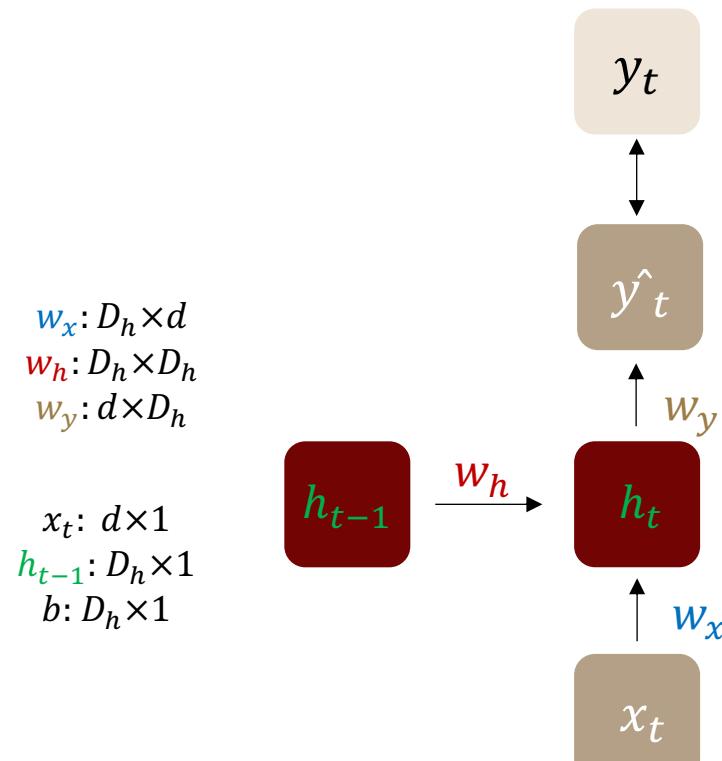
- 가중치 행렬  $w_{xh}, w_{hh}, w_{hy}$ 가 매 time step마다 동일함(shared weight)
- 업데이트 해야 할 가중치(파라미터)가 input 길이에 상관없이 고정됨
- time step이 거듭될 때마다 context information이 누적됨



## 2-3. RNN(Recurrent Neural Net)



## 2-4. forward, backward of RNN



*loss function = cross entropy( $y_t, \hat{y}_t$ )*

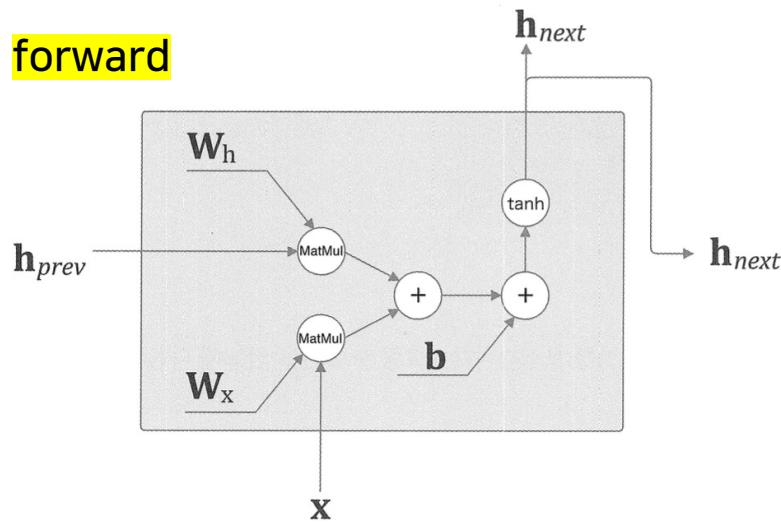
*output layer:  $\hat{y}_t = \text{softmax}(w_y h_t + b)$*

*hidden layer:  $h_t = \tanh(w_x x_t + w_h h_{t-1} + b)$*

$$\tanh\left(\begin{array}{|c|c|c|c|} \hline \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \hline \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array}\right) \times \begin{matrix} w_x \\ x_t \end{matrix} + \begin{array}{|c|c|c|} \hline \text{red} & \text{red} & \text{red} \\ \hline \text{red} & \text{red} & \text{red} \\ \hline \text{red} & \text{red} & \text{red} \\ \hline \end{array} \times \begin{matrix} w_h \\ h_{t-1} \end{matrix} + \begin{array}{|c|c|} \hline \text{green} & \text{green} \\ \hline \text{green} & \text{green} \\ \hline \end{array} + b$$

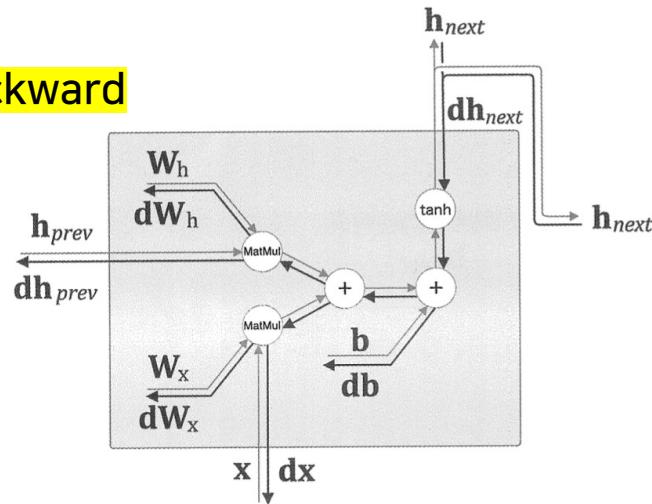
## 2-4. forward, backward of RNN

forward



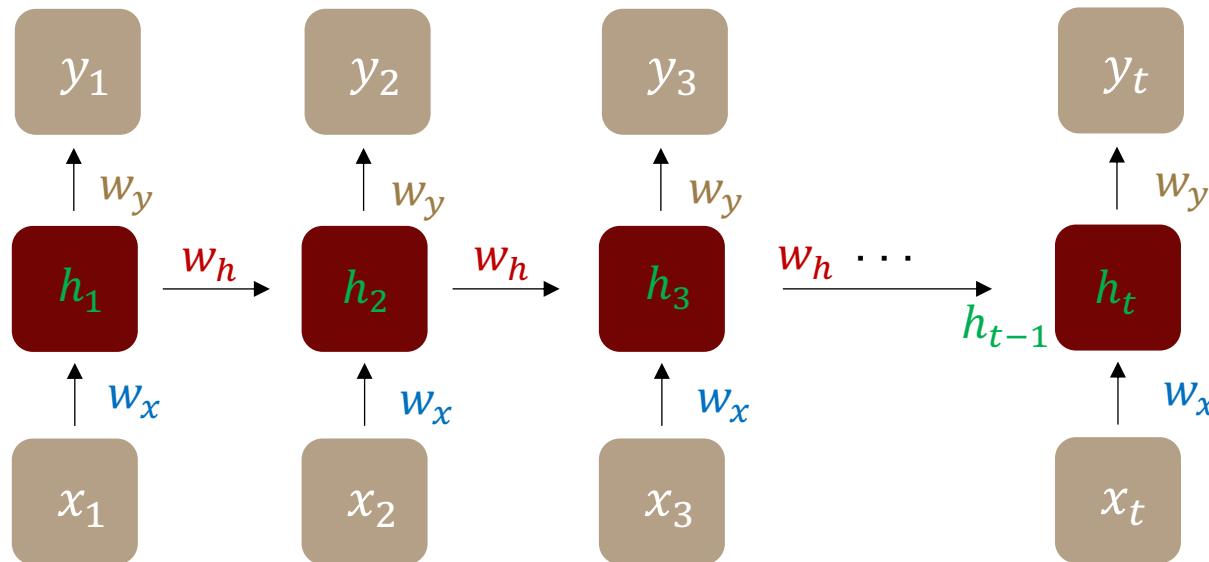
```
class RNN:  
    def __init__(self, Wx, Wh, b):  
        self.params = [Wx, Wh, b]  
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]  
        self.cache = None  
  
    def forward(self, x, h_prev):  
        Wx, Wh, b = self.params  
        t = np.matmul(h_prev,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

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
```

### BPTT(Backpropagation Through Time)

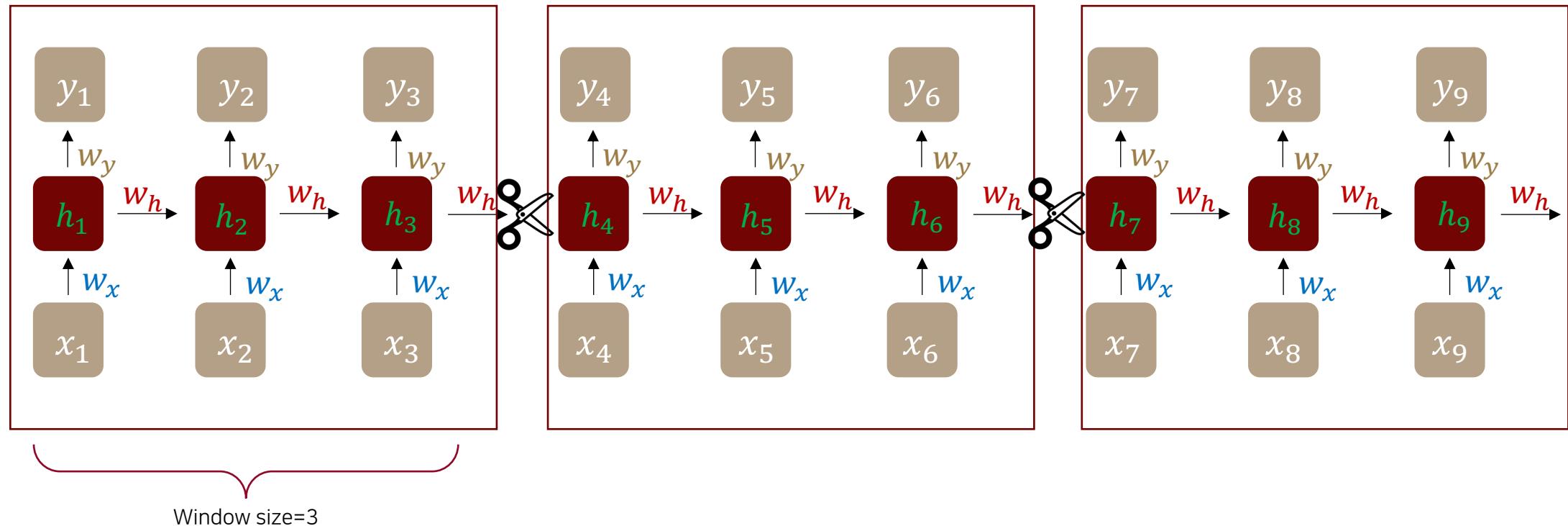


time step(=문장길이)가 길어짐에 따라

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

Sol) **Truncated BPTT**

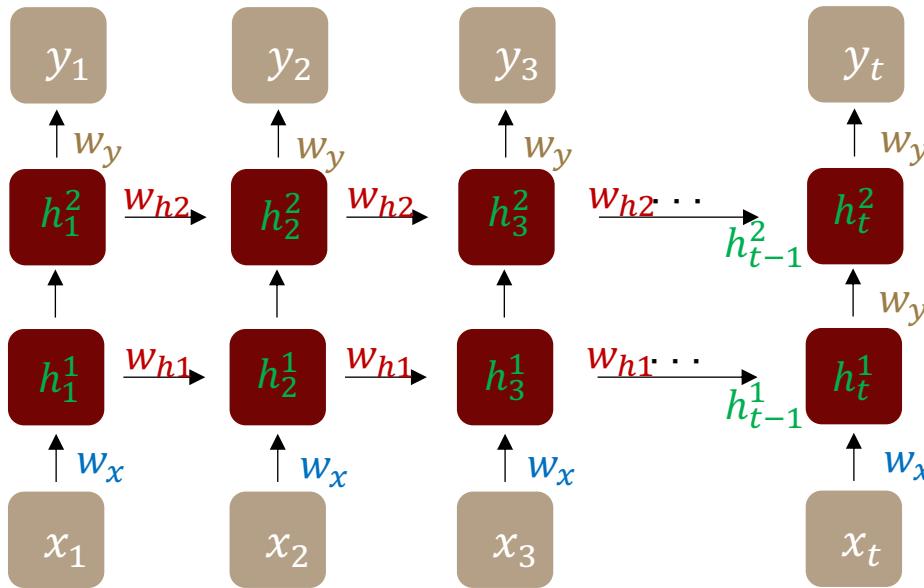
### Truncated BPTT



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

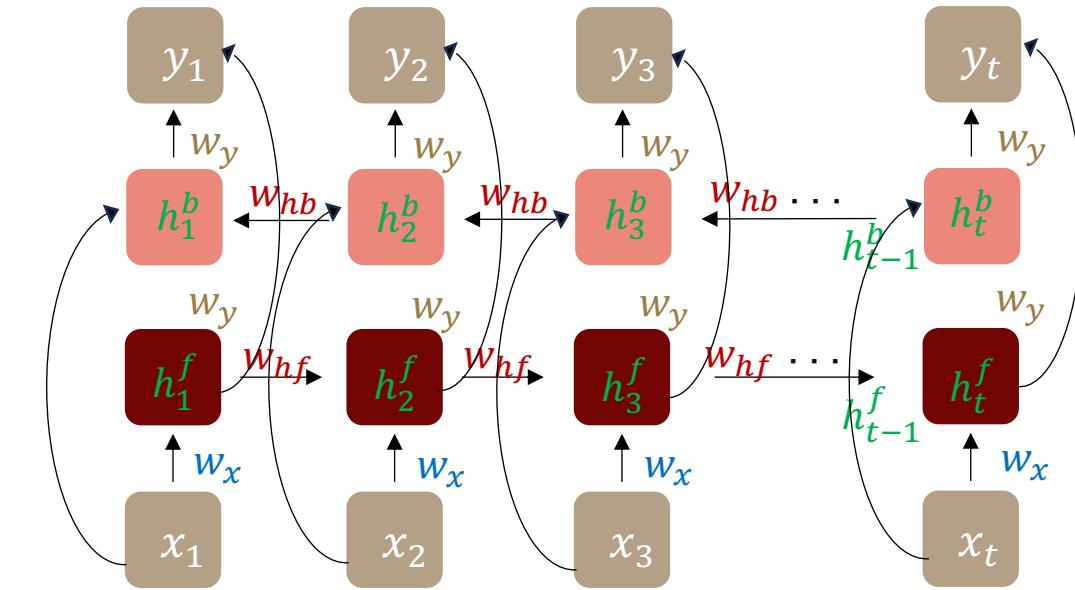
## 2-5. Deep, Bidirectional RNN

### Deep RNN



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

### Bidirectional RNN



- 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

## 2-5. 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_first=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_first=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_first=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

서연이는 수리통계학 스터디를 하러 스룸에 왔다.  
역시 부지런하다.  
오자마자 짐을 풀고 아메리카노 한잔을 했다.  
뒤이어 수민이도 스터디를 위해 두번째로 스룸에 도착했다.  
수민이는 \_\_\_\_\_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

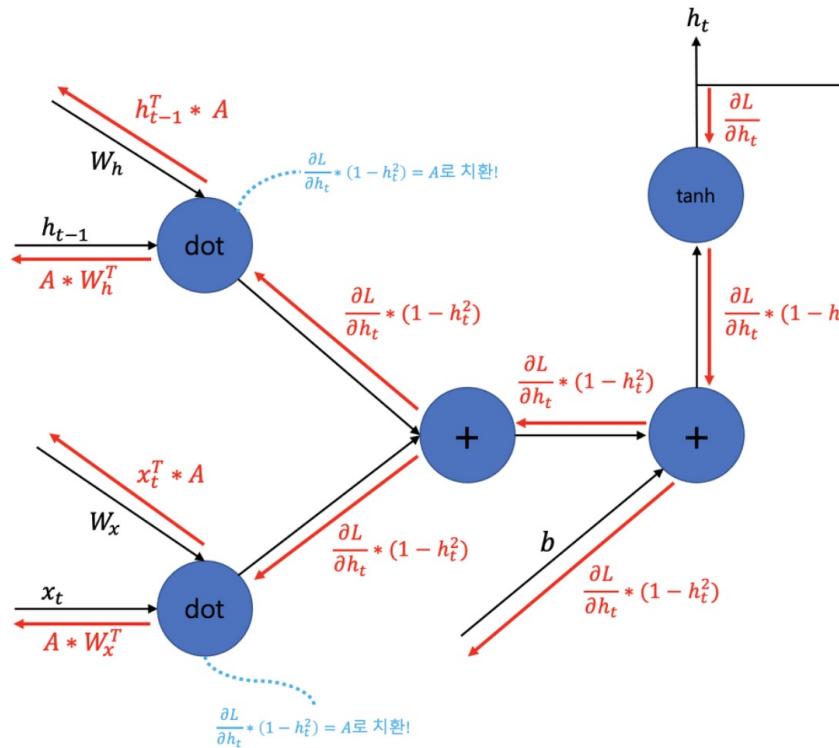
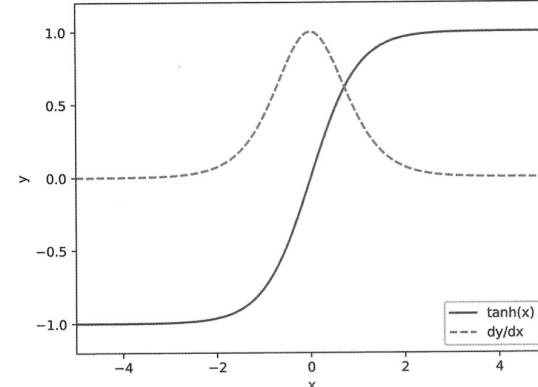


그림: 밑바닥부터 시작하는 딥러닝2

tanh 연산에서 backprop하면 기울기 소실 위험

그림 6-6  $y = \tanh(x)$ 의 그래프(점선은 미분)



matmul 연산에서 backprop하면 기울기 폭등 위험

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

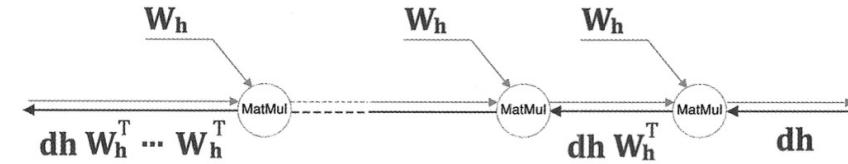
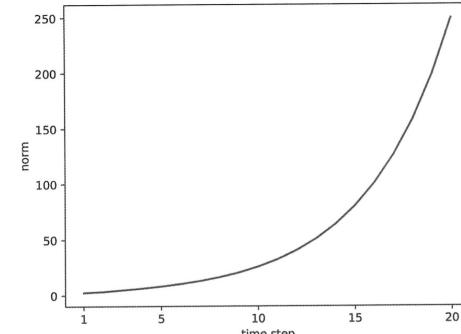


그림 6-8 기울기  $dh$ 는 시간 크기에 비례하여 지수적으로 증가한다.



# 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

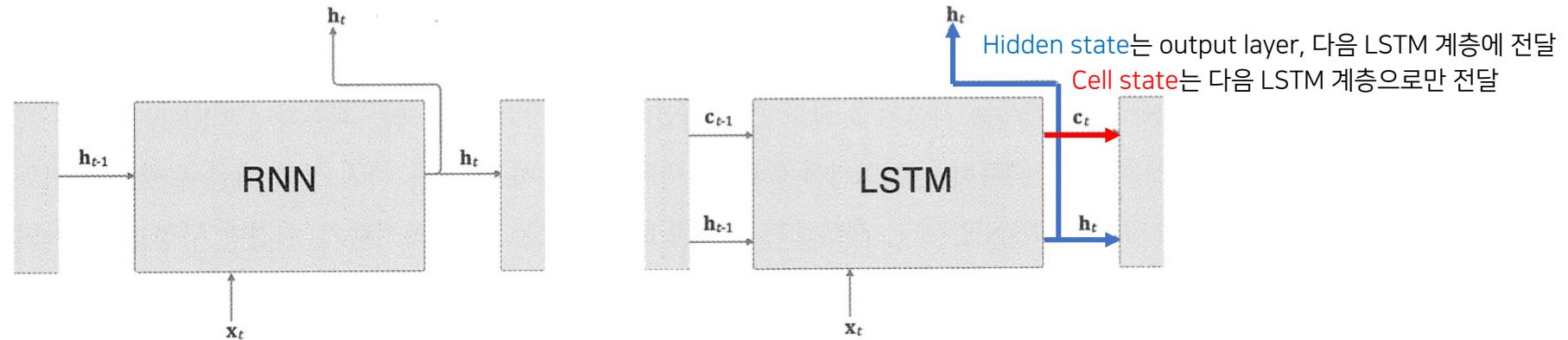
## LSTM

---

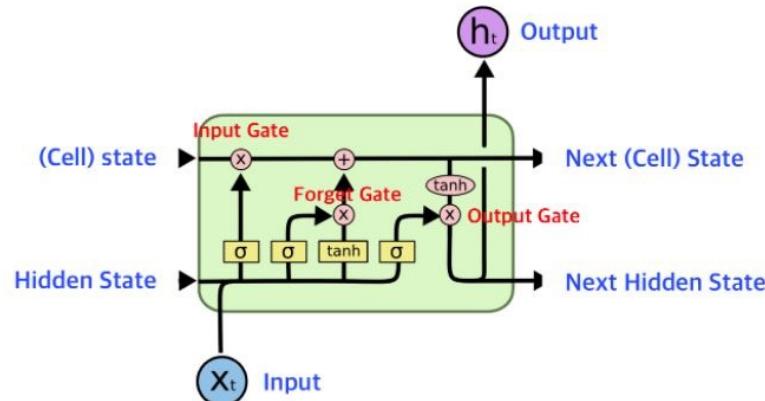
기존 RNN architecture에 Gate를 추가함으로써  
기억력을 개선하는 기법  
Long Short Term Memory

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.

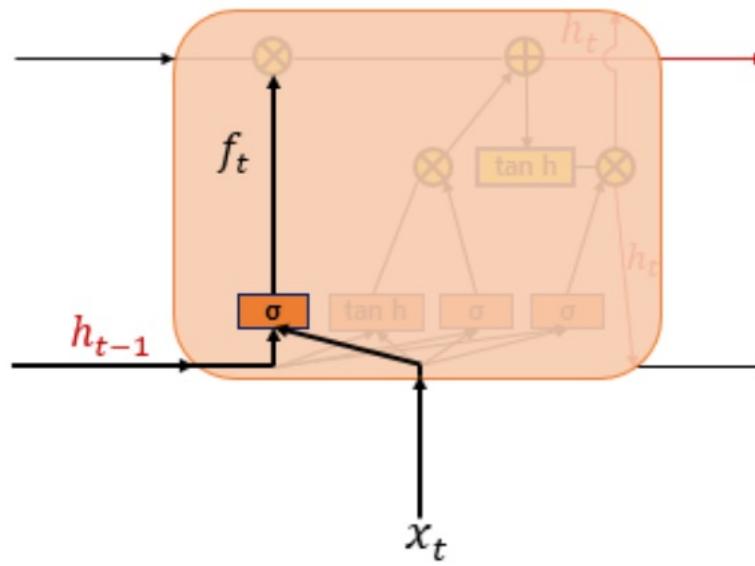


LSTM은 아래 세 가지 gate로 얼마나 기억하고, 잊을지를 정한다!

- Input gate
- Forget gate
- Output gate

그림: 밑바닥부터 시작하는 딥러닝2

#### forget gate



이전 기억을 얼마나 삭제할 것인지 정하는 gate

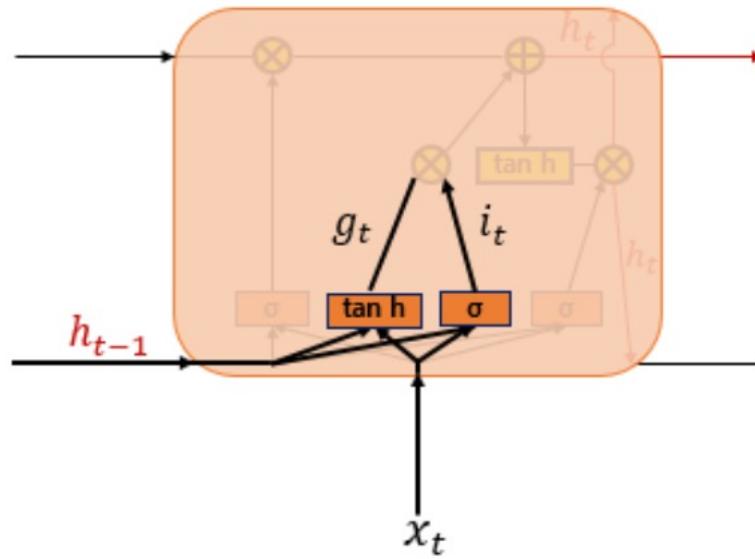
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \Rightarrow 0\text{에서 }1\text{의 값 출력}$$

- 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

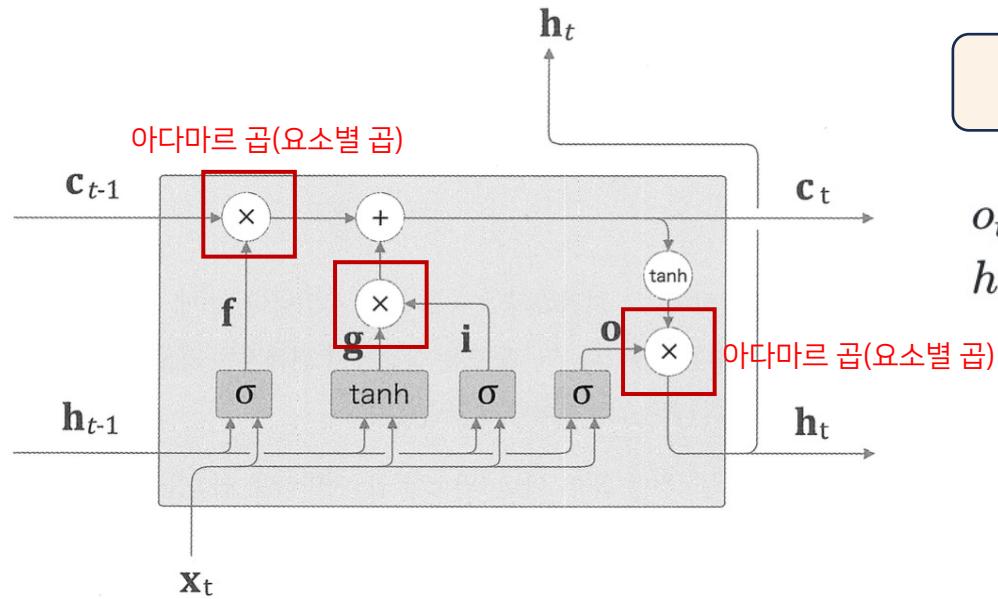
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \Rightarrow 0 \text{에서 } 1 \text{의 값 출력}$$

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

그림: 딥러닝을 이용한 자연어처리

### 3-3. Three Gates of LSTM

#### output gate



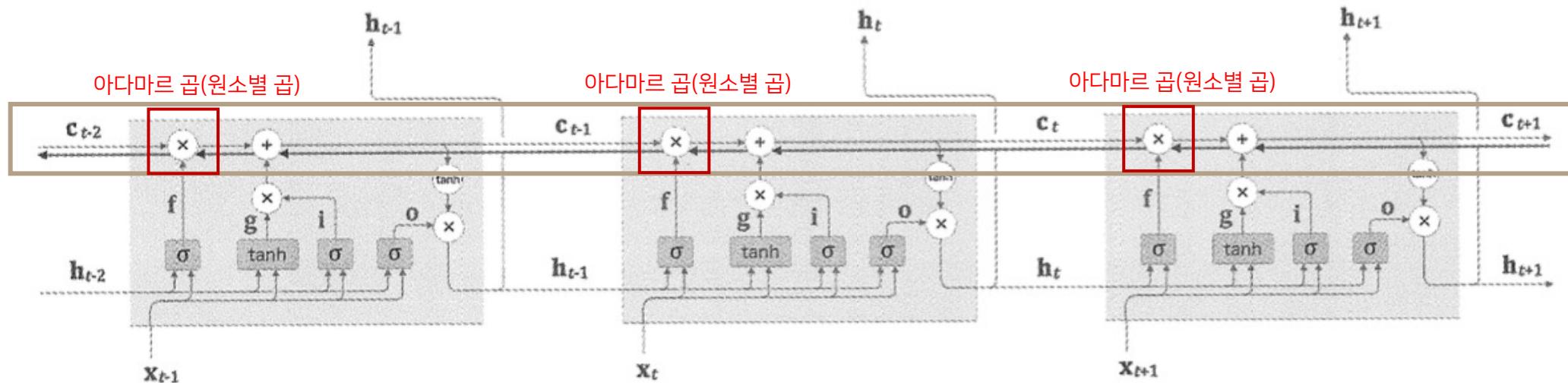
Input, forget gate의 값들을 다음 단계로 전달해주는 gate

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

그림: 밑바닥부터 시작하는 딥러닝2

### 3-3. Three Gates of LSTM

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



- + 노드는 기울기 변화 안 줌
- X 노드는 행렬곱이 아닌 아다마르 곱
- 매 시점마다 다른 게이트 값

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

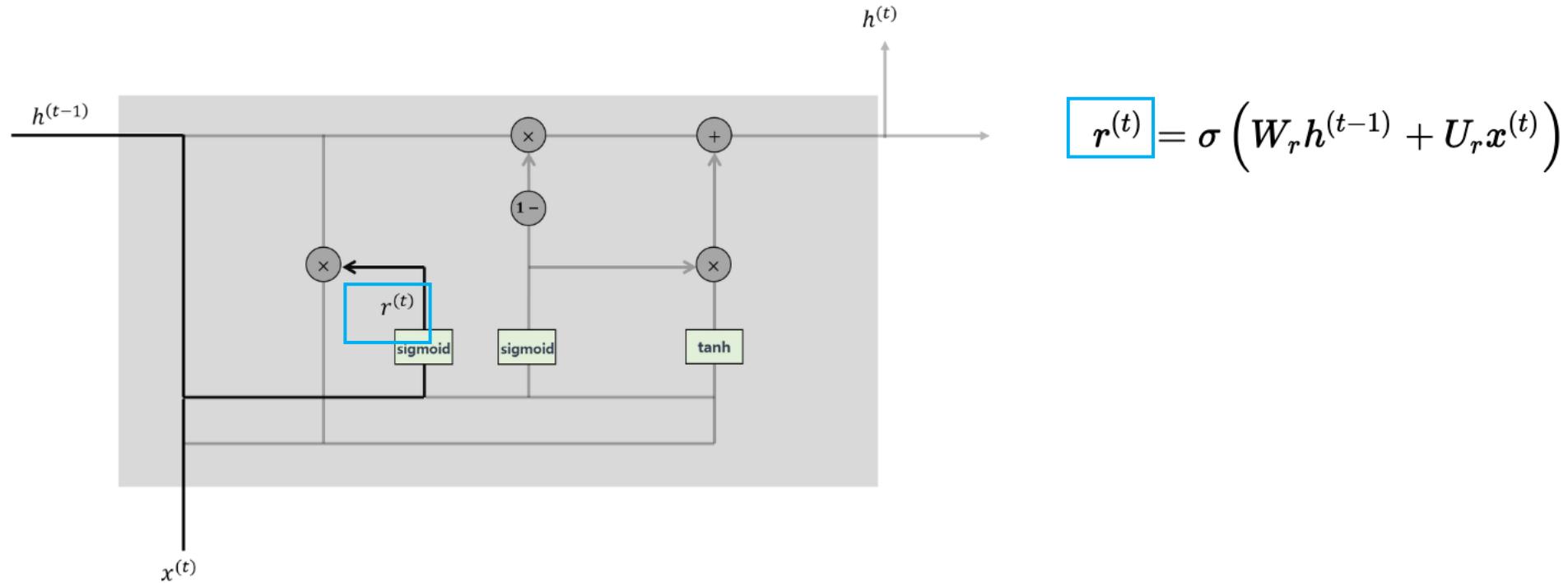


그림: <https://yjjo.tistory.com/18>

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

### update gate

과거 정보와 현재 정보의 업데이트 비율을 결정하는 gate

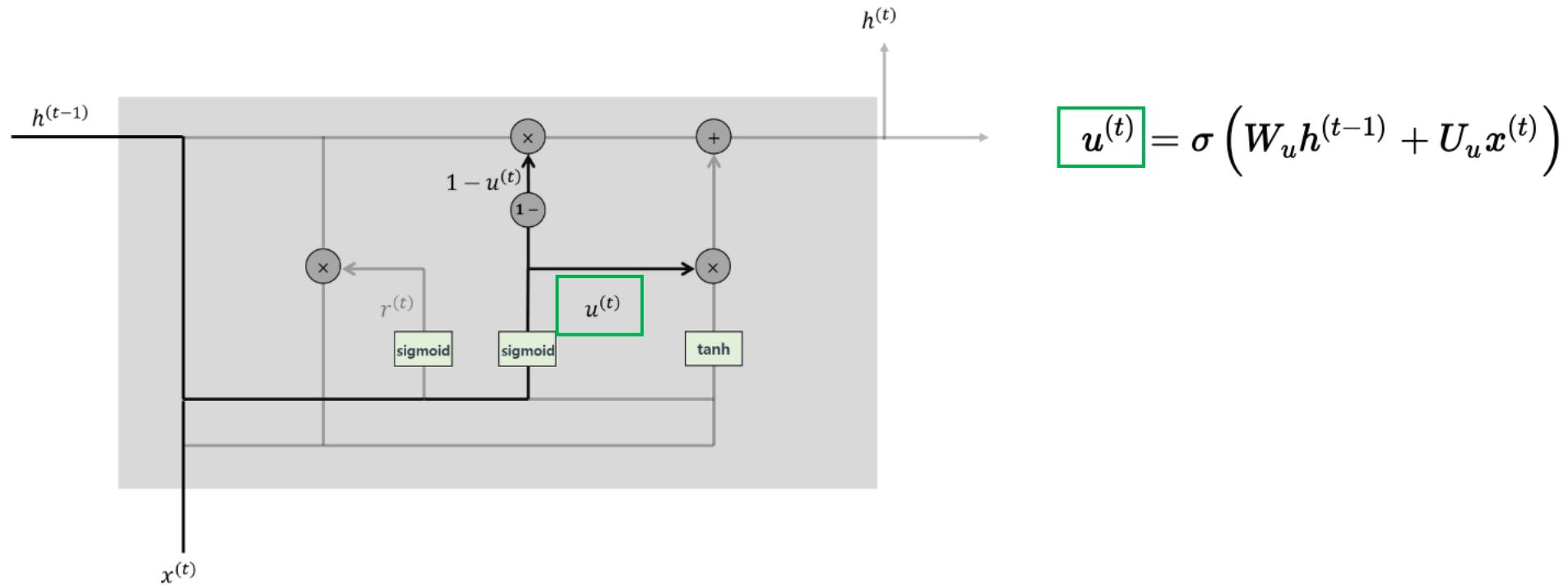


그림: <https://yjjo.tistory.com/18>

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

### $h_t$ calculation

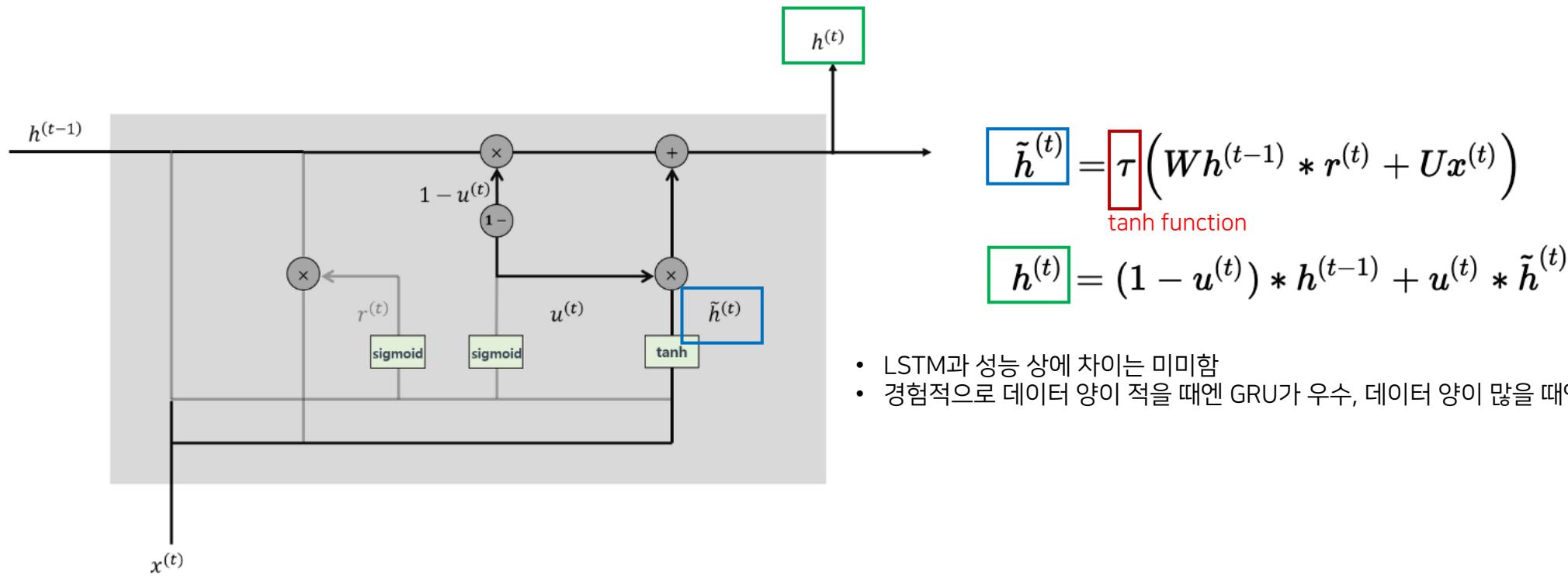


그림: <https://yjjo.tistory.com/18>

# 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

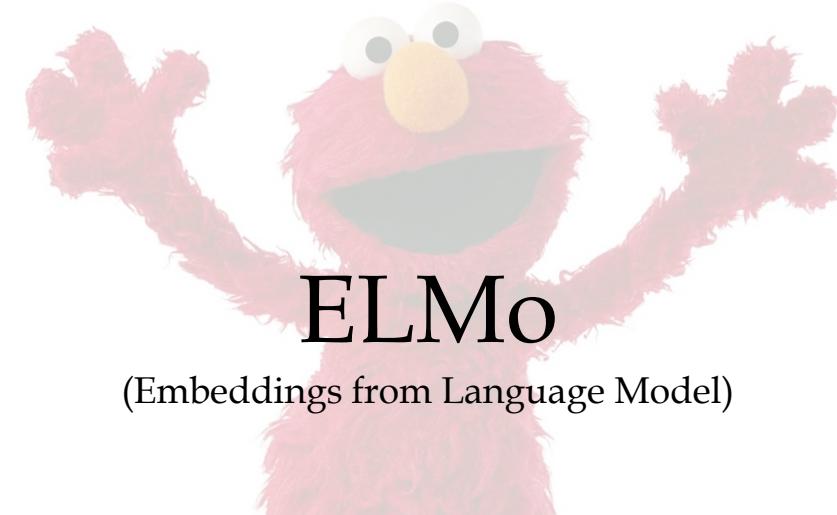
word2vec

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

"연패" embedding vector



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



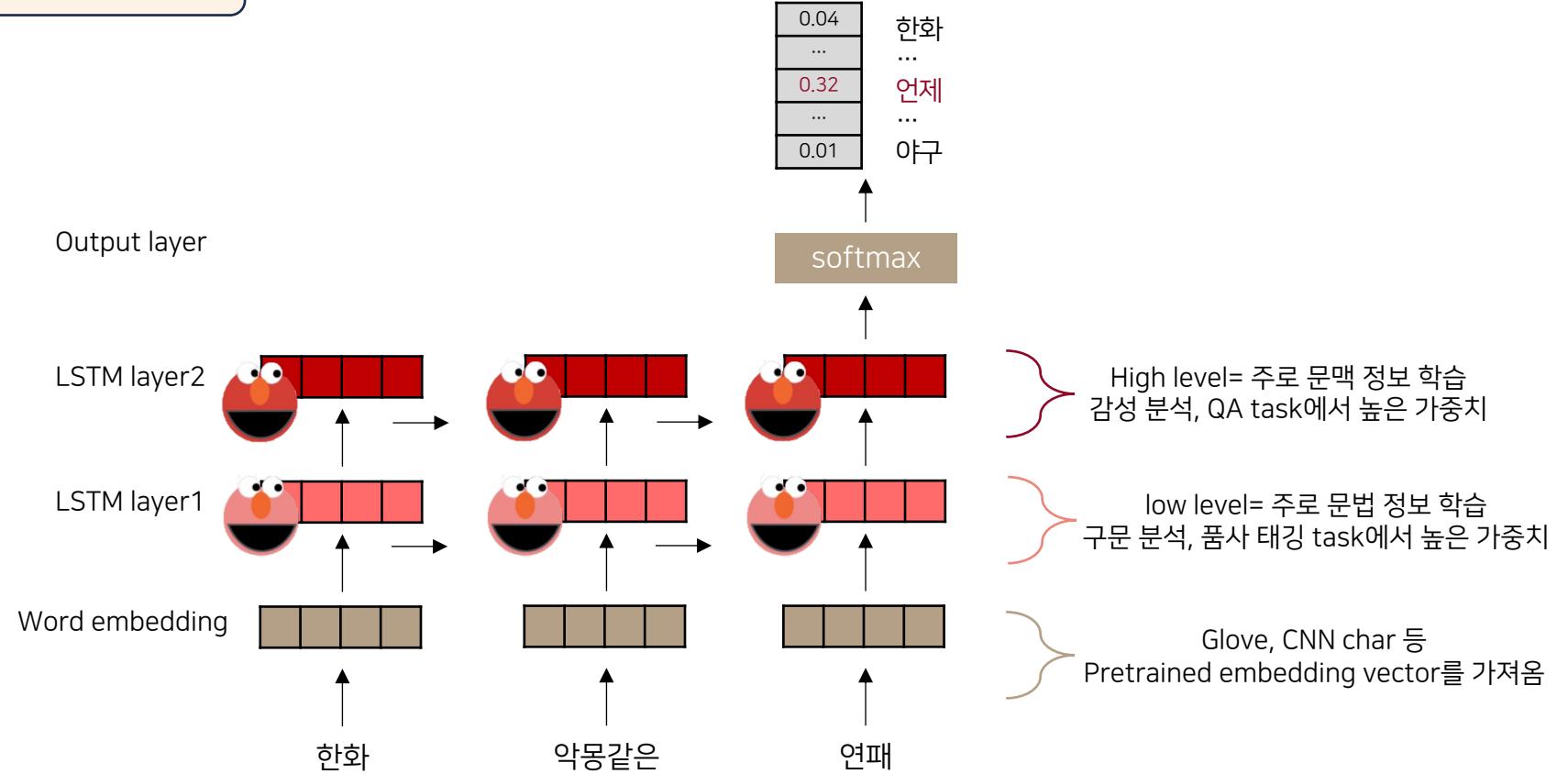
ELMo  
(Embeddings from Language Model)



BERT  
See you soon

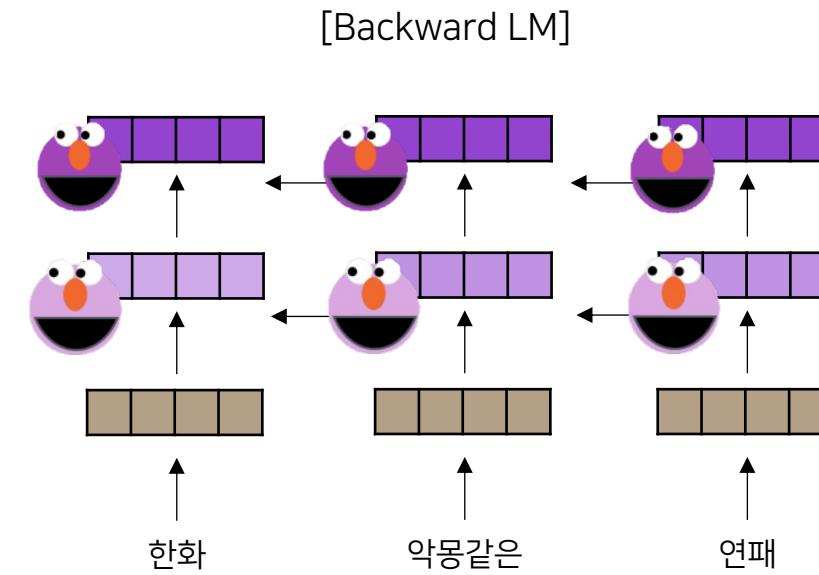
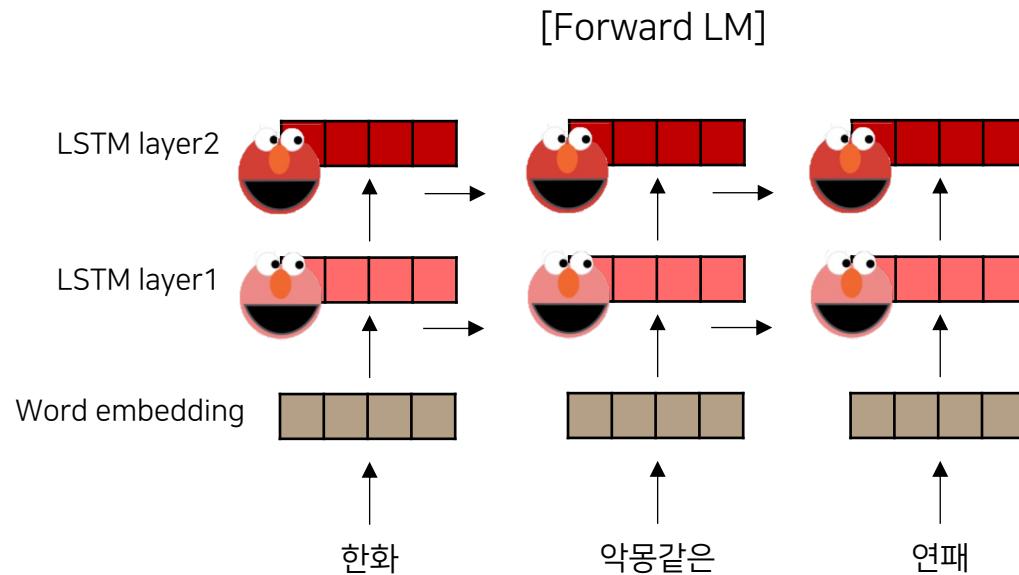
## 5-2. ELMo with biLSTM

예문: 한화, 악몽같은 연파 언제 탈출하나

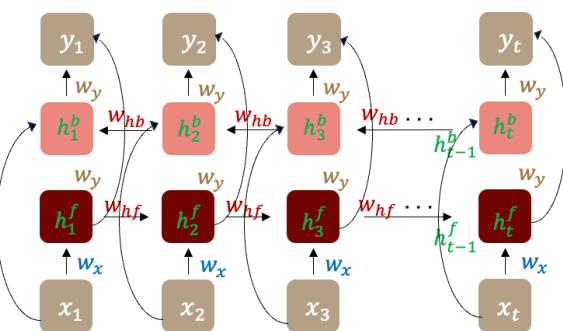


## 5-2. ELMo with biLSTM

예문: 한화, 악몽같은 연패 언제 탈출하나



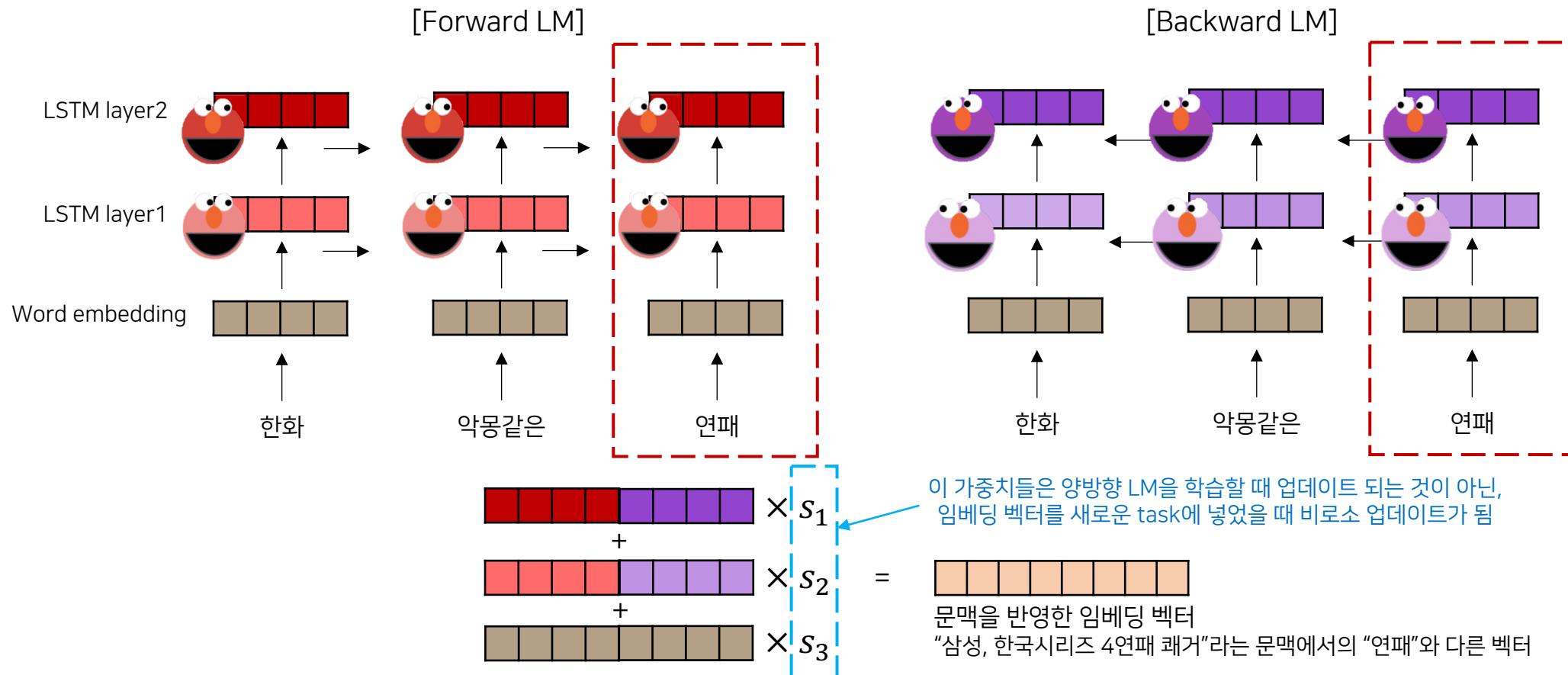
Bidirectional RNN이 hidden state만 앞뒤로 공유하는 반면,  
위 경우 두 개를 각각 다른 Language Model로 보고 개별적으로 학습



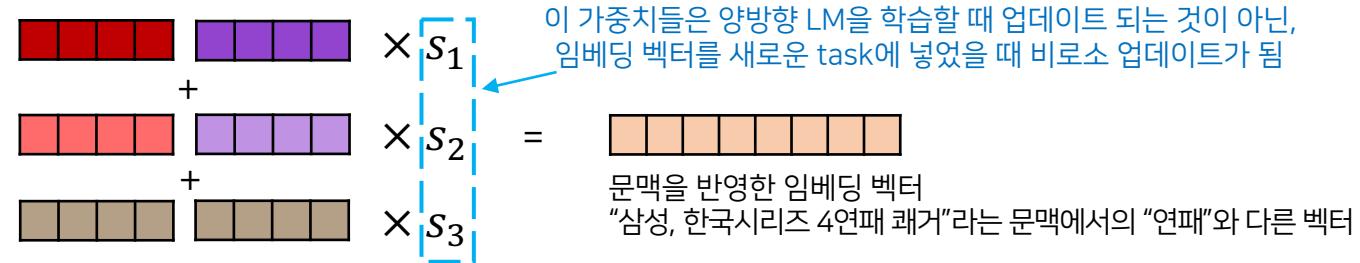
## 5-2. ELMo with biLSTM

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

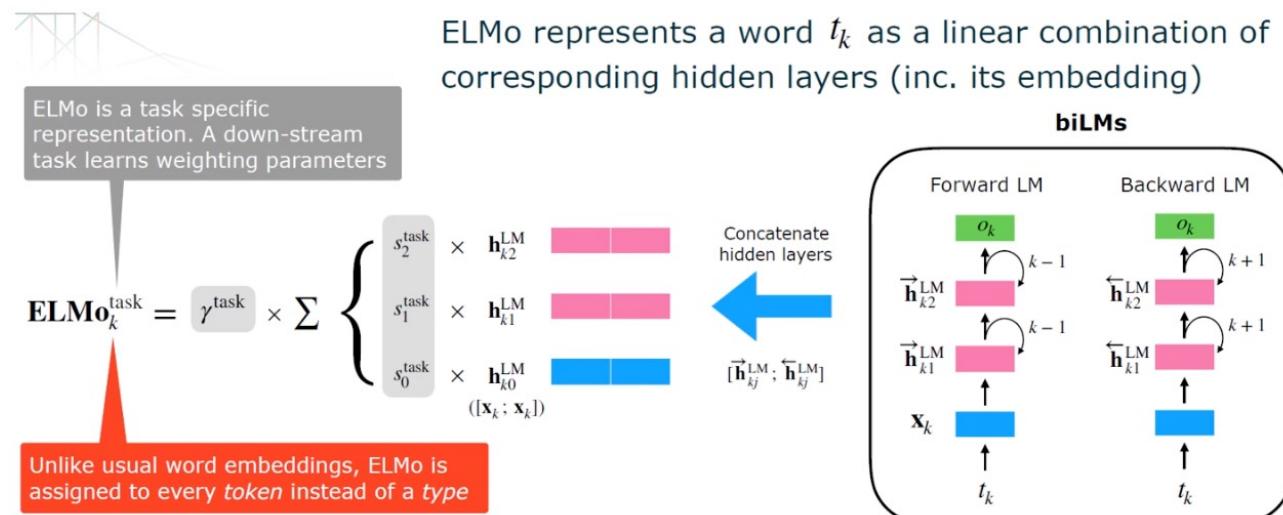


그림: 고려대학교 DSBA <https://www.youtube.com/watch?v=zV8klUwH32M&t=1053s>

# 06 Announcement

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

## 6-1. 우수 예습과제 Review

수민님

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!