# **Recurrent Neural Networks**

2017.08.19

최건호

# **INDEX**

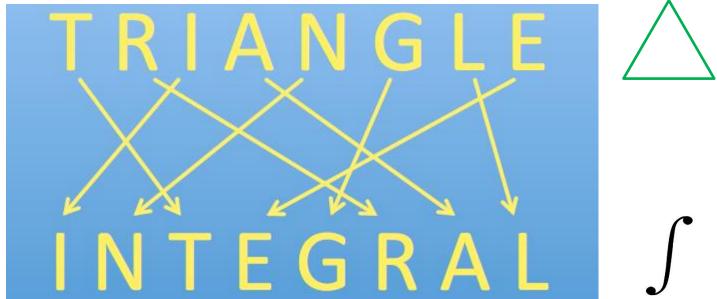
02 03 04 Why RNN Char RNN Naïve RNN LSTM & GRU Real World **RNN** Basic LSTM Preprocessing Data Naïve RNN의 GRU Word Limit without 한계 Embedding sequence 성능비교



# TRIANGLE

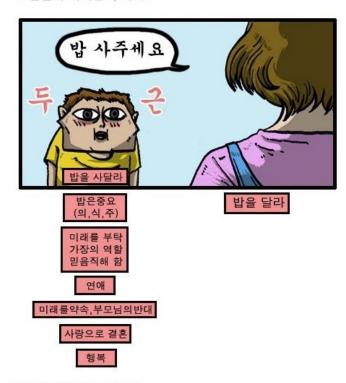








모든말에 의미를 부여하고



그 말의 의미를 고민한다

모든말에 의미를 부여하고



$$= \int E_{-j} \left[ \int_{Q_{1}} \frac{P(z,x)}{P_{1}(z_{j})} \right] f_{1}(z_{j}) dz_{j} - \sum_{i \neq j} \int_{Q_{1}} \frac{1}{P_{1}(z_{i})} \frac{1}{P_{1}(z_{j})} f_{2}(z_{j}) dz_{j} - \sum_{i \neq j} \int_{Q_{1}} \frac{1}{P_{1}(z_{i})} \frac{1}{P_{1}(z_{j})} \frac{1}{P_{1}(z_{j})} \frac{1}{P_{1}(z_{j})} \frac{1}{P_{1}(z_{j})} \int_{Q_{1}} \frac{1}{P_{1}(z_{j})} \frac{1}{P_{1}(z_{j})}$$

(출처: 웹툰 마음의 소리)





#### 흐름, 패턴, 순서



The Procter & Gamble Company

인간의 사고과정은 Sequential하게 이루어진다

인간의 사고과정은 Sequential하게 이루어진다



AI 역시 사람의 사고를 모방하기 위해서는 Sequential data를 통해 의미 있는 결론을 도출해 내야 함

인간의 사고과정은 Sequential하게 이루어진다

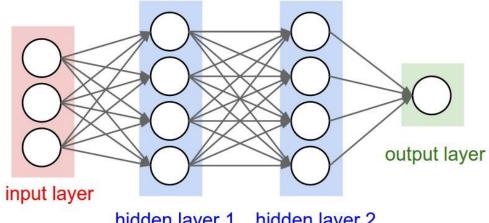


AI 역시 사람의 사고를 모방하기 위해서는 Sequential data를 통해 의미 있는 결론을 도출해 내야 함



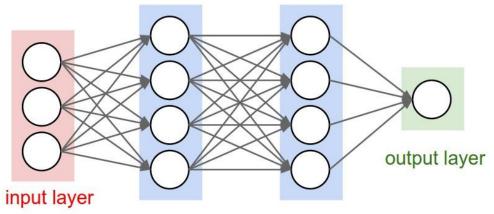
하지만 여태까지 배운 Simple Neural Network와 CNN에는 Sequence에 대한 고려가 없었음





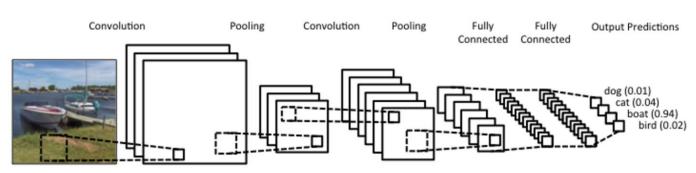
hidden layer 1 hidden layer 2

http://cs231n.github.io/convolutional-networks/

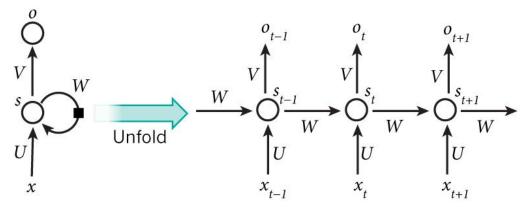


hidden layer 1 hidden layer 2

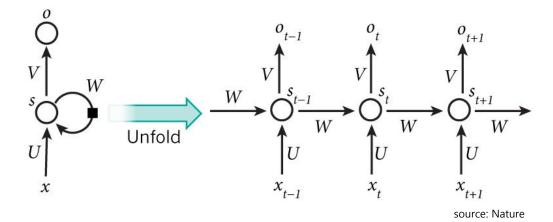
http://cs231n.github.io/convolutional-networks/

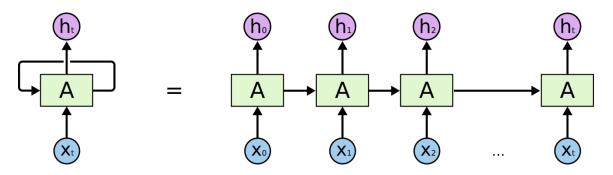


http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/



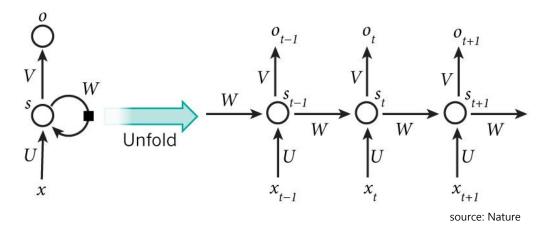
source: Nature

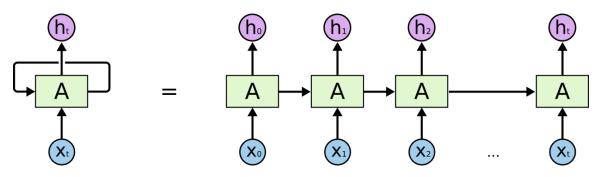




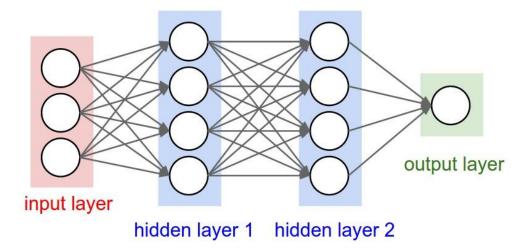
http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

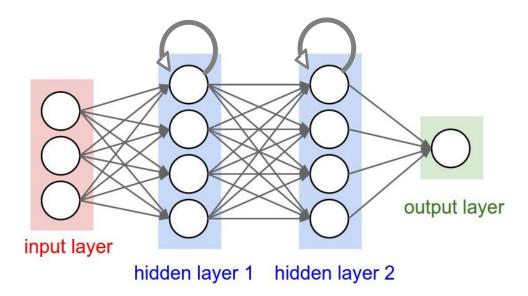
#### 뭔가 기존의 Neural Network와 연결이 잘 안됨



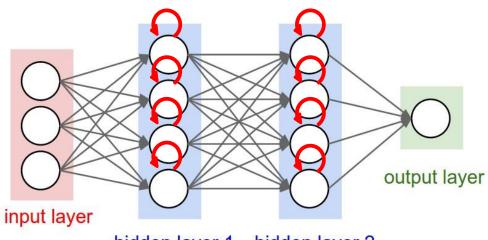


http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

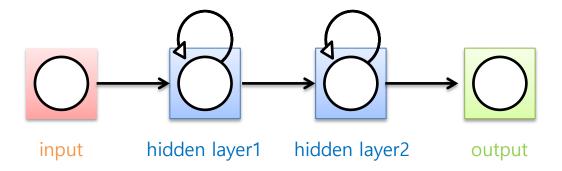


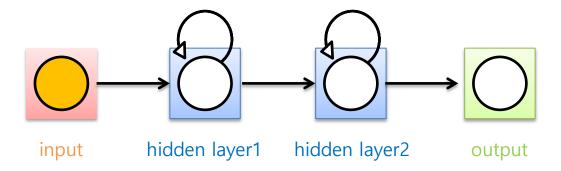


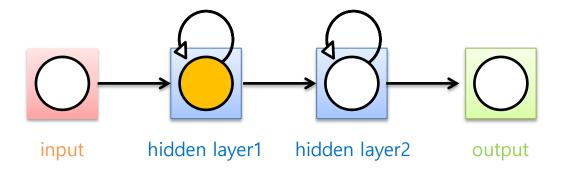
#### 정확하게 표현하면 각 노드들에 적용됨

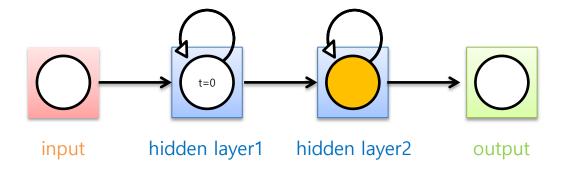


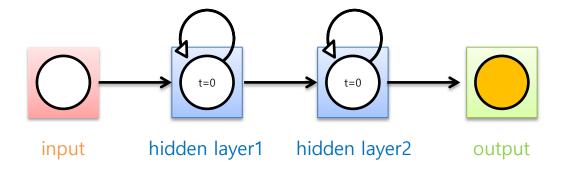
hidden layer 1 hidden layer 2

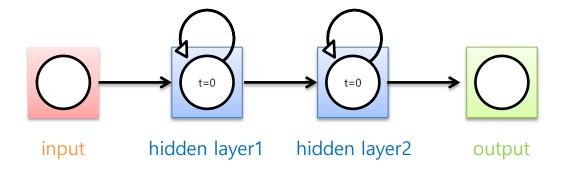


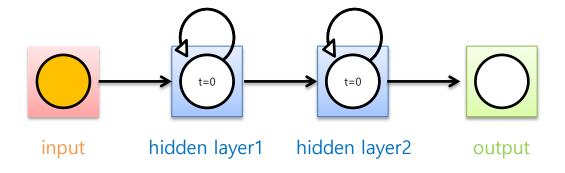


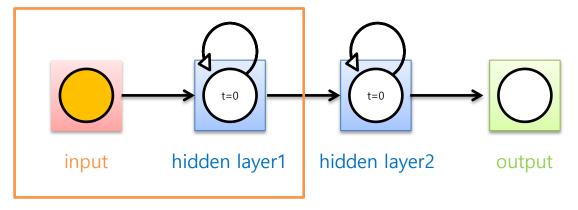


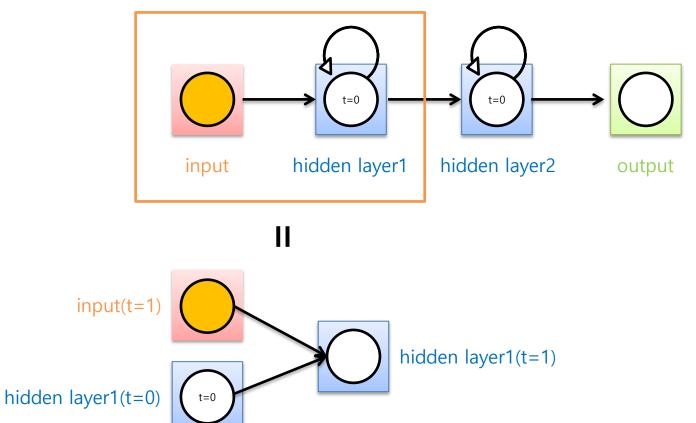


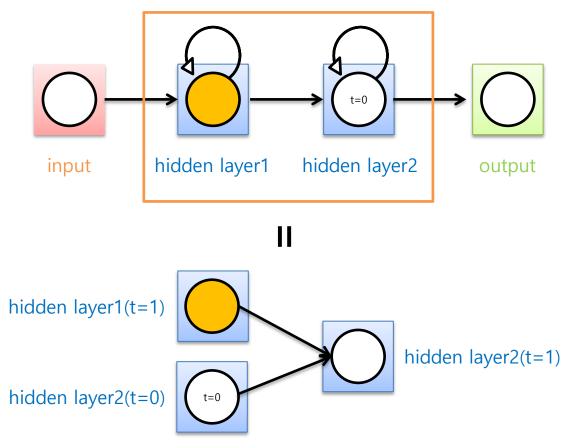


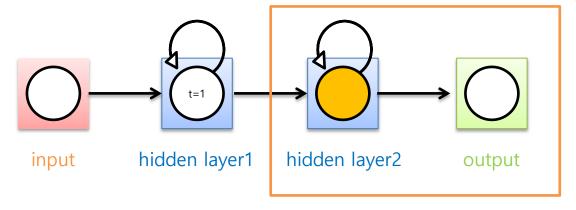


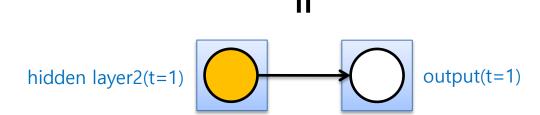


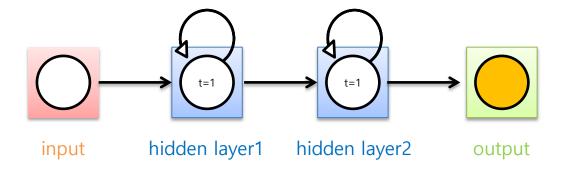


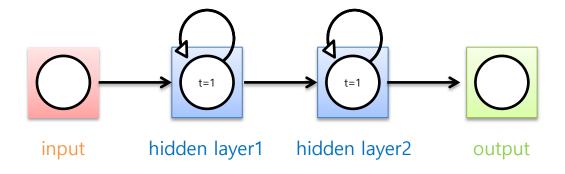


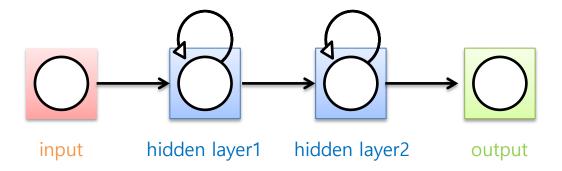


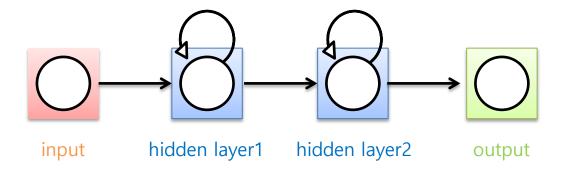




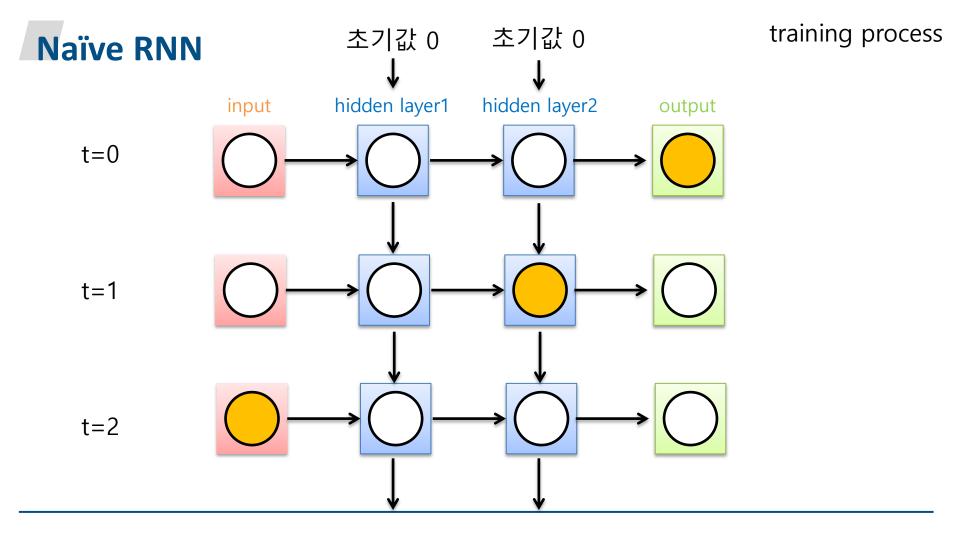


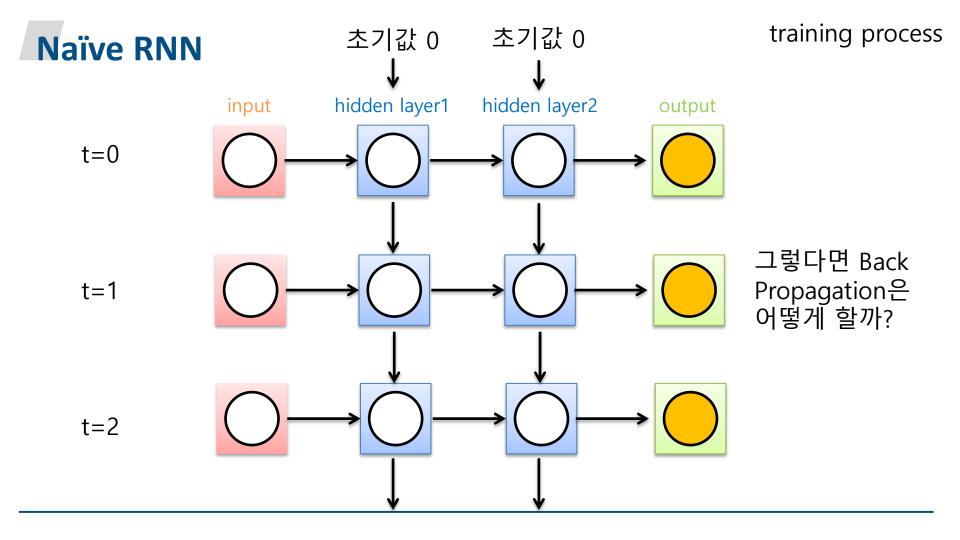


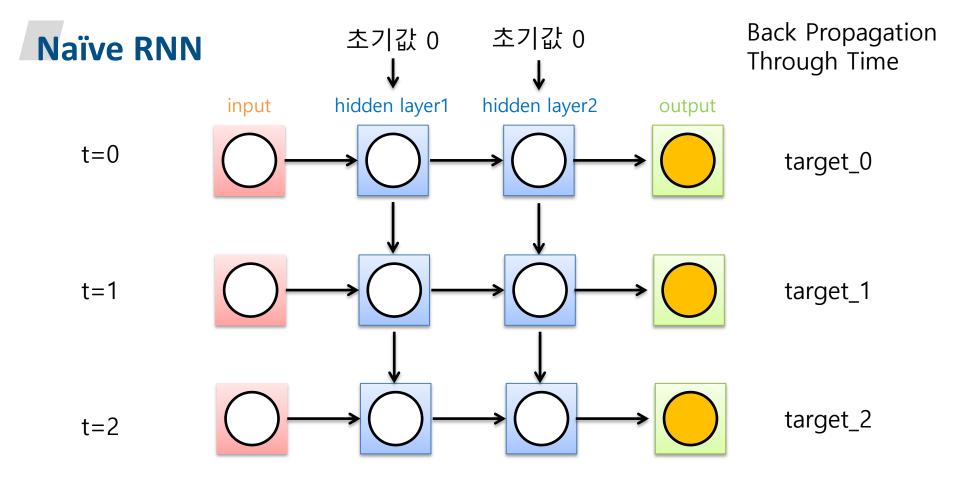


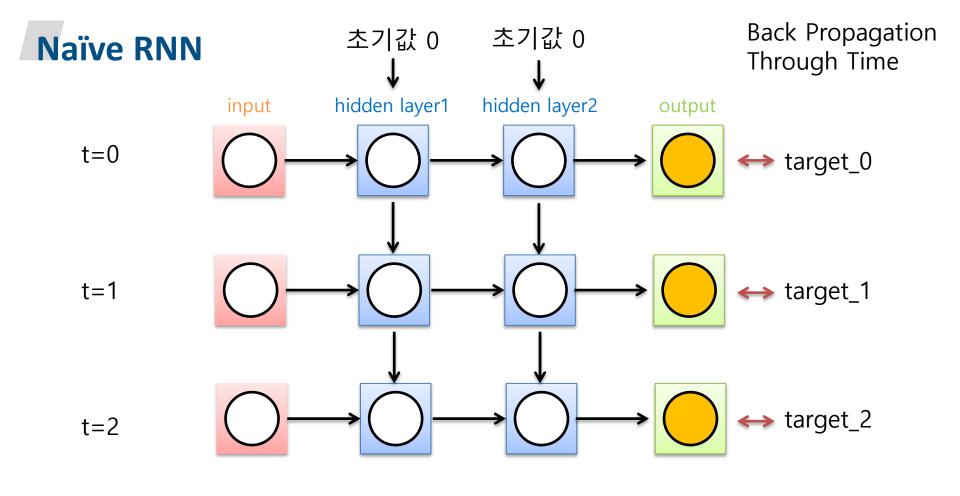


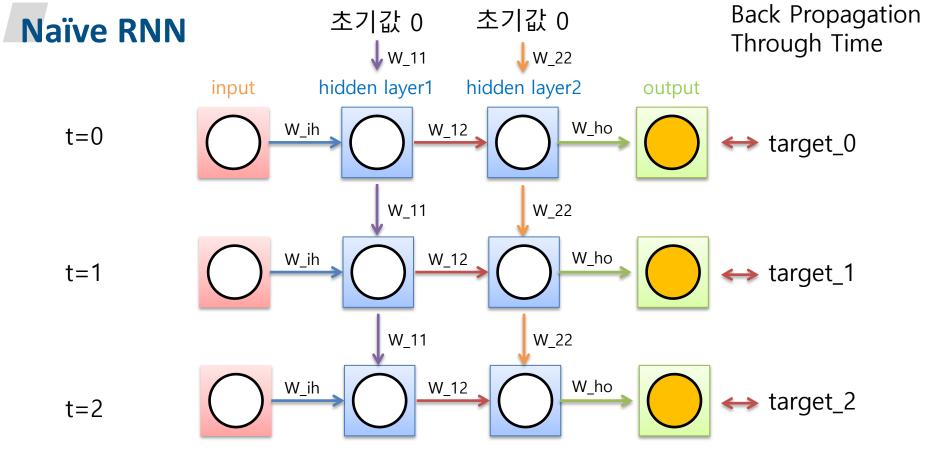
Recurrent 부분을 풀어서 다시 보면이제는 이해할 수 있음

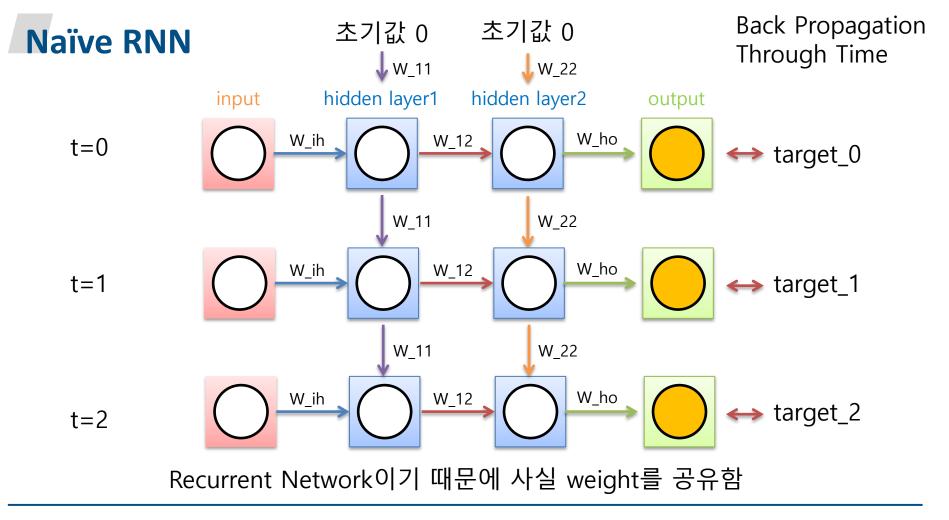


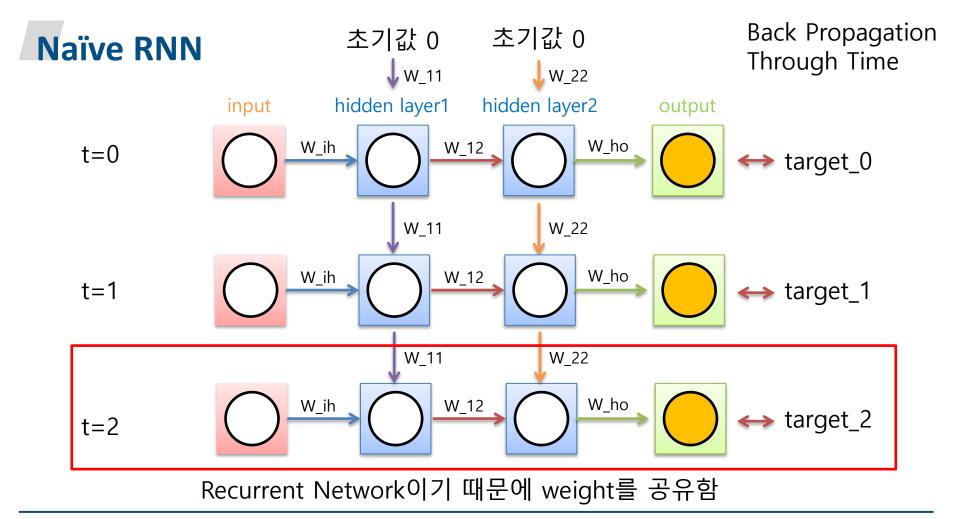


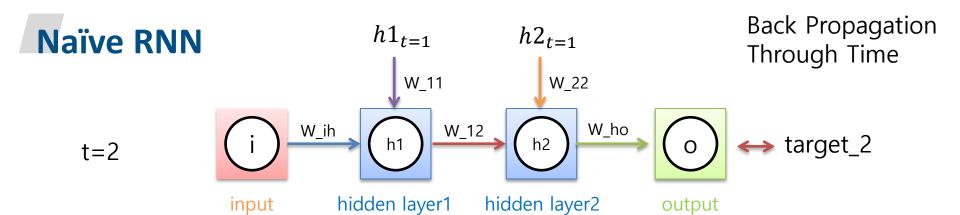


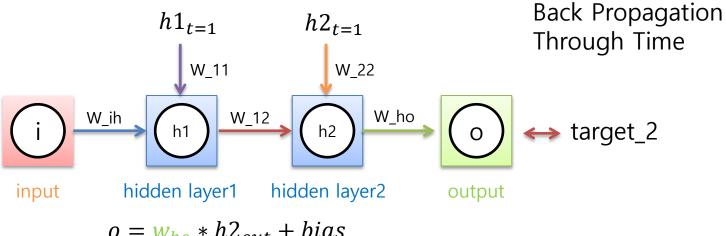






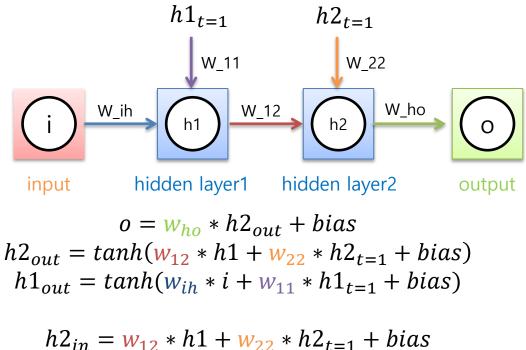






$$o = w_{ho} * h2_{out} + bias$$
  
 $h2_{out} = tanh(w_{12} * h1 + w_{22} * h2_{t=1} + bias)$   
 $h1_{out} = tanh(w_{ih} * i + w_{11} * h1_{t=1} + bias)$ 

t=2

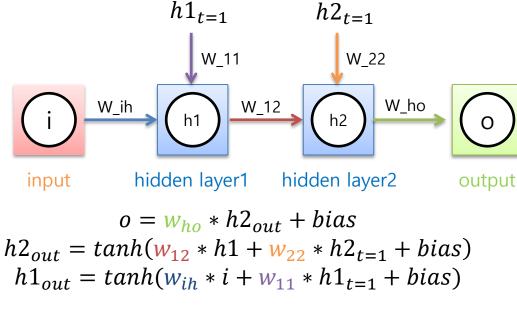


 $h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$ 

Back Propagation Through Time

→ target\_2

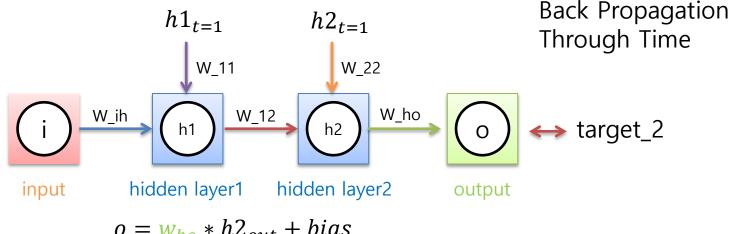
t=2



Back Propagation Through Time

$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$
  
 $h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$ 

$$\frac{\partial o}{\partial w_{22}} = \frac{\partial o}{\partial h 2_{out}} * \frac{\partial h 2_{out}}{\partial h 2_{in}} * \frac{\partial h 2_{in}}{\partial w_{22}}$$



$$o = w_{ho} * h2_{out} + bias$$

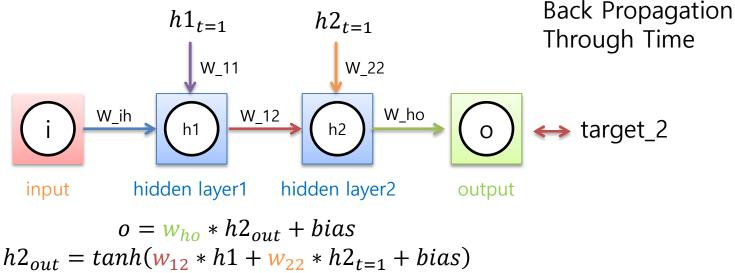
$$h2_{out} = tanh(w_{12} * h1 + w_{22} * h2_{t=1} + bias)$$

$$h1_{out} = tanh(w_{ih} * i + w_{11} * h1_{t=1} + bias)$$

$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$
  
 $h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$ 

$$\frac{\partial o}{\partial w_{22}} = \frac{\partial o}{\partial h 2_{out}} * \frac{\partial h 2_{out}}{\partial h 2_{in}} * \frac{\partial h 2_{in}}{\partial w_{22}}$$

$$h 2_{t=1}$$



$$o = w_{ho} * h2_{out} + bias$$

$$h2_{out} = tanh(w_{12} * h1 + w_{22} * h2_{t=1} + bias)$$

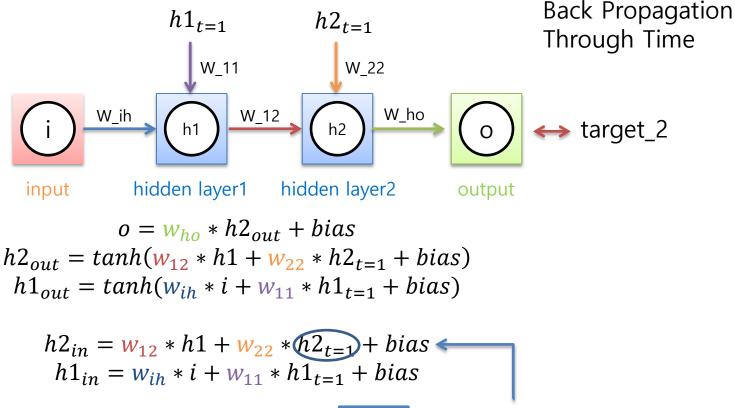
$$h1_{out} = tanh(w_{ih} * i + w_{11} * h1_{t=1} + bias)$$

$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$

$$h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$$

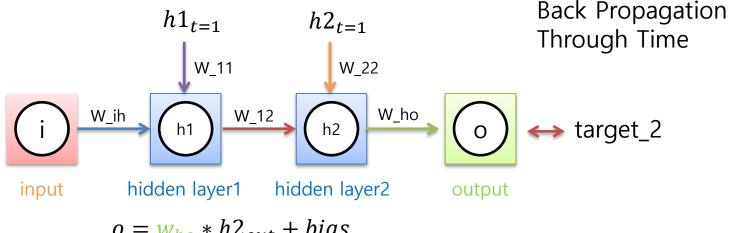
$$\frac{\partial o}{\partial w_{22}} = \frac{\partial o}{\partial h2_{out}} * \frac{\partial h2_{out}}{\partial h2_{in}} * \frac{\partial h2_{in}}{\partial w_{22}}$$

$$h2_{t=1}$$



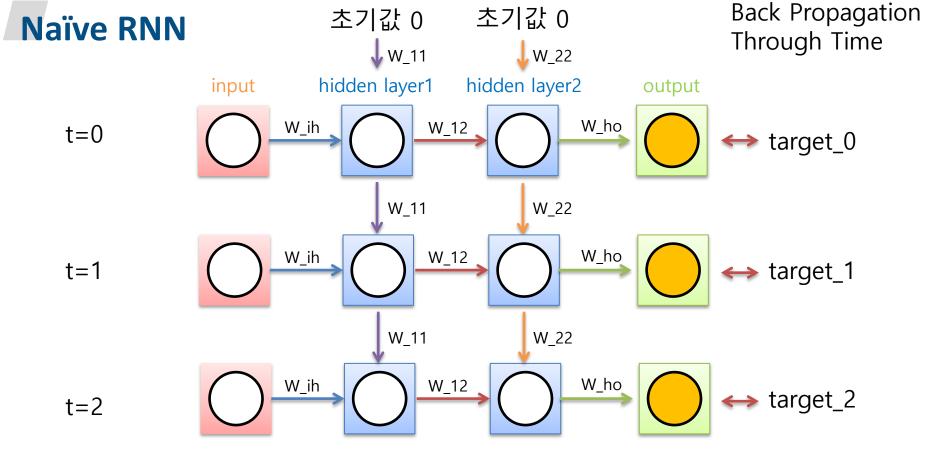
$$\frac{\partial o}{\partial w_{22}} = \frac{\partial o}{\partial h 2_{out}} * \frac{\partial h 2_{out}}{\partial h 2_{in}} * \frac{\partial h 2_{in}}{\partial w_{22}}$$

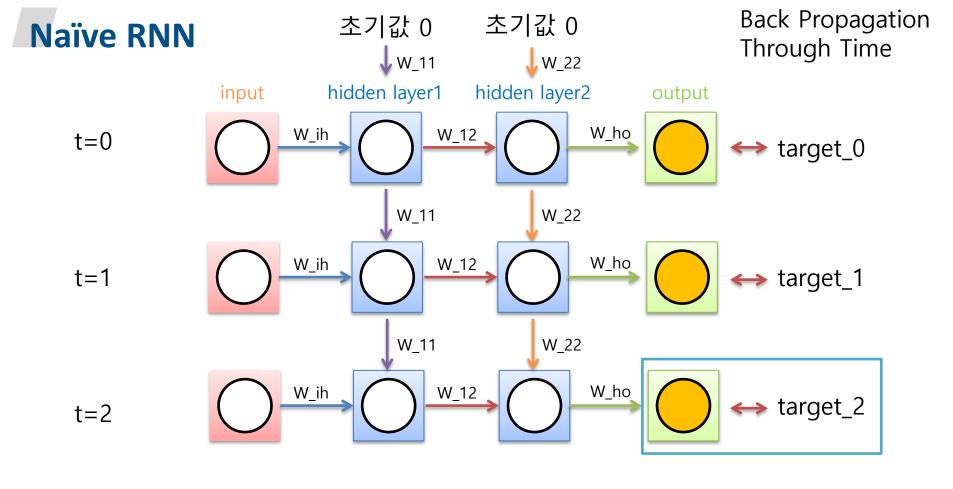
$$h 2_{t=0}$$

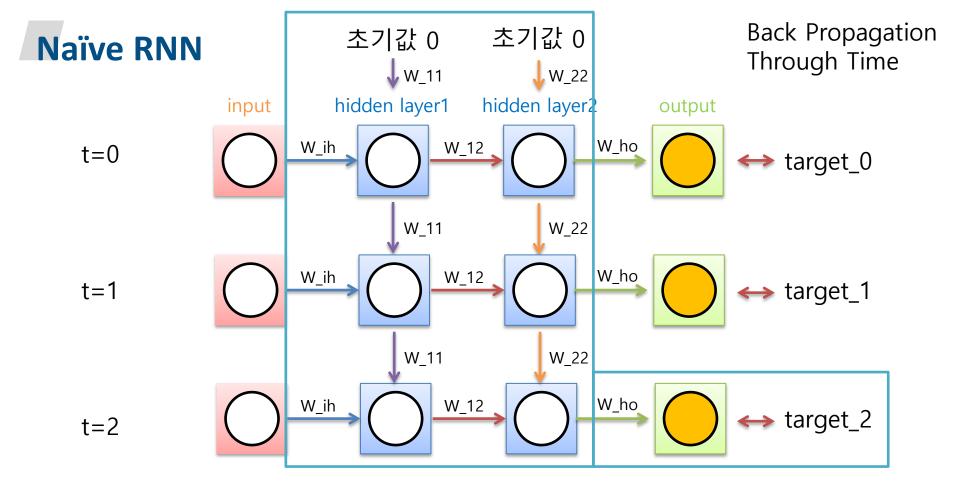


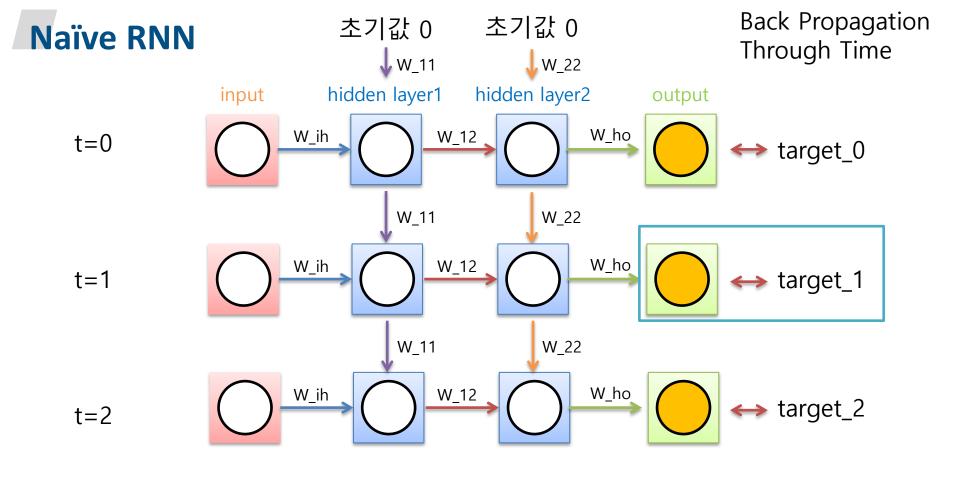
$$\begin{aligned} o &= w_{ho} * h2_{out} + bias \\ h2_{out} &= tanh(w_{12} * h1 + w_{22} * h2_{t=1} + bias) \\ h1_{out} &= tanh(w_{ih} * i + w_{11} * h1_{t=1} + bias) \end{aligned}$$

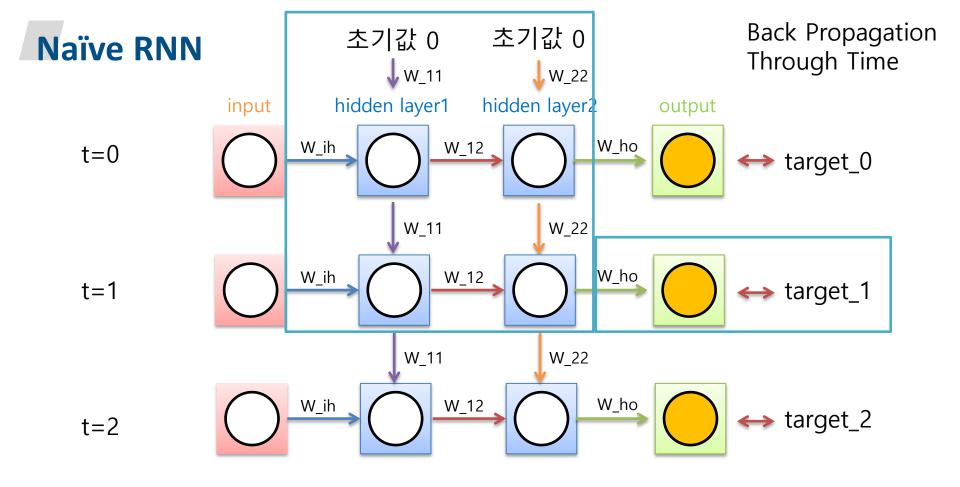
$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$
 결국 계산하려면  $h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$   $t=0$ 까지 가야 함  $\frac{\partial o}{\partial w_{22}} = \frac{\partial o}{\partial h2_{out}} * \frac{\partial h2_{out}}{\partial h2_{in}} * \frac{\partial h2_{in}}{\partial w_{22}}$ 

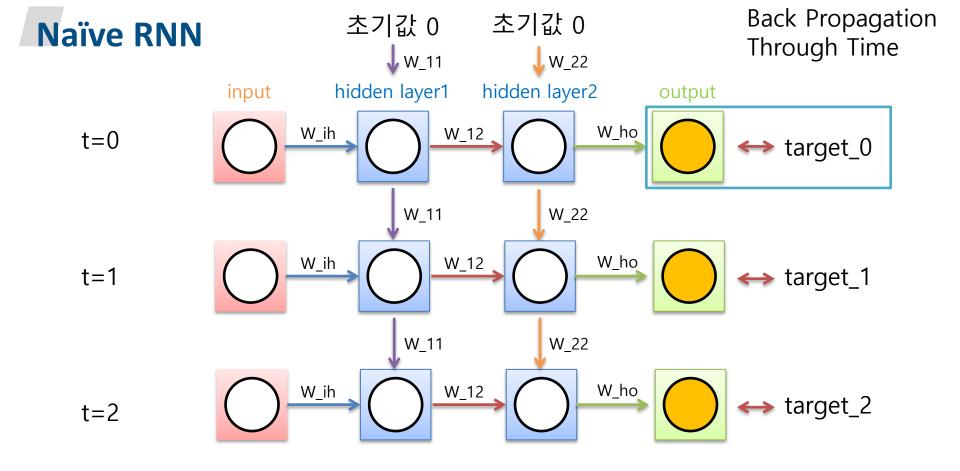


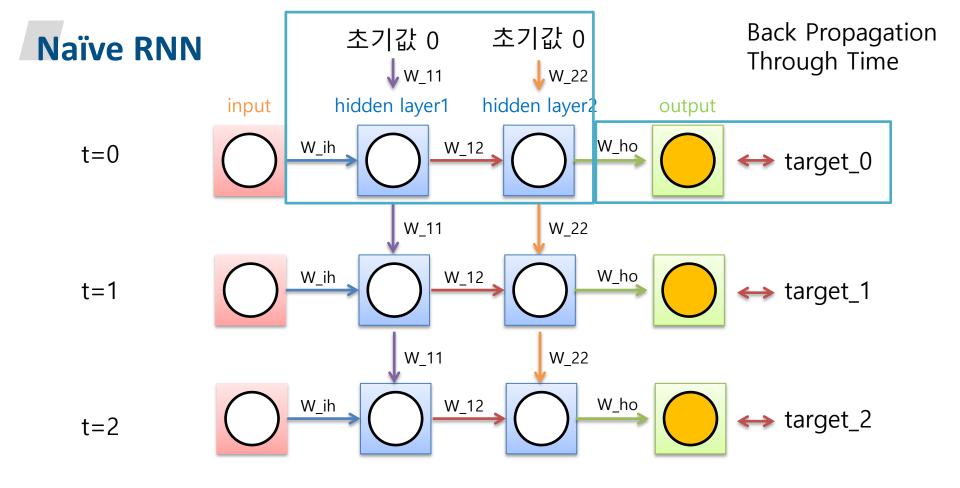


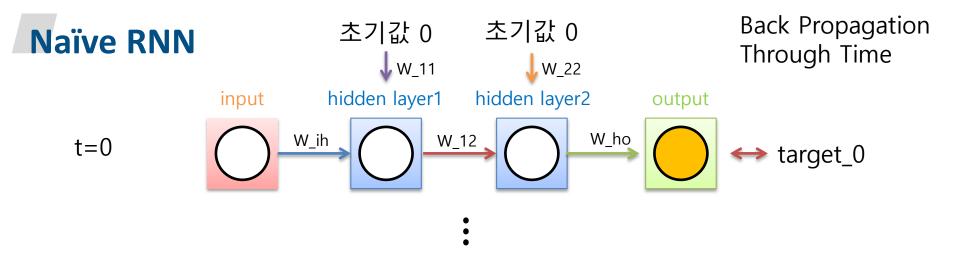




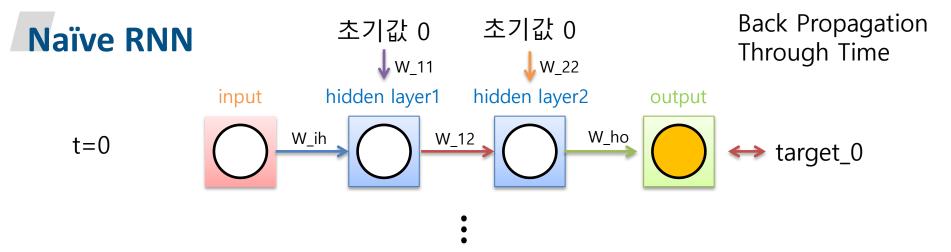






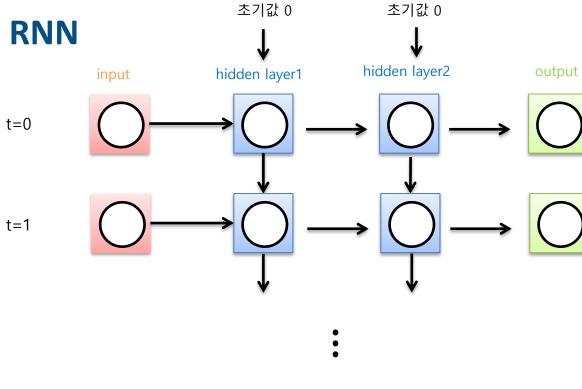


t의 범위에 따라 weight 값들이 반복적으로 gradient가 계산되는데 이를 다 더해놨다가 한번에 업데이트 함



t의 범위에 따라 weight 값들이 반복적으로 gradient가 계산되는데 이를 다 더해놨다가 한번에 업데이트 함

$$\frac{\partial Loss}{\partial w} = \sum_{t} \frac{\partial Loss}{\partial w}$$

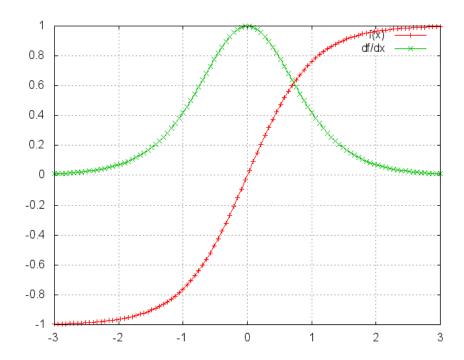


$$t=15$$
  $\longrightarrow$   $\bigcirc$   $\longrightarrow$   $\bigcirc$ 

# 초기값 0 초기값 0 **Naïve RNN** hidden layer2 output hidden layer1 input t=0t=1 이렇게 t의 범위가 늘어나면 성능 이 좋지 않던데 이유가 뭘까? t=15

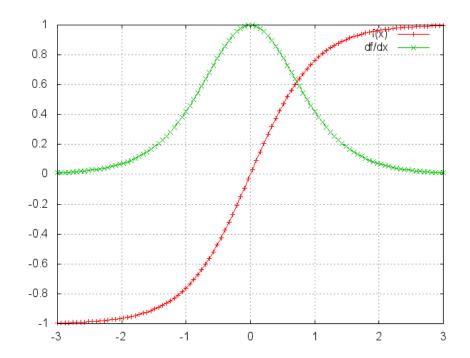
Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일 어나는데, 이를 미분해보면 오른 쪽과 같음

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일 어나는데, 이를 미분해보면 오른 쪽과 같음



Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일 어나는데, 이를 미분해보면 오른 쪽과 같음

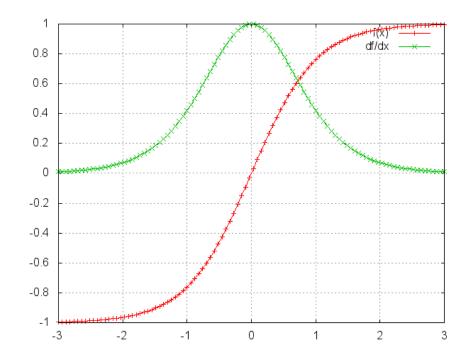
0과 1 사이의 값을 계속 곱하다 보면 gradient가 사라지고 결과 적으로 해당 loss에 대해 학습이 안됨



Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일 어나는데, 이를 미분해보면 오른 쪽과 같음

0과 1 사이의 값을 계속 곱하다 보면 gradient가 사라지고 결과 적으로 해당 loss에 대해 학습이 아됨

어떻게 해야 할까?



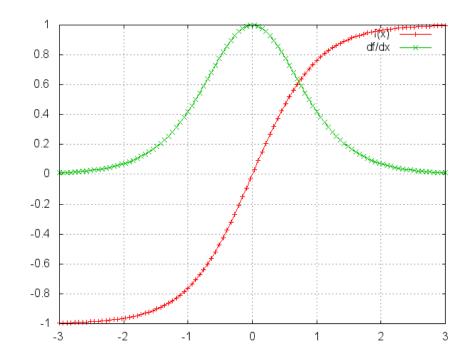
#### **Naïve RNN**

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일 어나는데, 이를 미분해보면 오른 쪽과 같음

0과 1 사이의 값을 계속 곱하다 보면 gradient가 사라지고 결과 적으로 해당 loss에 대해 학습이 아됨

어떻게 해야 할까?

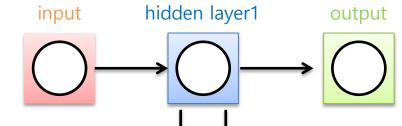
장기기억을 추가하자!



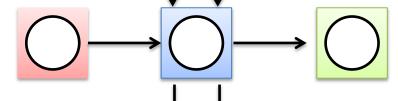
Hidden state Cell state



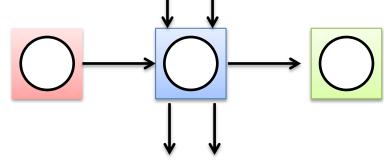








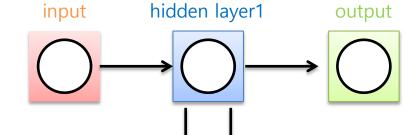




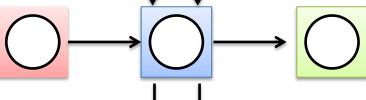
Hidden state Cell state → 중요 ▼ 전

중요한 정보는 쭉 전달하는 부분

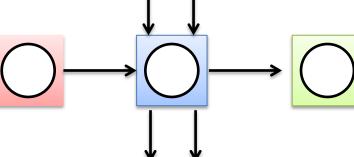




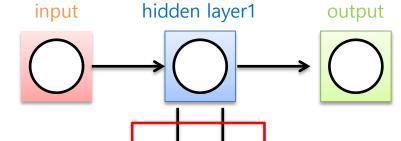




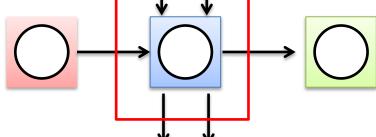
t=2



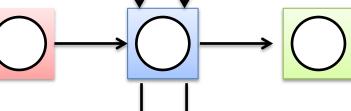


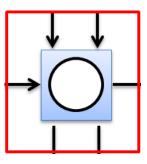


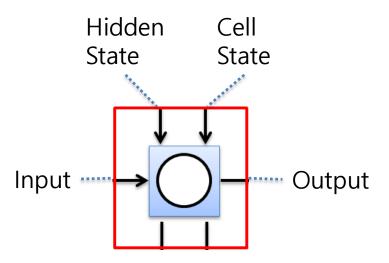
t=1

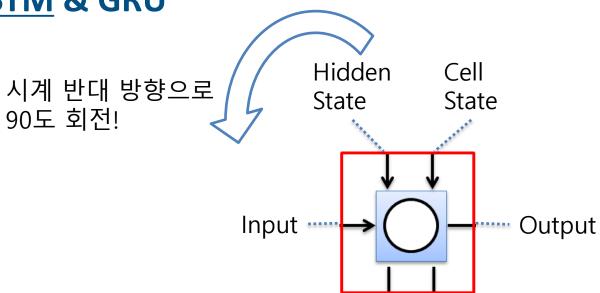


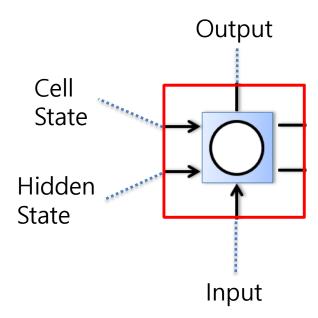
t=2

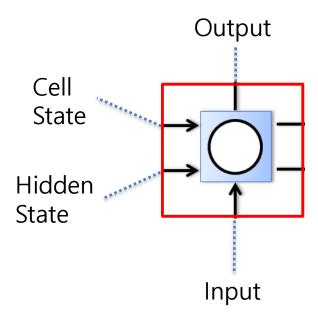


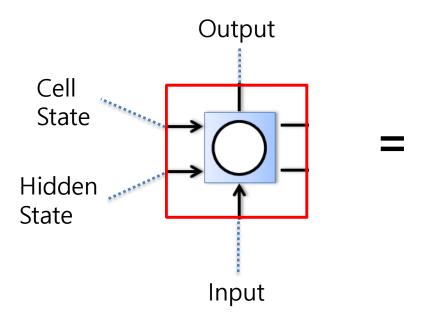


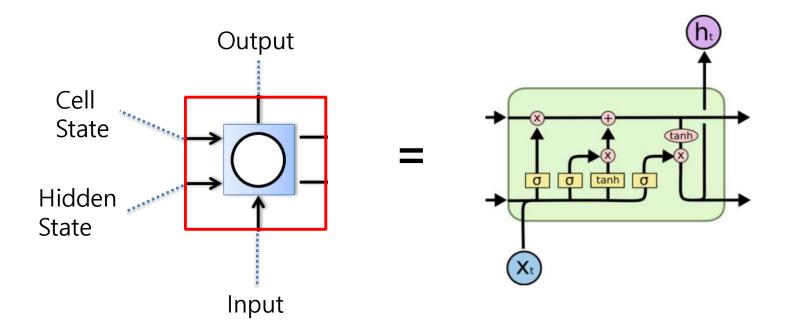


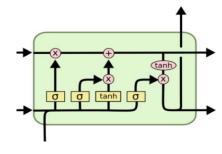


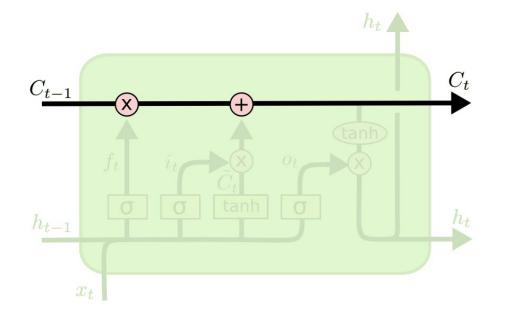




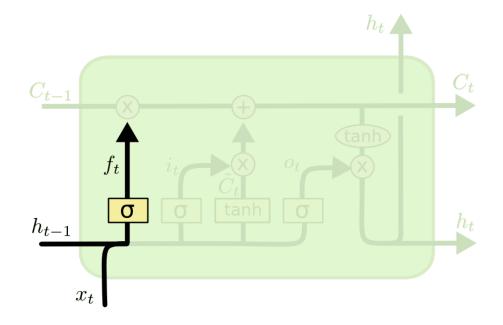




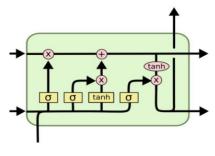




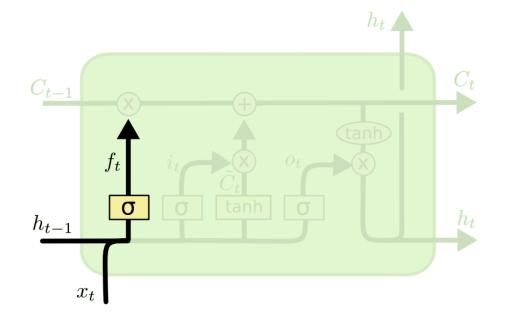
Cell state는 중요 정보를 그대로 쭉 전달하는 역할



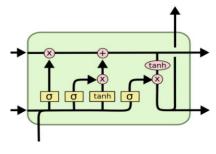
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$



#### "Forget Gate Layer"

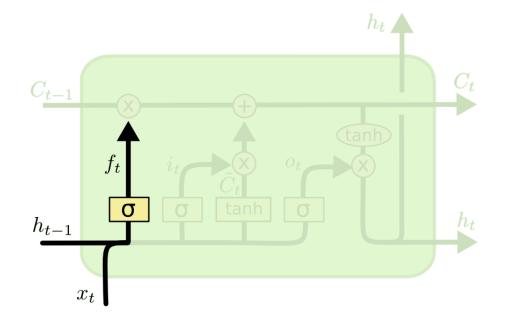


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

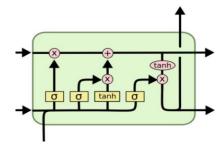


#### "Forget Gate Layer"

Input과 이전 hidden state를 기준으로 이전의 Cell State를 얼마나 계속 전달할 것인지 결정



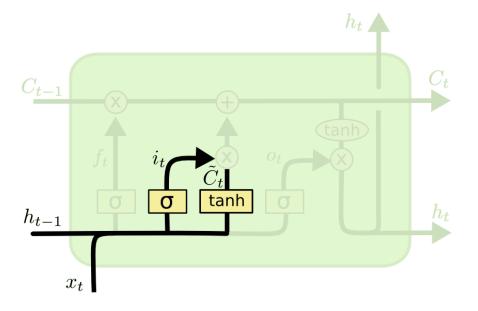
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$



#### "Forget Gate Layer"

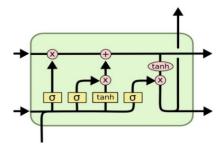
Input과 이전 hidden state를 기준으로 이전의 Cell State를 얼마나 계속 전달할 것인지 결정

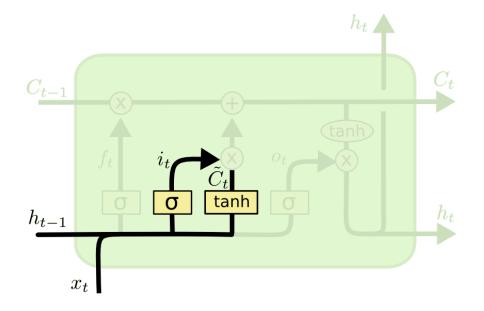
Activation func.이 sigmoid이기 때문에 0~1로 나오는데 0이면 완전 잊고 1이면 완전 그대로 넘기는 것



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

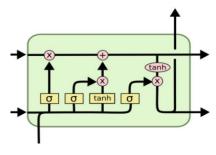
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C$$



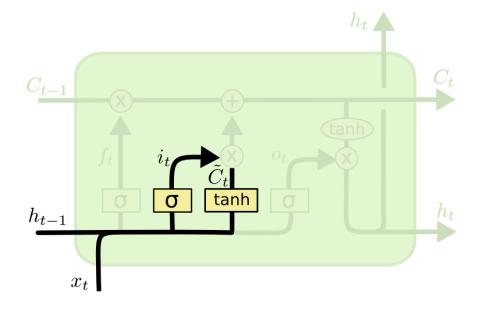


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C$$

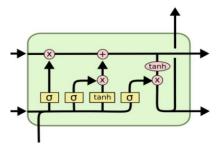


"Input Gate"



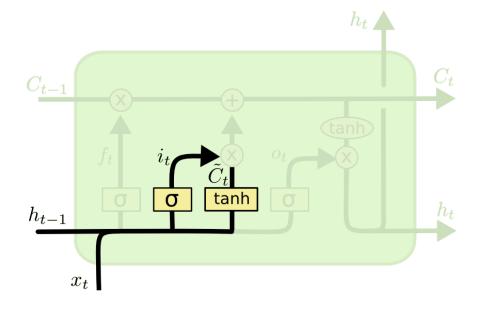
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C$$



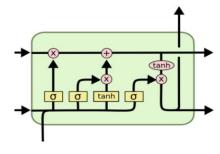
#### "Input Gate"

 $i_t$ 는 hidden state와 input을 통해 Cell state update 비중을 정함



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

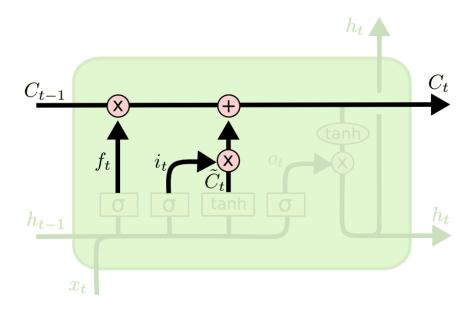
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C$$



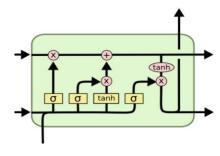
#### "Input Gate"

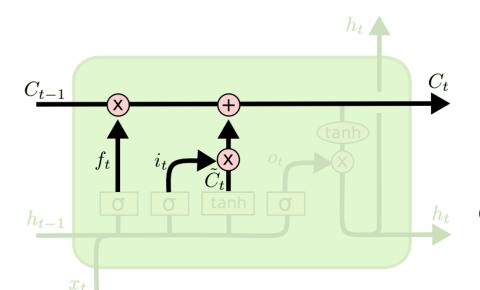
 $i_t$ 는 hidden state와 input을 통해 Cell state update 비중을 정함

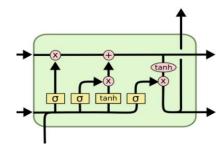
 $\widetilde{C}_t$ 는 cell state로 update될 정보를 생성



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

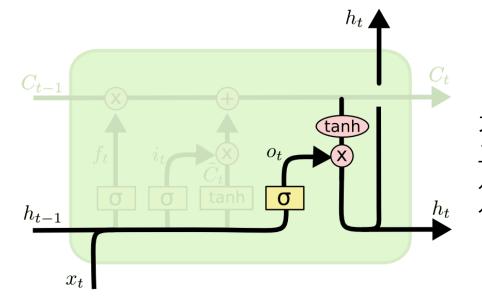




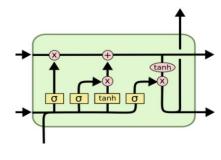


앞서 계산했던 얼마나 잊거나 기억할 것인지, 업데이트는 얼마의 비중으로 얼마나 업데이트 할 것인지 값들을 Cell state에 적용하는 단계

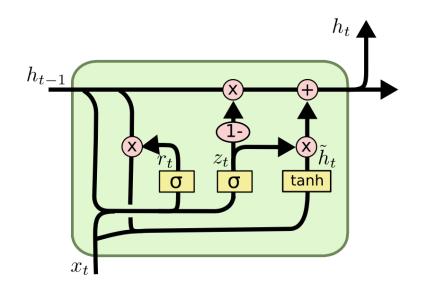
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh (C_t)$$



기존 hidden state, input으로  $o_t$ 를 정하고, 업데이트 된 cell state를 tanh에 통과시킨 비중을 곱해 새로운 hidden state를 생성

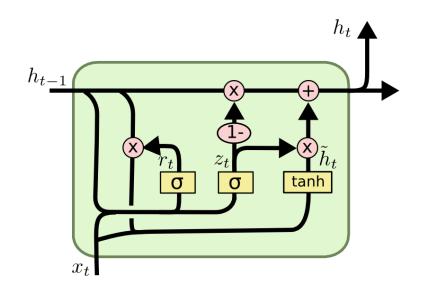


$$z_{t} = \sigma (W_{z} \cdot [h_{t-1}, x_{t}])$$

$$r_{t} = \sigma (W_{r} \cdot [h_{t-1}, x_{t}])$$

$$\tilde{h}_{t} = \tanh (W \cdot [r_{t} * h_{t-1}, x_{t}])$$

$$h_{t} = (1 - z_{t}) * h_{t-1} + z_{t} * \tilde{h}_{t}$$



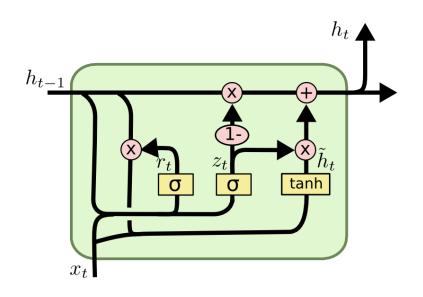
$$z_{t} = \sigma (W_{z} \cdot [h_{t-1}, x_{t}])$$

$$r_{t} = \sigma (W_{r} \cdot [h_{t-1}, x_{t}])$$

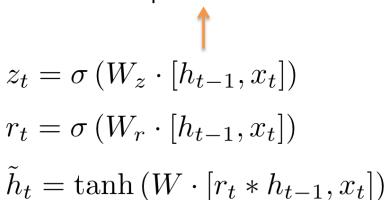
$$\tilde{h}_{t} = \tanh (W \cdot [r_{t} * h_{t-1}, x_{t}])$$

$$h_{t} = (1 - z_{t}) * h_{t-1} + z_{t} * \tilde{h}_{t}$$

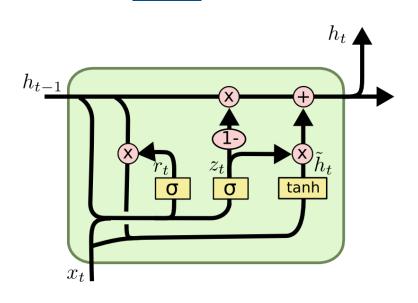
Forget gate와 Input gate를 Update gate 하나로 합침. Cell state와 Hidden state도 하나로 합침

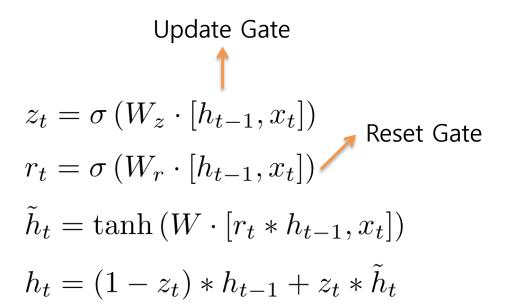


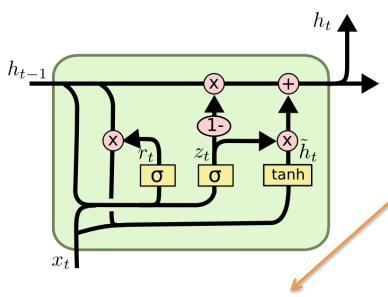
**Update Gate** 



$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

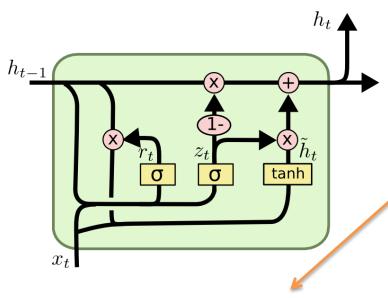






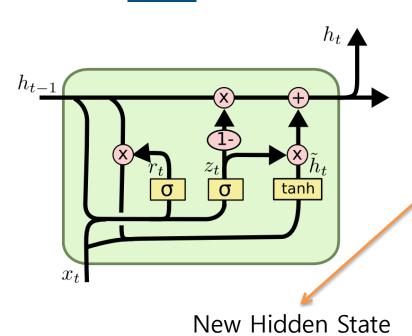
New Hidden State

**Update Gate**  $z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$ Reset Gate  $r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$  $\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$  $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$ 



New Hidden State

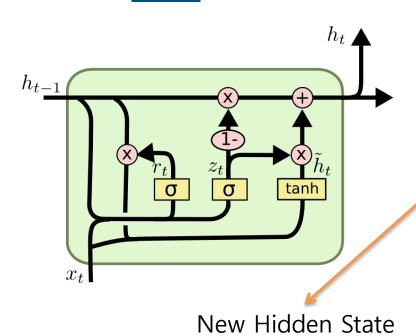
**Update Gate**  $z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$ Reset Gate  $r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$  $\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$  $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$ 



Update Gate

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
 Reset Gate 
$$\underline{r_t} = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
 
$$\tilde{h}_t = \tanh\left(W \cdot [\underline{r_t} * h_{t-1}, x_t]\right)$$
 
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

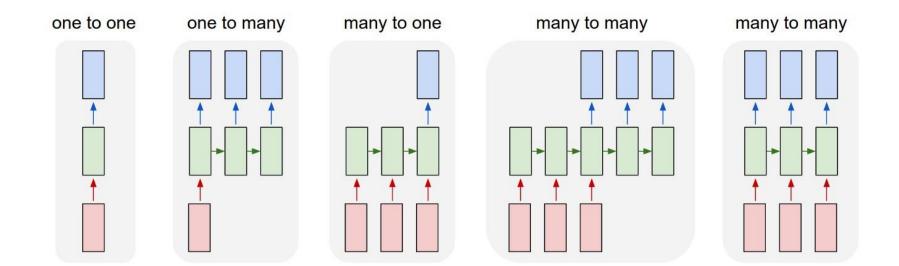
 $z_t$ 에 의해  $h_{t-1}$ 과  $\widetilde{h_t}$  의 weighted sum으로  $h_t$ 를 생성한다.

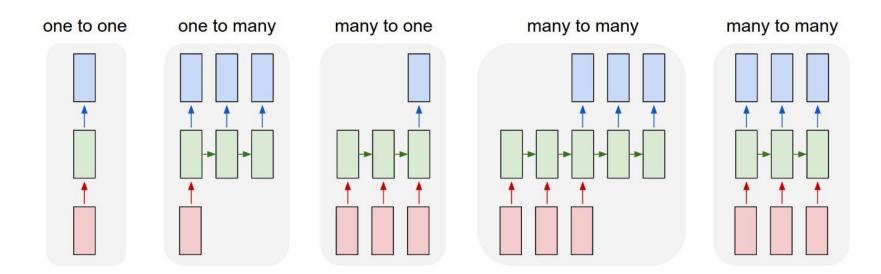


Update Gate

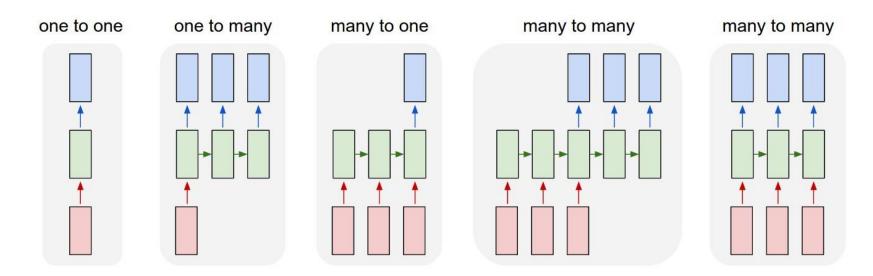
$$z_{t} = \sigma\left(W_{z} \cdot [h_{t-1}, x_{t}]\right)$$
 Reset Gate 
$$\underline{r_{t}} = \sigma\left(W_{r} \cdot [h_{t-1}, x_{t}]\right)$$
 
$$\tilde{h}_{t} = \tanh\left(W \cdot [\underline{r_{t}} * h_{t-1}, x_{t}]\right)$$
 
$$h_{t} = (1 - z_{t}) * h_{t-1} + \underline{z_{t}} * \tilde{h}_{t}$$

 $z_t$ 에 의해  $h_{t-1}$ 과  $\widetilde{h_t}$  의 weighted sum으로  $h_t$ 를 생성한다.

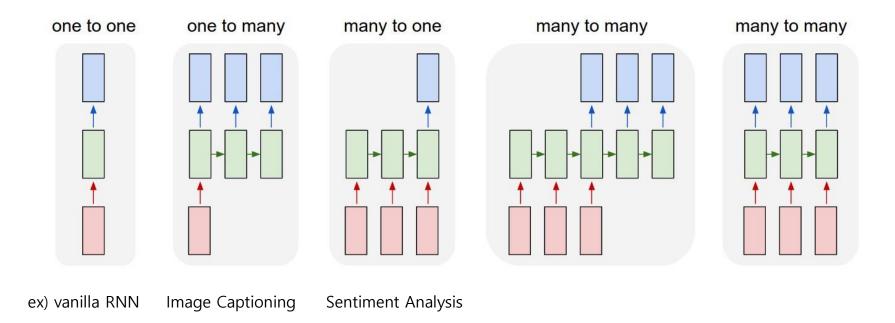


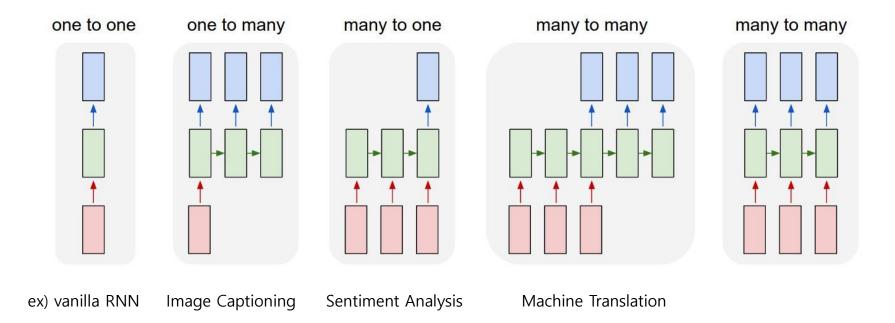


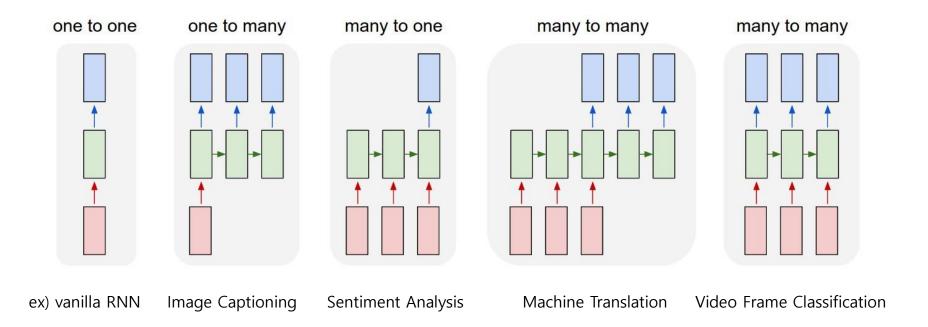
ex) vanilla RNN



ex) vanilla RNN Image Captioning







"hello pytorch"

"hello pytorch" --- 우선 컴퓨터가 알아볼 수 있게 바꿔야 함

"hello pytorch" --- 우선 컴퓨터가 알아볼 수 있게 바꿔야 함

"abcdefghijklmnopqrstuvwxyz"

"hello pytorch" --- 우선 컴퓨터가 알아볼 수 있게 바꿔야 함

"abcdefghijklmnopqrstuvwxyz " ----- one-hot vector

"hello pytorch" --- 우선 컴퓨터가 알아볼 수 있게 바꿔야 함

"abcdefghijklmnopqrstuvwxyz" ——— one-hot vector ex) [0 0 0 0 1 0 0 0]

"hello pytorch" --- 우선 컴퓨터가 알아볼 수 있게 바꿔야 함

"abcdefghijklmnopqrstuvwxyz" ——— one-hot vector ex) [0 0 0 0 1 0 0 0]

$$a = [1 \ 0 \ 0 \ 0 \ \dots \ 0]$$

$$b = [0 \ 1 \ 0 \ 0 \ \dots \ 0]$$

$$c = [0 \ 0 \ 1 \ 0 \ ... \ 0]$$

•

"hello pytorch" --- 우선 컴퓨터가 알아볼 수 있게 바꿔야 함

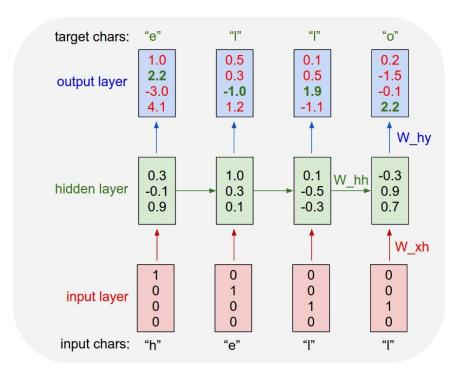
"abcdefghijklmnopqrstuvwxyz"

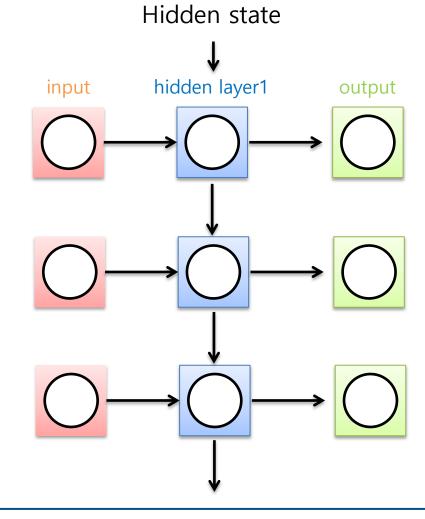
$$27$$

$$a = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \end{bmatrix}$$

$$c = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$
:

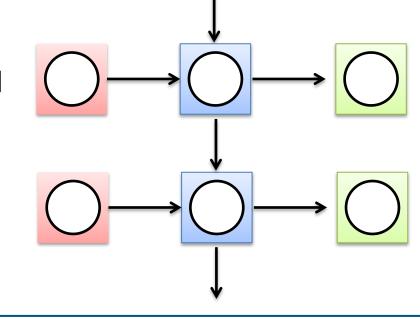




$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$

$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

$$I = [0 \ 0 \ 0 \ ...1... \ 0]$$



Hidden state

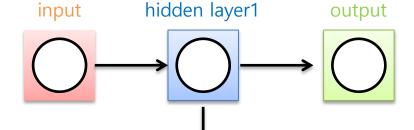
hidden layer1

output

input



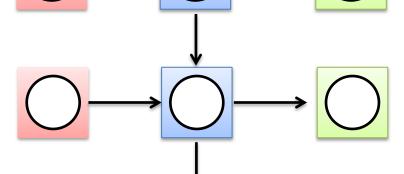
$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$



$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

 $I = [0 \ 0 \ 0 \ ...1... \ 0]$ 



$$I = [0 \ 0 \ 0 \ ...1... \ 0]$$

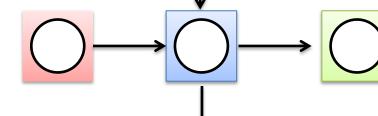
$$o = [0 \ 0 \ 0 \ ...1... \ 0]$$



$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$

$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

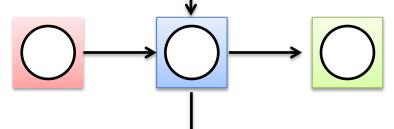
$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$



$$I = [0 \ 0 \ 0 \ ...1... \ 0]$$

$$I = [0 \ 0 \ 0 \ ...1... \ 0]$$





$$o = [0 \ 0 \ 0 \ ...1... \ 0]$$

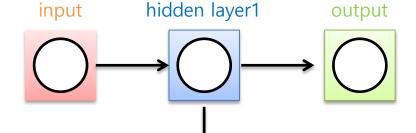
"hello pytorch"

Hidden state

**Training** 

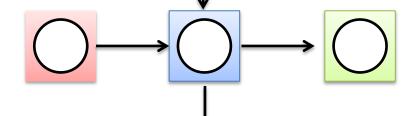


$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$



$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

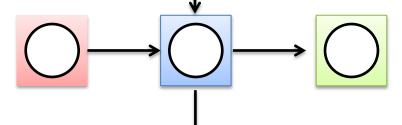
$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$



$$I = [0 \ 0 \ 0 \ ...1... \ 0]$$

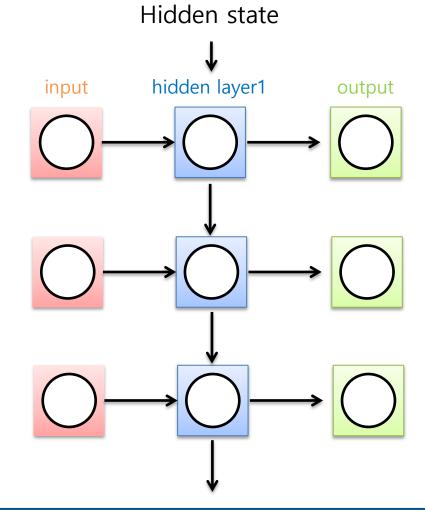
$$I = [0 \ 0 \ 0 \ ... 1... \ 0]$$



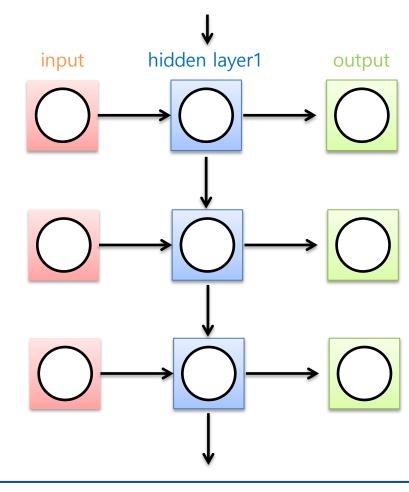


$$o = [0 \ 0 \ 0 \ ...1... \ 0]$$

"hello pytorch"

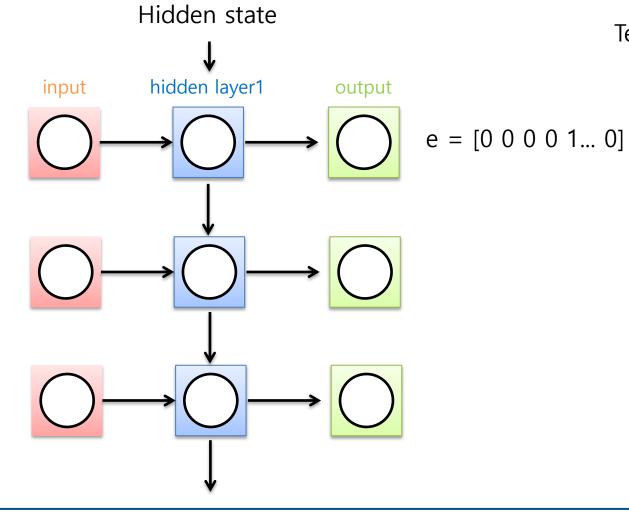


$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$



Hidden state

$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$

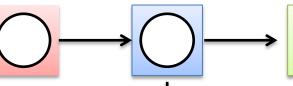








Hidden state

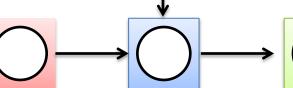




$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$

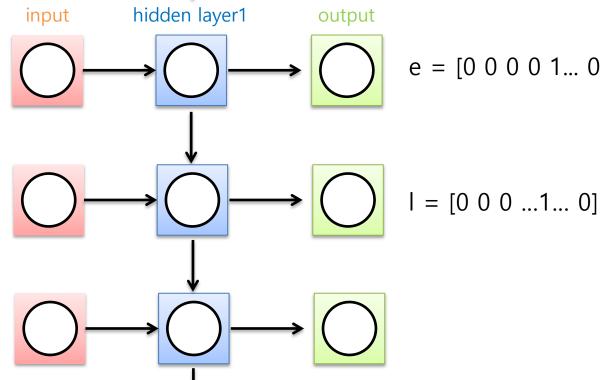




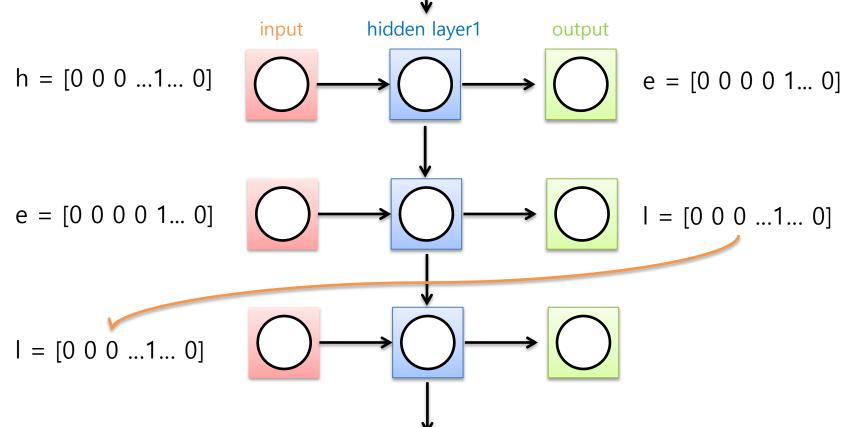


$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$

$$e = [0 \ 0 \ 0 \ 1... \ 0]$$



Hidden state

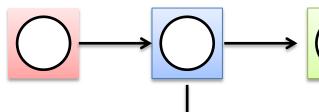


hidden layer1

Hidden state

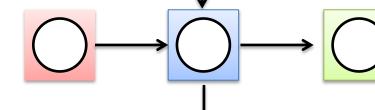
output

$$h = [0 \ 0 \ 0 \ ...1... \ 0]$$



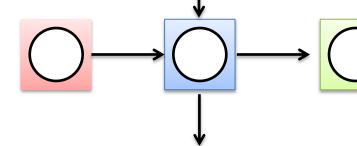
 $e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$ 

$$e = [0 \ 0 \ 0 \ 0 \ 1... \ 0]$$



 $I = [0 \ 0 \ 0 \ ...1... \ 0]$ 

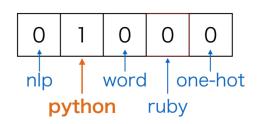
$$I = [0 \ 0 \ 0 \ ...1... \ 0]$$



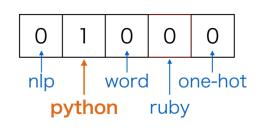
 $o = [0 \ 0 \ 0 \ ...1... \ 0]$ 

그런데 one-hot vector가 효율적인 방법일까?

그런데 one-hot vector가 효율적인 방법일까?



그런데 one-hot vector가 효율적인 방법일까?

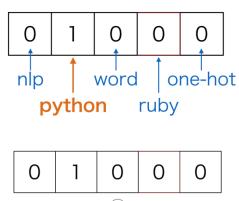




element-wise product is zero vector. Thus, Inner product is zero.

그런데 one-hot vector가 효율적인 방법일까?

 의미를 제대로 담지 못함. 또한 벡터 간의 연산도 무의미

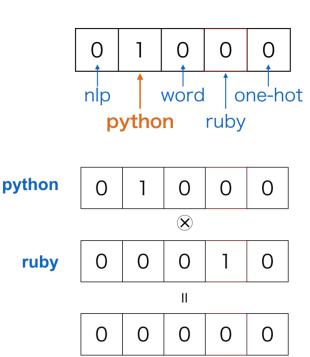




element-wise product is zero vector. Thus, Inner product is zero.

그런데 one-hot vector가 효율적인 방법일까?

- 의미를 제대로 담지 못함. 또한 벡터 간의 연산도 무의미
- 2. 만약 word를 one-hot으로 표현하려 한다면 길이도 무한정 늘어나고 새로 운 단어가 들어오면 전체 길이가 늘어 나서 기존의 모델이 무의미



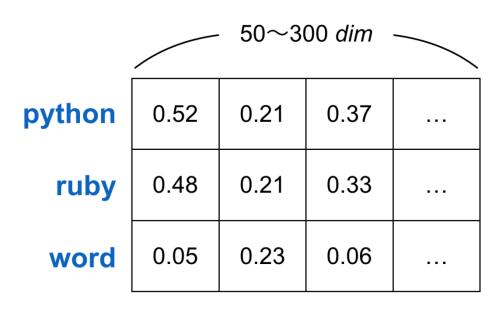
element-wise product is zero vector. Thus, Inner product is zero.

Word Embedding

## Word Embedding

50~300 dim								
python	0.52	0.21	0.37					
ruby	0.48	0.21	0.33					
word	0.05	0.23	0.06					

#### Word Embedding



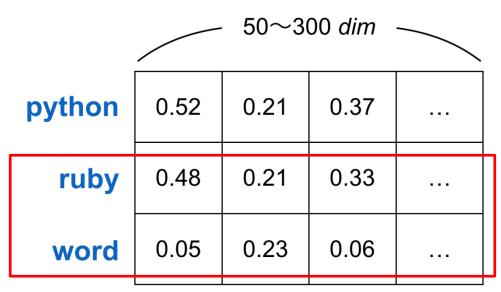
일정한 길이의 벡터에 continuous value로 단어들을 표현

#### Word Embedding

		50~300 dim						
python	0.52	0.21	0.37					
ruby	0.48	0.21	0.33					
word	0.05	0.23	0.06					

일정한 길이의 벡터에 continuous value로 단어들을 표현

#### Word Embedding



일정한 길이의 벡터에 continuous value로 단어들을 표현

semantic한 거리 차이가 있음

#### Word Embedding

50~300 dim								
python	0.52	0.21	0.37					
ruby	0.48	0.21	0.33					
word	0.05	0.23	0.06					

일정한 길이의 벡터에 continuous value로 단어들을 표현

embedding 하는 방법엔 여러 종류가 있지만 이번 예시에서는 embedding 자체도 학습을 통해 loss를 최소화하는 방향으로 학습되도록 설계

"The quick brown fox jumps over the lazy dog"

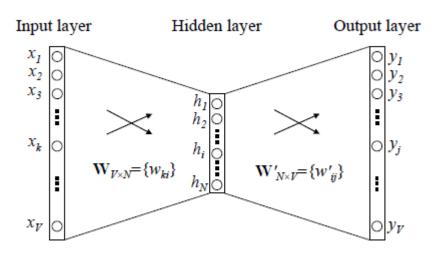
The	[1	0	0	0	0	0	0	0]
quick	[0	1	0	0	0	0	0	0]
brown	[0	0	1	0	0	0	0	0]
fox	[0	0	0	1	0	0	0	0]
jump	[0	0	0	0	1	0	0	0]
over	[0	0	0	0	0	1	0	0]
lazy	[0	0	0	0	0	0	1	0]
dog	[0	0	0	0	0	0	0	1]

"The quick brown fox jumps over the lazy dog"

 $[1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$ The quick  $[0\ 1\ 0\ 0\ 0\ 0\ 0]$  $[0\ 0\ 1\ 0\ 0\ 0\ 0]$ brown [0 0 0 1 0 0 0 0] fox [0 0 0 0 1 0 0 0] jump  $[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]$ over lazy  $[0\ 0\ 0\ 0\ 0\ 1\ 0]$ [0 0 0 0 0 0 0 1] dog

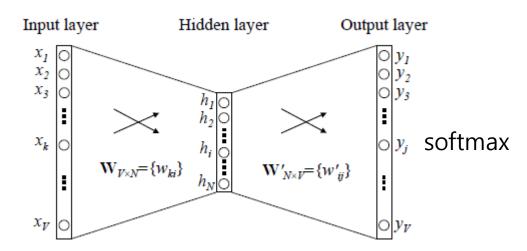


The	[1	0	0	0	0	0	0	0]
quick	[0	1	0	0	0	0	0	0]
brown	[0	0	1	0	0	0	0	0]
fox	[0	0	0	1	0	0	0	0]
jump	[0	0	0	0	1	0	0	0]
over	[0	0	0	0	0	1	0	0]
lazy	[0	0	0	0	0	0	1	0]
dog	[0	0	0	0	0	0	0	1]



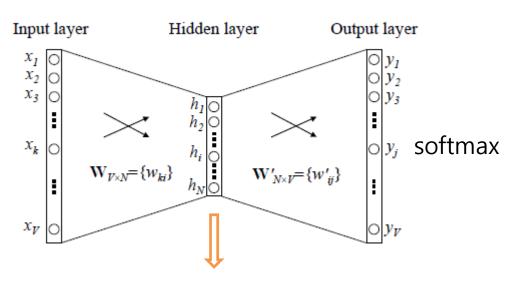


The	[1	0	0	0	0	0	0	0]
quick	[0	1	0	0	0	0	0	0]
brown	[0	0	1	0	0	0	0	0]
fox	[0	0	0	1	0	0	0	0]
jump	[0	0	0	0	1	0	0	0]
over	[0	0	0	0	0	1	0	0]
lazy	[0	0	0	0	0	0	1	0]
dog	[0	0	0	0	0	0	0	1]

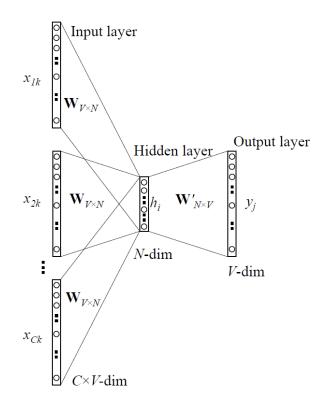


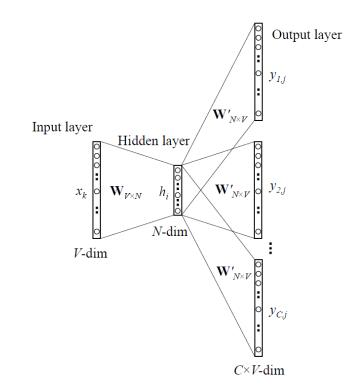


The	[1	0	0	0	0	0	0	0]
quick	[0	1	0	0	0	0	0	0]
brown	[0	0	1	0	0	0	0	0]
fox	[0	0	0	1	0	0	0	0]
jump	[0	0	0	0	1	0	0	0]
over	[0	0	0	0	0	1	0	0]
lazy	[0	0	0	0	0	0	1	0]
dog	[0	0	0	0	0	0	0	1]



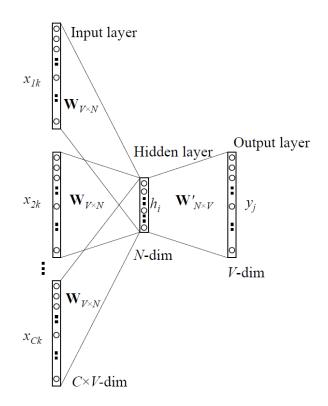
# embedding



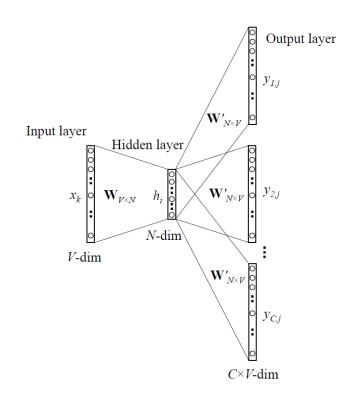


주변 단어로 중심 단어가 나오도록

중심 단어에서 주변 단어들이 나오도록



Continuous Bag of Words (CBOW) Learning



Skip-Gram Model

#### Shakespeare plays dataset

#### PANDABUS:

Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

#### Second Senator:

They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

#### DUKE VINCENTIO:

Well, your wit is in the care of side and that.

#### Second Lord:

They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

#### Clown:

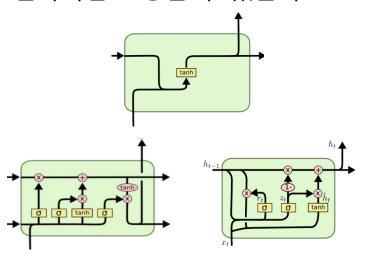
Come, sir, I will make did behold your worship.

#### VIOLA:

I'll drink it.

Shakespeare plays dataset

RNN, LSTM, GRU를 써서 얼마나 진짜처럼 모방할 수 있을까?



#### PANDABUS:

Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

#### Second Senator:

They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

#### DUKE VINCENTIO:

Well, your wit is in the care of side and that.

#### Second Lord:

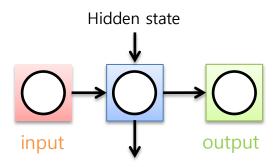
They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

#### Clown:

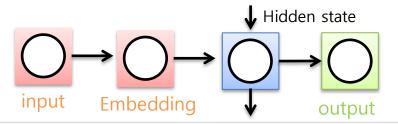
Come, sir, I will make did behold your worship.

#### VIOLA:

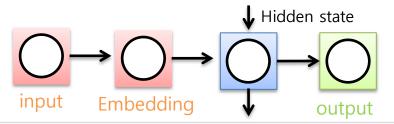
I'll drink it.



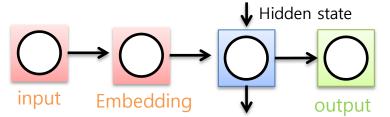
```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(hidden size,hidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out,hidden = self.rnn(out,hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(hidden size,hidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out,hidden = self.rnn(out,hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

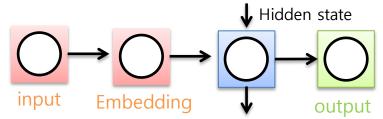


```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(nidden size,nidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



nn.Embedding(num\_embeddings, embedding\_dim)

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(nidden size,nidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

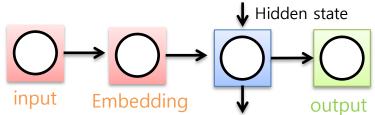


embedding할 가지 수 ex) 알파벳 소문자면 26



nn.Embedding(num\_embeddings, embedding\_dim)

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(nidden size,nidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



embedding할 가지 수 ex) 알파벳 소문자면 26

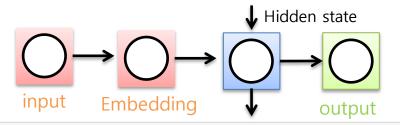
T mbodding

 $nn. Embedding (num\_embeddings,\ embedding\_dim)$ 

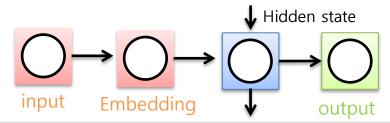
+1.01

목표 embedding 차원

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(nidden size,nidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
   def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self_encoder = nn_Embedding(input_size_hidden_size)
       self.rnn = nn.RNN(hidden size,hidden size,num layers)
        setf.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



num\_layers는 파란색 RNN Cell을 몇 개 쌓을 것 인지

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self_encoder = nn_Embedding(input_size_hidden_size)
        self.rnn = nn.RNN(hidden size,hidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

num\_layers는 파란색 RNN Cell을 몇 개 쌓을 것 인지

ex) 2개 였다면?

```
Hidden state

Hidden state

Hidden state

Hidden state

Output
```

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self_encoder = nn_Embedding(input_size_hidden_size)
        self.rnn = nn.RNN(hidden size,hidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

num\_layers는 파란색 RNN Cell을 몇 개 쌓을 것 인지

ex) 2개 였다면?

GRU로 바꾸려면 nn.RNN을 nn.GRU로 바꿔주면 됨

```
Hidden state

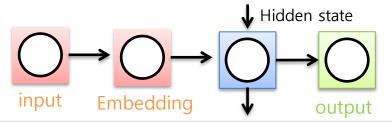
Hidden state

Hidden state

Hidden state

Output
```

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
       self_encoder = nn_Embedding(input_size_hidden_size)
        self.rnn = nn.RNN(hidden size,hidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(hidden size hidden size num lavers)
       self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

Hidden state

Hidden state

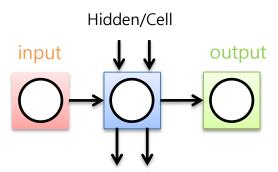
output

output

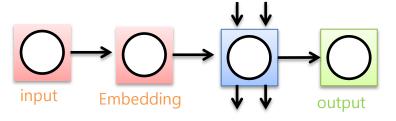
hidden state size로 나온 결과값을 원하는 모양으로 맞춰주는 부분

```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.RNN(hidden size hidden size num lavers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden):
        out = self.encoder(input.view(1,-1))
        out, hidden = self.rnn(out, hidden)
        out = self.decoder(out.view(batch size,-1))
        return out, hidden
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers, batch size, hidden size)).cuda()
        return hidden
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

#### LSTM

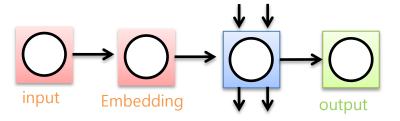


```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.encoder = nn.Embedding(input size, hidden size)
        self.rnn = nn.LSTM(hidden size, hidden size, num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden,cell):
        out = self.encoder(input.view(1,-1))
        out,(hidden,cell) = self.rnn(out,(hidden,cell))
        out = self.decoder(out.view(batch size,-1))
        return out, hidden, cell
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        cell = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        return hidden, cell
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```



```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.rnn = nn.LSTM(hidden size,hidden size,num layers)
        self.decoder = nn.Linear(hidden size, output size)
    def forward(self, input, hidden,cell):
        out - colf encoder/input view/1 -1))
        out, (hidden, cell) = self.rnn(out, (hidden, cell))
        out = self.decoder(out.view(batch size,-1))
        return out, hidden, cell
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        cell = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        return hidden, cell
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

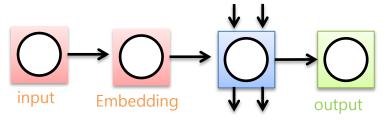
구현적인 측면에서는 hidden state에다가 cell state를 추가 한 정도



```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.rnn = nn.LSTM(hidden size,hidden size,num lavers)
        self.decoder = nm.Linear(hidden size, output size)
    def forward(self, input, hidden,cell):
        out - colf encoder/input view(1 -1))
        out, (hidden, cell) = self.rnn(out, (hidden, cell))
        out = self.decoder(out.view(batch size.-1))
        return out, hidden, cell
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        cell = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        return hidden, cell
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

구현적인 측면에서는 hidden state에다가 cell state를 추가 한 정도

성능은 실습으로 확인



```
class RNN(nn.Module):
    def init (self, input size, hidden size, output size, num layers=1):
        super(RNN, self). init ()
        self.input size = input size
        self.hidden size = hidden size
        self.output size = output size
        self.num layers = num layers
        self.rnn = nn.LSTM(hidden size,hidden size,num layers)
        self.decoder = nm.Linear(hidden size, output size)
    def forward(self, input, hidden,cell):
        out - colf encoder/input view(1 -1))
        out,(hidden,cell) = self.rnn(out,(hidden,cell))
        out = self.decoder(out.view(batch size.-1))
        return out, hidden, cell
    def init hidden(self):
        hidden = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        cell = Variable(torch.zeros(num layers,batch size,hidden size)).cuda()
        return hidden, cell
model = RNN(n characters, hidden size, n characters, num layers).cuda()
```

# Q&A