KUBIG 25-W
겨울방학 BASIC STUDY SESSION

NLP SESSION
WEEK3

KUBIG
DATA SCIENCE & AI

CONTENTS

KUBIG
DATA SCIENCE & AI

01 Announcement, 우수 복습과제 Review

KUBIG
DATA SCIENCE & AI

1-1. Announcement

✔ 설 연휴(WEEK 4) 세션 진행 방식

- 녹화 강의로 진행 (출석 x)
- 과제, 팀 회의 정상 진행

✔ WEEK 5 대면 세션 진행

- 대면 참여가 불가능한 분들에겐 줌 링크 제공 예정 (캠 켜야 출석 인정)
- 세션 후 각 팀별로 중간발표 진행 예정
- CV 세션과 함께 뒤풀이 예정

1-2. 우수 복습과제 코드 Review

KUBIG DATA SCIENCE & AI

남동연 님	이우진 님	이연호 님
2주차 복습과제1	2주차 복습과제2	2주차 복습과제3
IMDB 텍스트 감성분석	FastText vs Word2Vec	Word2Vec CBOW 구현

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

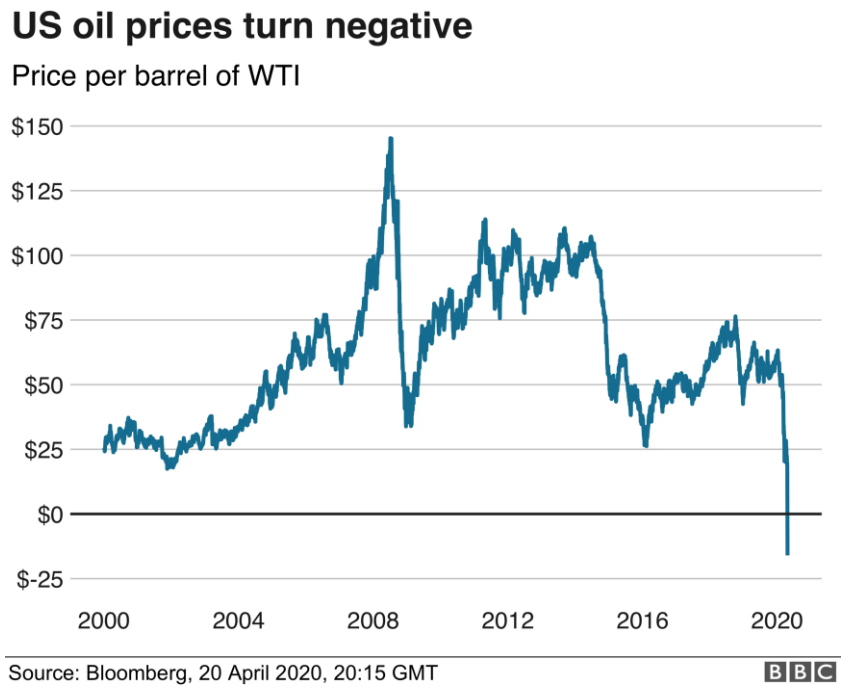
02 Sequential Data, Language Model

KUBIG
DATA SCIENCE & AI

KUBIG DATA SCIENCE & AI

Sequential Data

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



Time Series

강화학습

Sequential Data : 순서적으로 인접한 정보들이 Highly Correlated

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

❓ Sentence가 Sequential Data인 이유!

2-1. What is Sequential Data

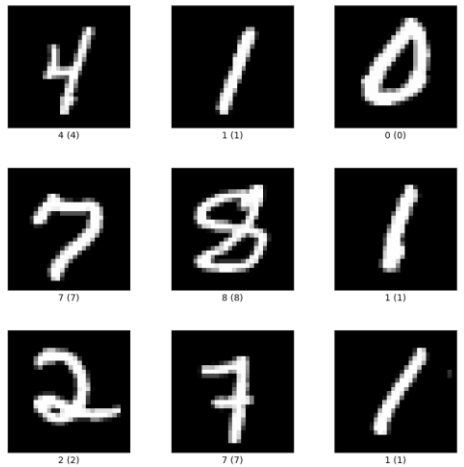


Image Data

- 공간적 특성에 focus

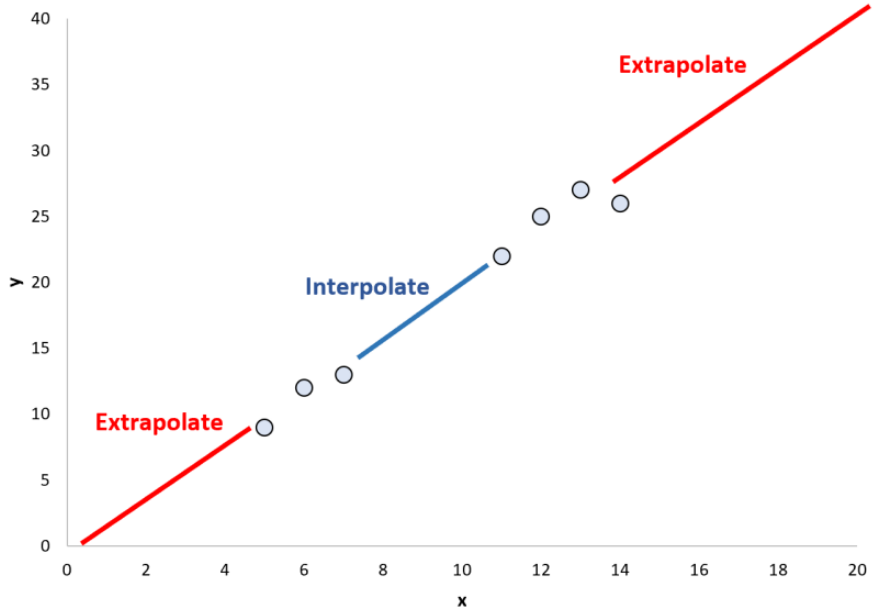
Graphical Data

- Node와 edge의 연결이 중요

KUBIG DATA SCIENCE & AI

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



2-2. Invariance In Sequential Data

KUBIG DATA SCIENCE & AI

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

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

$$P(man\ bites\ dog) \neq P(dog\ bites\ man)$$

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

- **Temporal Invariance**: Translation invariance in sequence order

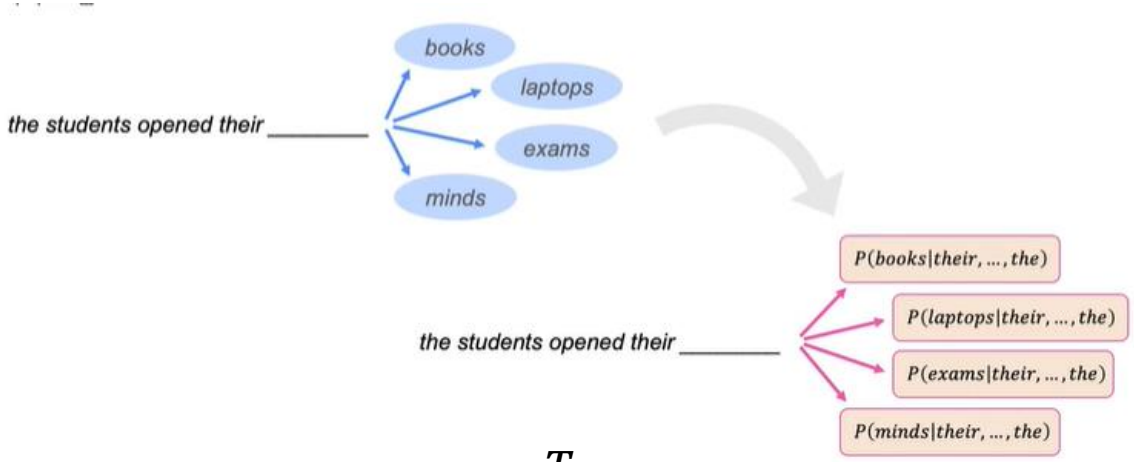
$$P\left(X_{1+j}, \dots, X_{n+j}\right) = \prod_{k=1} P\left(X_k | X_1, \dots, X_{k-1}\right) = \prod_{k=1} P\left(X_{k+j} \middle| X_{1+j} = \mathbf{x_1}, \dots, X_{k-1+j} = \mathbf{x_{k-1}}\right)$$

$$= \mathrm{f}_\theta(\mathbf{x_1}, \dots, \mathbf{x}_{\mathrm{k}-1})$$

KUBIG DATA SCIENCE & AI

Language Model

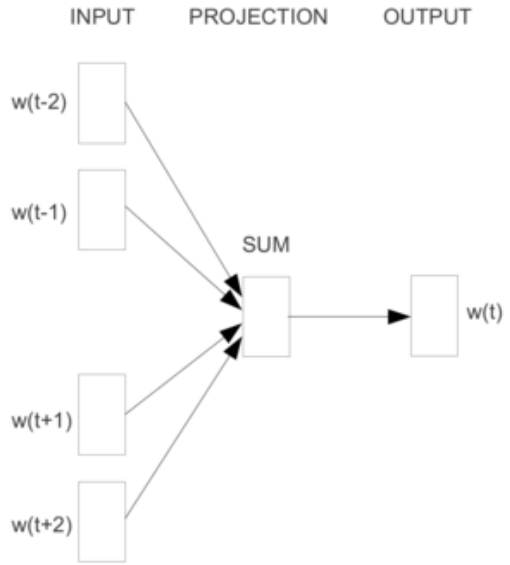
- 단어의 시퀀스(문장)에 대해서 얼마나 자연스러운 문장인지 확률을 이용해 예측하는 모델
- 주어진 단어의 시퀀스에 대해 다음에 나타날 단어가 어떤 것인지 예측하는 작업을 Language Modeling



$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_1, \ldots, w_{t-1})$$

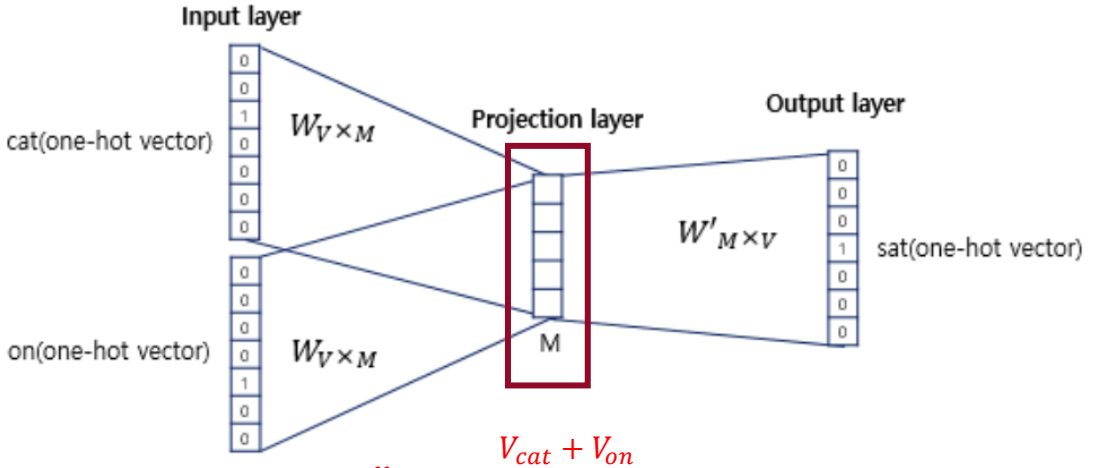
그림: 서울대학교 DSBA https://www.youtube.com/watch?v=OyCZvSDxMAk

2-4. 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 W가
과하게 많아져 비효율적



$$v = \frac{V_{cat} + V_{on}}{2 * n(window\ size)}$$

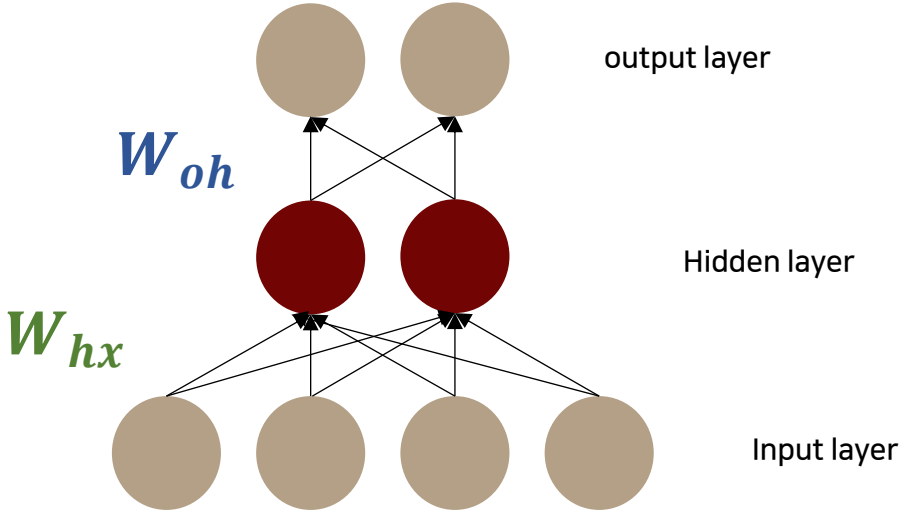
문맥을 반영하기는 하나,
Projection layer에서 벡터들이 합해지기 때문에
앞, 뒤 순서 맥락을 잃어버림

03 RNN

Sequence data를 처리하는 순환신경망

KUBIG
DATA SCIENCE & AI

3-0. DNN(Deep Neural Network)



RNN과의 비교를 위해
아래에서 위로 올라가는 구조로 도식화

$$\boldsymbol{o_t} = \boldsymbol{g}(\boldsymbol{W_{oh}}\mathbf{h_t})$$

$$\mathbf{h_t} = f(\boldsymbol{W_{hx}}\boldsymbol{x_t})$$

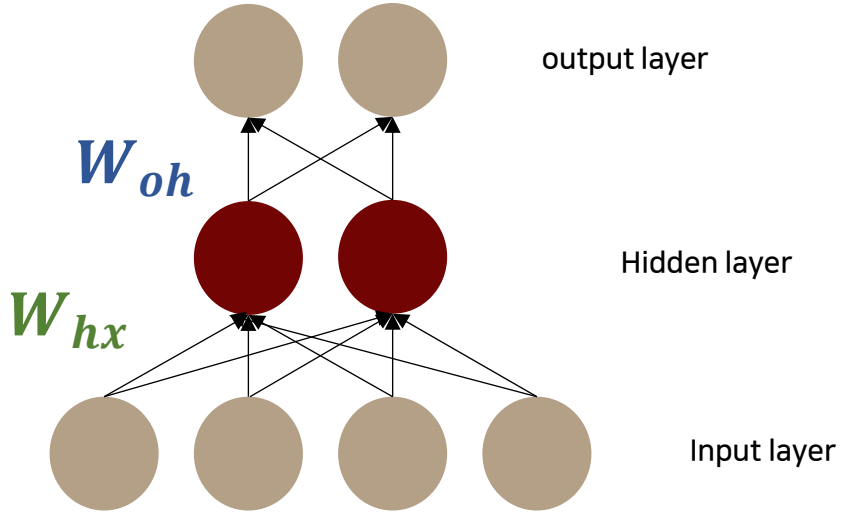
$$f(\bullet) = \boldsymbol{tanh}, \boldsymbol{g}(\bullet) = \boldsymbol{softmax}$$

RNN과의 비교를 위해
벡터 단위로 간단하게 도식화
시점 t에 대해 일반화한 표현

참고: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

$$\mathbf{h}_{t-1} = f(W_{hx}\boldsymbol{x}_{t-1})$$
$$\boldsymbol{o}_{t-1} = g(W_{oh}\mathbf{h}_{t-1})$$

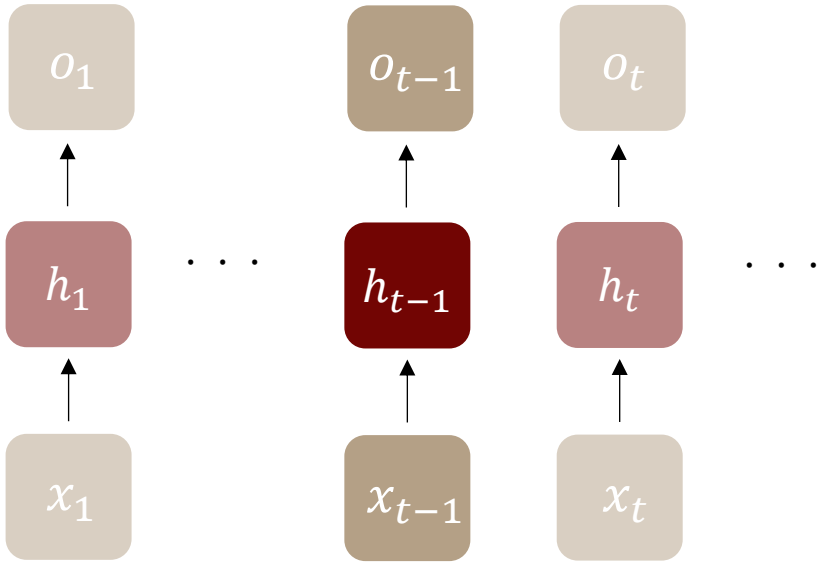
$$f(\bullet) = \boldsymbol{tanh}, g(\bullet) = \boldsymbol{softmax}$$



output layer

Hidden layer

Input layer

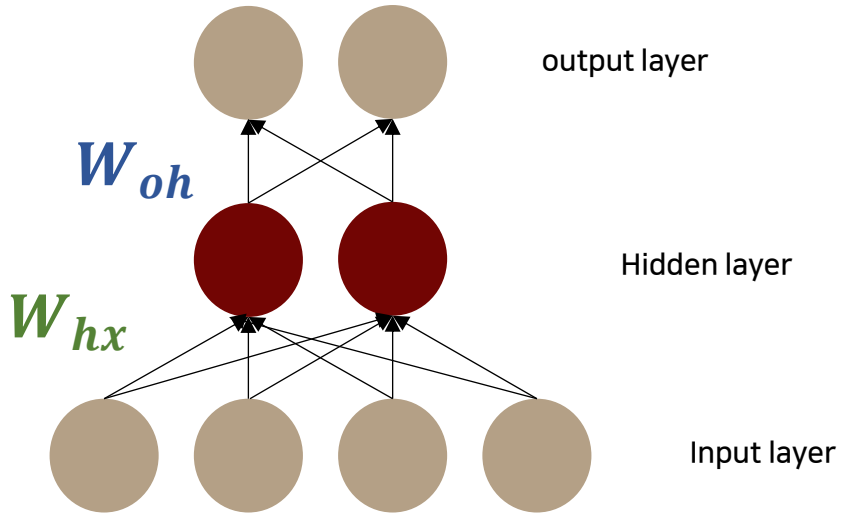


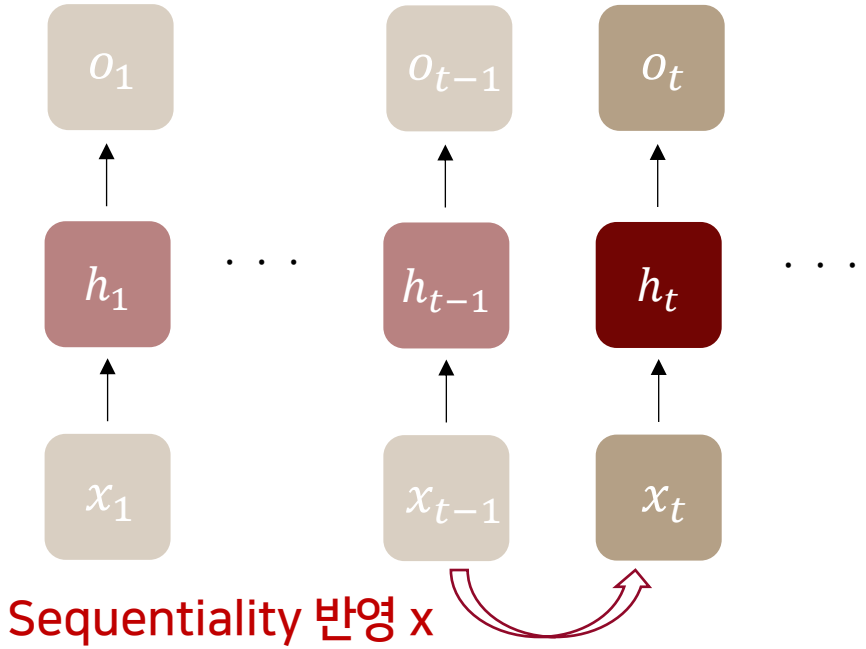
참고: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

$$\mathbf{h}_t = f(W_{hx}\boldsymbol{x}_t)$$
$$\boldsymbol{o}_t = g(W_{oh}\mathbf{h}_t)$$

이전 시점(t-1)에 대한 정보 x

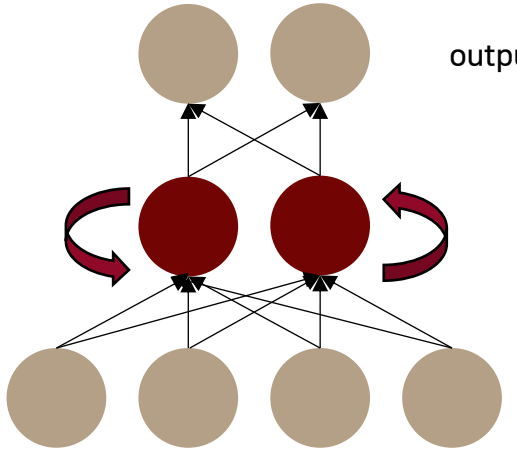
$$f(\bullet) = \boldsymbol{tanh}, g(\bullet) = \boldsymbol{softmax}$$



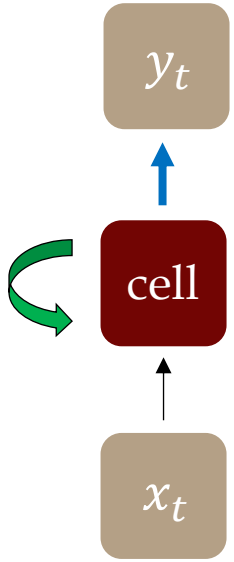


참고: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

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



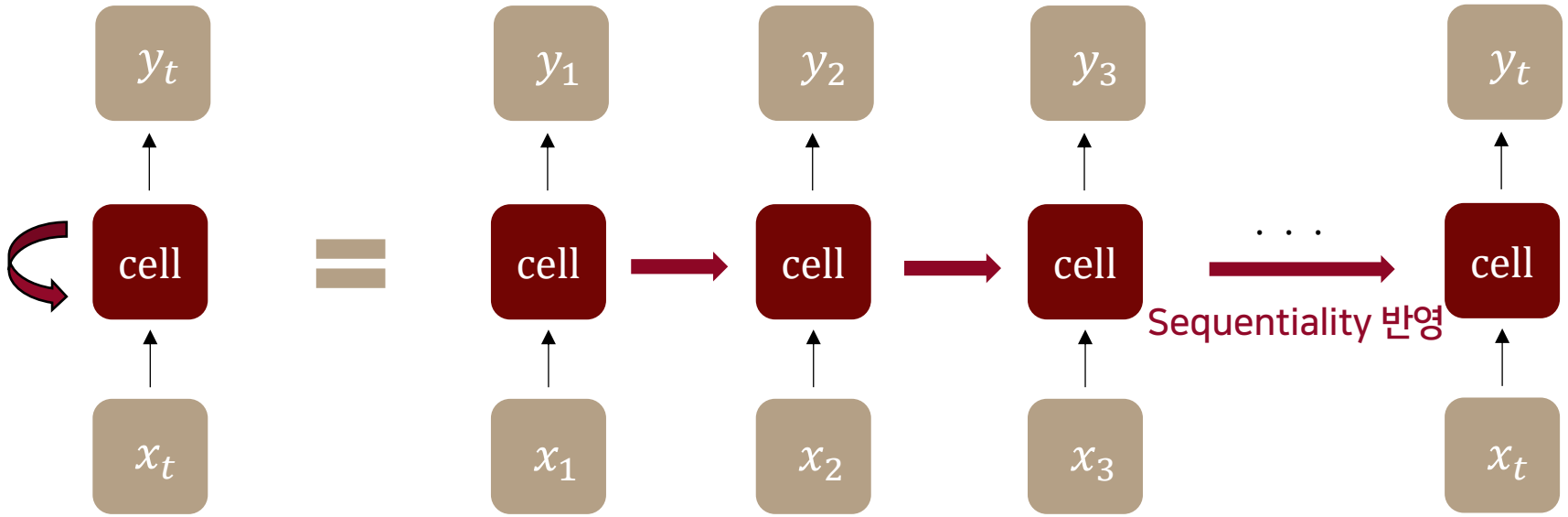
NNLM은 앞으로만 향하는 feed forward 방식이었다면,
RNN은 output을 다시 hidden layer로 순환시키는
Recurrent 방식!



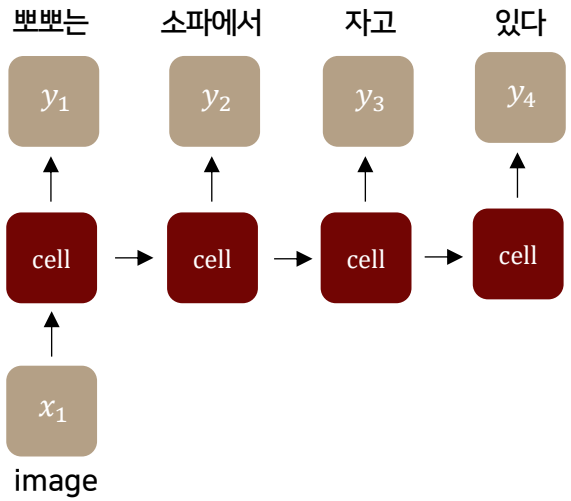
Hidden layer의 output=hidden state
Hidden state는 output layer 방향으로도 전달되고,
다시 hidden layer(RNN cell) 방향으로도 전달된다.
즉, 이전의 값을 기억하려는 성질을 가지므로
이 cell을 memory cell이라고도 부름.

$$\mathbf{h}_t = f(W_{hx}\boldsymbol{x}_t) + \text{이전 시점들에 대한 정보}$$
$$\boldsymbol{o}_t = g(W_{oh}\mathbf{h}_t)$$

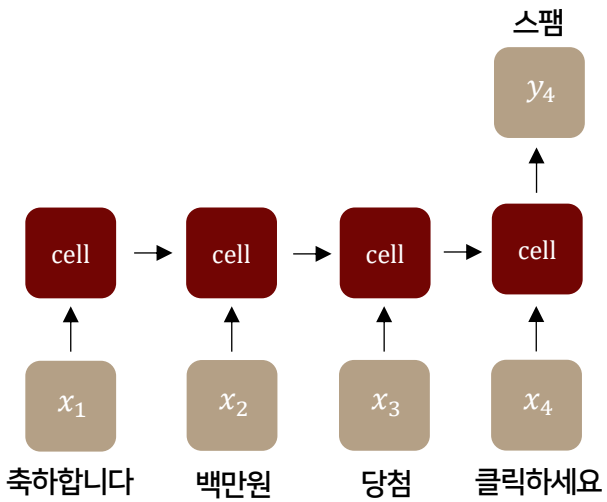
$$f(\bullet) = \boldsymbol{tanh}, g(\bullet) = \boldsymbol{softmax}$$



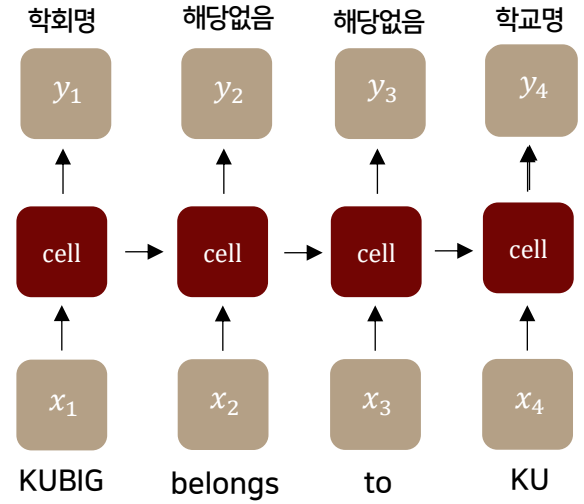
RNN can be applied to various task in NLP



one-to-many
Ex) image captioning

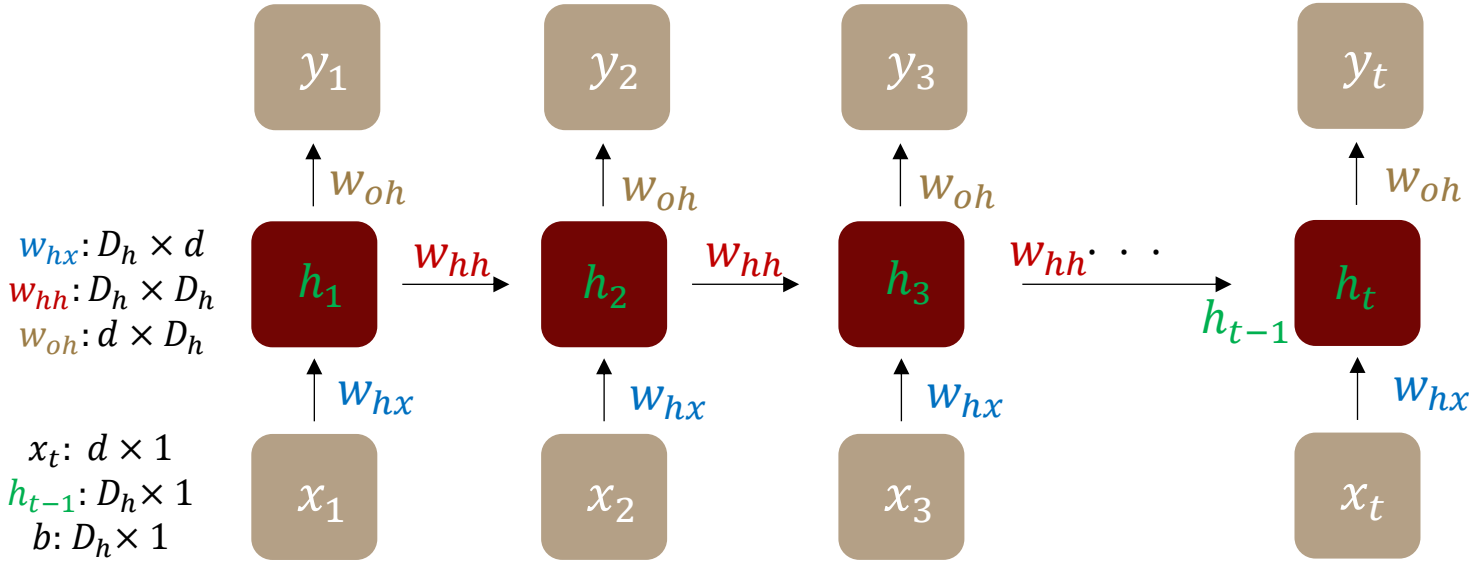


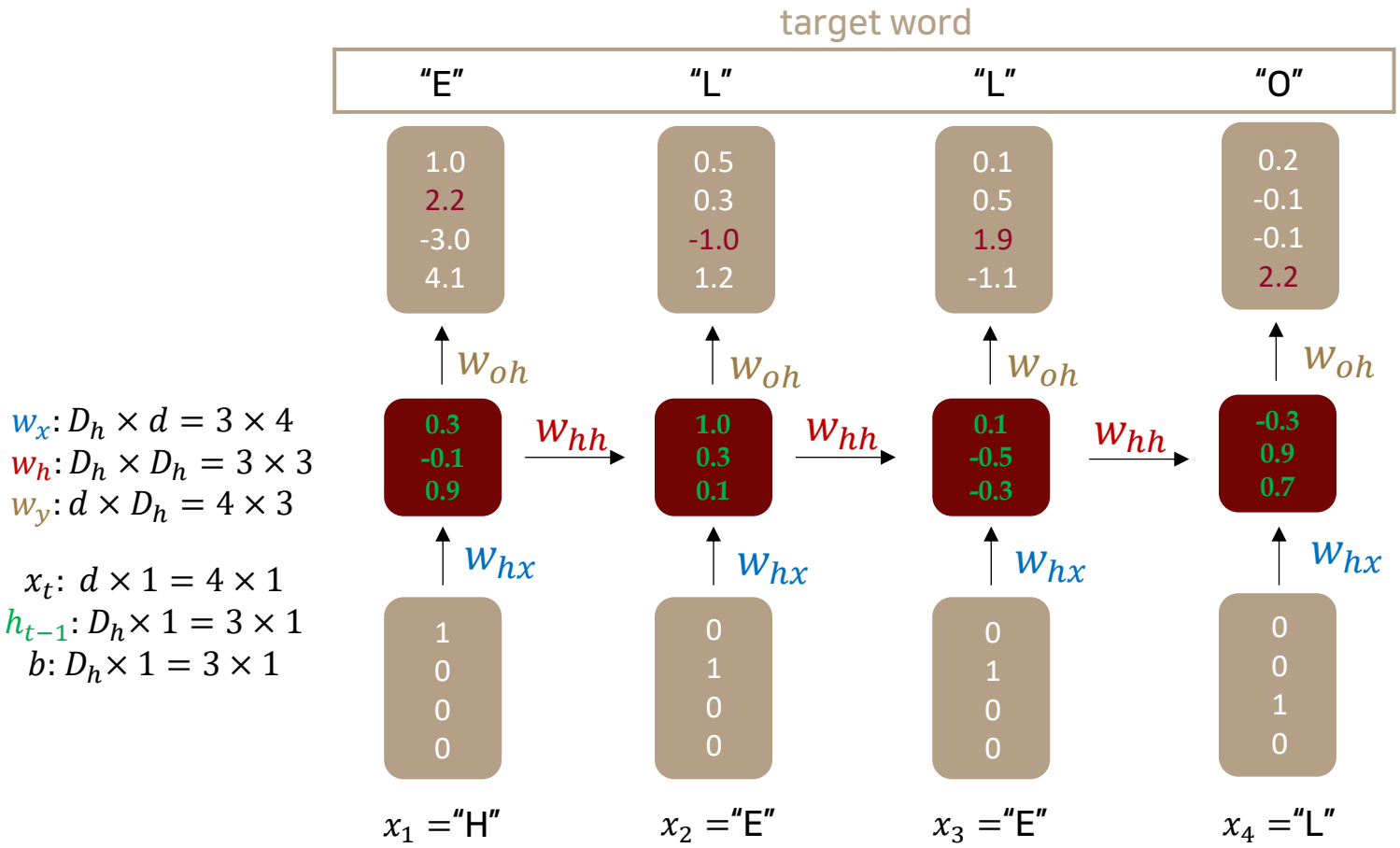
many-to-one
Ex) spam classification
Sentiment analysis



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

- 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





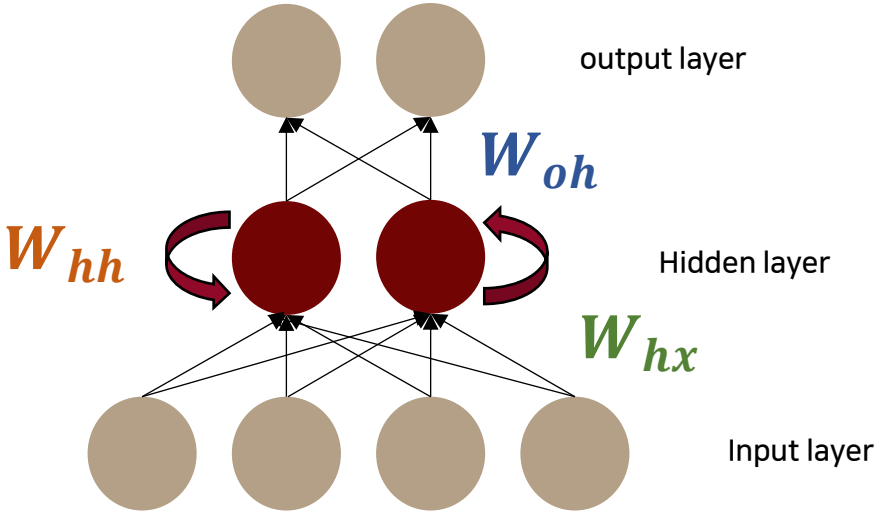
$w_x: D_h \times d = 3 \times 4$
$w_h: D_h \times D_h = 3 \times 3$
$w_y: d \times D_h = 4 \times 3$

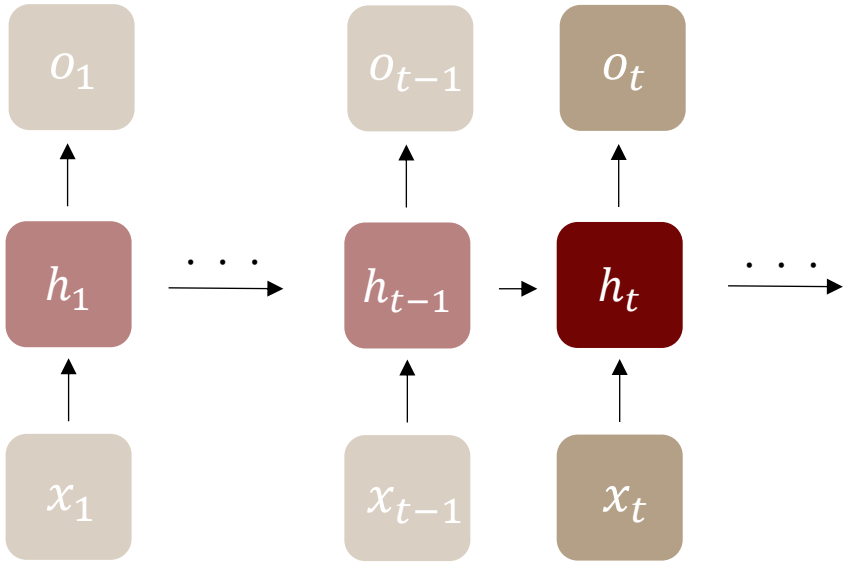
$x_t: d \times 1 = 4 \times 1$
$h_{t-1}: D_h \times 1 = 3 \times 1$
$b: D_h \times 1 = 3 \times 1$

3-2. Forward pass of RNN

$$\mathbf{h}_t = f(W_{hx}x_t + W_{hh}h_{t-1})$$
$$o_t = g(W_{oh}\mathbf{h}_t)$$

$$f(\bullet) = tanh, g(\bullet) = softmax$$



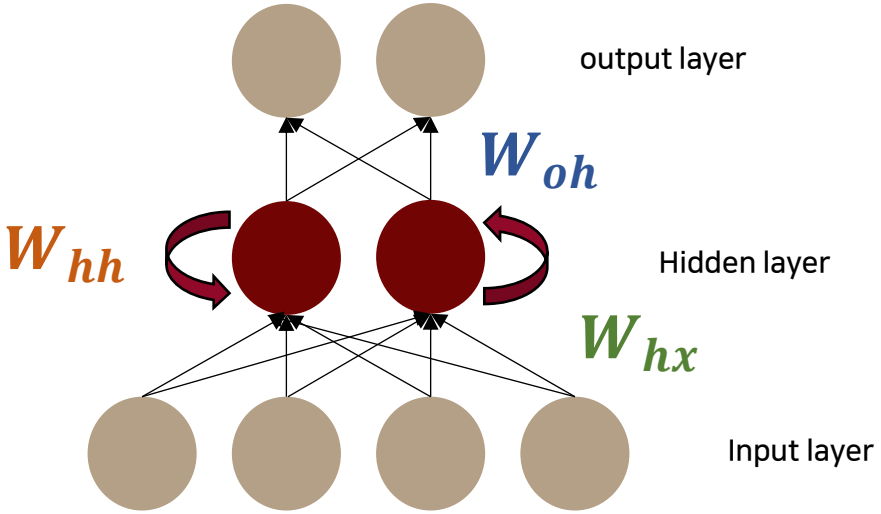


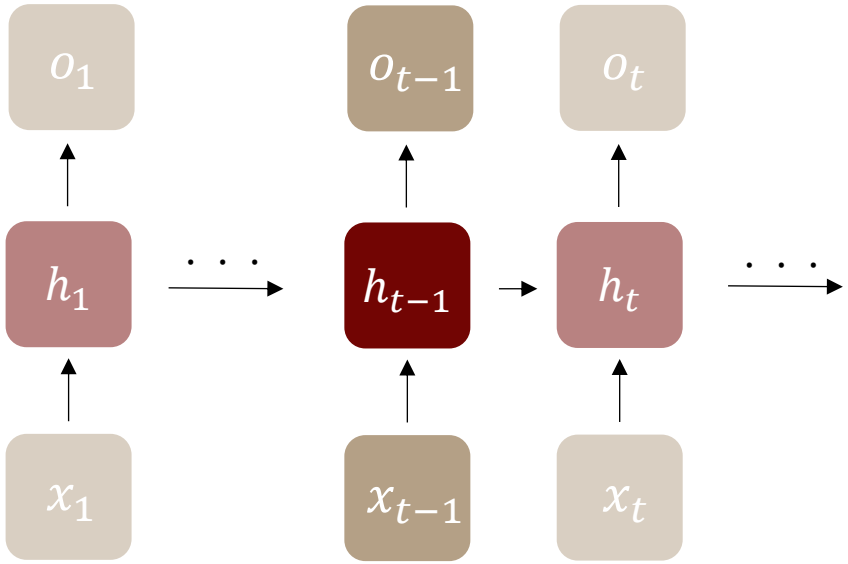
참고: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

3-2. Forward pass of RNN

$$\mathbf{h}_{t-1} = f(W_{hx}\boldsymbol{x}_{t-1} + W_{hh}\boldsymbol{h}_{t-2})$$
$$\mathbf{h}_t = f(W_{hx}\boldsymbol{x}_t + W_{hh}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \boldsymbol{g}(W_{oh}\mathbf{h}_t)$$

$$f(\bullet) = \boldsymbol{tanh}, \boldsymbol{g}(\bullet) = \boldsymbol{softmax}$$

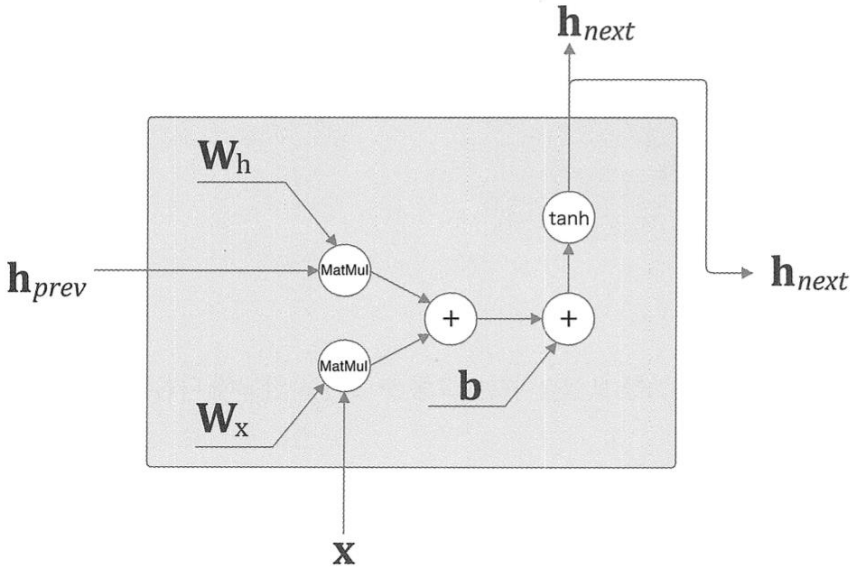




참고: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

3-2. Forward pass of RNN

forward



$$\mathbf{h}_t = tanh(W_{hx}x_t + W_{hh}h_{t-1})$$

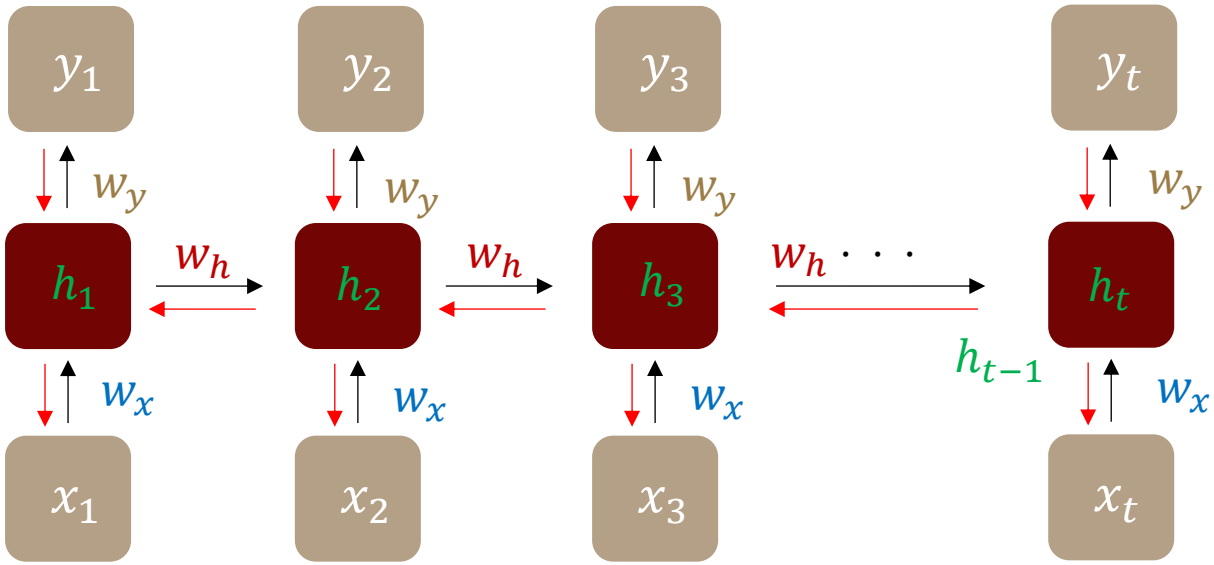
```
class RNN:
  def __init__(self, Wx, Wh, b):
    self.params = [Wx,Wh,b]
    self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
    self.cache = None

  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)
```

$$o_t = softmax(W_{oh}\mathbf{h}_t)$$

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

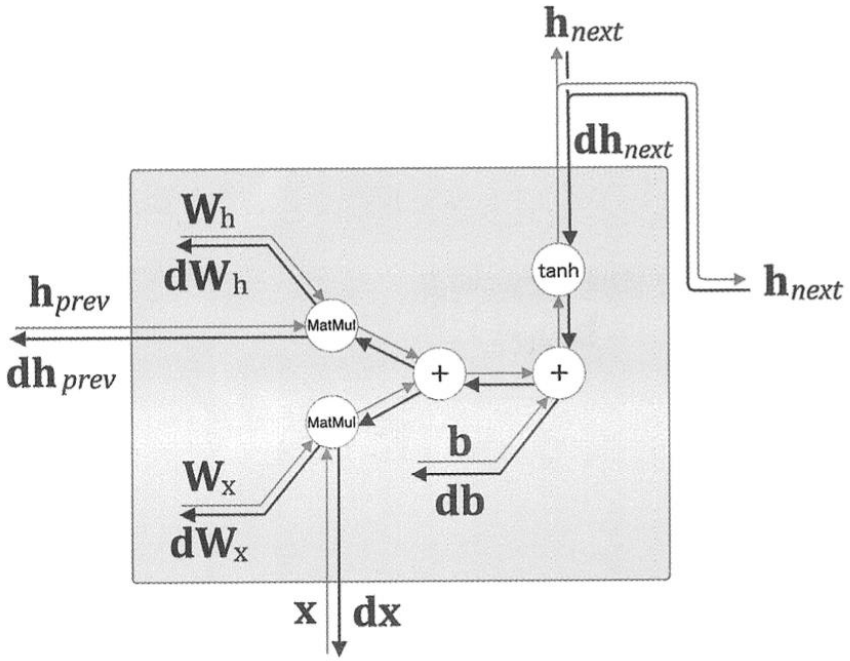
BPTT(Backpropagation Through Time)



Time step에 따라 Back Propagation

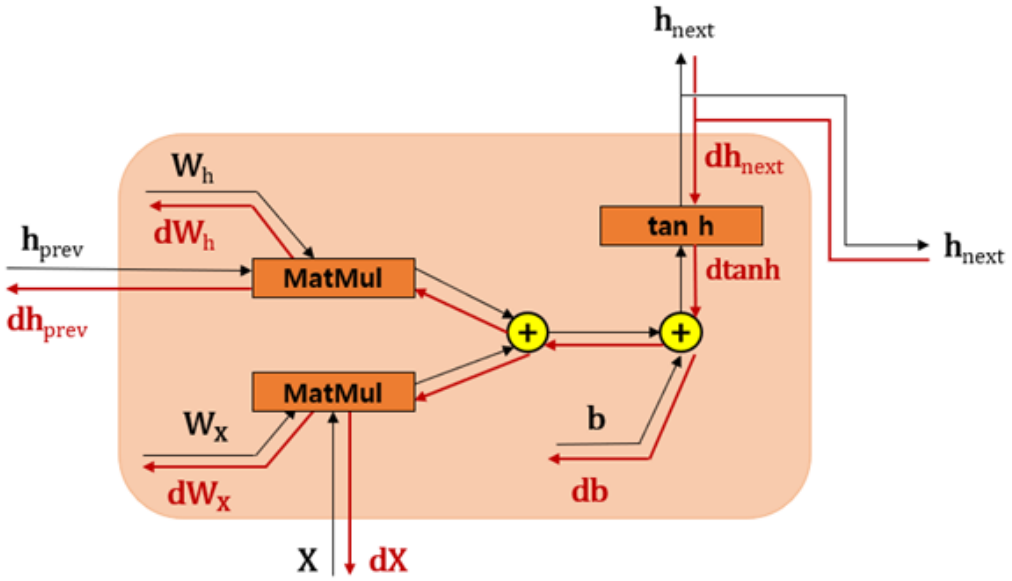
3-3. Backpropagation of RNN

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



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

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

$\mathbf{db}$: 데이터의 0번째 column의 합

$\mathbf{dWh} = \mathbf{dtanh} * \mathbf{h_{prev}} = \mathbf{dh_{next}} * \left(\mathbf{1} - \mathbf{tanh^2}\right) * \mathbf{h_{prev}}$
$\mathbf{dh_{prev}} = \mathbf{dtanh} * \mathbf{W_h} = \mathbf{dh_{next}} * \left(\mathbf{1} - \mathbf{tanh^2}\right) * \mathbf{W_h}$

Given $\mathbf{h_t} = f_{w_h}(\mathbf{x_t}, \mathbf{h_{t-1}})$, $\boldsymbol{o}_t = g_{w_o}(\mathbf{h_t})$, and $L(x, y, w_h, w_o) = \sum_{t=1}^{T} l(y_t, o_t)$

Loss function

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

⛓ Chain rule!

$$\frac{\mathrm{d}f(y(x), x)}{\mathrm{d}x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial x}$$

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

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

Given $\mathbf{h_t} = \tanh(\mathbf{x_t}, \mathbf{h_{t-1}})$, $\boldsymbol{o}_t = g_{w_o}(\mathbf{h_t})$, and $L(x, y, w_h, w_o) = \sum_{t=1}^{T} l(y_t, o_t)$

Loss function

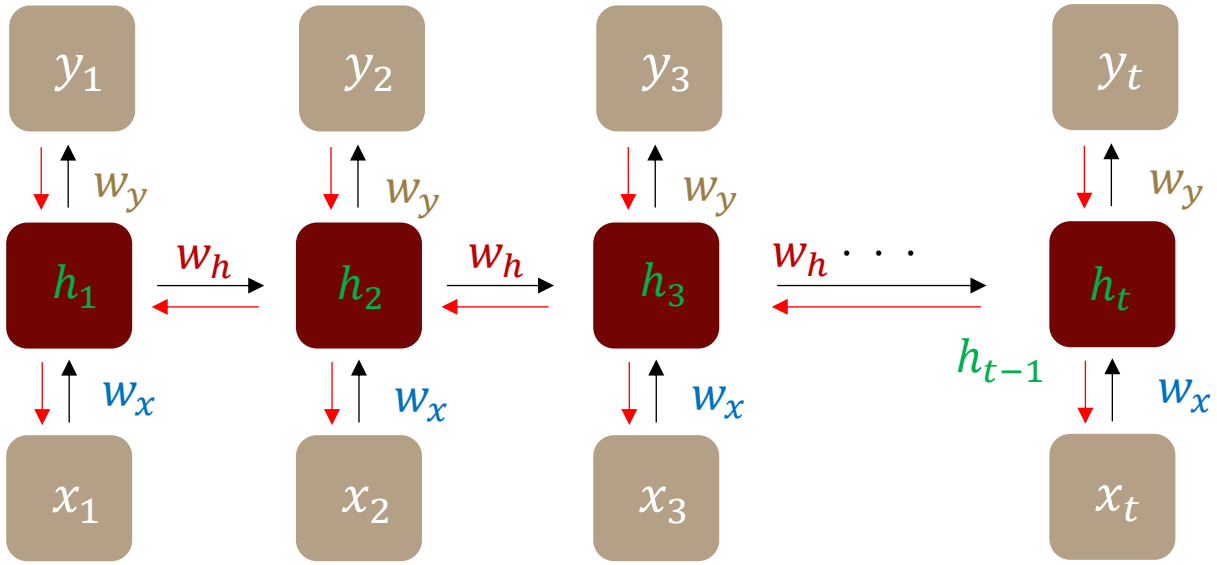
$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^{T} \partial_{w_h} l(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} l(y_t, o_t) \partial_{h_t} g(h_t, w_h) \partial_{w_h} h_t$$

⛓ Chain rule!

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

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

BPTT(Backpropagation Through Time)

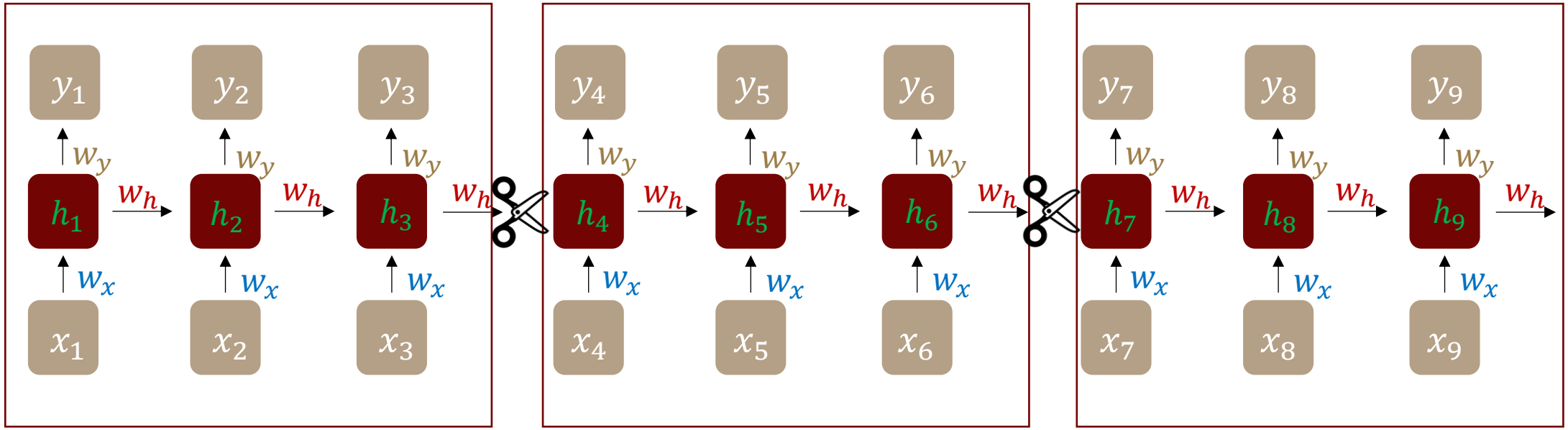


time step(=문장길이)가 길어짐에 따라

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

Sol) Truncated BPTT

Truncated BPTT

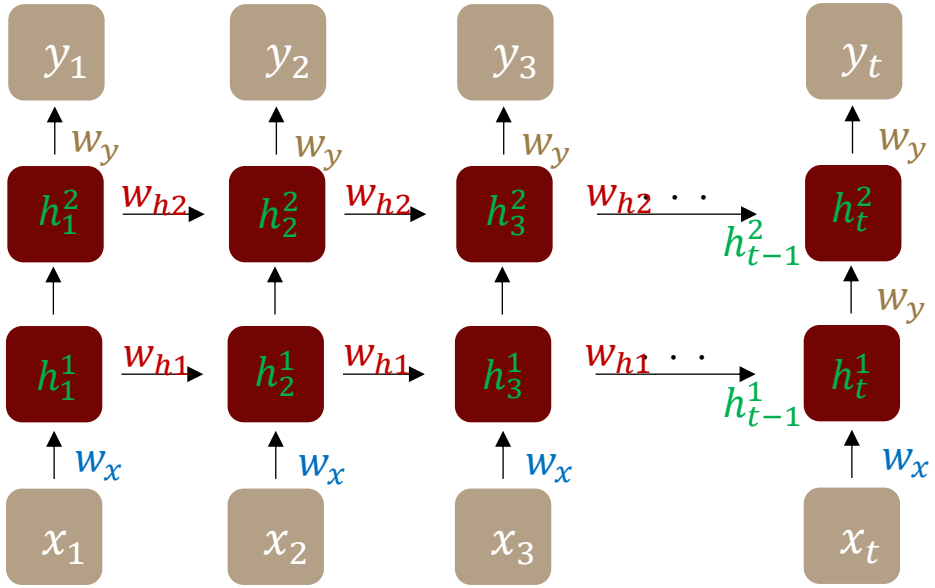


기울기 소실, 메모리 공간 낭비, 연산량 문제를 해결하기 위해
Backpropagation할 시에 적당한 길이로 끊음
forward propagation에선 끊어지지 않음!

3-5. Deep, Bidirectional RNN

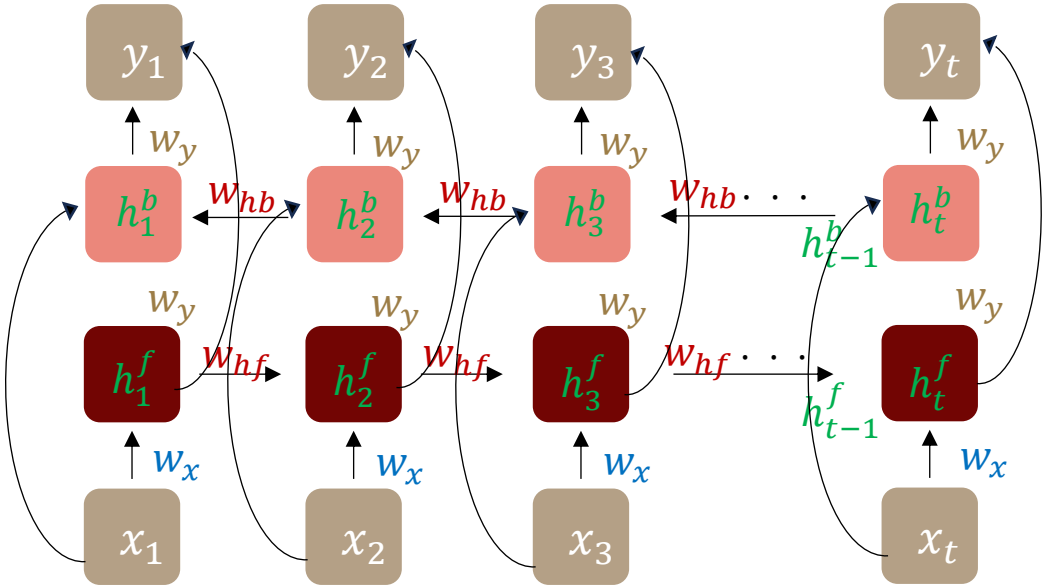
KUBIG DATA SCIENCE & AI

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: 정찬이가 랩 미팅을 다녀온 뒤로 얼굴에 _____ 가득해서 참 다행이야
1) 화색이 2) 슬픔이 3) 정색이

3-5. Deep, Bidirectional RNN

KUBIG DATA SCIENCE & AI

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])
```

04 LSTM, GRU

게이트가 추가된 RNN

KUBIG
DATA SCIENCE & AI

4-1. Vanishing or Exploding Gradient

승현이는 CS 스터디를 위해 과도 스룸에 도착했다.
역시 부지런하다.
오자마자 짐을 풀고 아메리카노 한잔을 했다.
뒤이어 우진이도 두번째로 스룸에 도착했다.
우진이는 _____에게 인사했다.

🤔 문장이 길어져도 빈칸에 들어갈 사람이 '승현'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)

4-1. Vanishing or Exploding Gradient

Gradient Vanishing

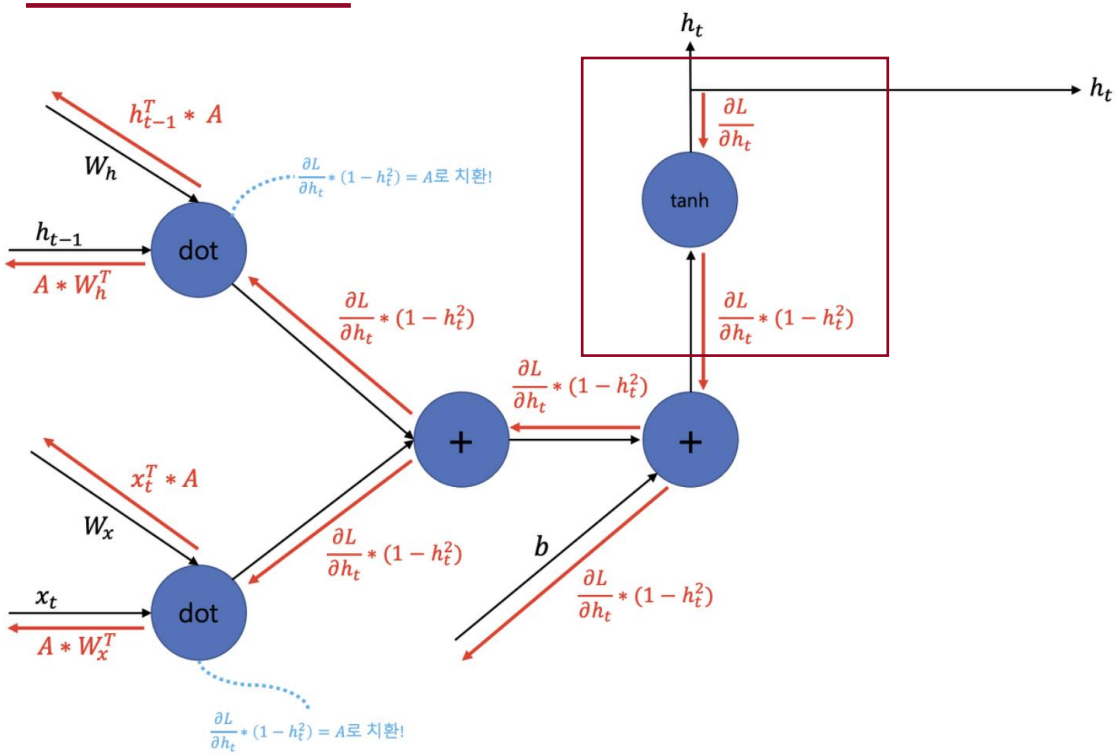
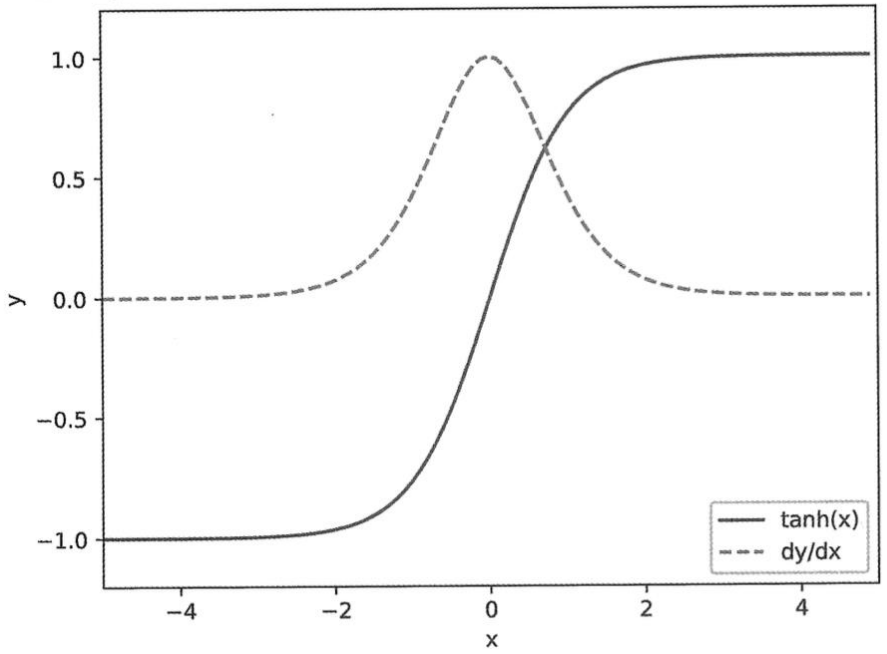


그림: 밑바닥부터 시작하는 딥러닝2

$y = tanh(x)$일 때 $\frac{\partial y}{\partial x} = 1 - y^2$

tanh 연산에서 backprop하면 기울기 소실 위험

그림 6-6 $y = \tanh(x)$의 그래프(점선은 미분)



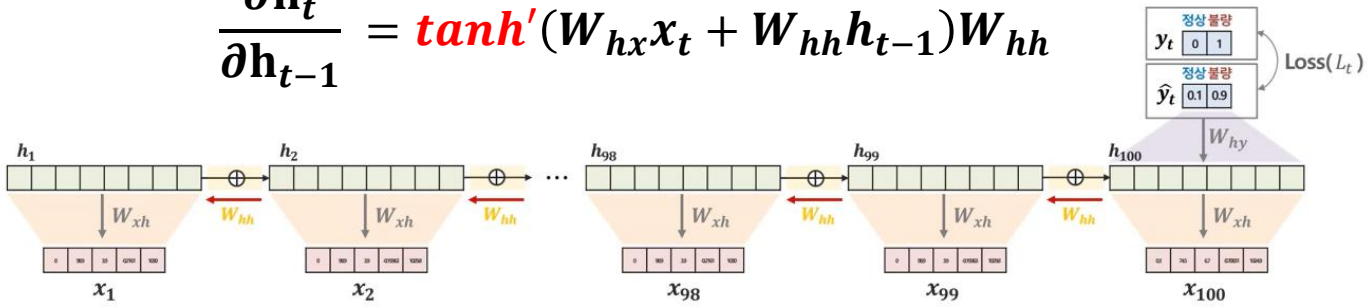
$\frac{\partial y}{\partial x}$ 이 0과 1사이의 값, 즉 tanh 노드를 지날 때마다 값이 계속 작아짐

만약 time step이 매우 길다면?

Gradient Vanishing

$$\mathbf{h}_t = tanh(W_{hx}x_t + W_{hh}h_{t-1})$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = tanh'(W_{hx}x_t + W_{hh}h_{t-1})W_{hh}$$



T_{100}: 시점 100 에서의 영향 T_{99}: 시점 100 으로부터 전해진 영향 고려 T_{98}: 시점 99 으로부터 전해진 영향 고려

$$\frac{\partial Loss}{\partial W_{hh}} = \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \mathbf{h}_{100}} \times \frac{\partial \mathbf{h}_{100}}{\partial W_{hh}}} + \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \mathbf{h}_{100}} \times \frac{\partial \mathbf{h}_{100}}{\partial \mathbf{h}_{99}} \times \frac{\partial \mathbf{h}_{99}}{\partial W_{hh}}} + \boxed{\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial y}{\partial \mathbf{h}_{100}} \times \frac{\partial \mathbf{h}_{100}}{\partial \mathbf{h}_{99}} \times \frac{\partial \mathbf{h}_{99}}{\partial \mathbf{h}_{98}} \times \frac{\partial \mathbf{h}_{98}}{\partial W_{hh}}}$$

T_1: 시점 2 으로부터 전해진 영향 고려

$$\boxed{+\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial y}{\partial \mathbf{h}_{100}} \times \frac{\partial \mathbf{h}_{100}}{\partial \mathbf{h}_{99}} \times \cdots \times \frac{\partial \mathbf{h}_6}{\partial \mathbf{h}_5} \times \frac{\partial \mathbf{h}_5}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \times \frac{\partial \mathbf{h}_1}{\partial W_{hh}}}$$

$$W_{hh}^{new} = W_{hh}^{old} - \eta * \frac{\partial Loss}{\partial W_{hh}}$$

$$W_{hh}^{new} = W_{hh}^{old} - \eta * \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial \mathbf{h}_{100}} \times \frac{\partial \mathbf{h}_{100}}{\partial W_{hh}} + \cdots + \frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial y}{\partial \mathbf{h}_{100}} \times \frac{\partial \mathbf{h}_{100}}{\partial \mathbf{h}_{99}} \times \cdots \times \frac{\partial \mathbf{h}_6}{\partial \mathbf{h}_5} \times \frac{\partial \mathbf{h}_5}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \times \frac{\partial \mathbf{h}_1}{\partial W_{hh}}\right)$$

그림: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

4-1. Vanishing or Exploding Gradient

Gradient Exploding

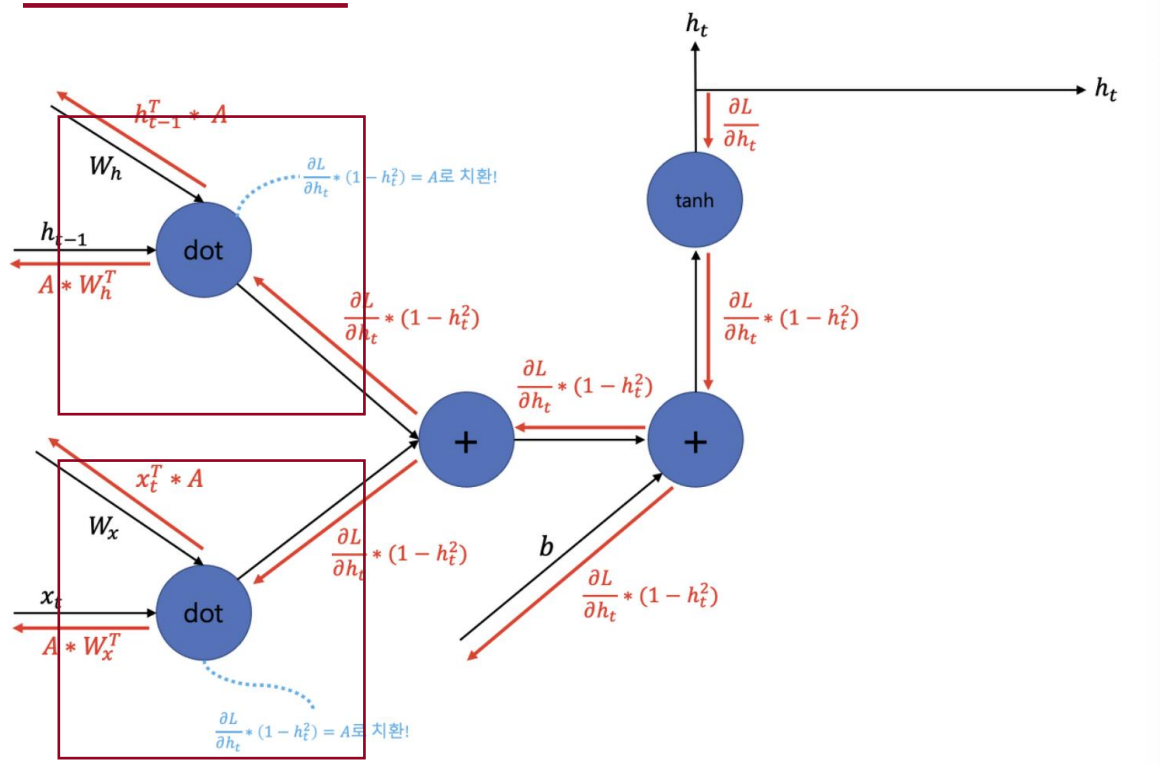


그림: 밑바닥부터 시작하는 딥러닝2

matmul 연산에서 backprop하면 기울기 폭등 위험

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

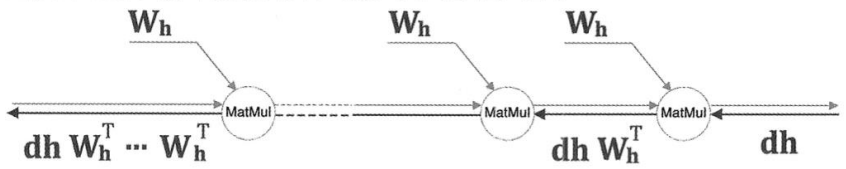
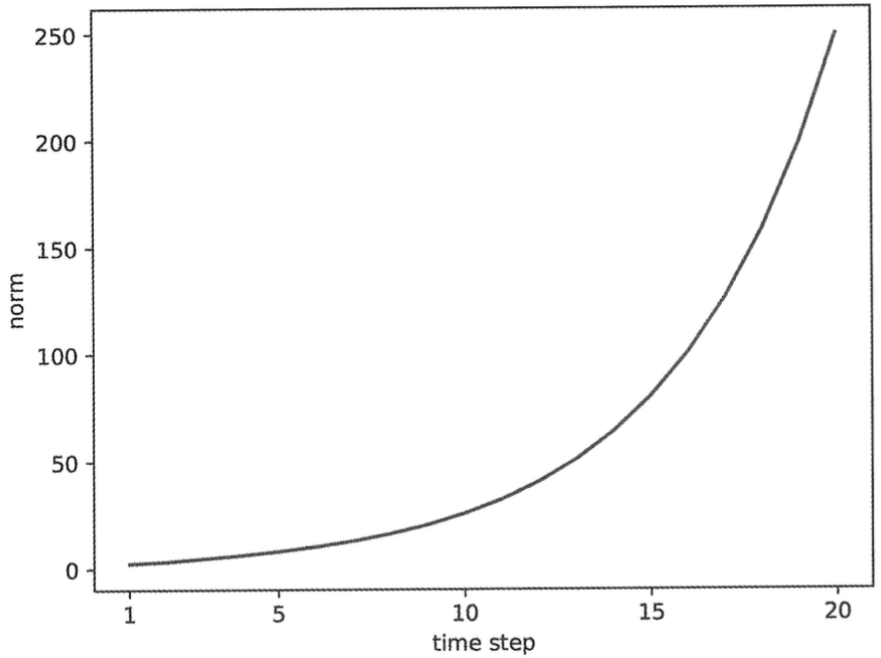


그림 6-8 기울기 dh는 시간 크기에 비례하여 지수적으로 증가한다.



4-1. Vanishing or Exploding Gradient

Solution to exploding gradient

Clipping

기울기가 exploding하려 할 때
인위적으로 그 기울기 값에 특정 조치를 취해주는 기법

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

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

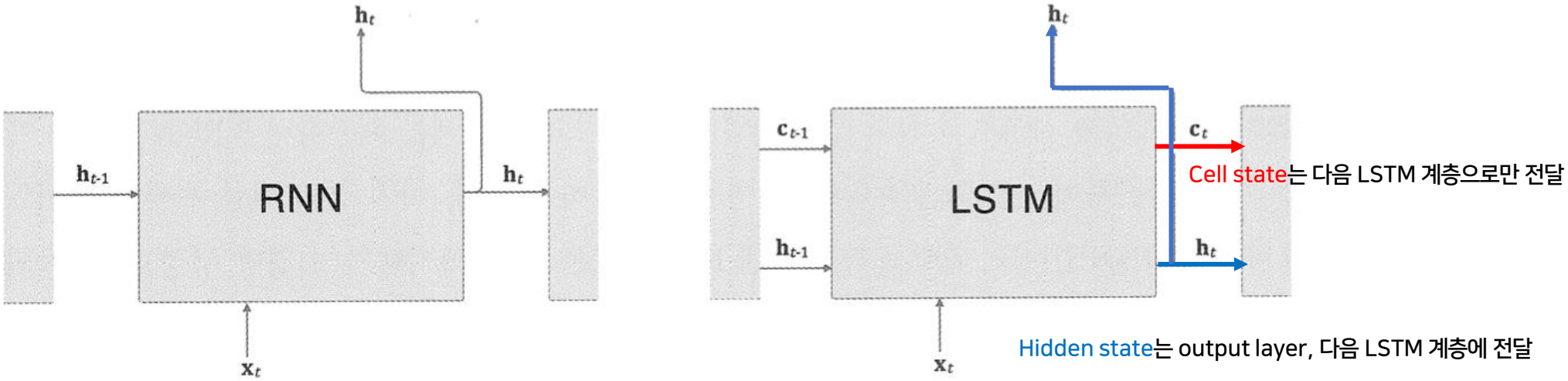
Solution to vanishing gradient

LSTM

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

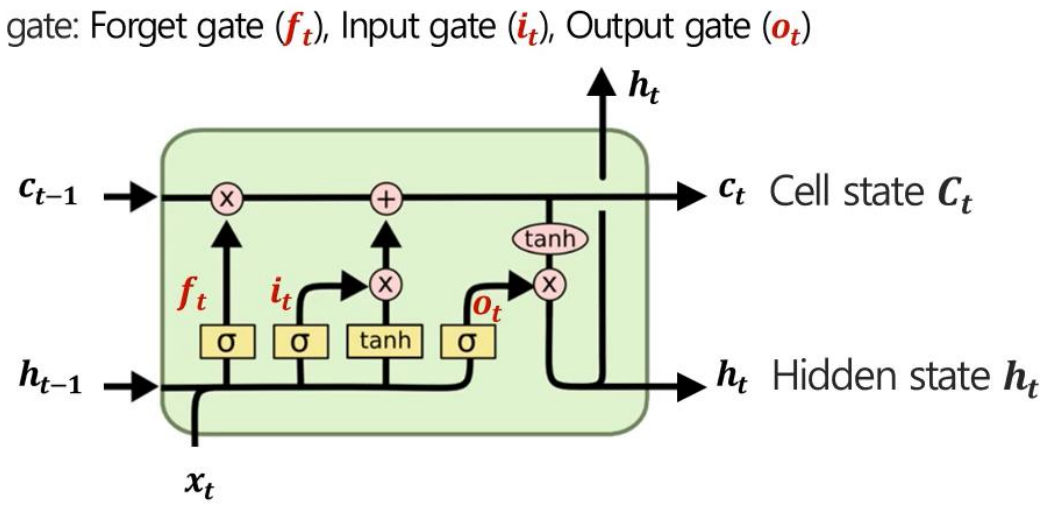
Then, how?

그림 6-11 RNN 계층과 LSTM 계층 비교



- Hidden state만 존재하던 RNN과 달리 Cell state가 추가됨.
- c_t에는 t 시점까지의 기억 정보들이 누적되어있음. 즉, 장기적으로 정보들을 유지.
- 이 때 정보들을 얼마나 기억할지를 정할 수 있음.

그림: 밑바닥부터 시작하는 딥러닝2



LSTM은 아래 세 가지 gate로 얼마나 기억하고, 잊을지를 정한다!

- Forget gate
- Input gate
- Output gate

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

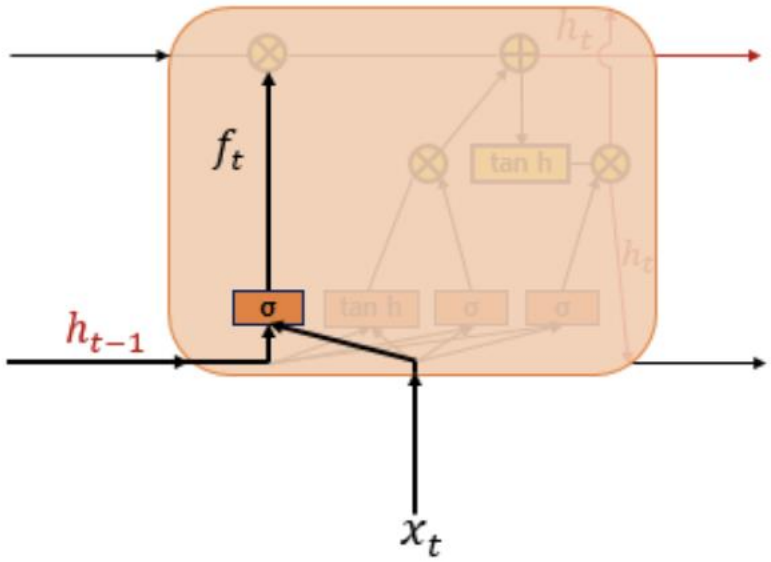
→ 세 게이트 모두 sigmoid를 통해 0과 1사이 값 출력 !

그림: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

4-3. Three Gates of LSTM

KUBIG DATA SCIENCE & AI

forget gate



이전 기억을 얼마나 삭제할 것인지 정하는 gate

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

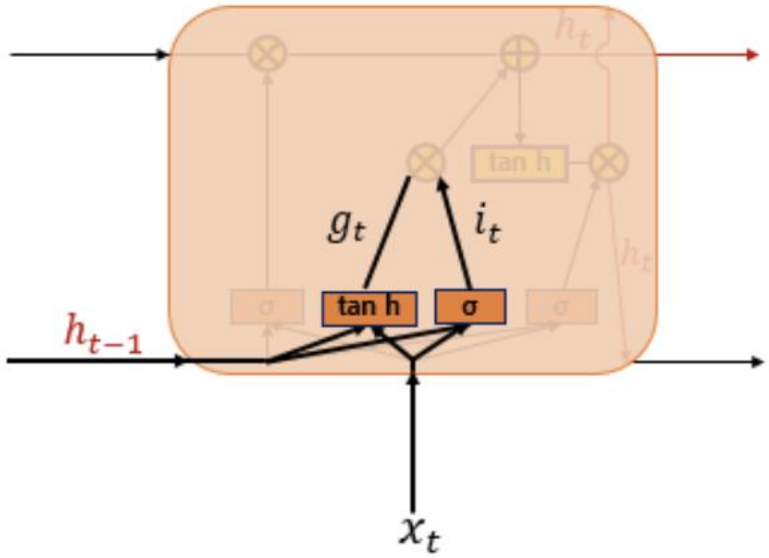
- f_t는 직전 시점의 cell state(c_{t-1})와 곱해짐.
- 0에 가까울수록 기억이 많이 삭제된 것이고, 1에 가까울수록 온전히 기억한 것임
- 얼마나 기억할 것인지를 LSTM이 가중치를 업데이트하며 스스로 학습함.

그림: 딥러닝을 이용한 자연어처리

4-3. Three Gates of LSTM

KUBIG DATA SCIENCE & AI

input gate



새로운 정보를 얼마나 누적할 것인지 정하는 gate

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

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

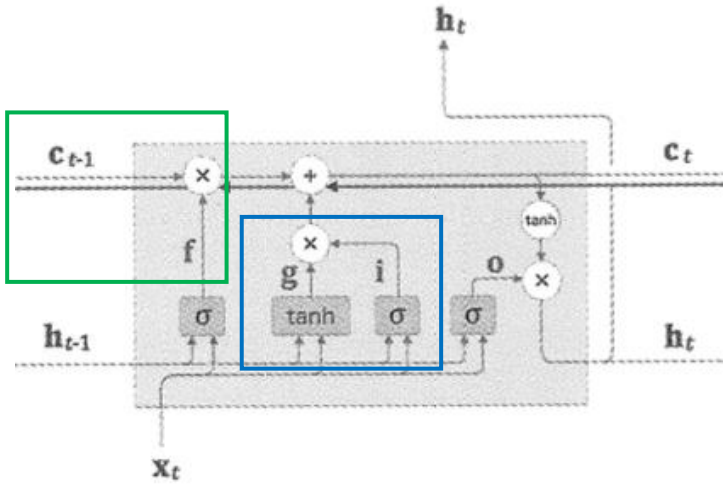
- g_t는 현재 입력 값과 과거 hidden state의 정보 요약 값
- i_t는 g_t와 곱해지며 1에 가까울수록 새로운 정보를 온전히 누적.
- 얼마나 누적할 것인지를 LSTM이 가중치를 업데이트하며 스스로 학습함.

그림: 딥러닝을 이용한 자연어처리

4-3. Three Gates of LSTM

KUBIG DATA SCIENCE & AI

Cell state



Forget gate Input gate

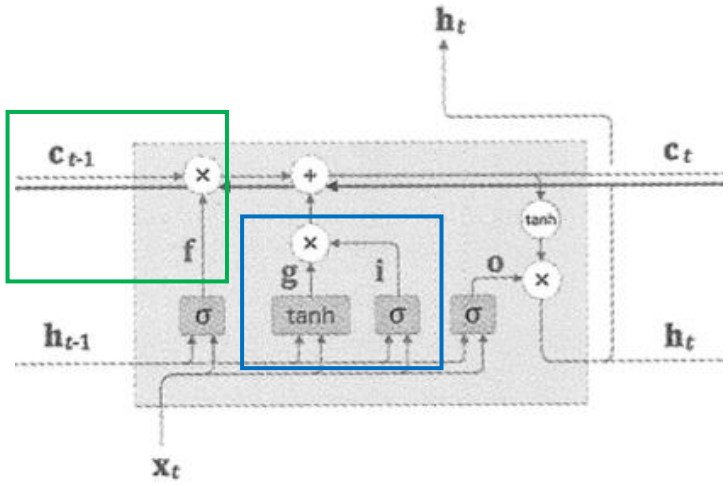
$$C_t = f_t \otimes C_{t-1} \oplus i_t \otimes g_t$$

- Forget gate를 통해 불필요한 기억은 지우고 input gate를 통해 중요한 현재 정보는 살려 현재 시점의 cell state(C_t) 업데이트
- $\otimes$는 원소별 곱, $\oplus$는 합을 뜻함
- 이 cell state는 장기적으로 정보를 유지하는 역할

그림: 밑바닥부터 시작하는 딥러닝2

KUBIG DATA SCIENCE & AI

Cell state



$$C_t = f_t \otimes C_{t-1} \oplus i_t \otimes g_t$$

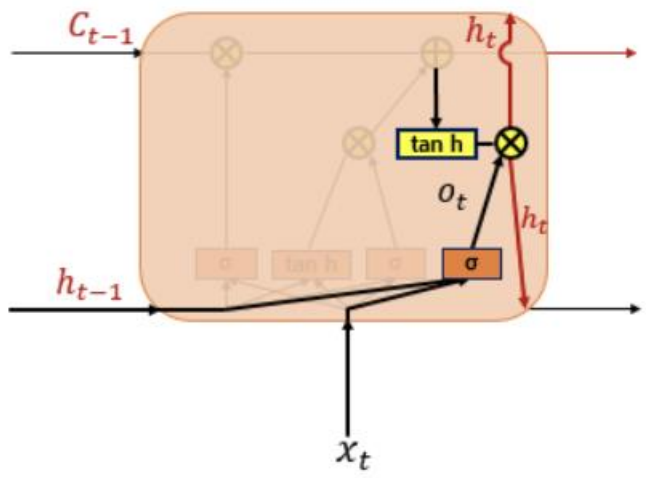
여기서 ⊗는 원소별 곱, 즉 elementwise product !

f_t	0 | 0.2 | 0.9 | 0.1 | 1	0.1 | 0 | 0.8 | 0.2 | 0.8	i_t
C_{t-1}	-0.2 | 0.1 | 0.5 | 0.7 | 0.9	-0.1 | 0.3 | 0.6 | 0.2 | 0.9	g_t
$f_t \otimes C_{t-1}$	0 | 0.02 | 0.45 | 0.07 | 0.9	⊕ -0.01 | 0 | 0.48 | 0.04 | 0.72	$i_t \otimes g_t$

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

그림: 밑바닥부터 시작하는 딥러닝2

output gate



Input, forget gate의 값들을 다음 단계로 전달해주는 gate

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

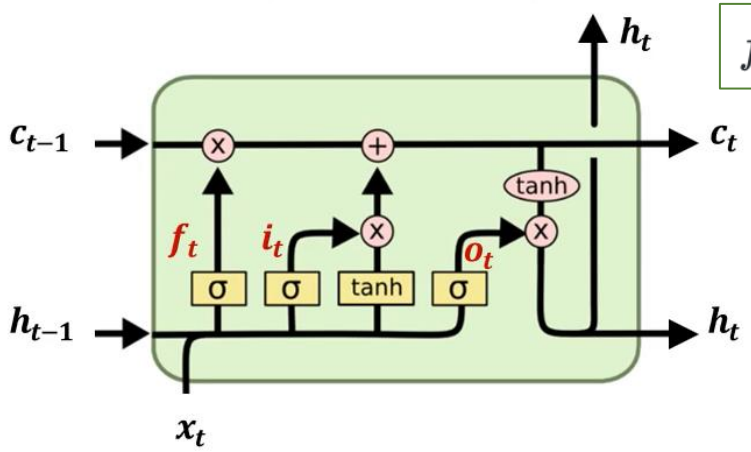
$$h_t = o_t \circ tanh(c_t)$$

- o_t는 tanh(c_t)와 곱해져 hidden state h_t으로 업데이트
- 즉 o_t를 통해 hidden state에 cell state를 얼마나 반영할 것인지 결정
- 마찬가지로 원소별 곱 $\otimes$ 사용

그림: 딥러닝을 이용한 자연어처리

4-3. Three Gates of LSTM

gate: Forget gate (f_t), Input gate (i_t), Output gate (o_t)



$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$g_t = tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$

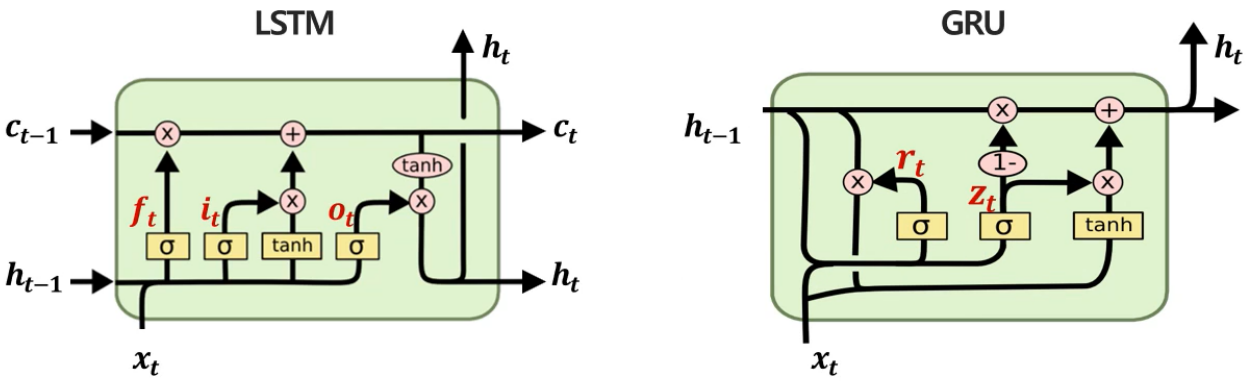
Cell state $C_t = f_t \otimes C_{t-1} \oplus i_t \otimes g_t$

Hidden state $h_t = o_t \otimes \tanh(C_t)$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

그림: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

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



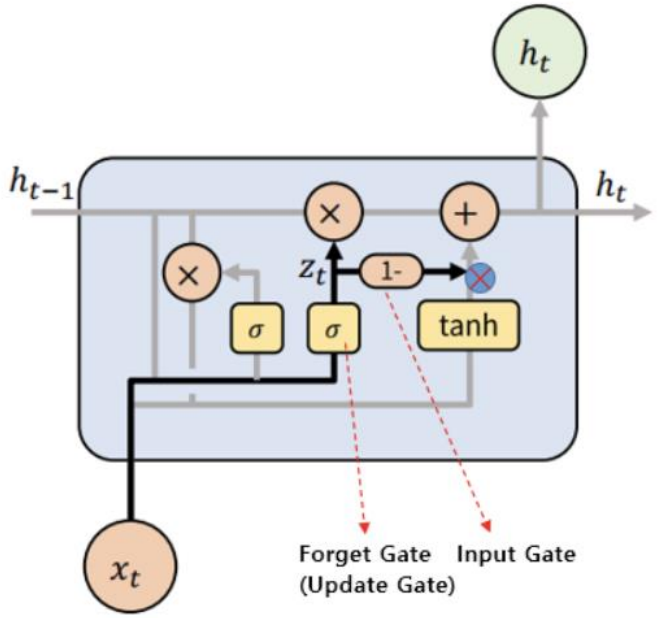
GRU는 LSTM의 구조를 간단히 개선하여 파라미터 수를 줄임

- Forget gate, input gate를 update gate(z_t)로 통합
- Output gate를 없애고 reset gate(r_t)를 정의
- Cell state, hidden state를 hidden state로 통합

그림: 고려대학교 산업경영공학부 [핵심머신러닝] https://www.youtube.com/watch?v=006BjyZicCo

update gate

과거 정보와 현재 정보의 업데이트 비율을 결정하는 gate



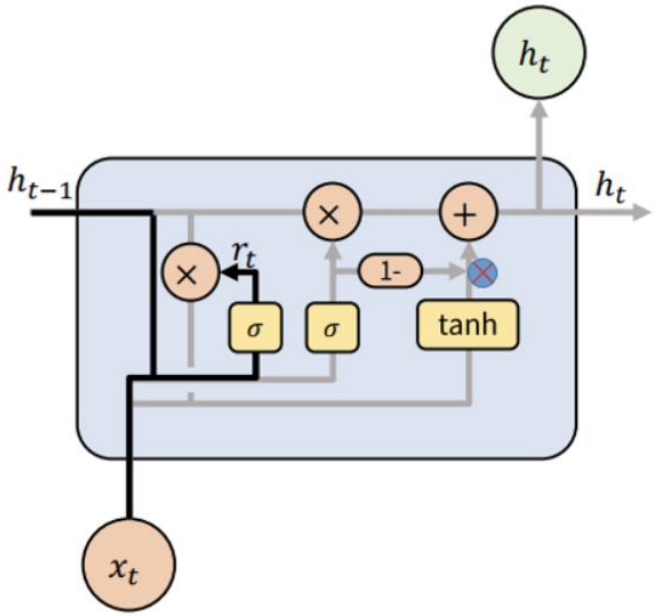
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$ => 0에서 1의 값 출력

- z_t는 forget gate(f_t)의 역할로, 이전 hidden state h_{t-1}와 곱해짐
- $1 - z_t$ 는 input gate(i_t)의 역할로, 임시 hidden state g_t와 곱해짐

그림: https://velog.io/@lighthouse97/Gated-Recurrent-UnitGRU%EC%9D%98-%EC%9D%B4%ED%95%B4

reset gate

과거의 정보를 적당히 reset해주는 gate

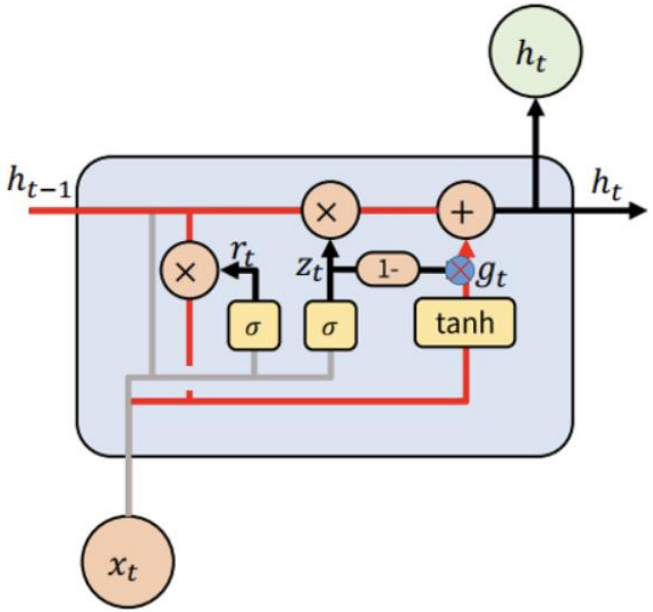


$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$ => 0에서 1의 값 출력

- r_t 는 이전 hidden state h_{t-1} 와 곱해져 임시 hidden state g_t 를 구하는데 활용

그림: https://velog.io/@lighthouse97/Gated-Recurrent-UnitGRU%EC%9D%98-%EC%9D%B4%ED%95%B4

Candidate Hidden State & Final Hidden State



○ = ⊗, 즉 elementwise product

$$g_t = tanh(W_{hg}(r_t \circ h_{t-1}) + W_{xg}x_t + b_g)$$
$$h_t = (1 - z_t) \circ g_t + z_t \circ h_{t-1}$$

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

그림: https://velog.io/@lighthouse97/Gated-Recurrent-UnitGRU%EC%9D%98-%EC%9D%B4%ED%95%B4

05 ELMo

문맥을 반영한 워드임베딩

KUBIG
DATA SCIENCE & AI

연합뉴스

삼성, 사상 첫 통합우승 4연패(종합2보)

word2vec → 0.45 | 0.02 | … | 0.89

"연패" embedding vector

데일리안

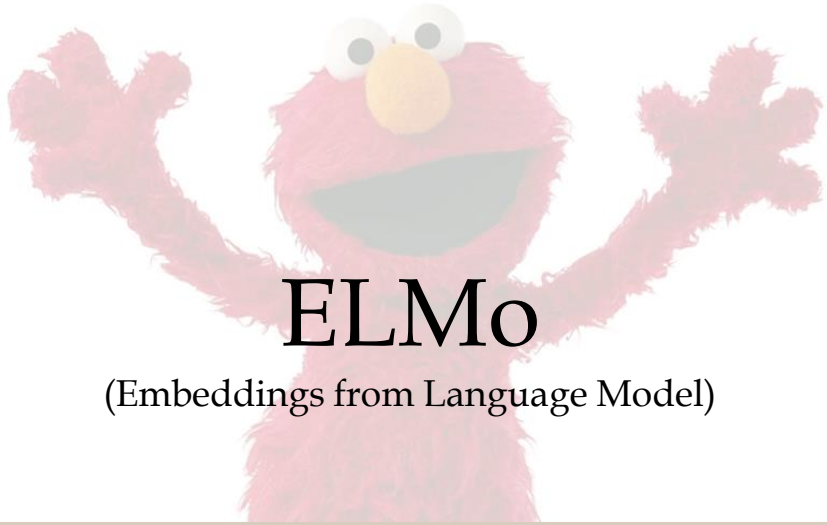
'18연패' 한화 이글스 무한 추락, KBO리그도 우울

word2vec → 0.45 | 0.02 | … | 0.89

"연패" embedding vector

다른 의미임에도 같은 벡터로 임베딩

🤔 문맥을 고려한 워드임베딩이 필요할 것 같은데…



ELMo

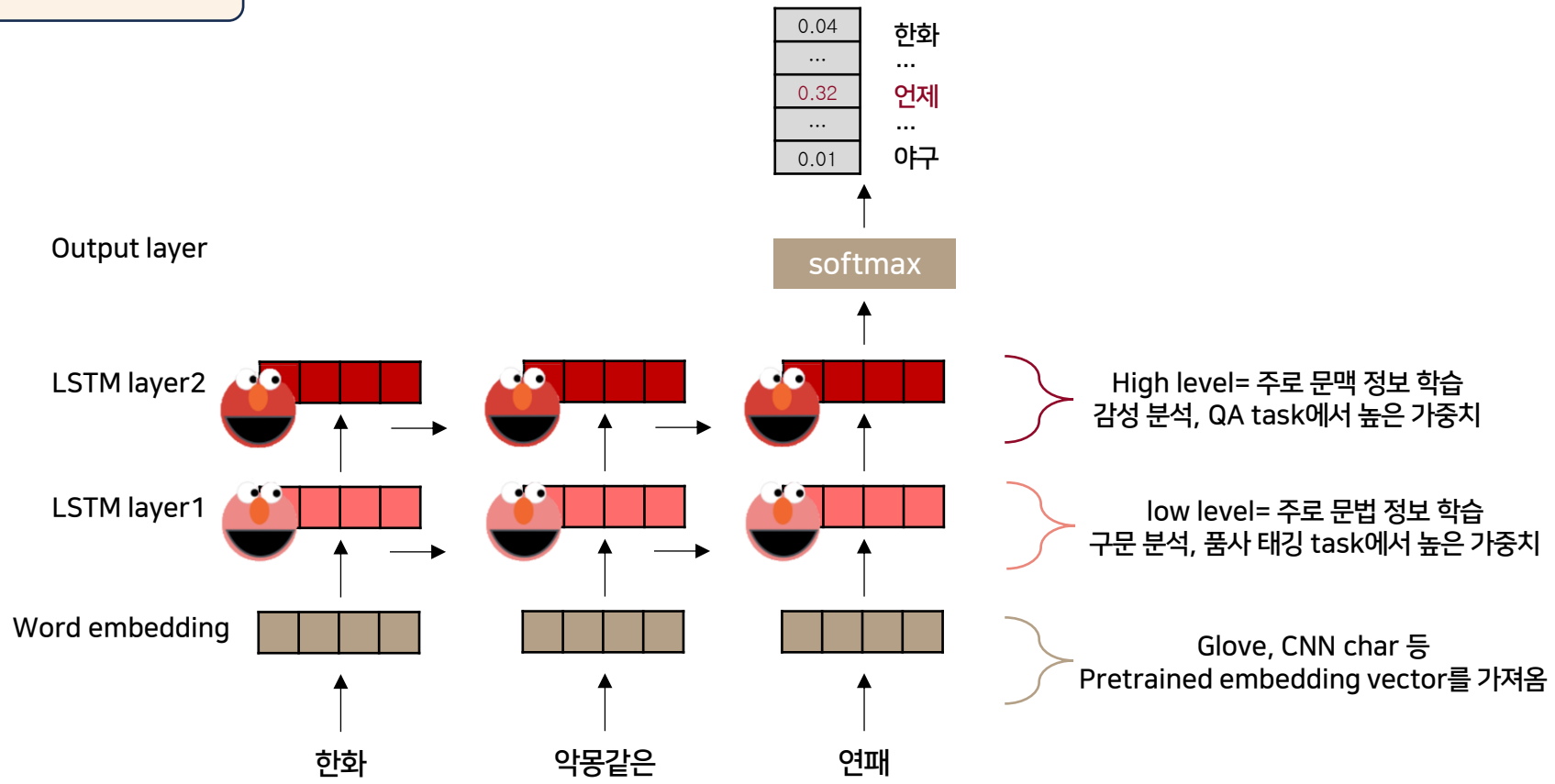
(Embeddings from Language Model)

BERT

See you soon

5-2. ELMo with biLSTM

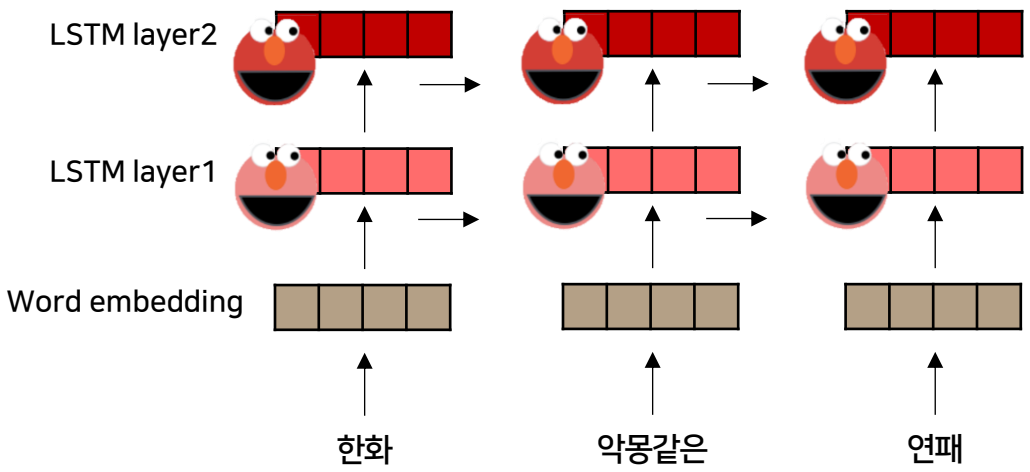
예문: 한화, 악몽같은 연패 언제 탈출하나



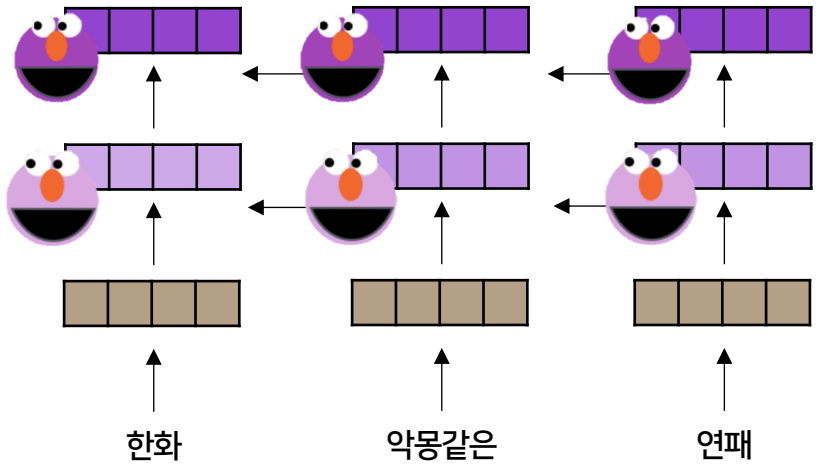
5-2. ELMo with biLSTM

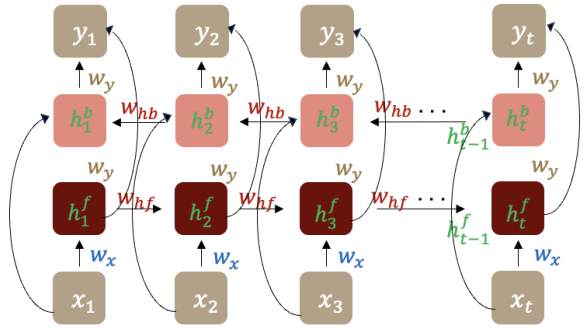
예문: 한화, 악몽같은 연패 언제 탈출하나

[Forward LM]



[Backward LM]



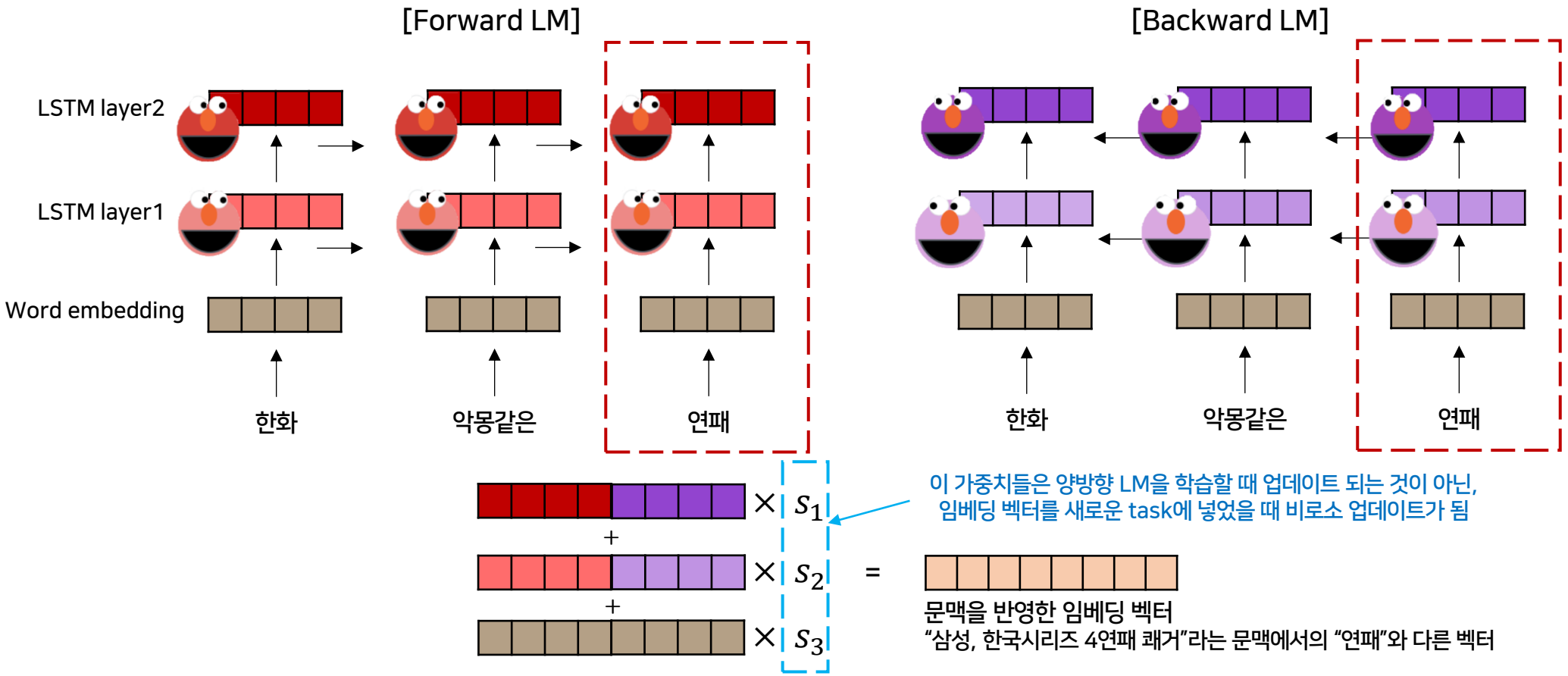


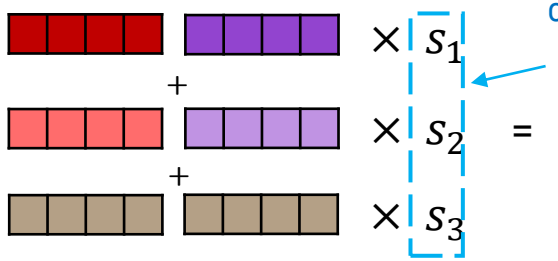
Bidirectional RNN이 hidden state만 앞뒤로 공유하는 반면,
위 경우 두 개를 각각 다른 Language Model로 보고 개별적으로 학습

5-2. ELMo with biLSTM

예문: 한화, 악몽같은 연패 언제 탈출하나

GOAL: 예문의 문맥 안에서 "연패"의 임베딩 벡터 구하기





이 가중치들은 양방향 LM을 학습할 때 업데이트 되는 것이 아닌,
임베딩 벡터를 새로운 task에 넣었을 때 비로소 업데이트가 됨

=

문맥을 반영한 임베딩 벡터
"삼성, 한국시리즈 4연패 쾌거"라는 문맥에서의 "연패"와 다른 벡터

ELMo: Embeddings from Language Models

Peters et. al (2018)

- ELMo for downstream task

ELMo represents a word t_k as a linear combination of corresponding hidden layers (inc. its embedding)

ELMo is a task specific representation. A down-stream task learns weighting parameters

$$\mathbf{ELMo}_k^{\text{task}} = \gamma^{\text{task}} \times \sum \left\{ s_2^{\text{task}} \times \mathbf{h}_{k2}^{\text{LM}},\ s_1^{\text{task}} \times \mathbf{h}_{k1}^{\text{LM}},\ s_0^{\text{task}} \times \mathbf{h}_{k0}^{\text{LM}} \right.$$

$$\left. \left([\mathbf{x}_k ; \mathbf{x}_k] \right) \right.$$

Concatenate hidden layers

$[\overrightarrow{\mathbf{h}}_{kj}^{\text{LM}} ; \overleftarrow{\mathbf{h}}_{kj}^{\text{LM}}]$

Unlike usual word embeddings, ELMo is assigned to every *token* instead of a *type*

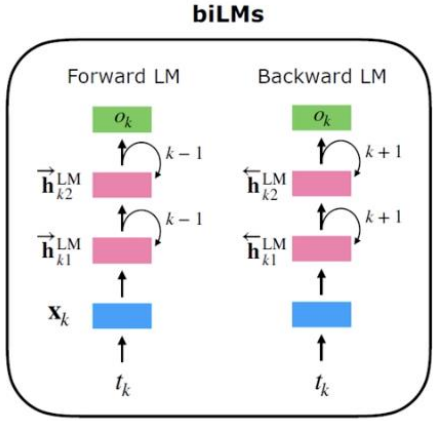


그림: 고려대학교 DSBA https://www.youtube.com/watch?v=zV8klUwH32M&t=1053s

06 우수 예습과제 Review , Announcement

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

KUBIG
DATA SCIENCE & AI

6-1. 우수 예습과제 Review

강서연 님

RNN vs LSTM

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

6-2. Week3 예,복습과제 안내, Week4 진도 안내

코드과제의 파일형식은 ipynb로, KUBIG 25-1 Github repo에 업로드 될 예정입니다!
Colab 환경에서 제작된 과제들이므로 google colab에서 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

수고하셨습니다!