Recurrent Neural Networks

2017.08.19

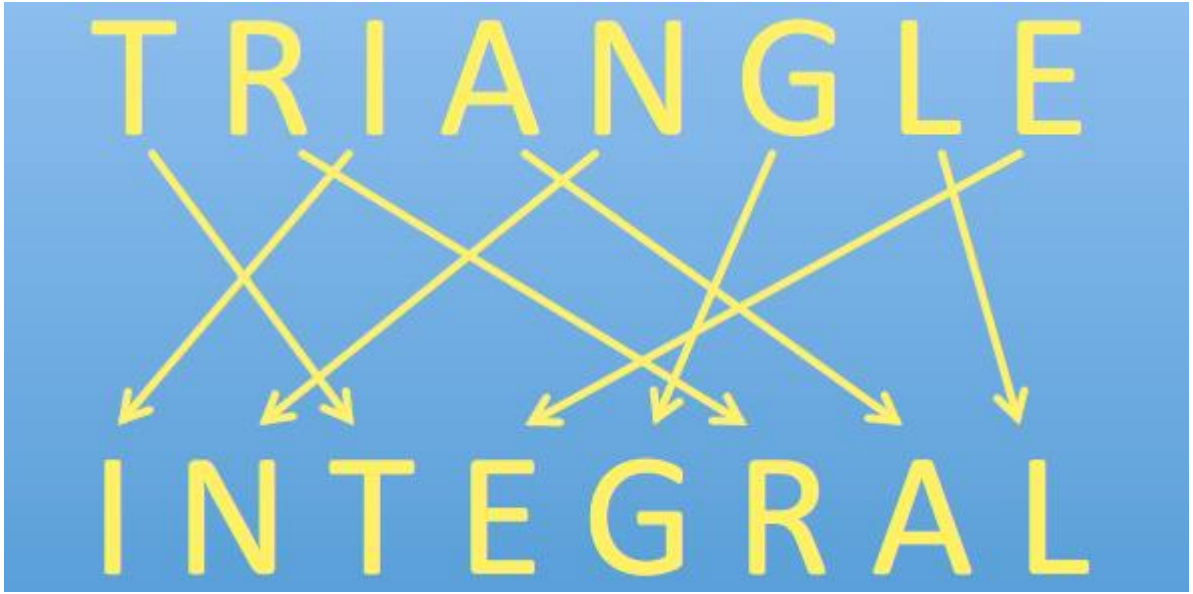
최건호

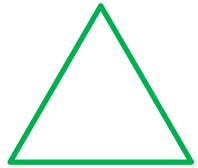
INDEX

Why RNN

(출처: https://mathjokes4mathyfolks.files.wordpress.com/2016/03/anagramslide.png)

Why RNN





(출처: https://mathjokes4mathyfolks.files.wordpress.com/2016/03/anagramslide.png)

Why RNN

모든말에 의미를 부여하고

그 말의 의미를 고민한다

(출처: 웹툰 마음의 소리)

Why RNN

모든말에 의미를 부여하고

밥은중요
(의,식,주)

밥을 달라

미래를 부탁
가장의 역할
믿음직해 함

연애

미래를약속,부모님의반대

사랑으로 결혼

행복

그 말의 의미를 고민한다

$$= \int E_{-j}\left[\log \frac{p(z,x)}{q_j(z_j)}\right] q_j(z_j)\, dz_j - \sum_{i\neq j}\int \log q_i(z_i) \prod_{l\neq i} q_l(z_l)\, dz$$

$$= \int E_{-j}\left[\log \frac{p(z,x)}{q_j(z_j)}\right] q_j(z_j)\, dz_j - \sum_{i\neq j}\int \left[\int \prod_{l\neq i} q_l(z_l)\, dz_{-i}\right] q_i(z_i)\log q_i(z_i)\, dz_i$$

$= 1$ 엔트로피(H)

$$= \int E_{-j}\left[\log \frac{p(z,x)}{q_j(z_j)}\right] q_j(z_j)\, dz_j - \sum_{i\neq j}\int H_{q_i}(z_i)$$

$-j$에 대해 constant　　j 외 모두 → constant

$$= \int E_{-j}\left[\log p(z,x)\right] q_j(z_j)\, dz_j - \int \log q_j(z_j)\, q_j(z_j)\, dz_j + C$$

$$= \int \log\left(\exp\left(E_{-j}[\log p(z,x)]\right)\right) q_j(z_j)\, dz_j - \int \log q_j(z_j)\, q_j(z_j)\, dz_j + C$$

상쇄

$$= \int q_j(z_j) \log \frac{\exp\left(E_{-j}[\log p(z,x)]\right)}{q_j(z_j)}\, dz_j + C$$

$$= -\int q_j(z_j) \log \frac{q_j(z_j)}{\exp\left(E_{-j}[\log p(z,x)]\right)}\, dz_j + C$$

$$= -KL\left(q_j(z_j) \,\|\, \exp\left(E_{-j}[\log p(z,x)]\right)\right) + C$$

⇒ ELBO를 maximize 하기위해서는 $KL\left(q_j(z_j) \,\|\, \exp\left(E_{-j}[\log p(z,x)]\right)\right)$ 를 minimize.

⇒ 결론적으로, $q_j^*(z_j) \propto \exp\left(E_{-j}[\log p(z,x)]\right)$

(출처: 웹툰 마음의 소리)

Why RNN


PG
Aug 12
O:85.44 H:87.45 L:85.13 C:87.04
Vol:74.96M
+0.31 (0.36%)
finviz.com
MONTHLY
90
87.04
80
70
60
50
40
30
800M
600M
400M
200M
2009
2010
2011
2012
2013
2014
2015
2016
PG [NYSE]
The Procter & Gamble Company

Why RNN

흐름, 패턴, 순서

Why RNN

인간의 사고과정은 Sequential하게 이루어진다

Why RNN

인간의 사고과정은 Sequential하게 이루어진다

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

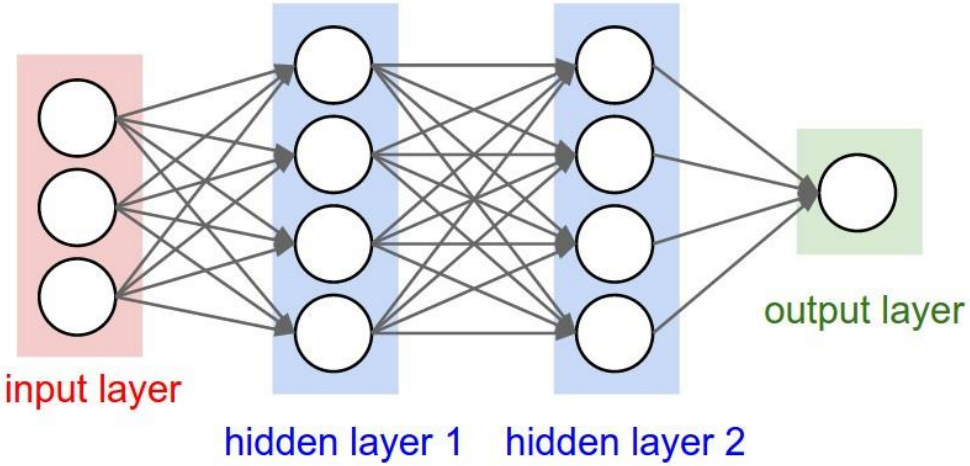
Why RNN

인간의 사고과정은 Sequential하게 이루어진다

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

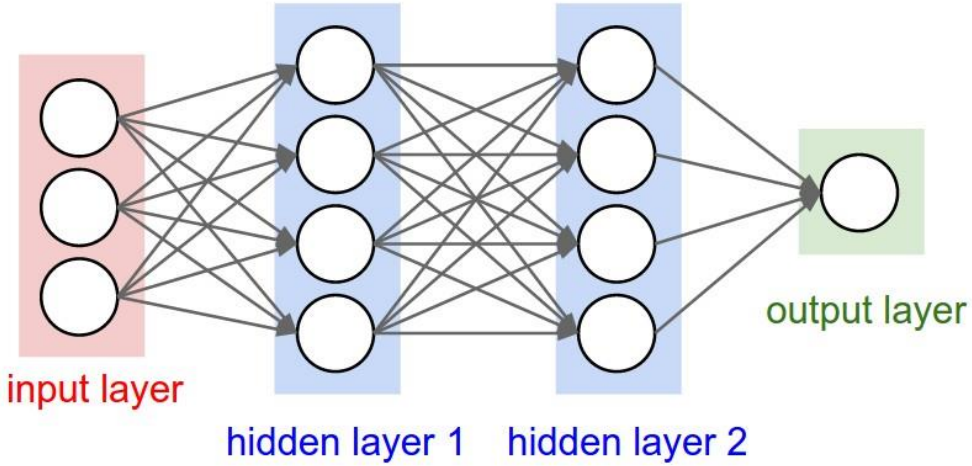
하지만 여태까지 배운 Simple Neural Network와 CNN에는 Sequence에 대한 고려가 없었음

Why RNN

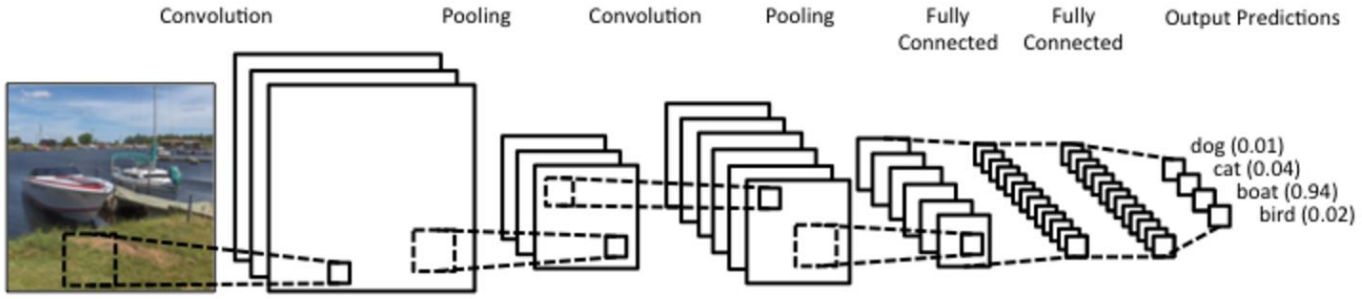


http://cs231n.github.io/convolutional-networks/

Why RNN

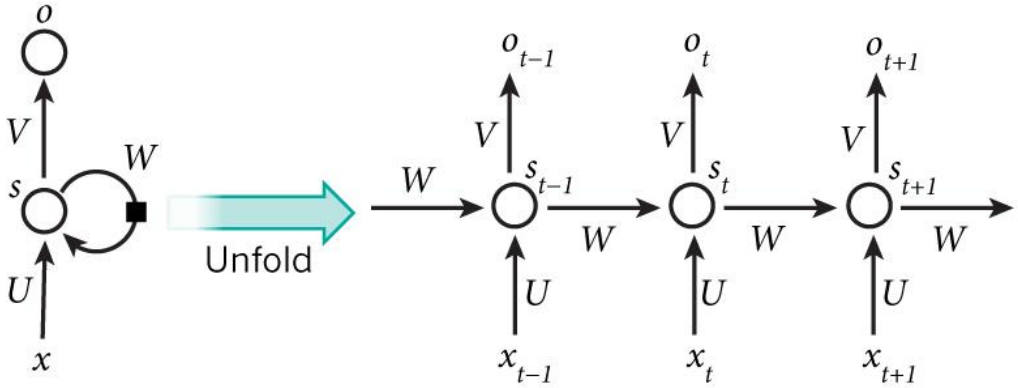


http://cs231n.github.io/convolutional-networks/



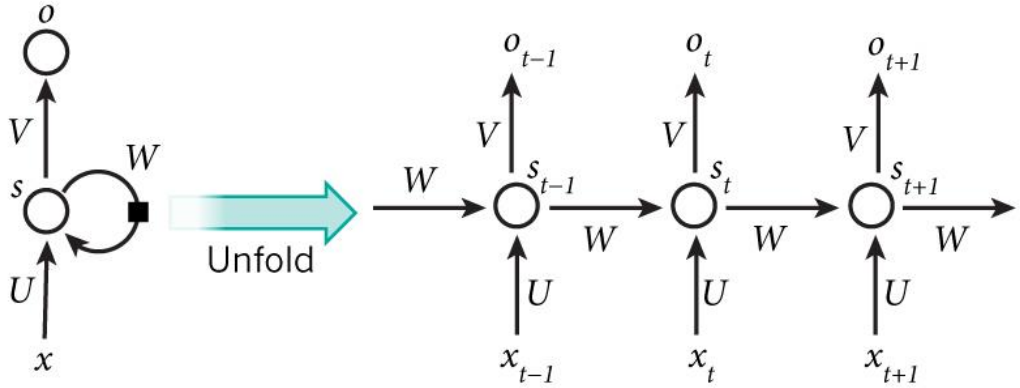
http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Naïve RNN

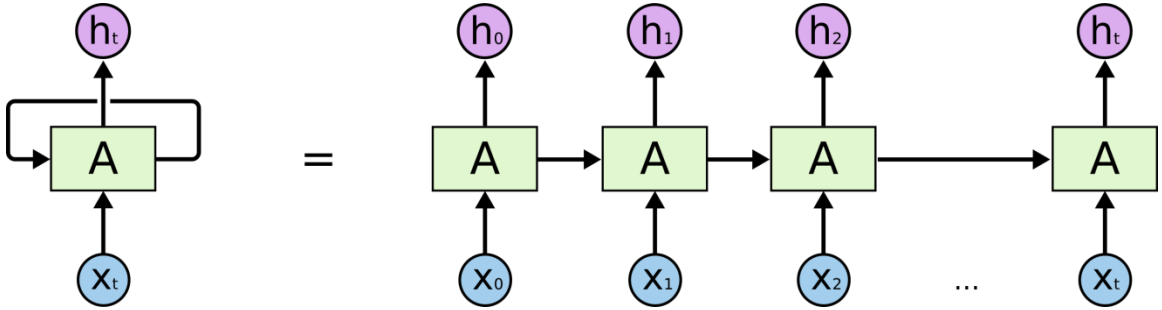


source: Nature

Naïve RNN



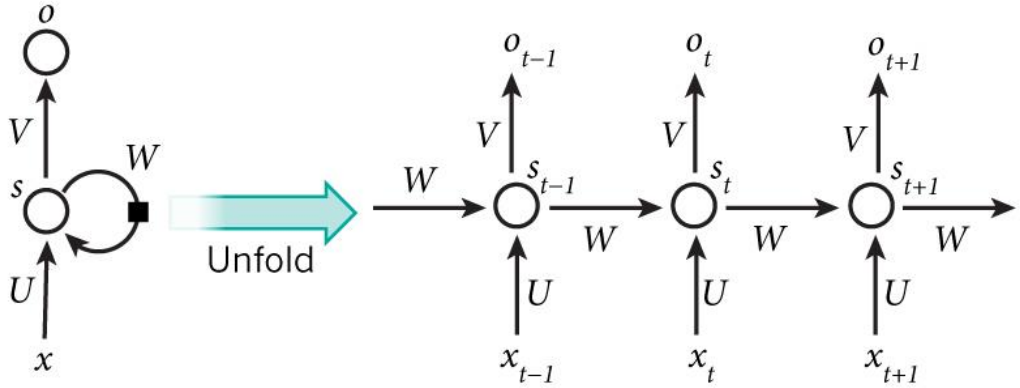
source: Nature



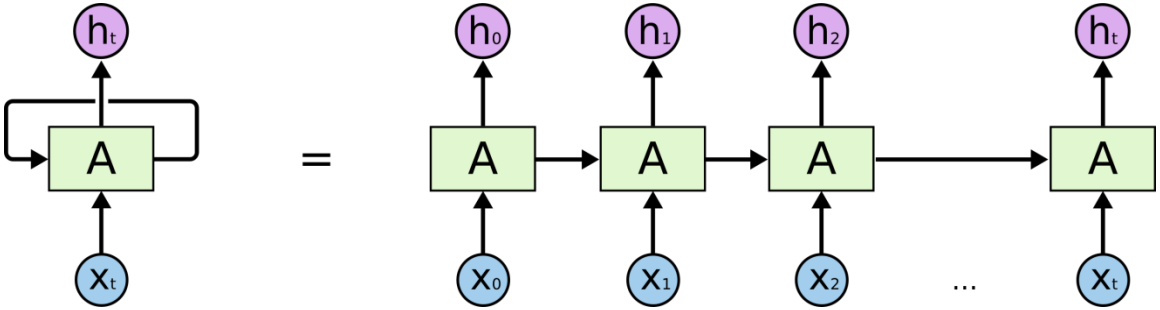
http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

Naïve RNN

뭔가 기존의 Neural Network와 연결이 잘 안됨

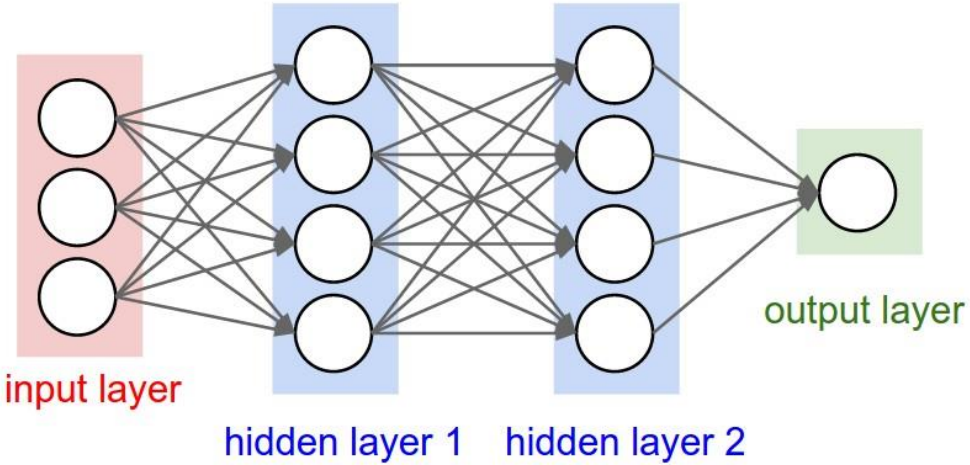


source: Nature

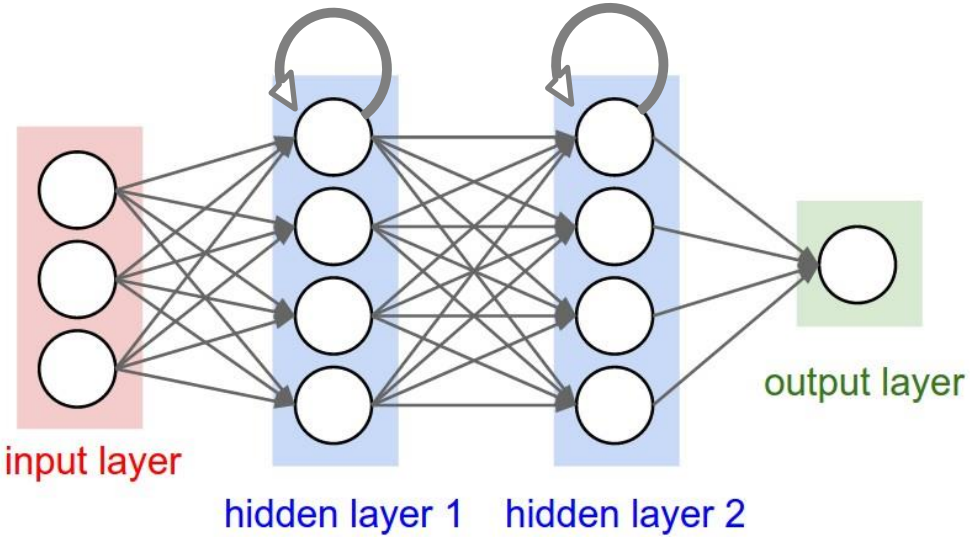


http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

Naïve RNN

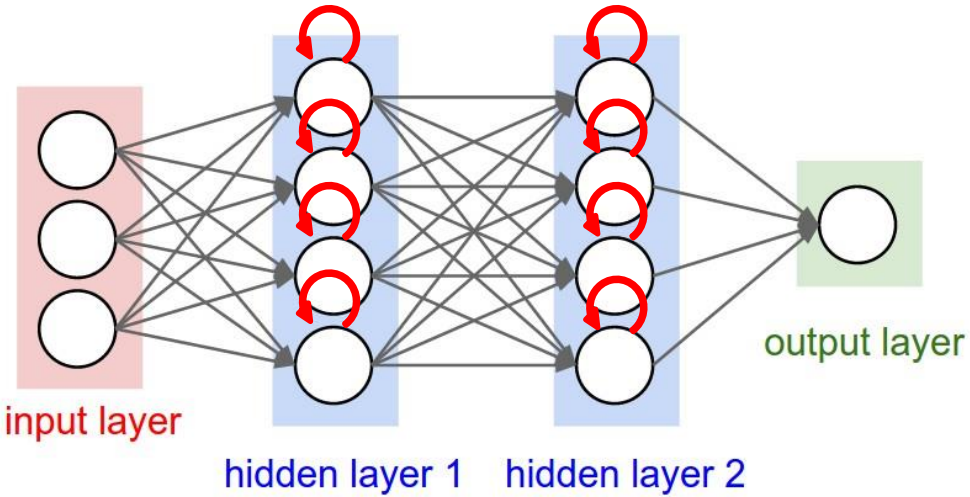


Naïve RNN

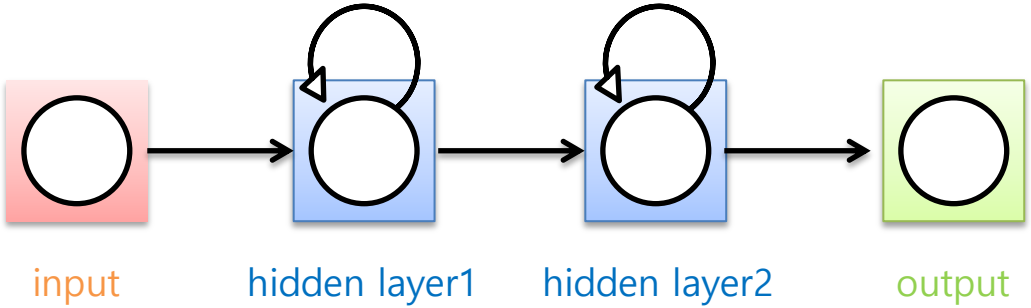


Naïve RNN

정확하게 표현하면
각 노드들에 적용됨

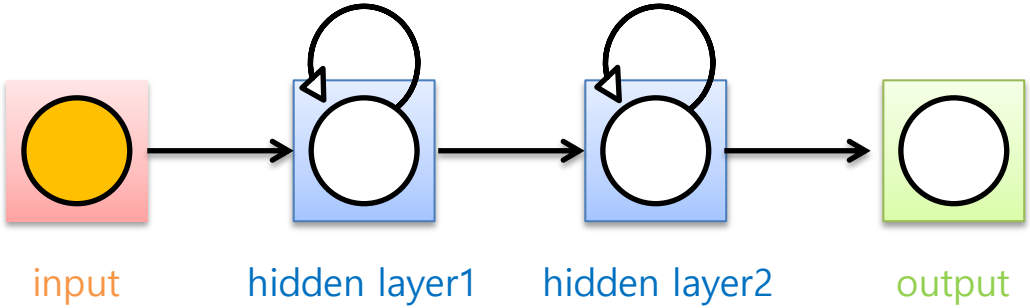


Naïve RNN



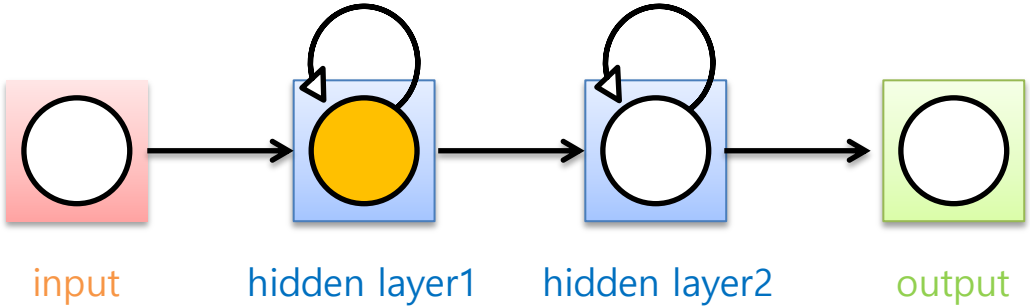
Naïve RNN

t=0 일 때



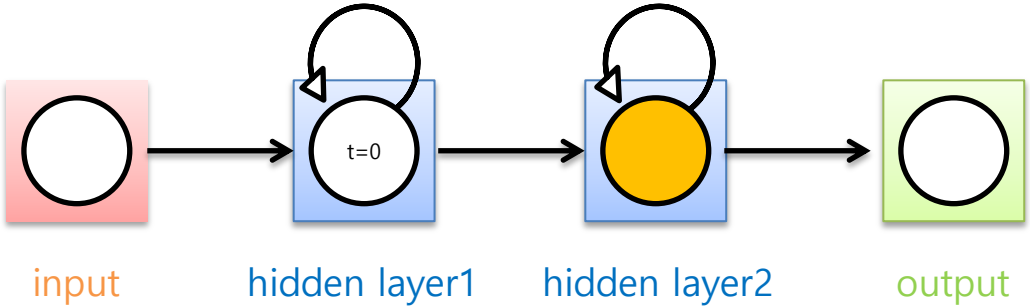
Naïve RNN

t=0 일 때



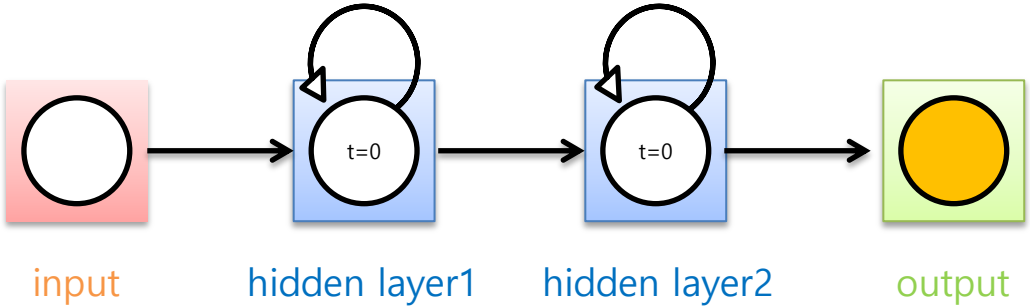
Naïve RNN

t=0 일 때



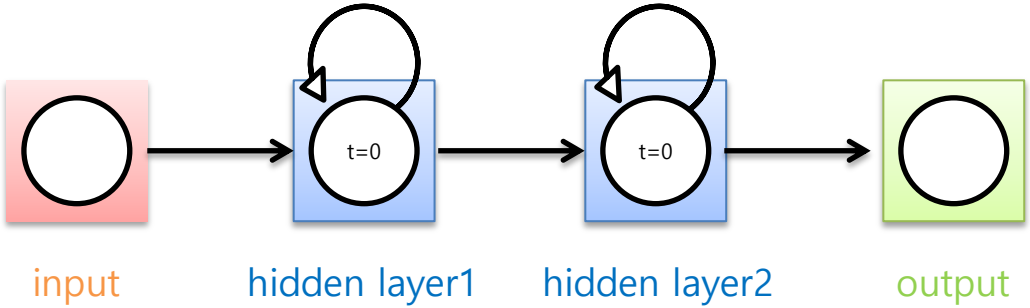
Naïve RNN

t=0 일 때



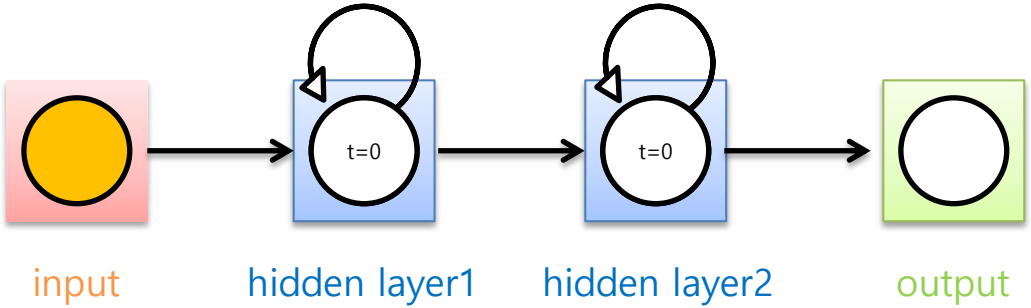
Naïve RNN

t=0은 지나가고
t=1일 때



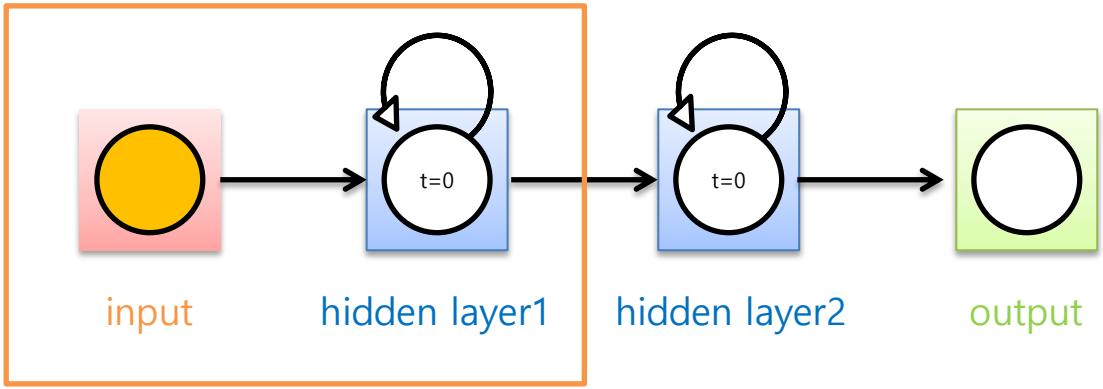
Naïve RNN

t=0은 지나가고
t=1일 때



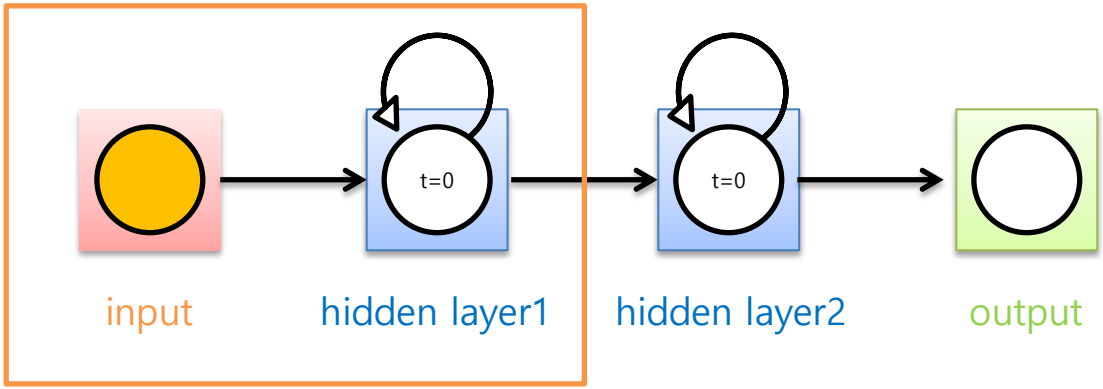
Naïve RNN

t=0은 지나가고
t=1일 때

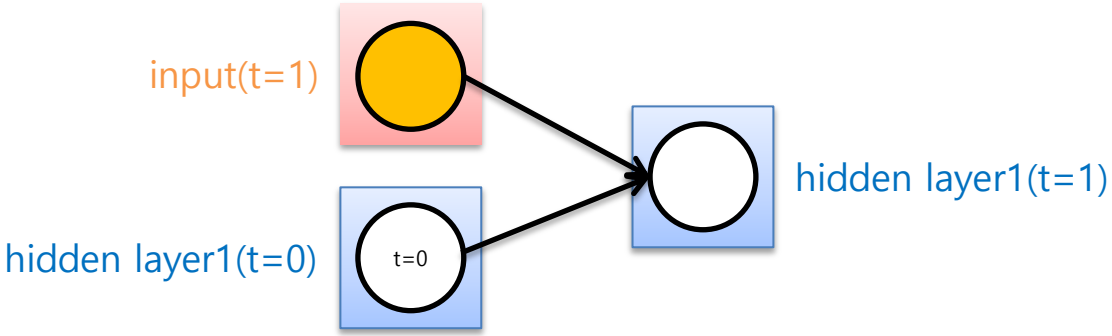


Naïve RNN

t=0은 지나가고
t=1일 때

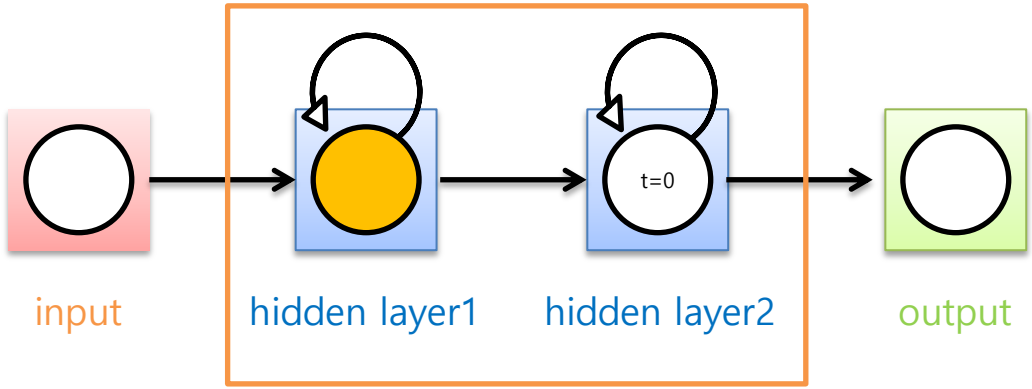


=

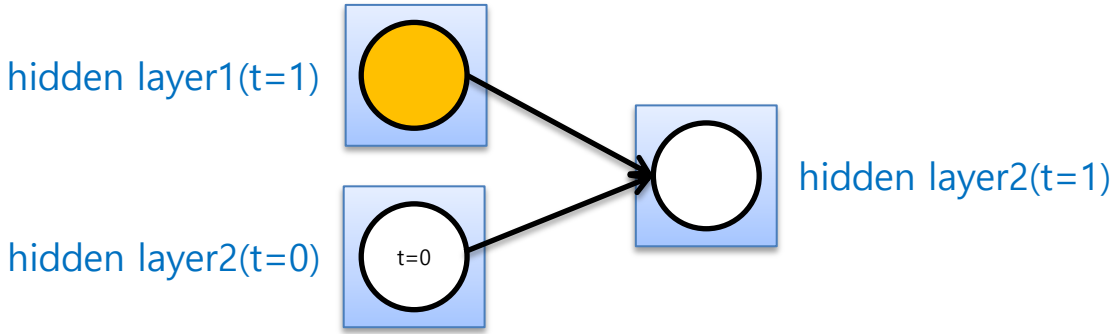


Naïve RNN

t=0은 지나가고
t=1일 때

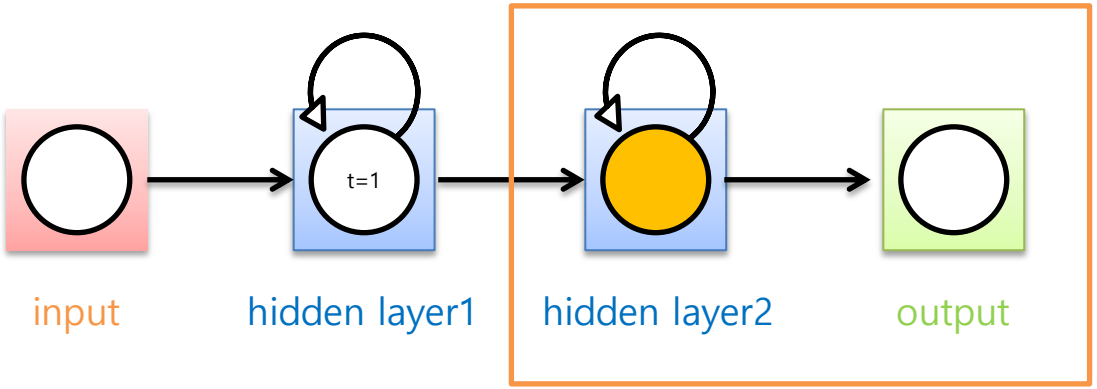


=

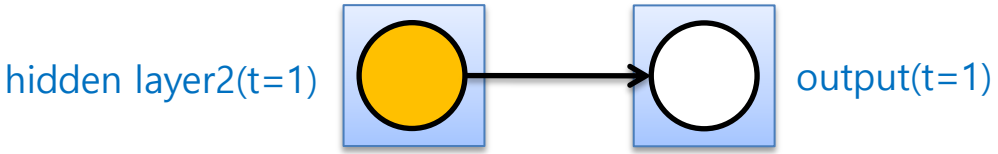


Naïve RNN

t=0은 지나가고
t=1일 때

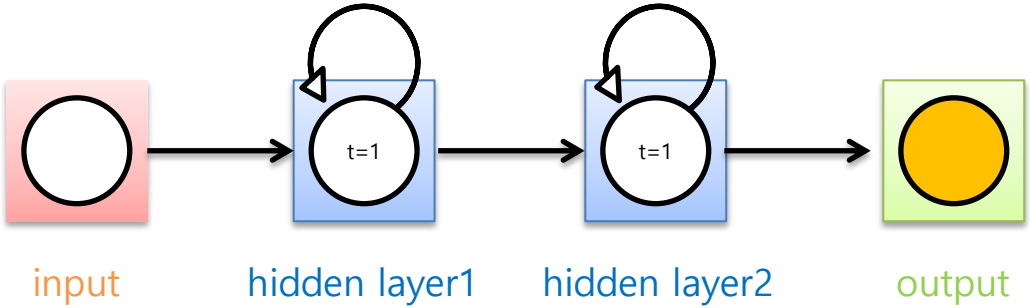


=



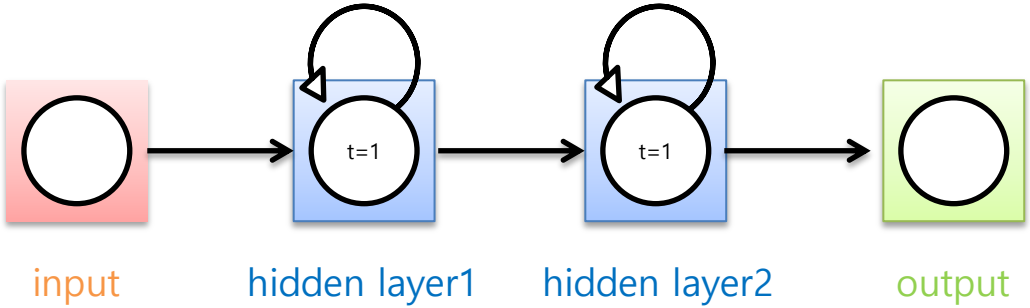
Naïve RNN

t=0은 지나가고
t=1일 때

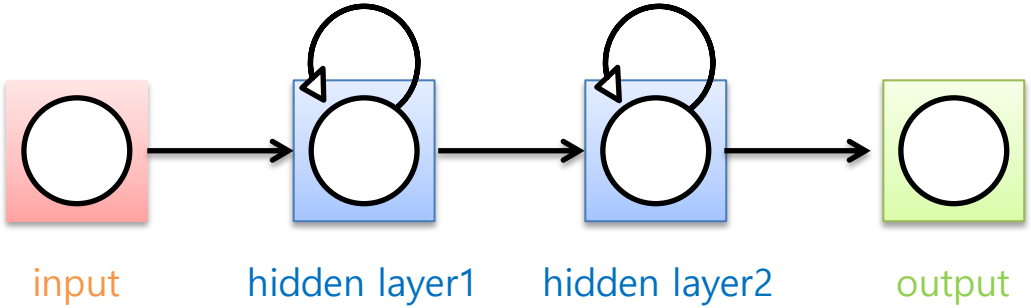


Naïve RNN

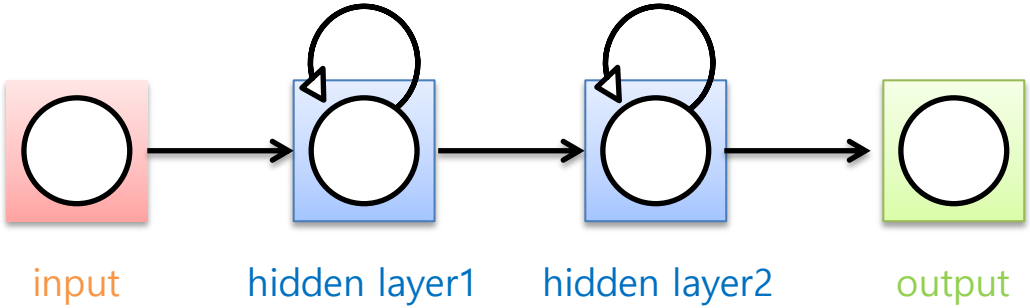
t=0은 지나가고
t=1일 때



Naïve RNN



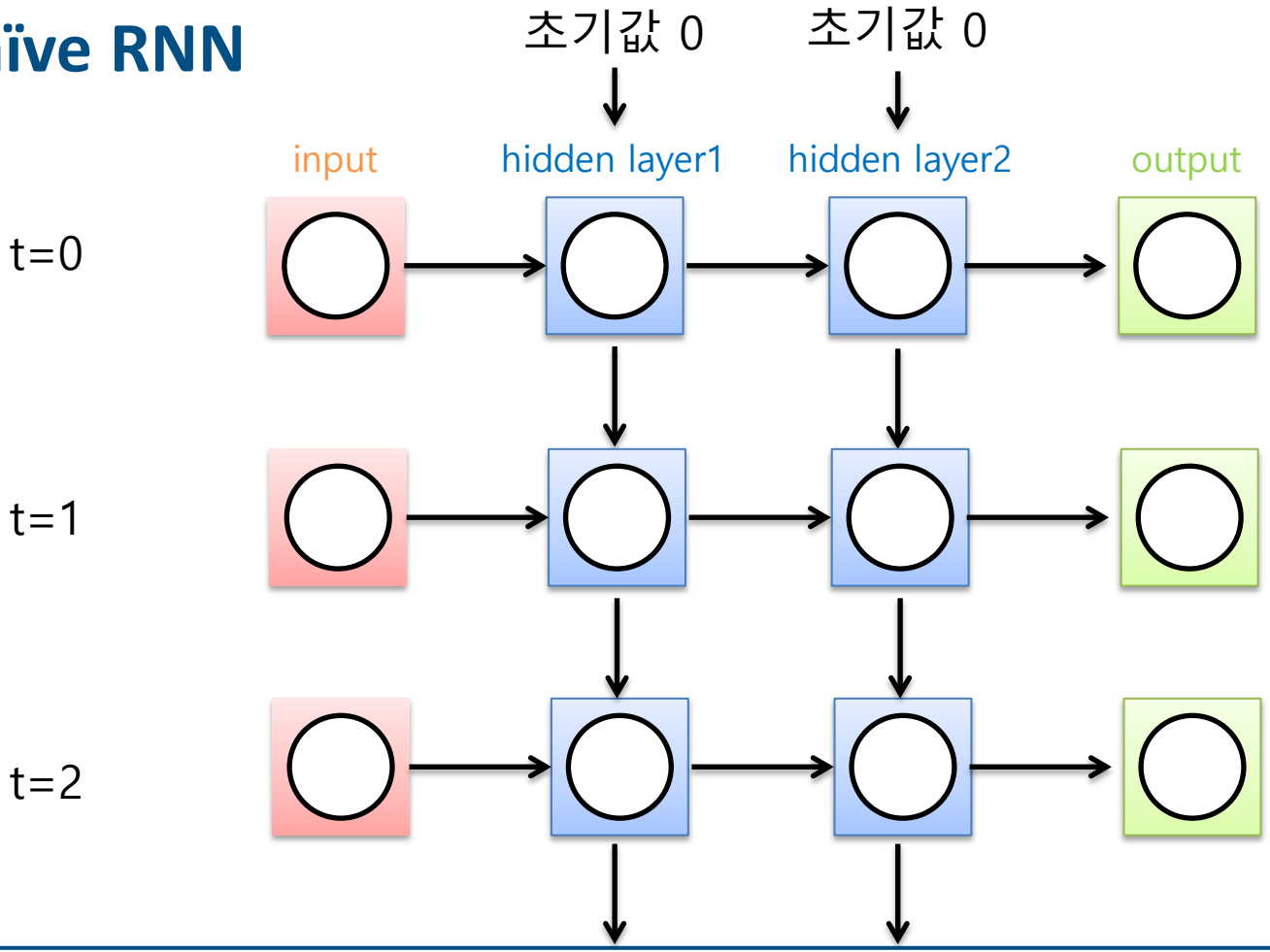
Naïve RNN



Recurrent 부분을 풀어서 다시 보면
이제는 이해할 수 있음

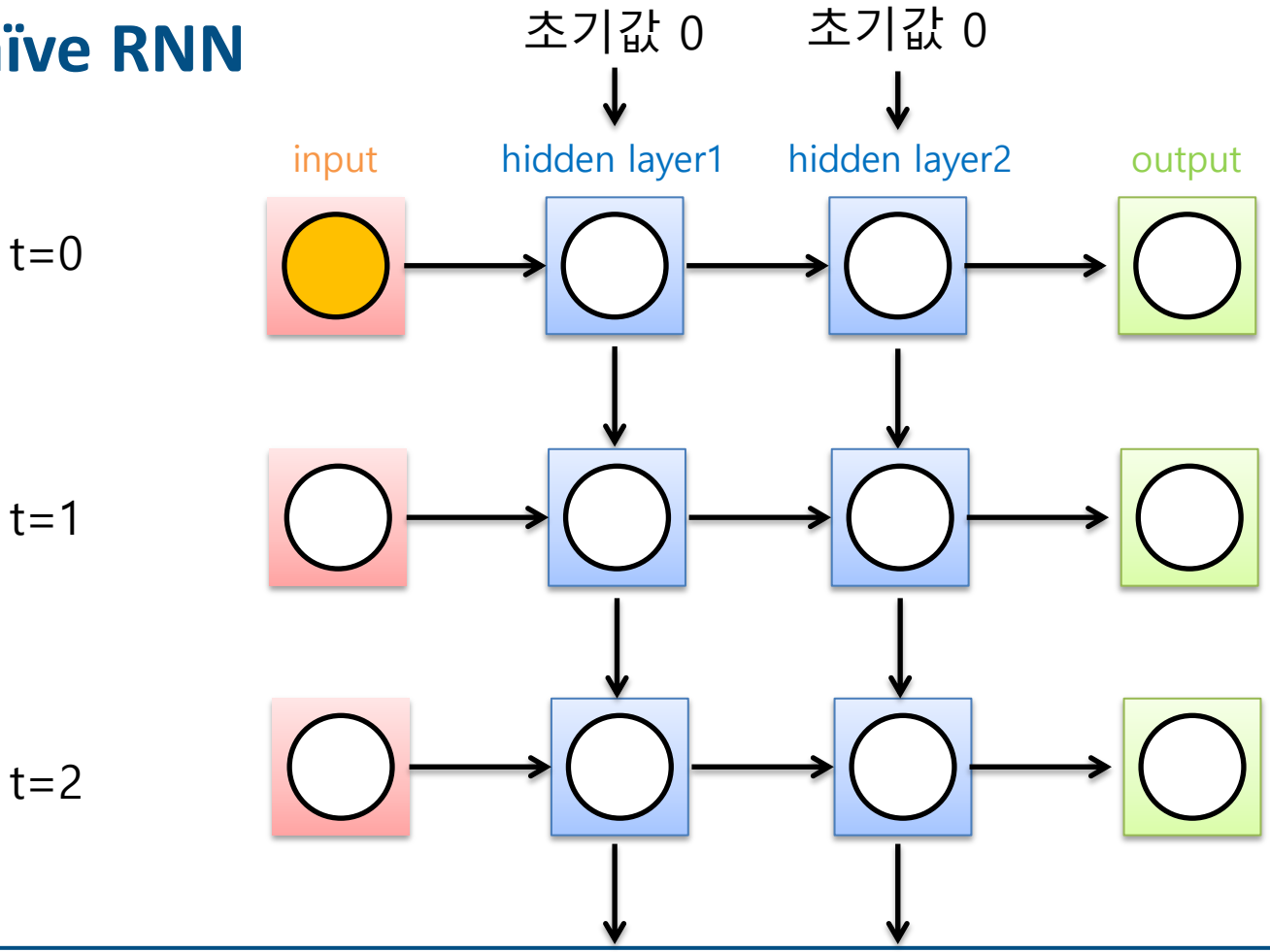
Naïve RNN

training process



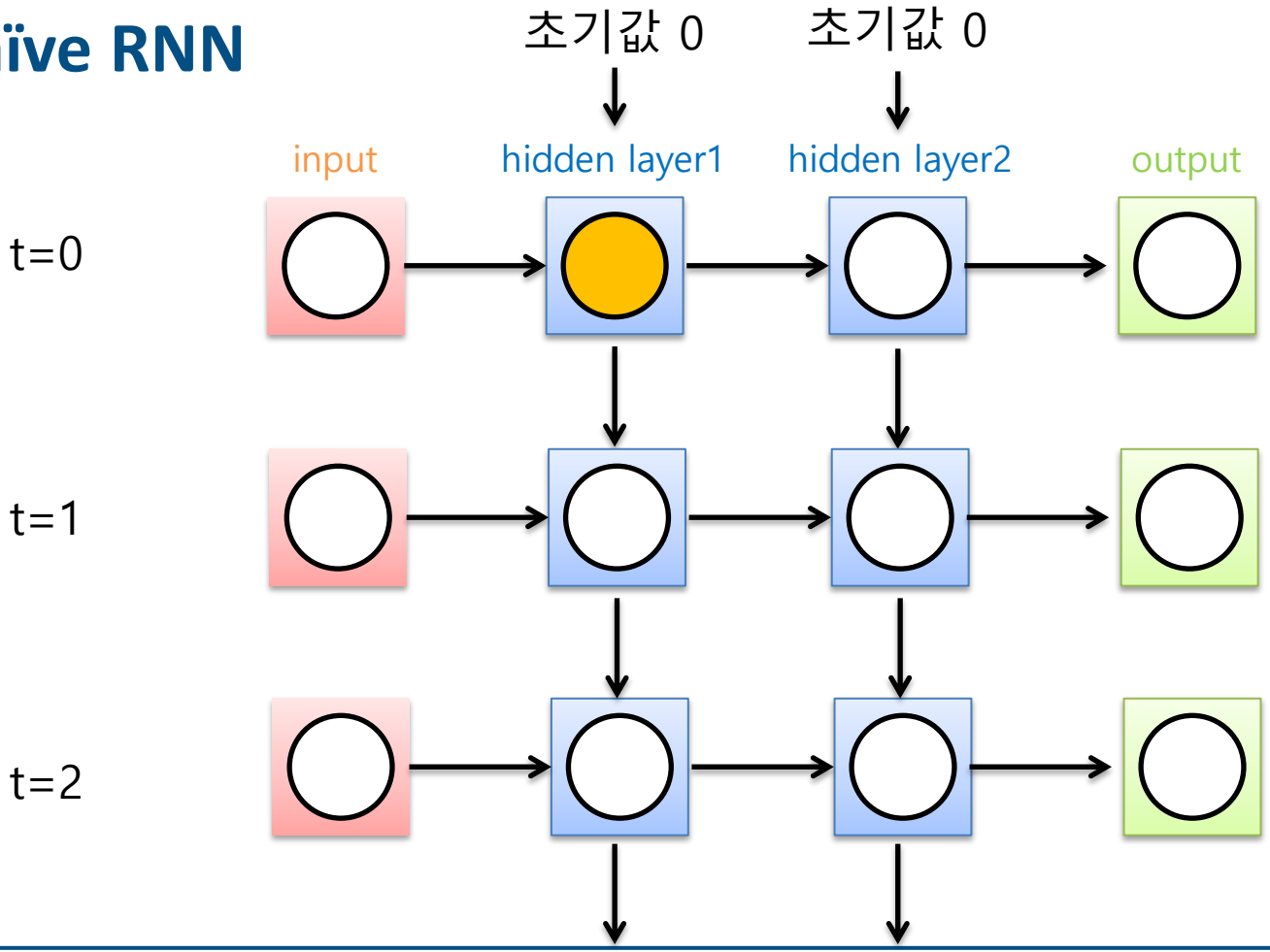
Naïve RNN

training process



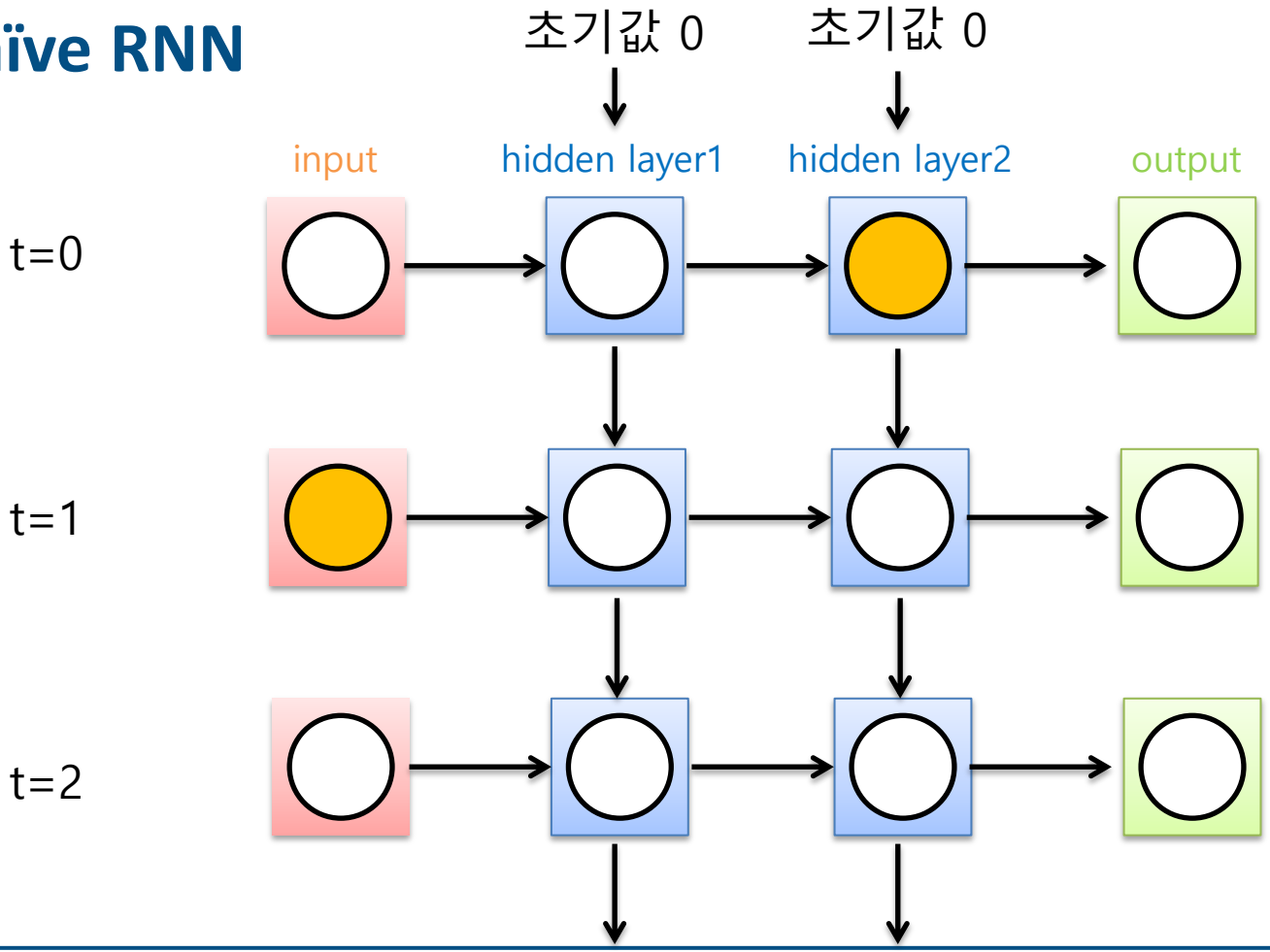
Naïve RNN

training process



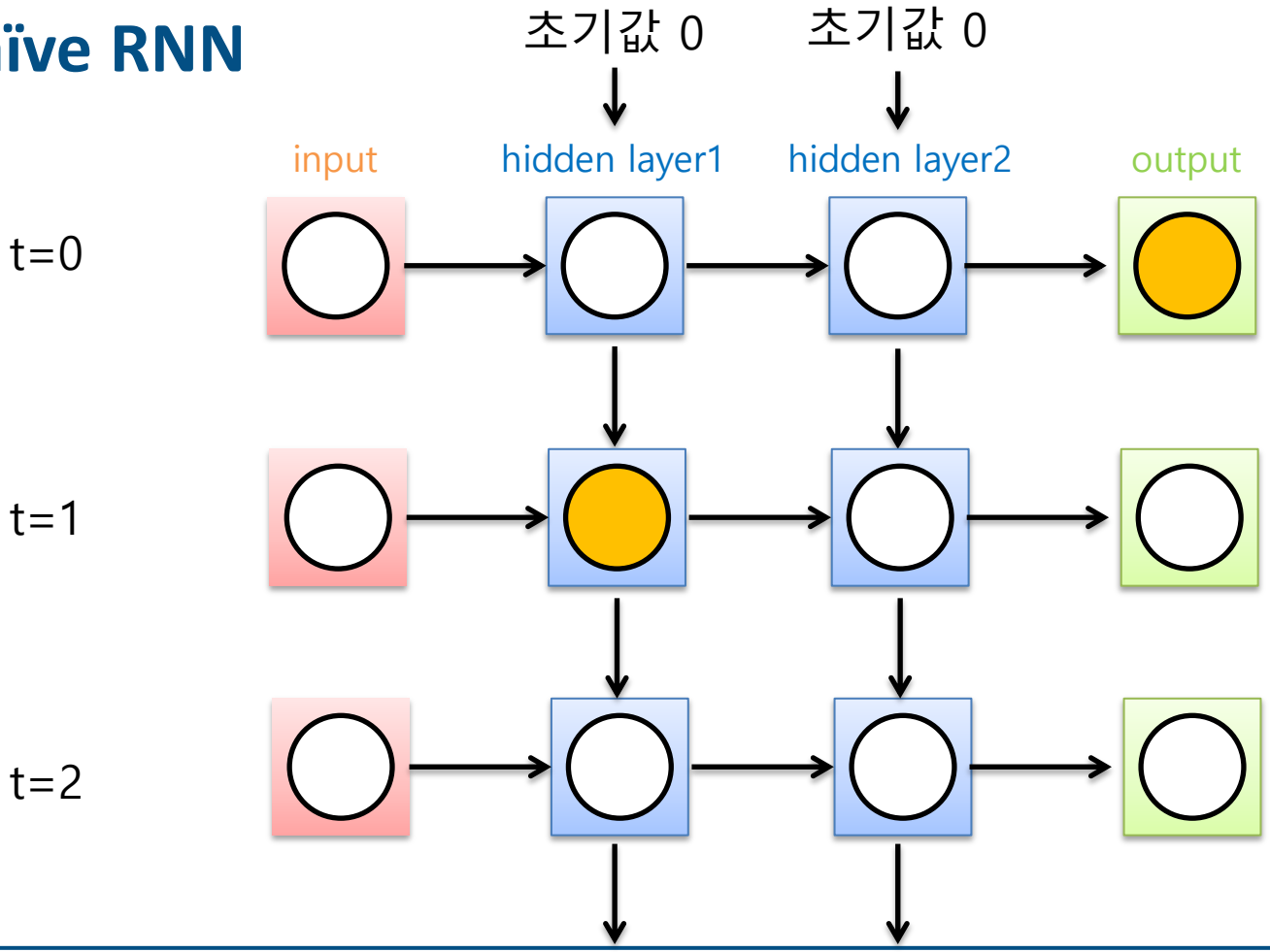
Naïve RNN

training process



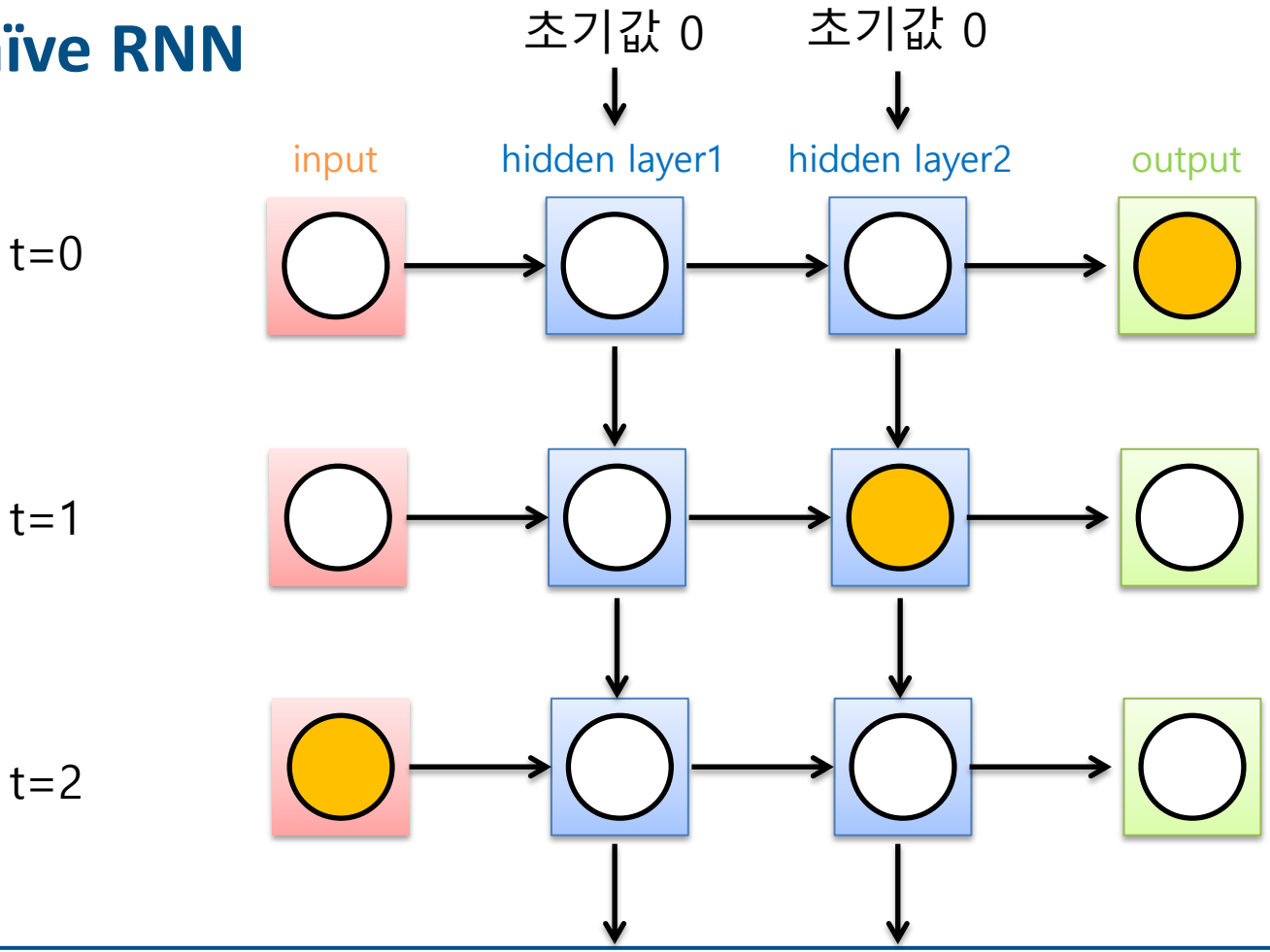
Naïve RNN

training process



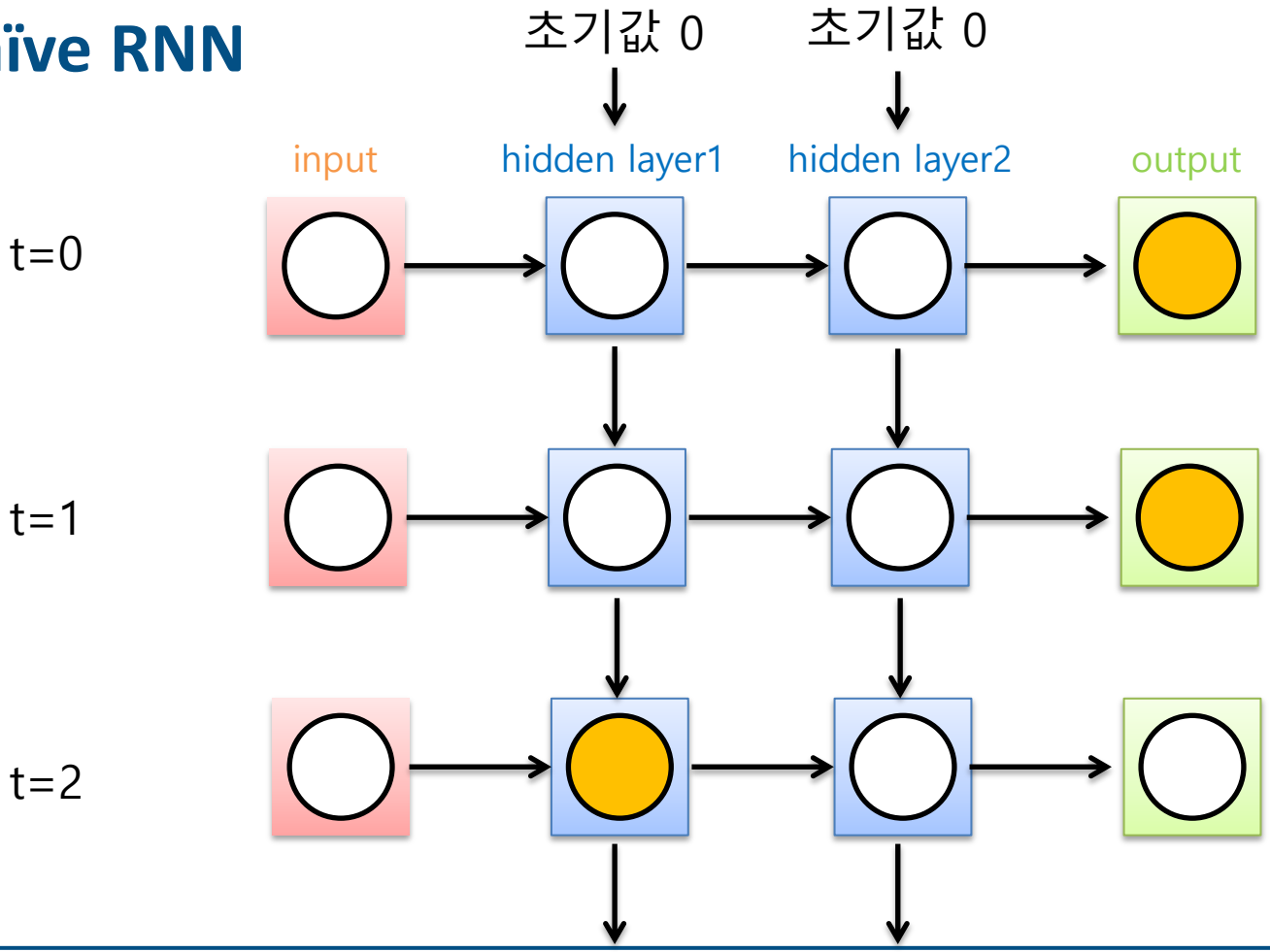
Naïve RNN

training process



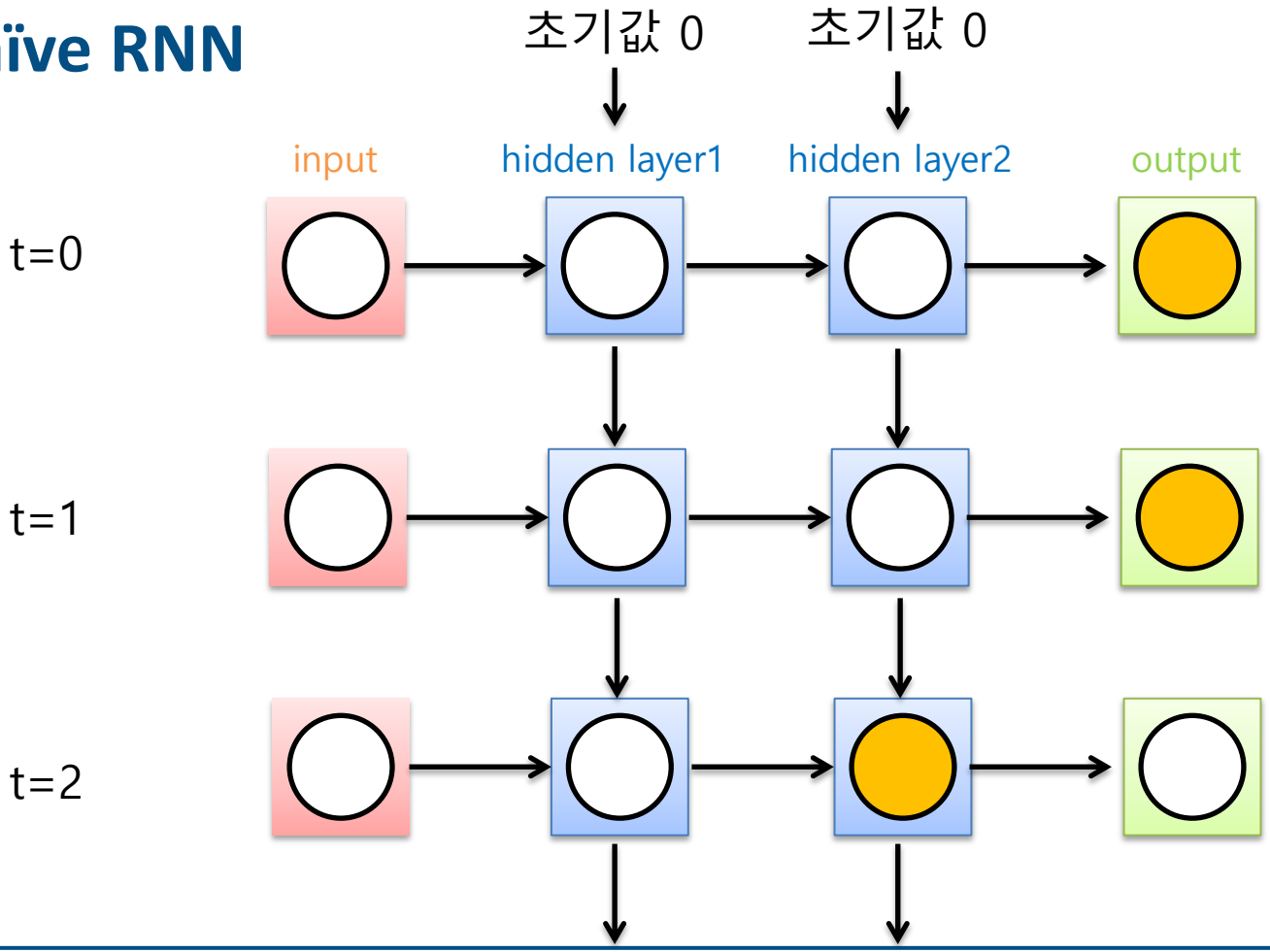
Naïve RNN

training process



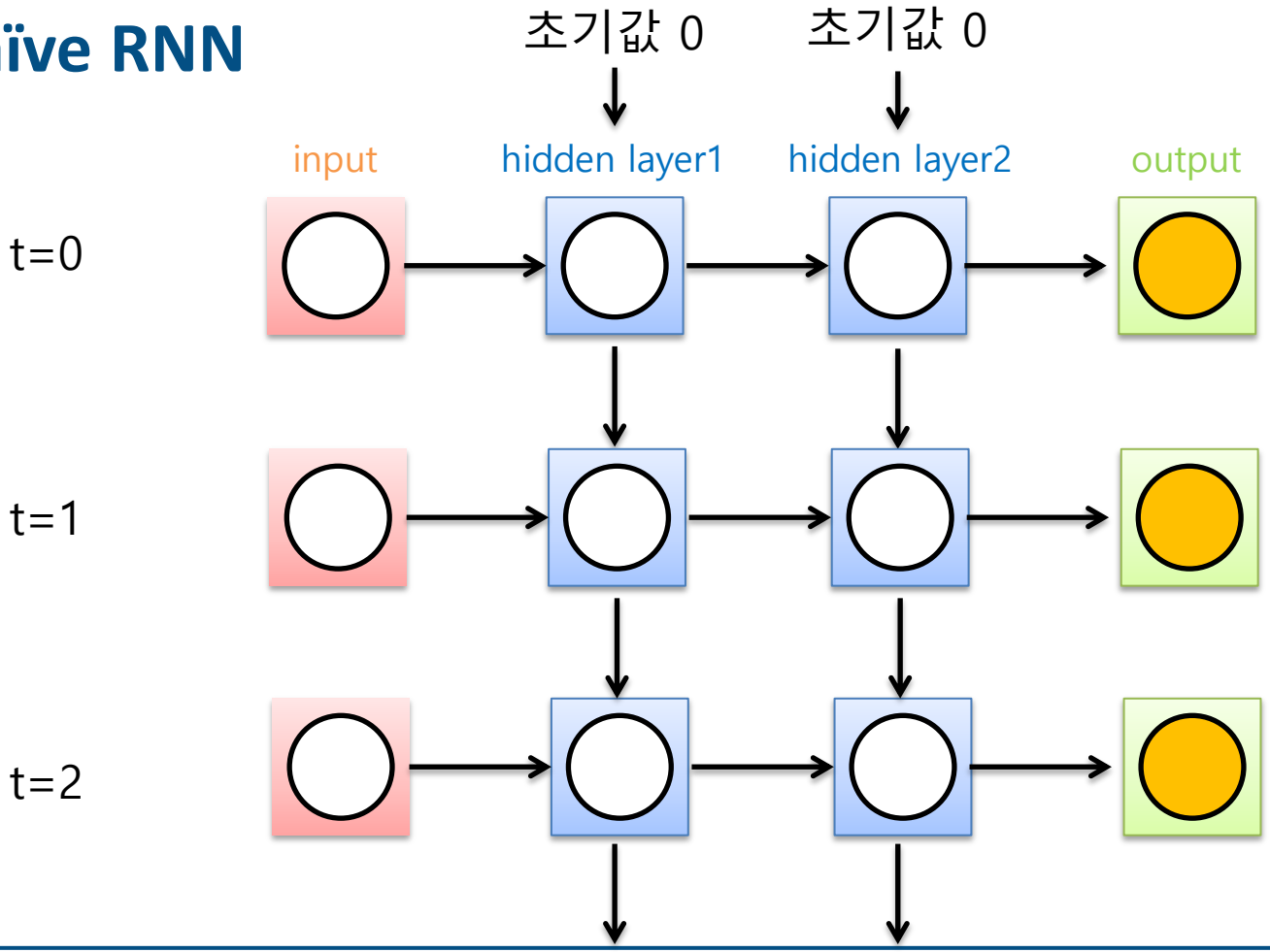
Naïve RNN

training process



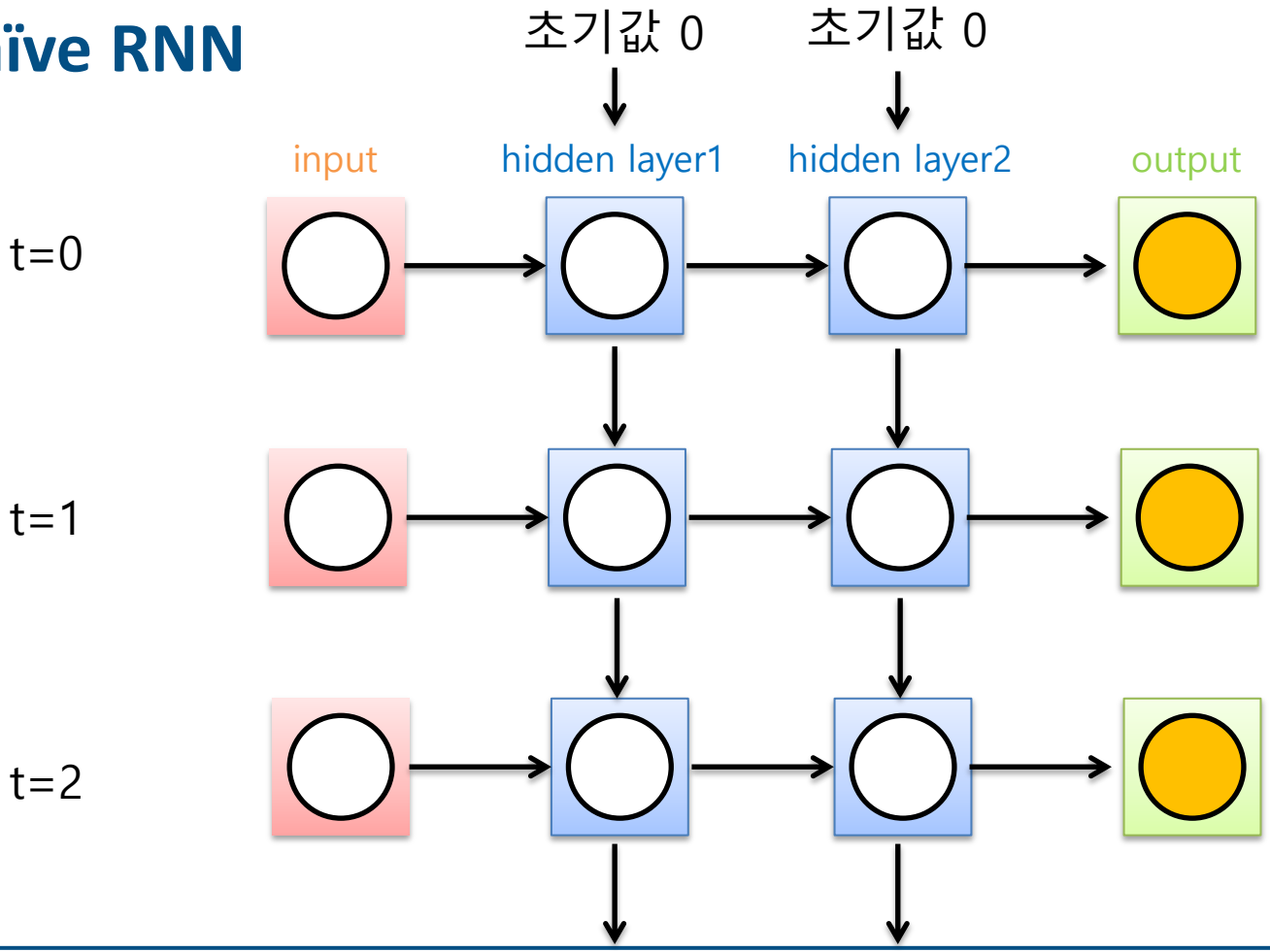
Naïve RNN

training process



Naïve RNN

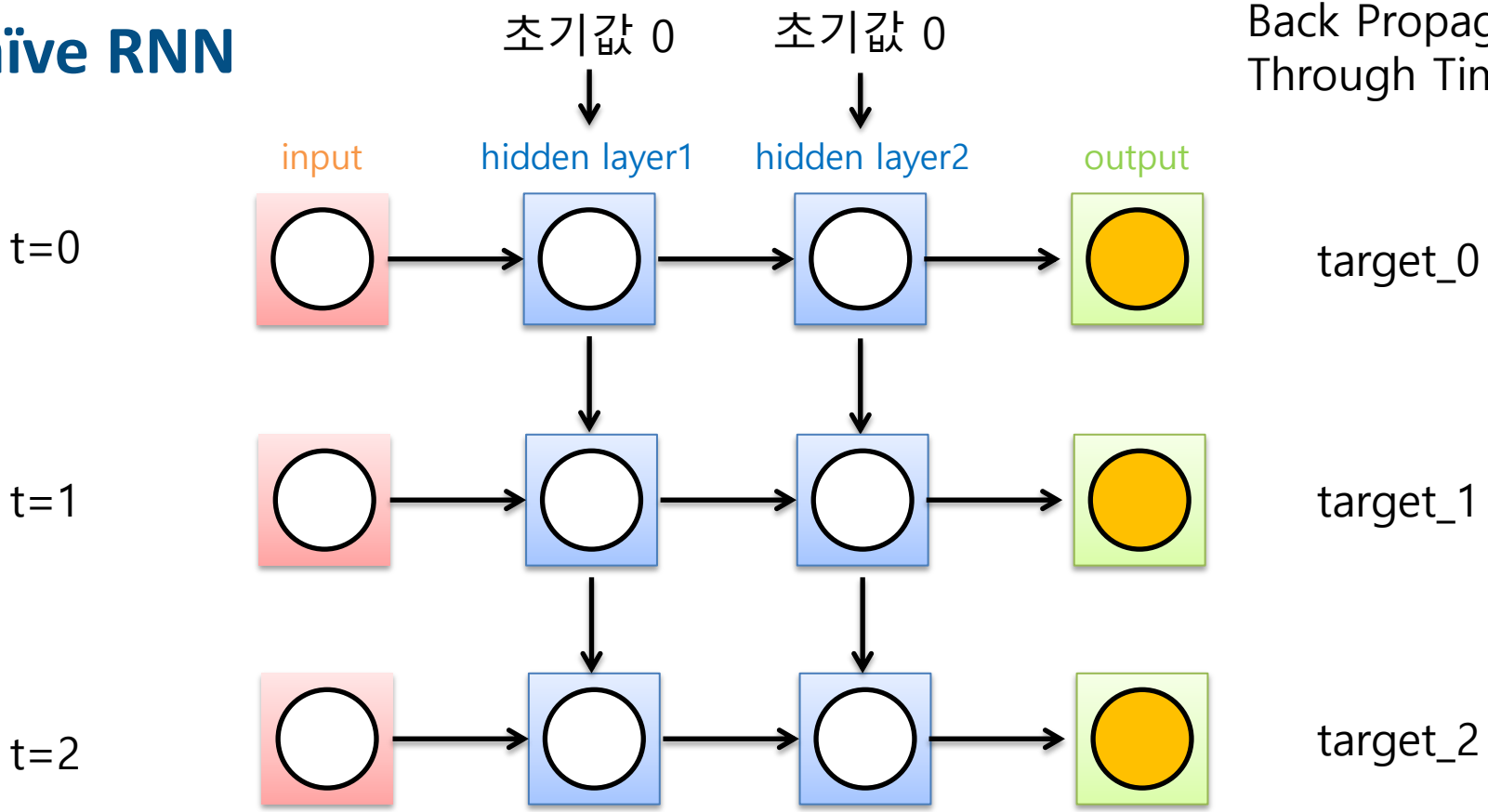
training process



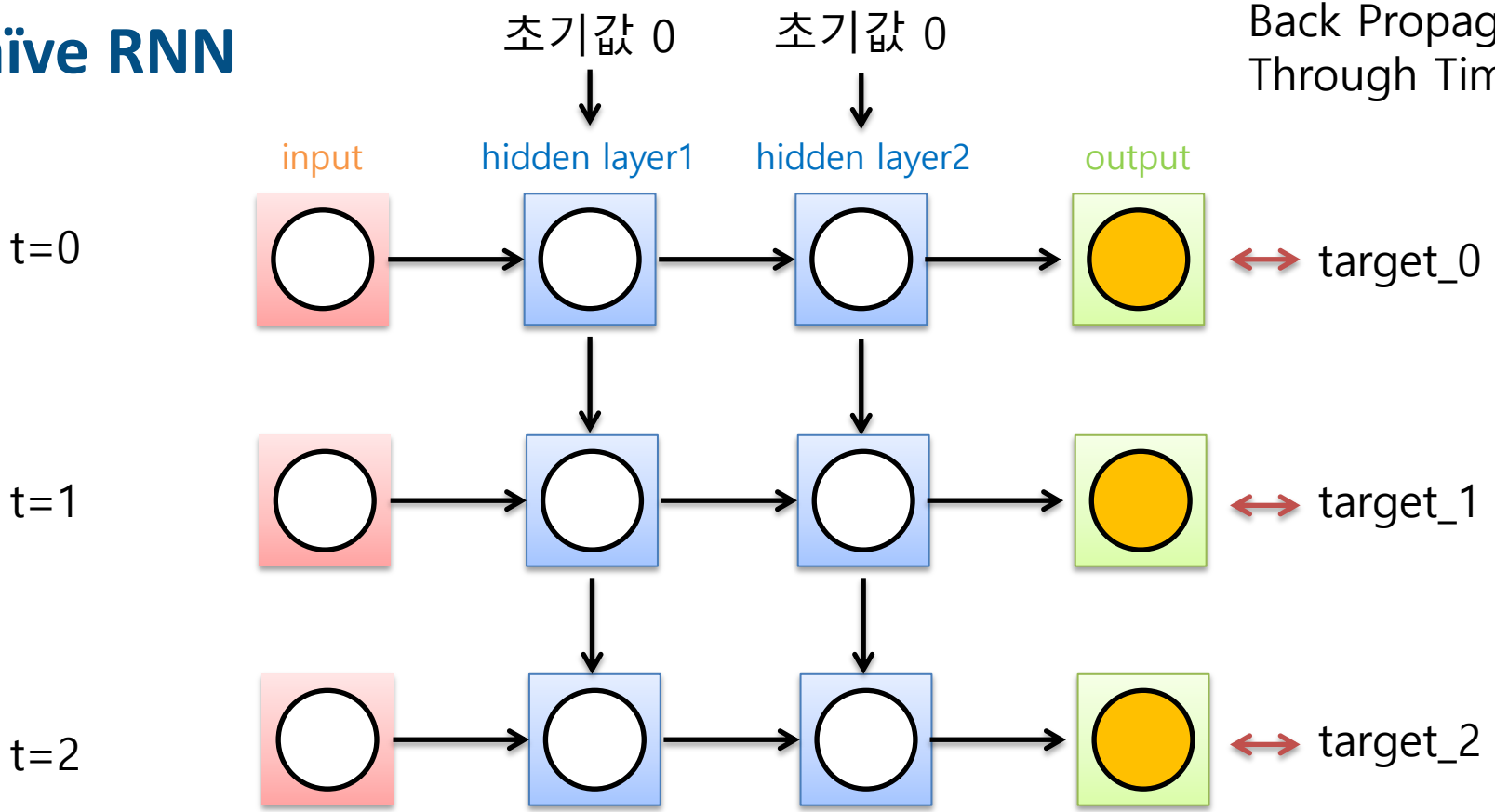
그렇다면 Back Propagation은 어떻게 할까?

Naïve RNN

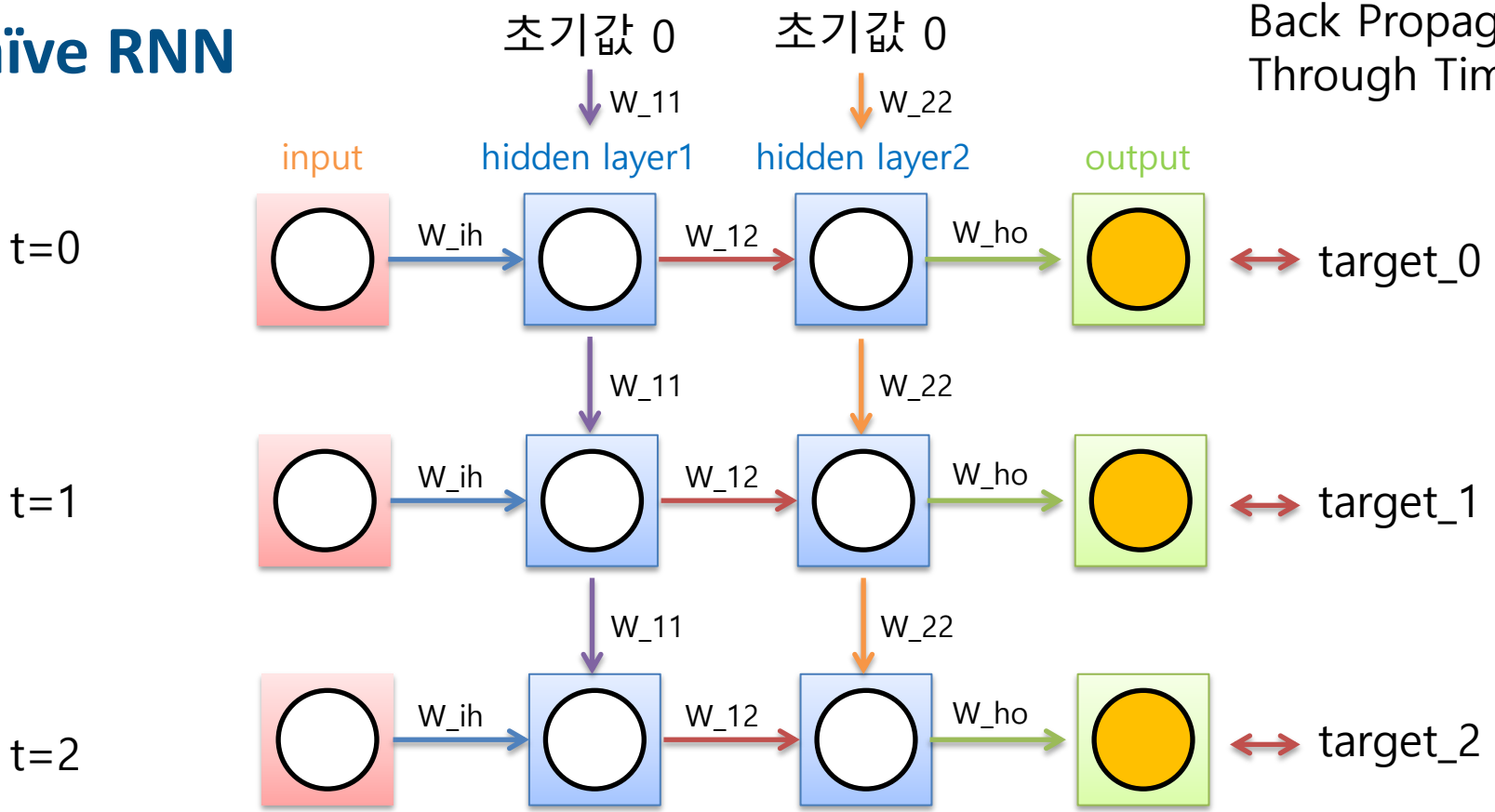
Back Propagation Through Time



Naïve RNN

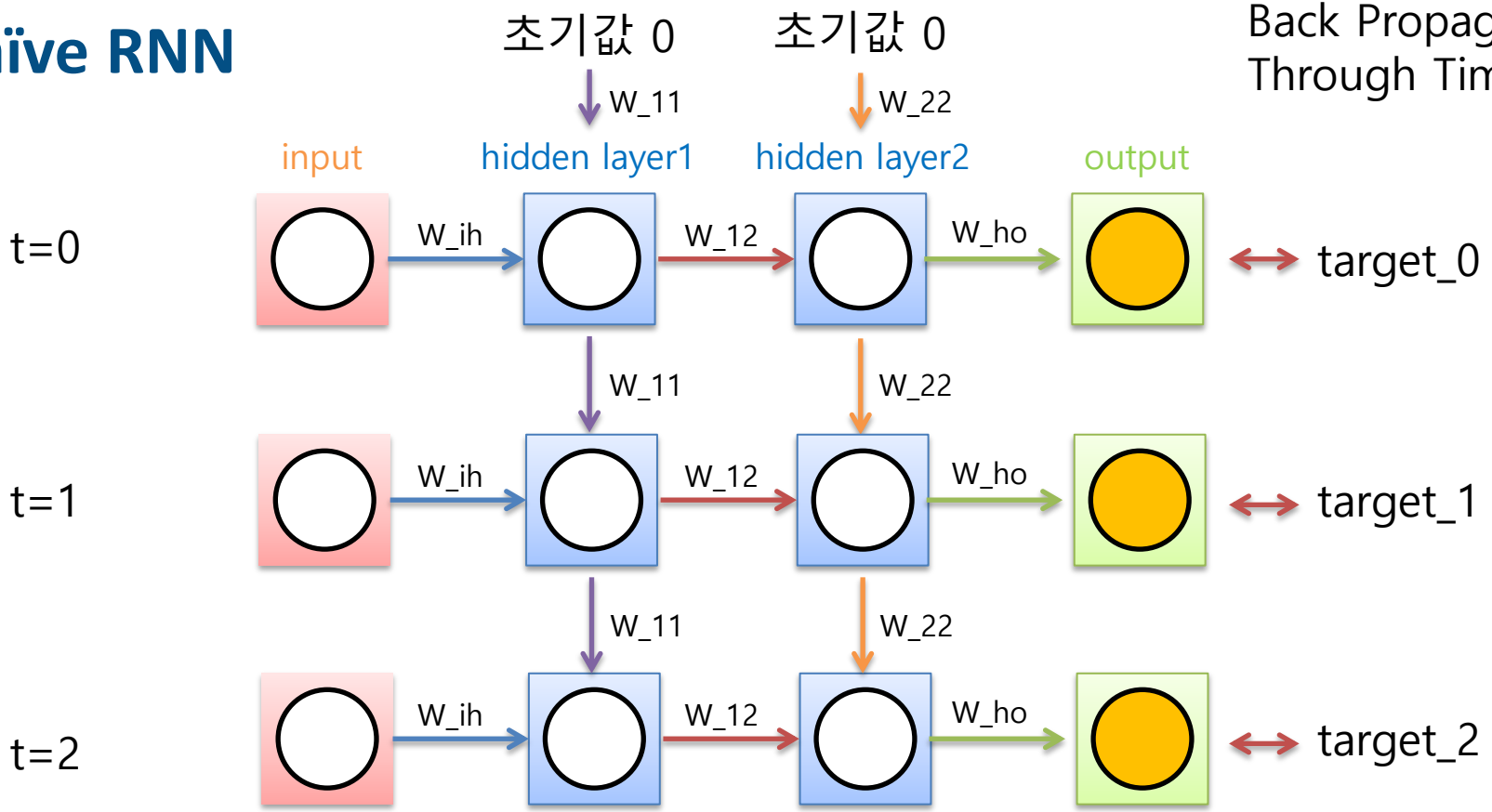

초기값 0
초기값 0
Back Propagation
Through Time
input
hidden layer1
hidden layer2
output
t=0
t=1
t=2
target_0
target_1
target_2

Naïve RNN


초기값 0
초기값 0
Back Propagation Through Time
W_11
W_22
input
hidden layer1
hidden layer2
output
t=0
t=1
t=2
W_ih
W_12
W_ho
target_0
target_1
target_2

Naïve RNN

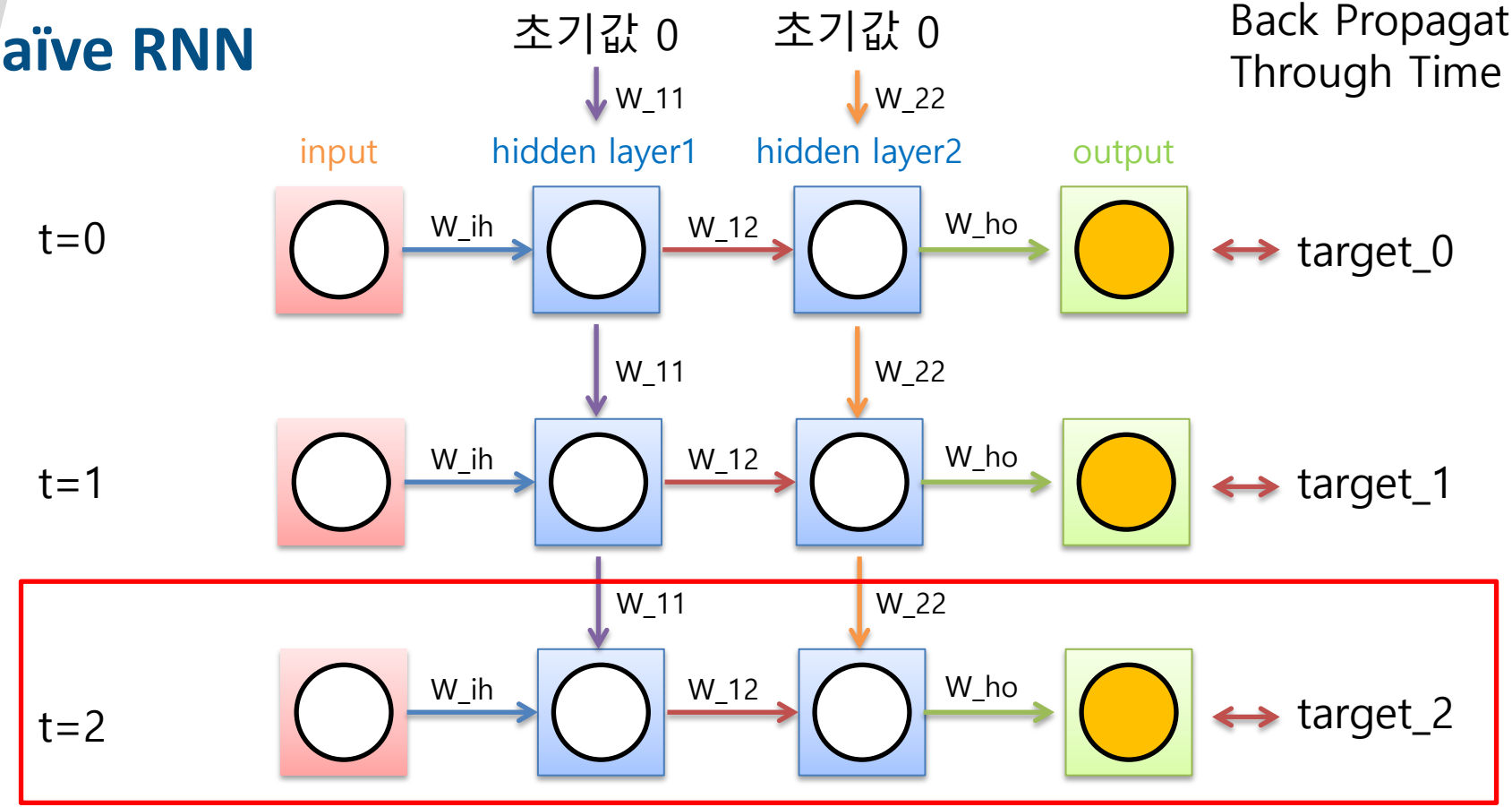
Back Propagation Through Time



Recurrent Network이기 때문에 사실 weight를 공유함

Naïve RNN

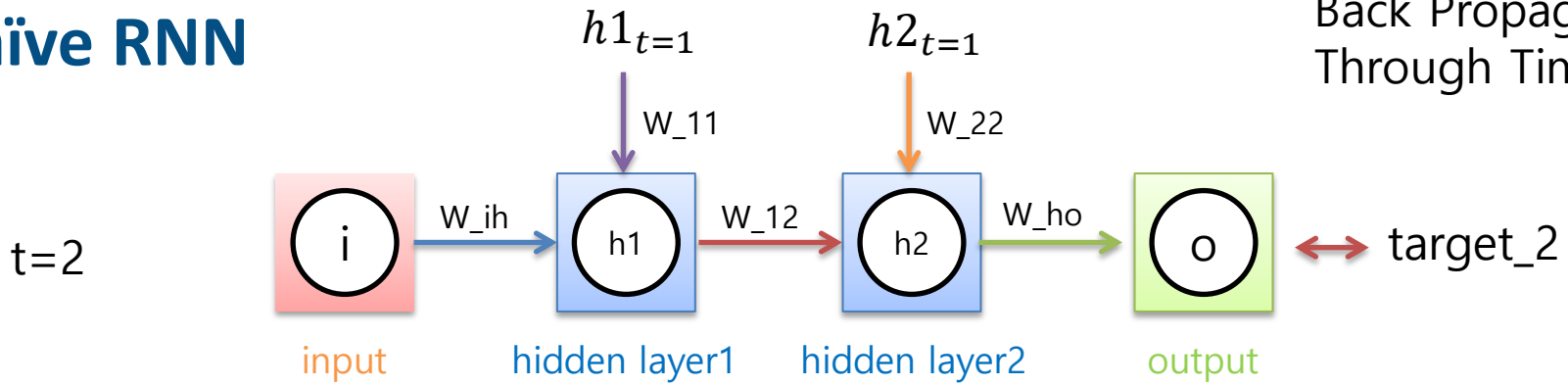
Back Propagation Through Time



Recurrent Network이기 때문에 weight를 공유함

Naïve RNN

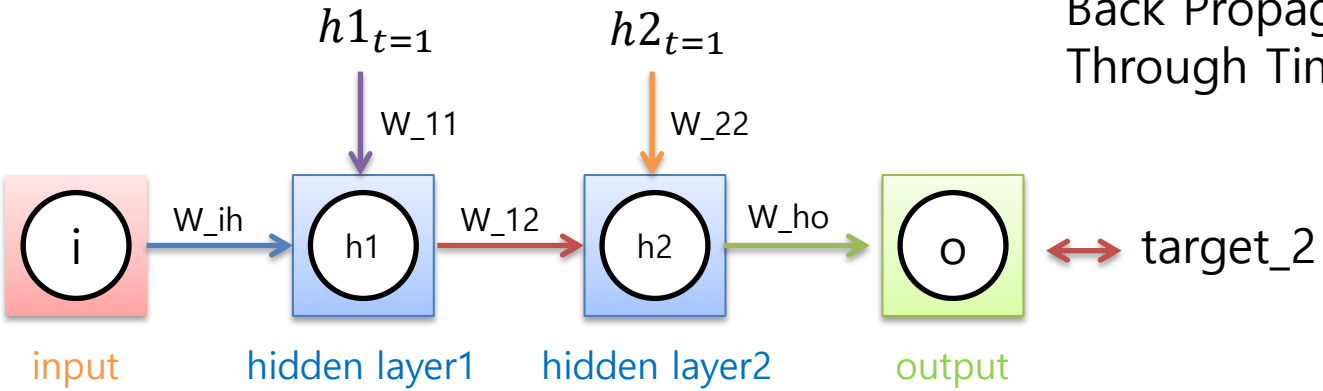
Back Propagation Through Time



Naïve RNN

Back Propagation Through Time

t=2



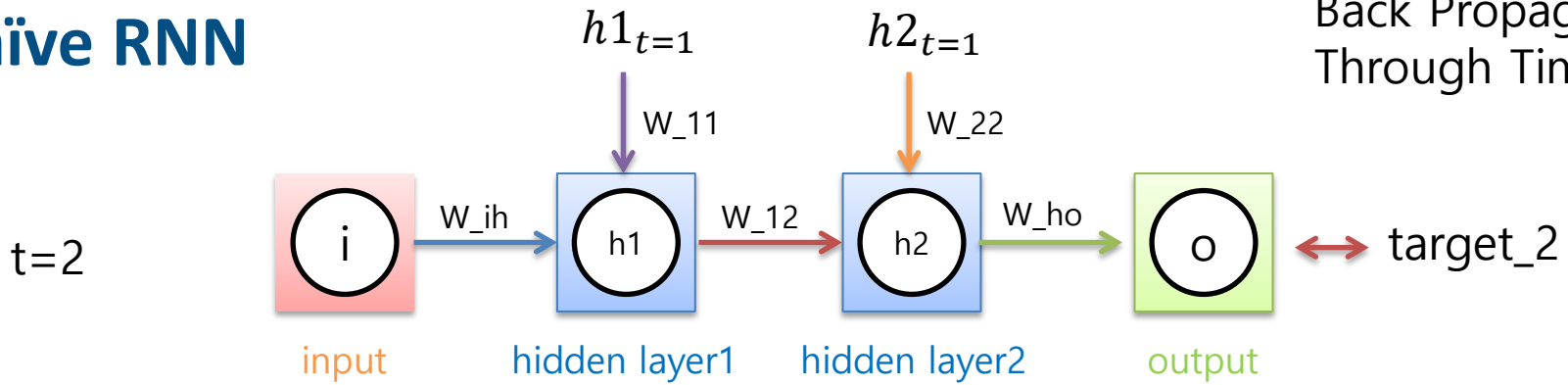
$$o = w_{ho} * h2_{out} + bias$$

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

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

Naïve RNN

Back Propagation Through Time

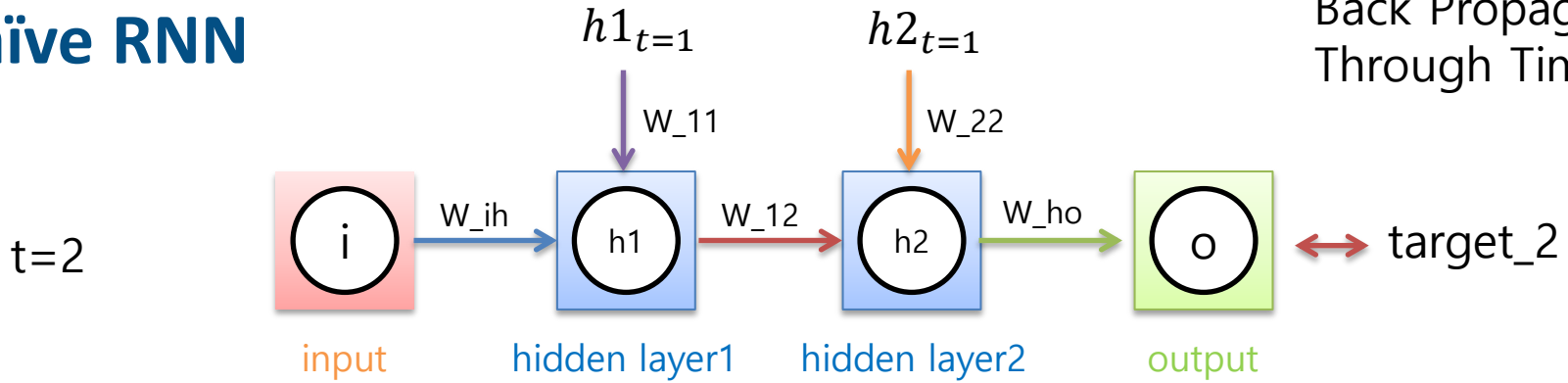


$$o = w_{ho} * h2_{out} + bias$$
$$h2_{out} = tanh(w_{12} * h1 + w_{22} * h2_{t=1} + bias)$$
$$h1_{out} = tanh(w_{ih} * i + w_{11} * h1_{t=1} + bias)$$

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

Naïve RNN

Back Propagation Through Time



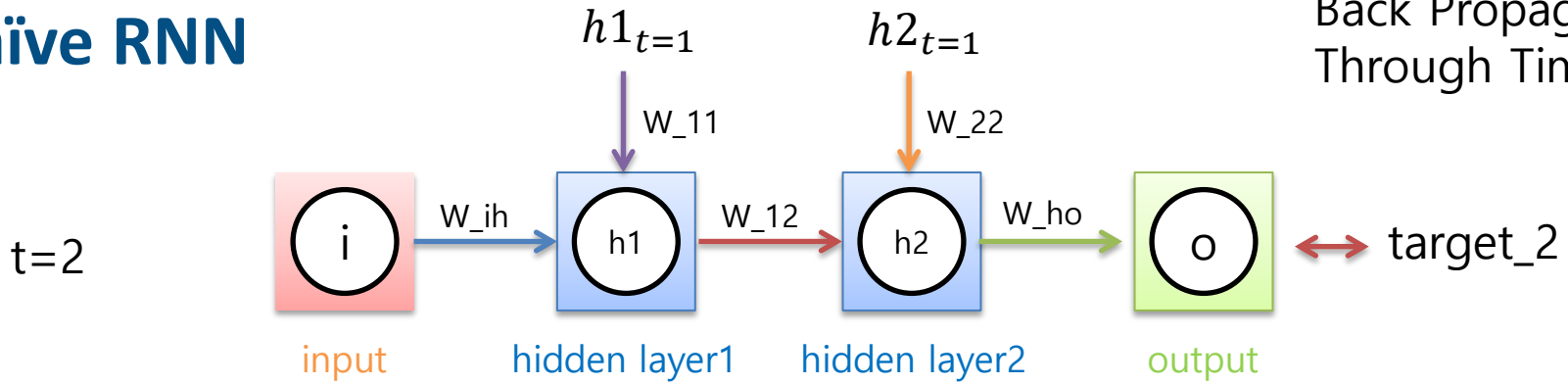
$$o = w_{ho} * h2_{out} + bias$$
$$h2_{out} = tanh(w_{12} * h1 + w_{22} * h2_{t=1} + bias)$$
$$h1_{out} = tanh(w_{ih} * i + w_{11} * h1_{t=1} + bias)$$

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

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

Naïve RNN

Back Propagation Through Time



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

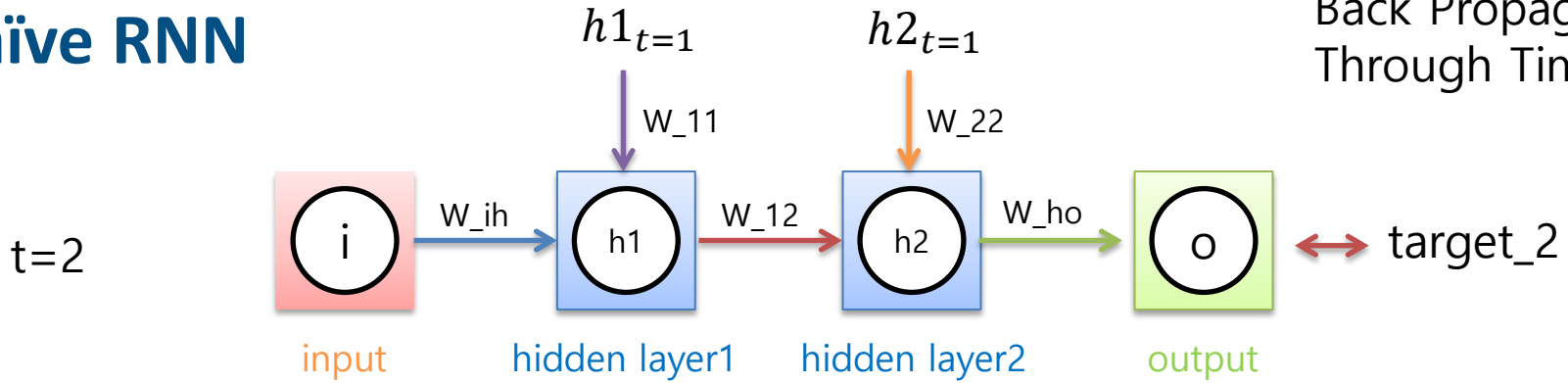
$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$
$$h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$$

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

$h2_{t=1}$

Naïve RNN

Back Propagation Through Time



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

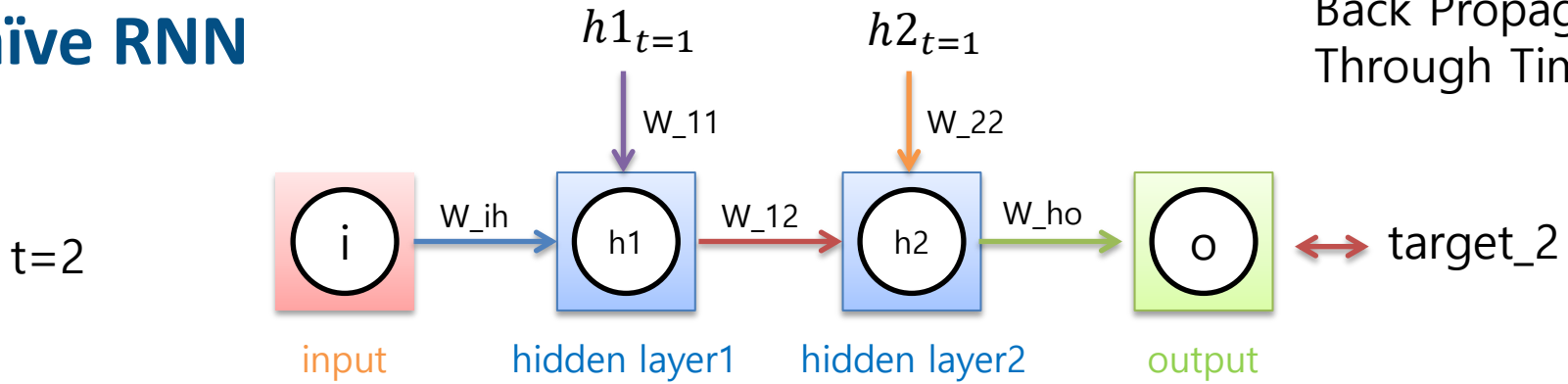
$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$
$$h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$$

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

$h2_{t=1}$

Naïve RNN

Back Propagation Through Time



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

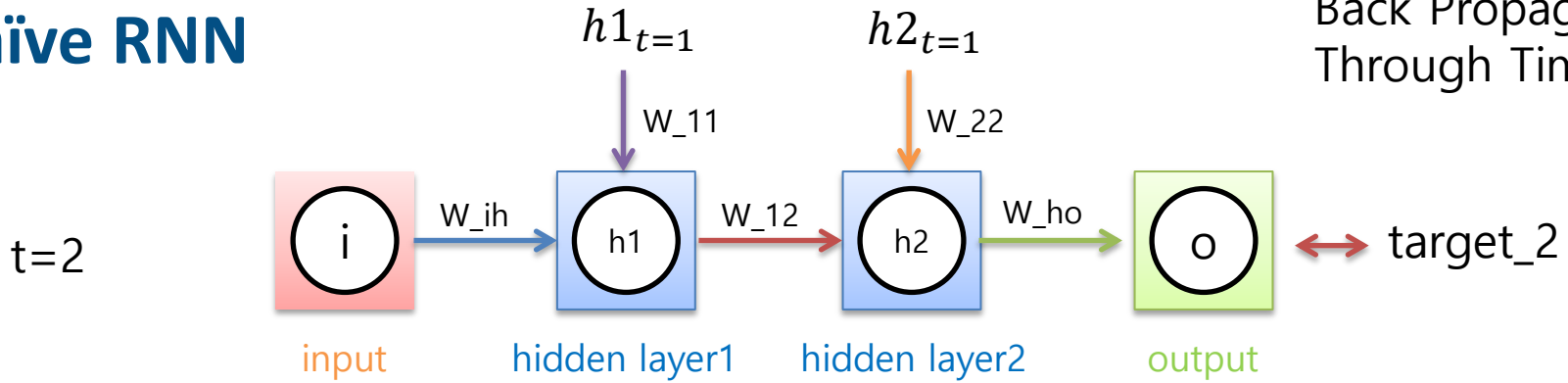
$$h2_{in} = w_{12} * h1 + w_{22} * h2_{t=1} + bias$$
$$h1_{in} = w_{ih} * i + w_{11} * h1_{t=1} + bias$$

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

$h2_{t=1}$

Naïve RNN

Back Propagation Through Time



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

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

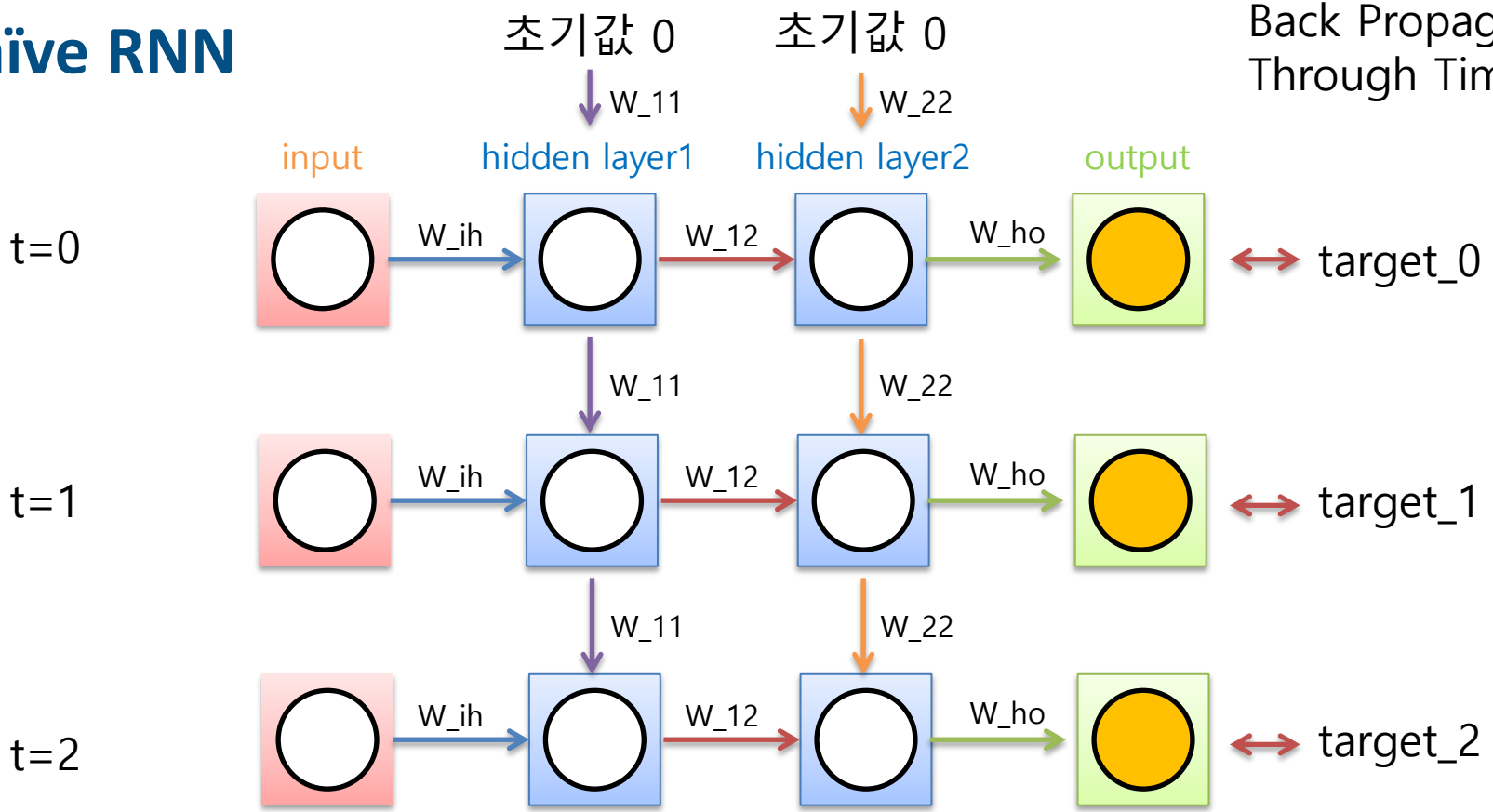
결국 계산하려면
t=0까지 가야 함

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

$h2_{t=1}$

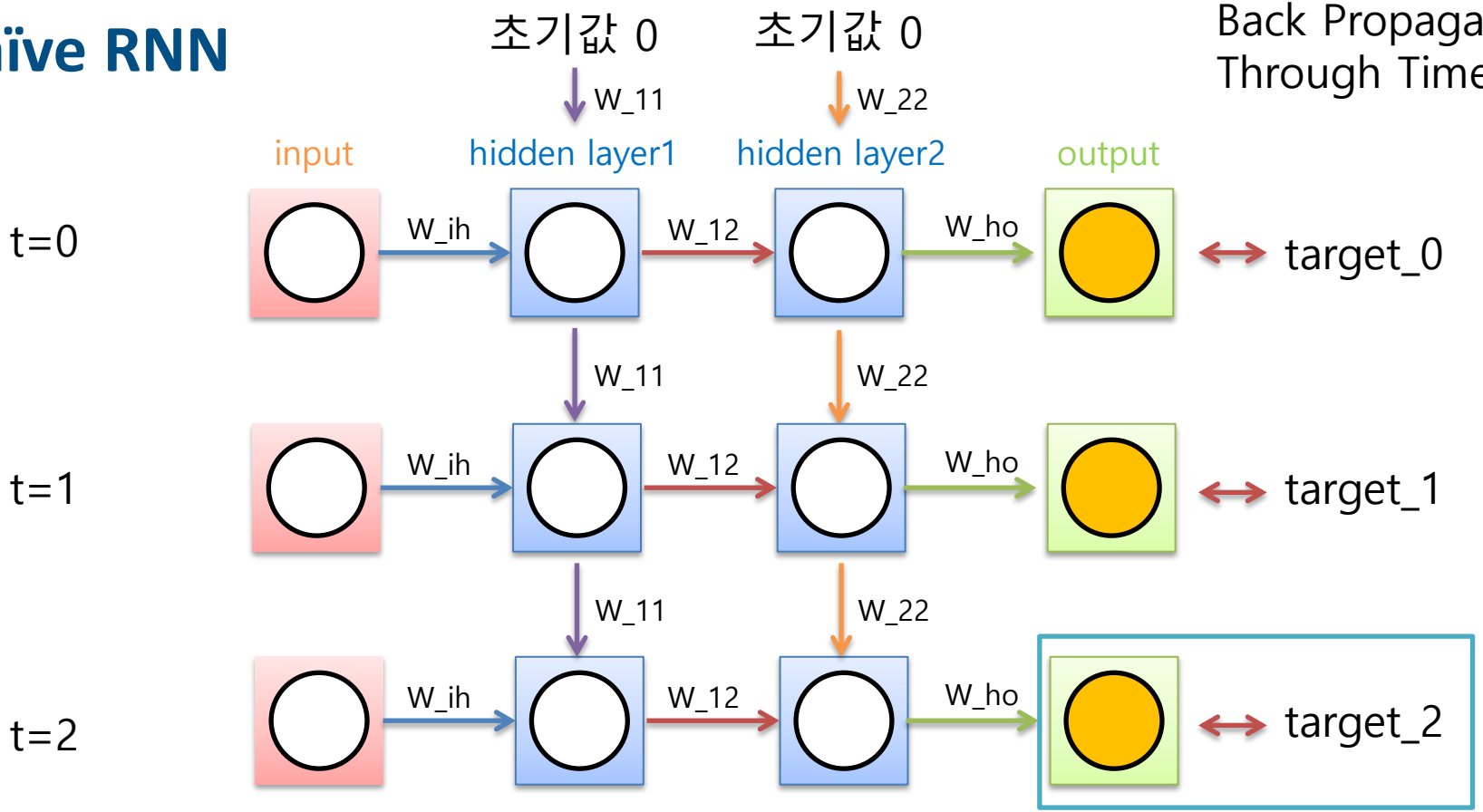
Naïve RNN

Back Propagation Through Time



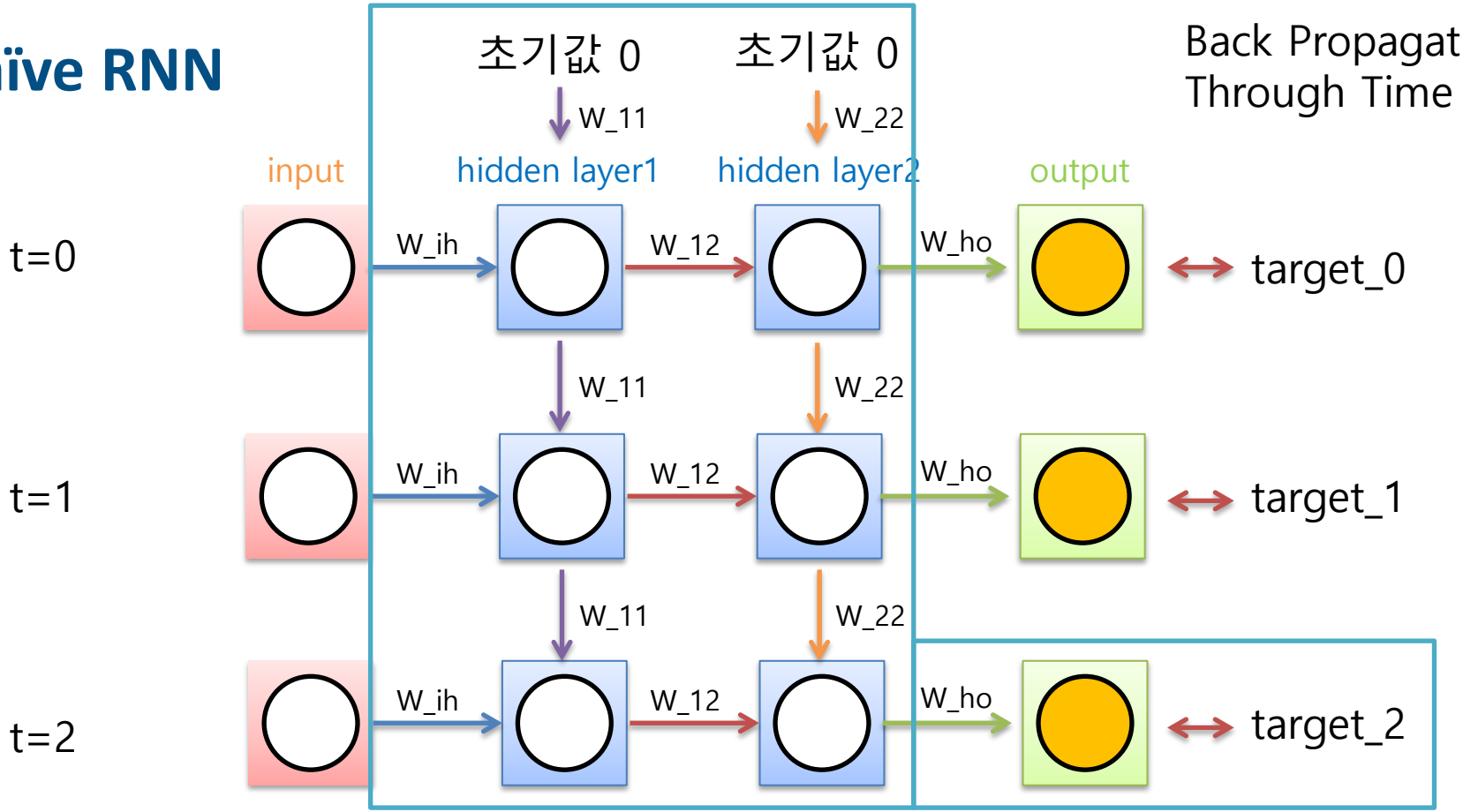
Naïve RNN

Back Propagation Through Time



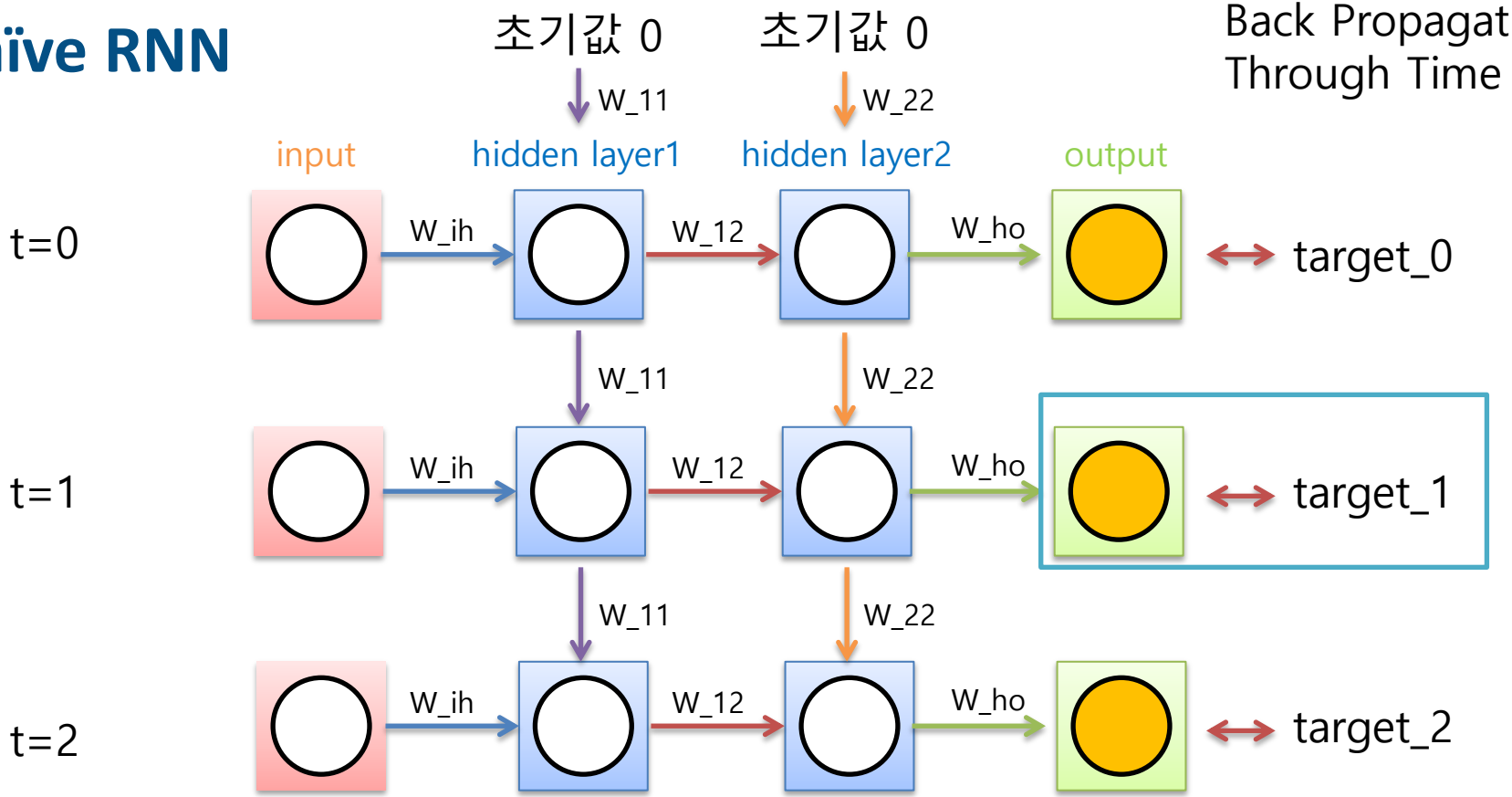
Naïve RNN

Back Propagation Through Time



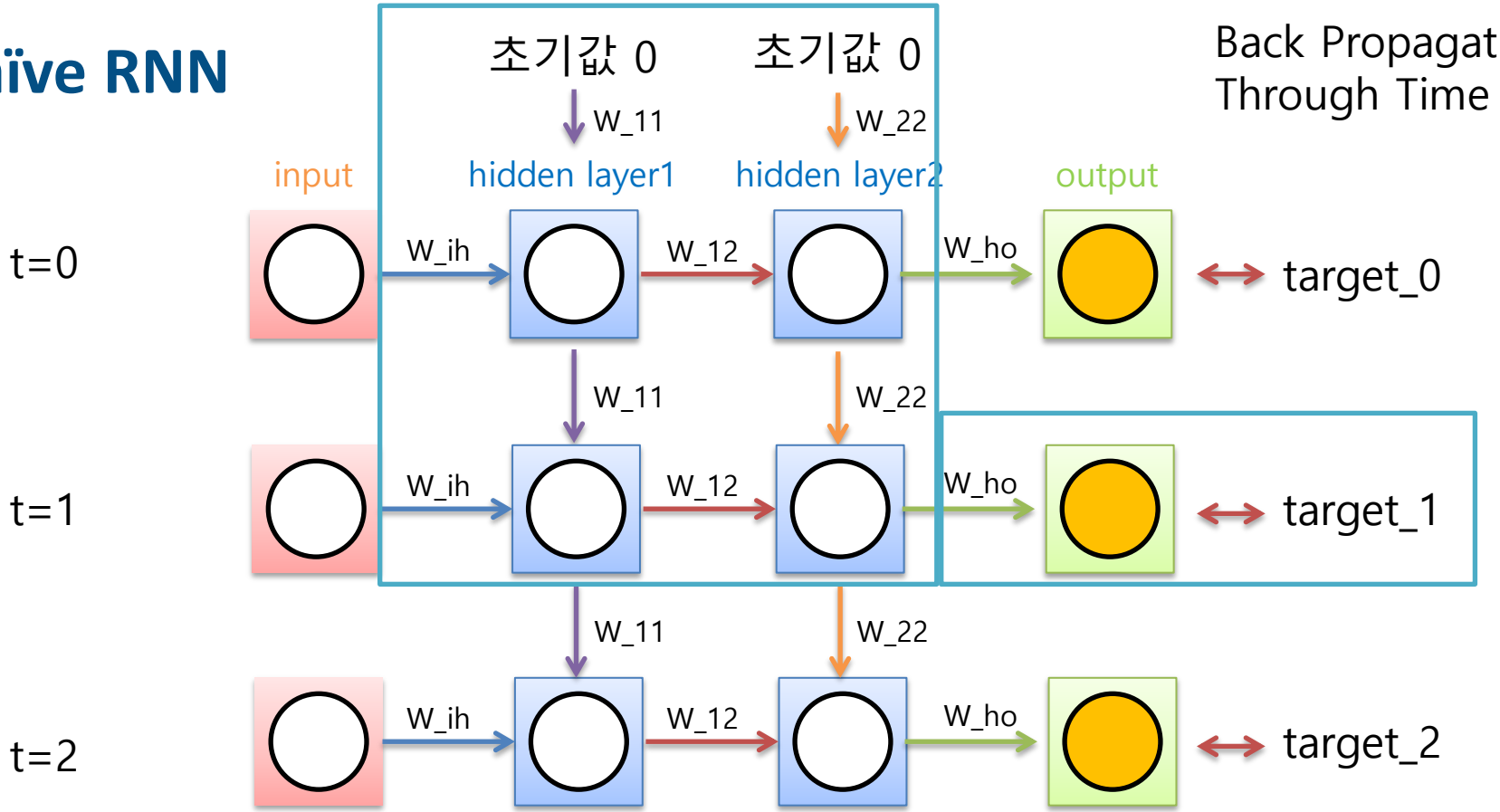
Naïve RNN

Back Propagation Through Time

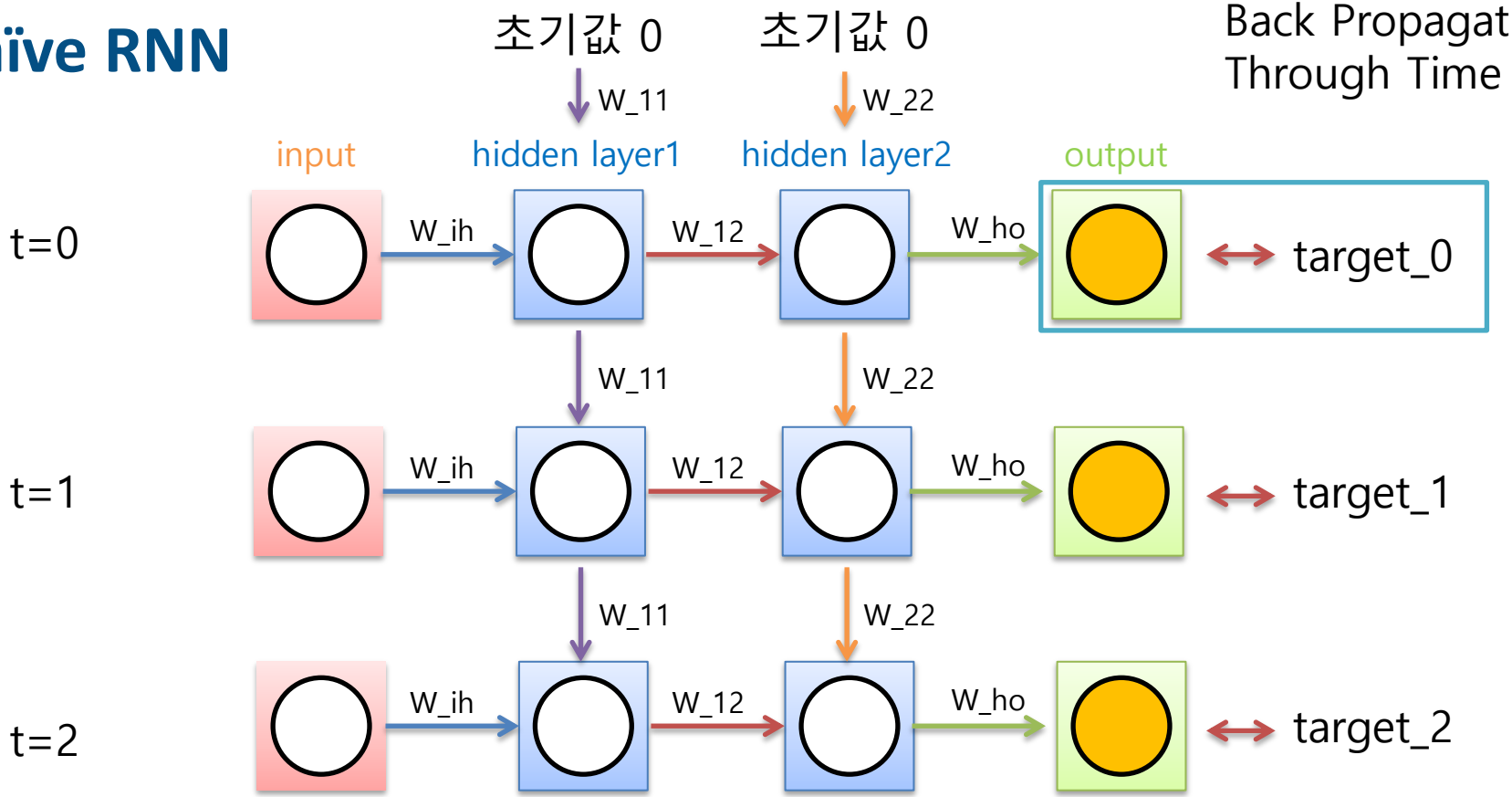


Naïve RNN

Back Propagation Through Time



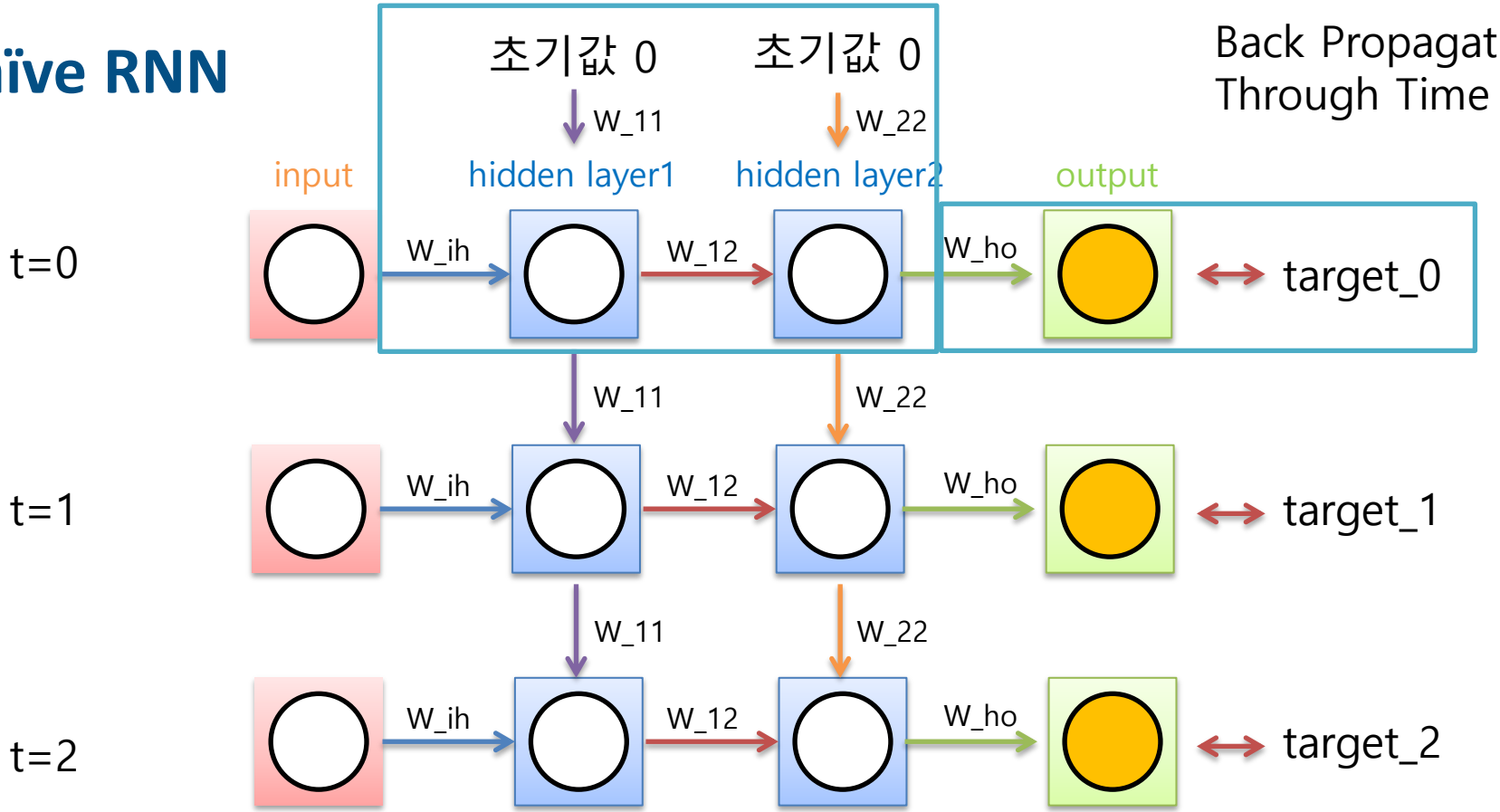
Naïve RNN



Back Propagation Through Time

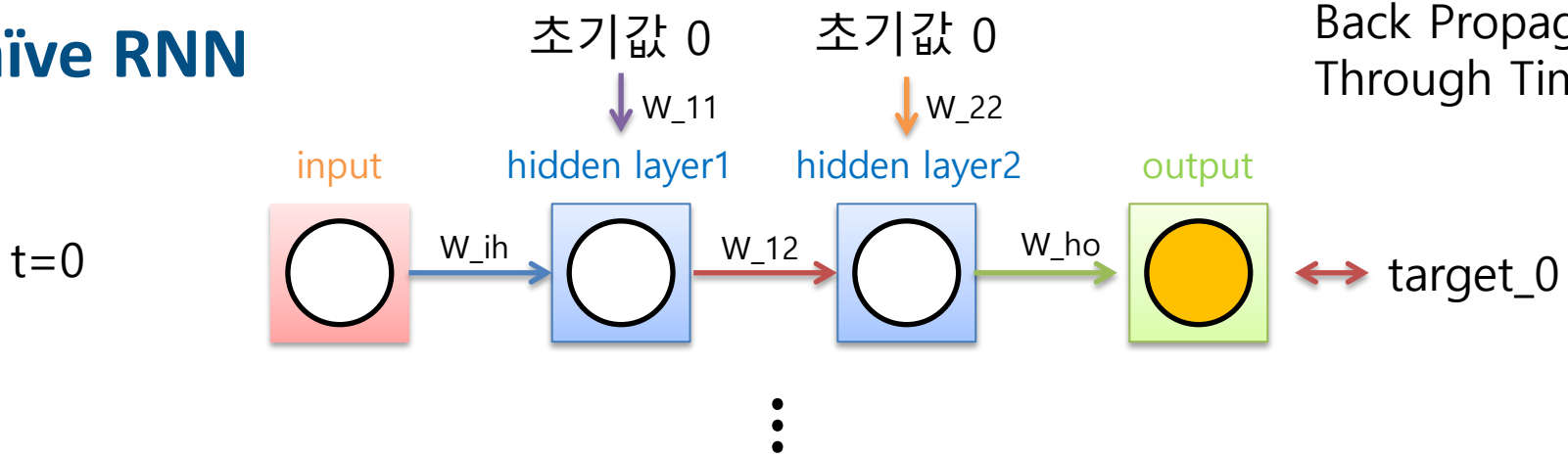
Naïve RNN

Back Propagation Through Time



Naïve RNN

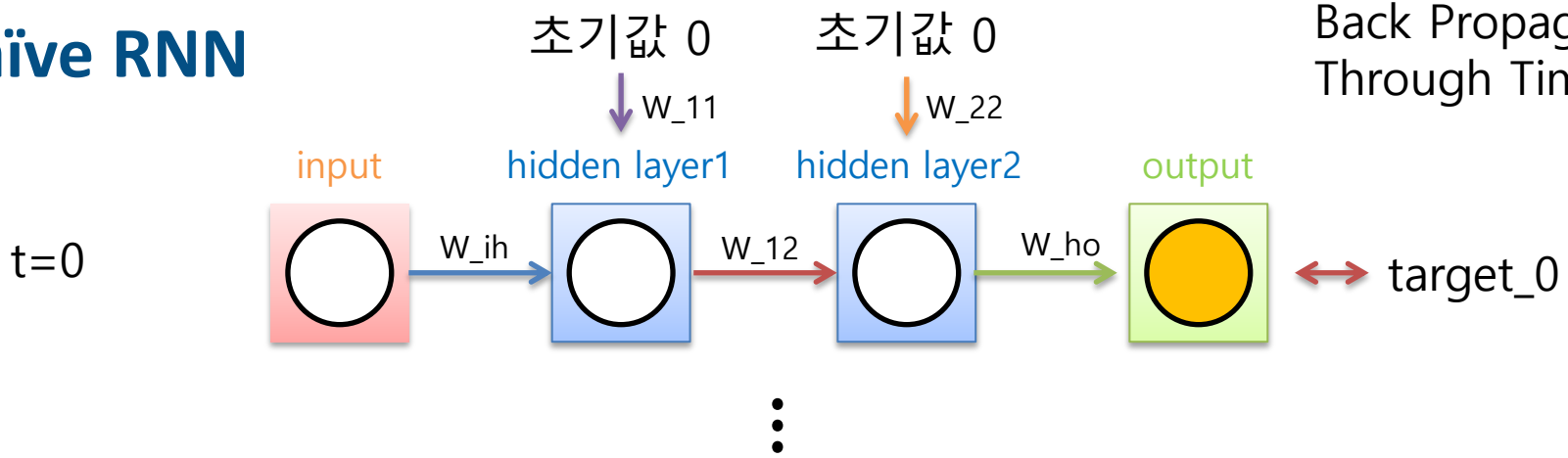
Back Propagation Through Time



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

Naïve RNN

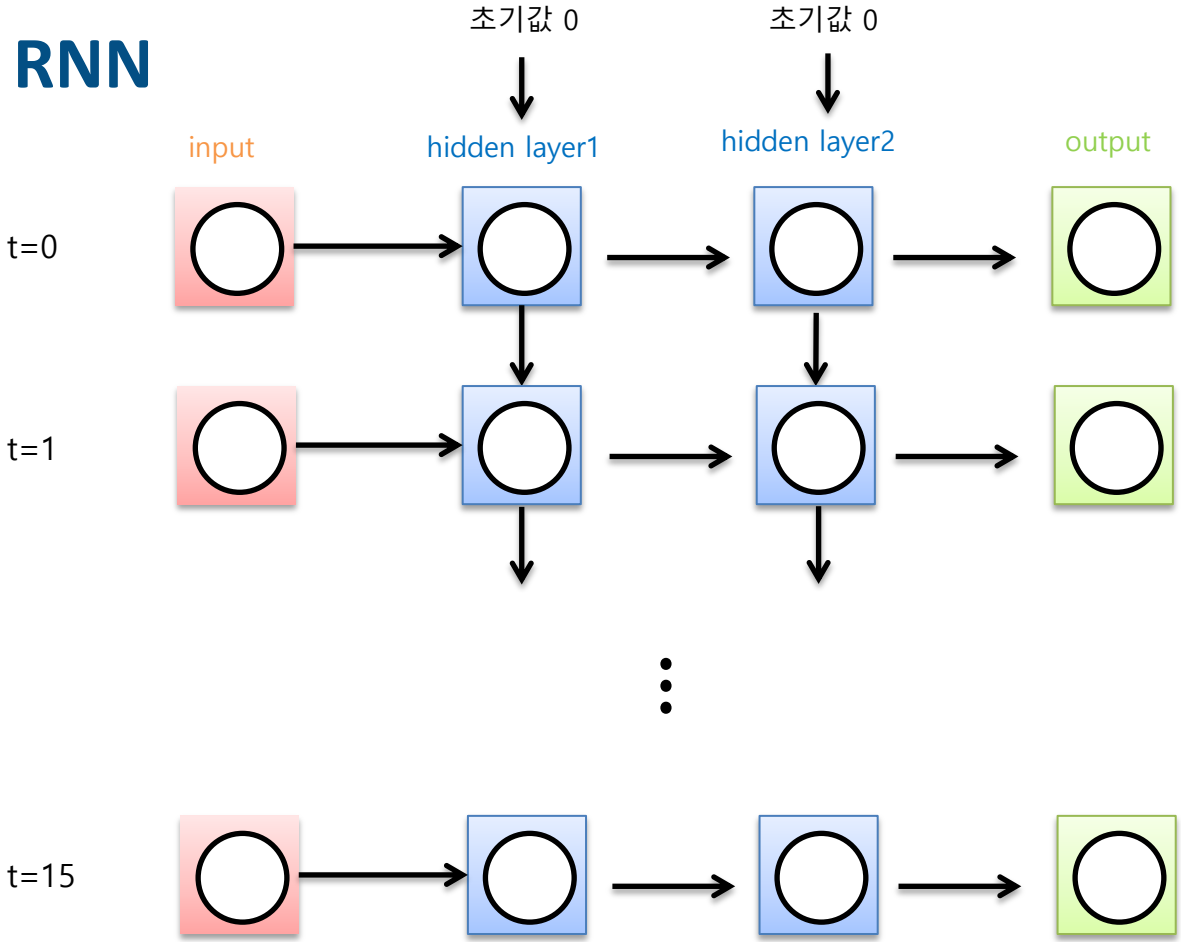
Back Propagation Through Time



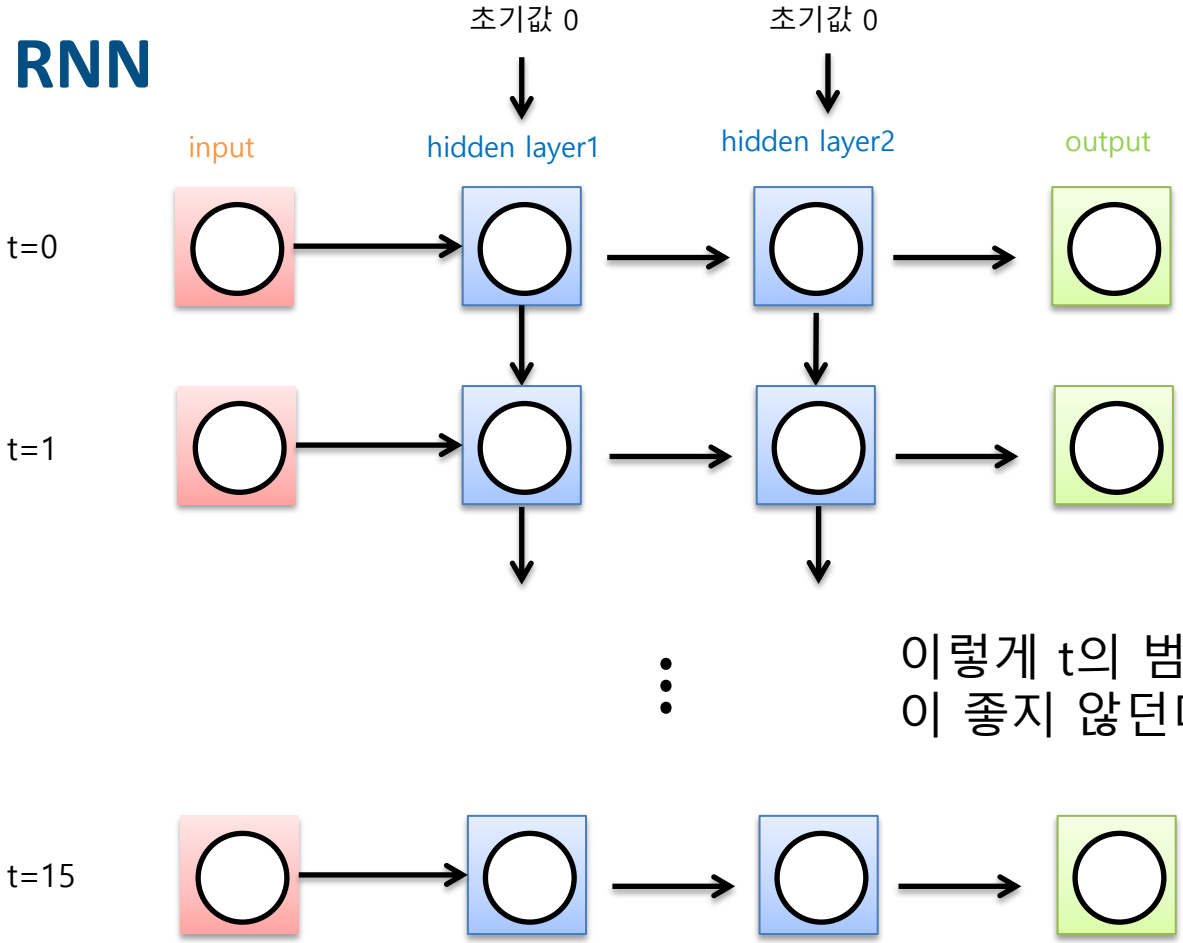
t의 범위에 따라 weight 값들이 반복적으로 gradient가 계산되는데 이를 다 더해놨다가 한번에 업데이트 함

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

Naïve RNN


초기값 0
초기값 0
input
hidden layer1
hidden layer2
output
t=0
t=1
t=15

Naïve RNN



이렇게 t의 범위가 늘어나면 성능이 좋지 않던데 이유가 뭘까?

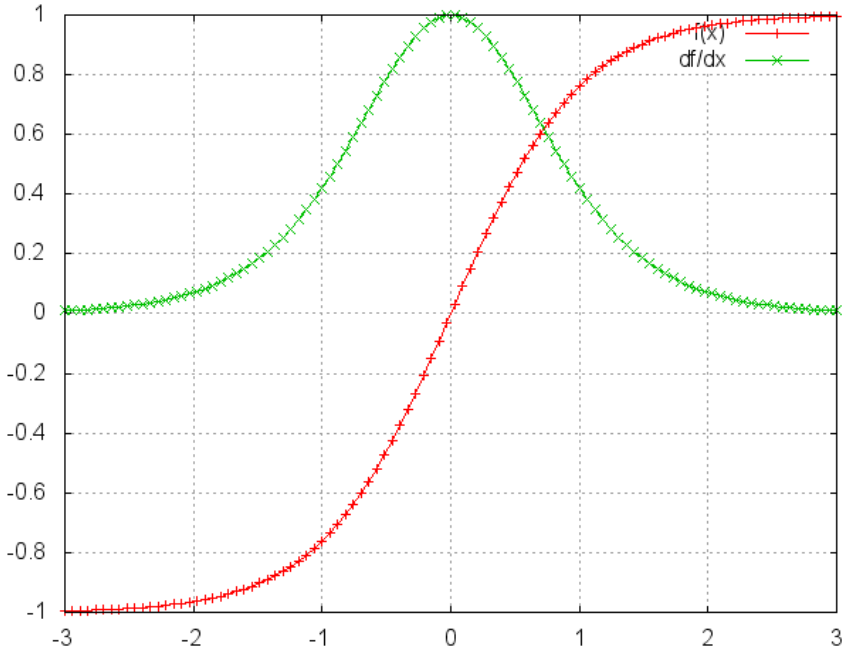
Naïve RNN

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일어나는데, 이를 미분해보면 오른쪽과 같음

(출처:http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

Naïve RNN

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일어나는데, 이를 미분해보면 오른쪽과 같음

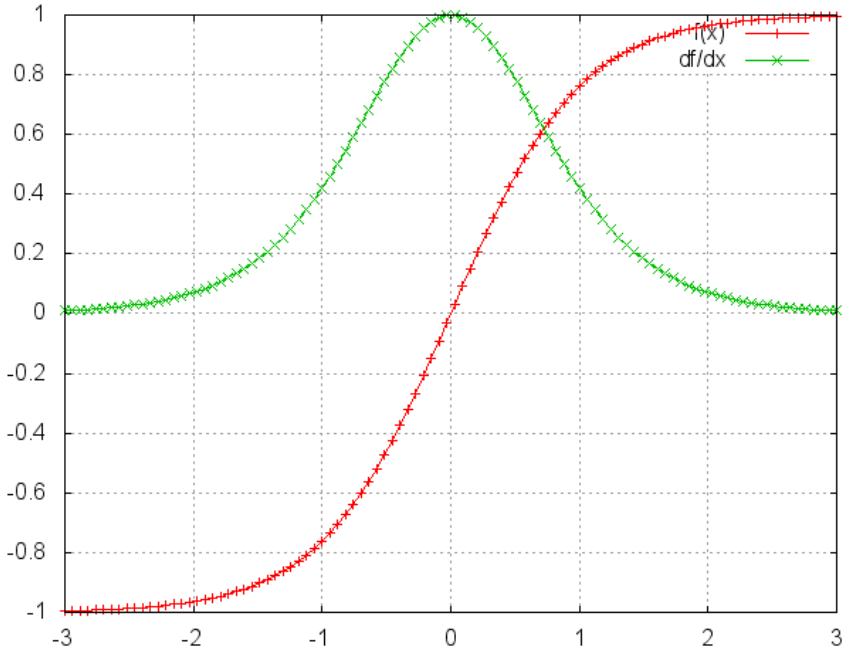


(출처:http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

Naïve RNN

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일어나는데, 이를 미분해보면 오른쪽과 같음

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



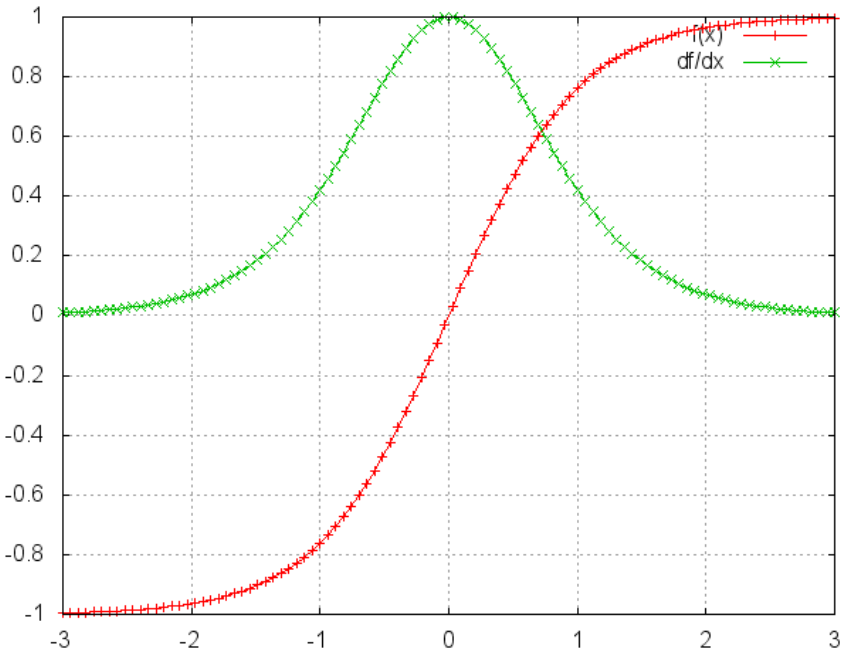
(출처:http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

Naïve RNN

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일어나는데, 이를 미분해보면 오른쪽과 같음

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

어떻게 해야 할까?



(출처:http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

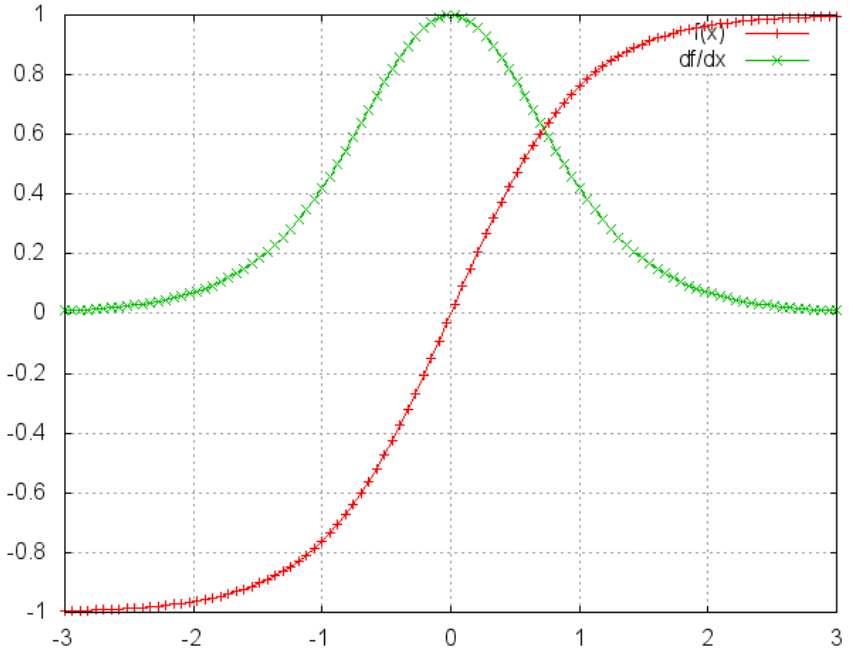
Naïve RNN

Back Propagation이 진행되면서 tanh 함수의 미분도 여러 번 일어나는데, 이를 미분해보면 오른쪽과 같음

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

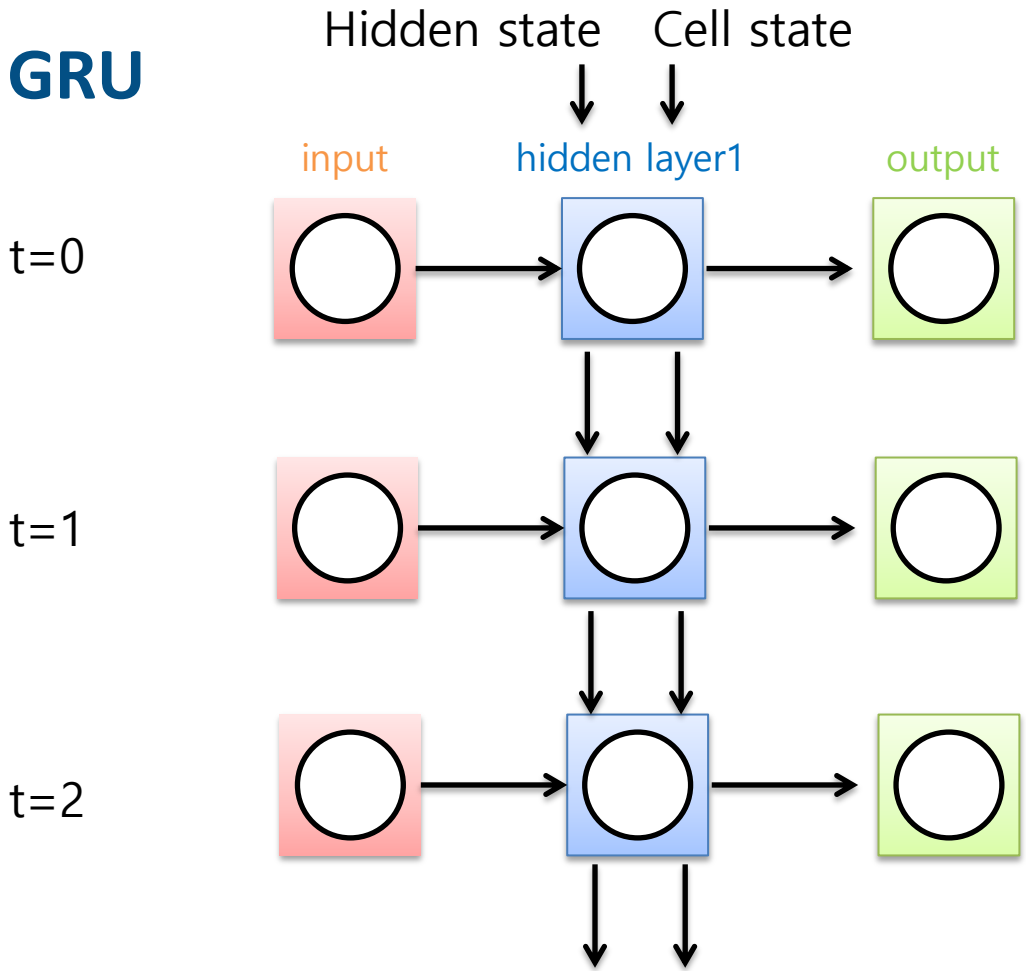
어떻게 해야 할까?

장기기억을 추가하자!

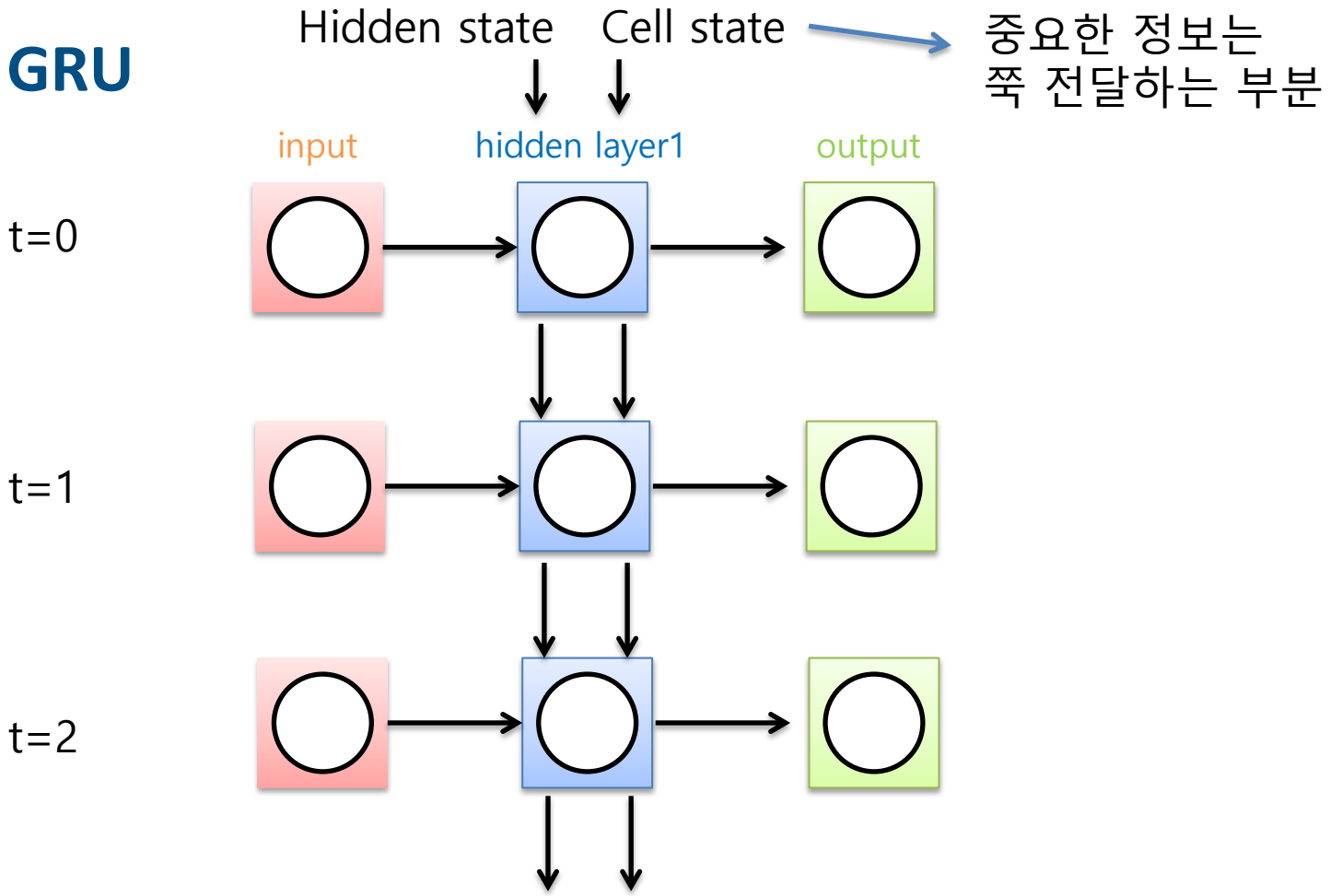


(출처:http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/)

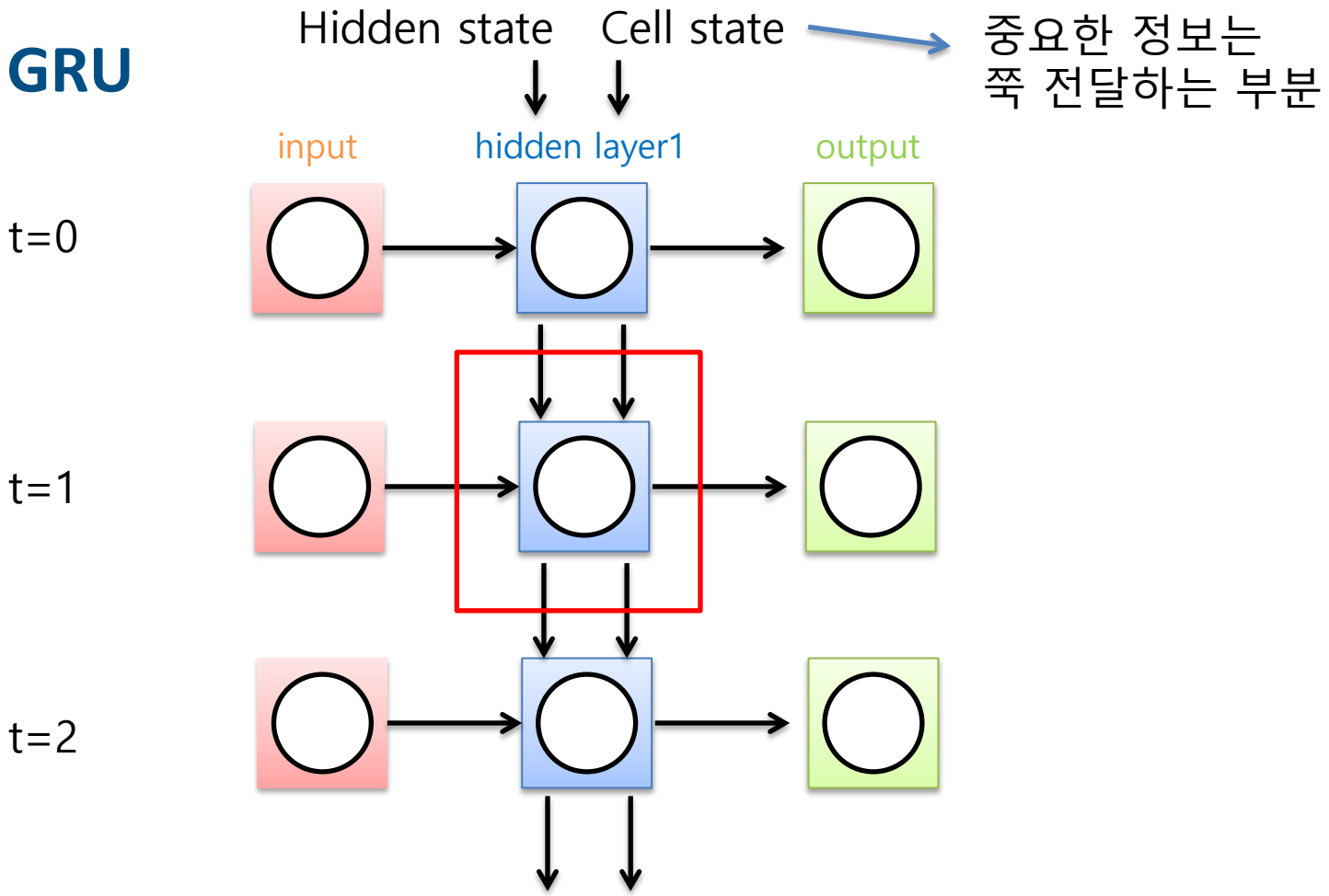
LSTM & GRU



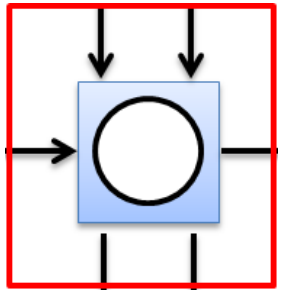
LSTM & GRU


Hidden state
Cell state
중요한 정보는
쪽 전달하는 부분
input
hidden layer1
output
t=0
t=1
t=2

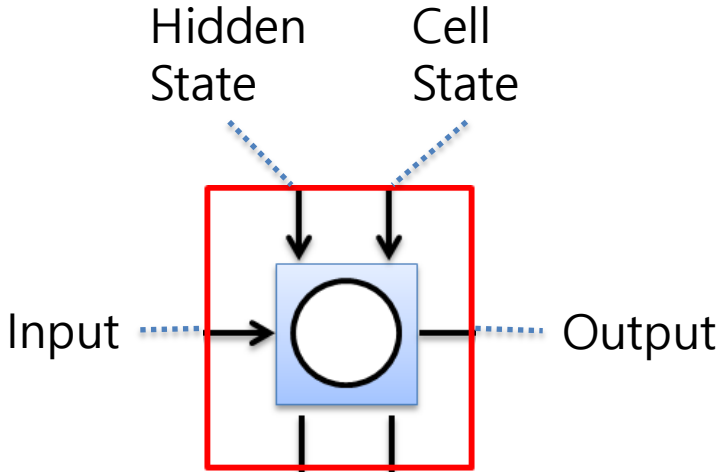
LSTM & GRU


Hidden state
Cell state
중요한 정보는
쭉 전달하는 부분
input
hidden layer1
output
t=0
t=1
t=2

LSTM & GRU

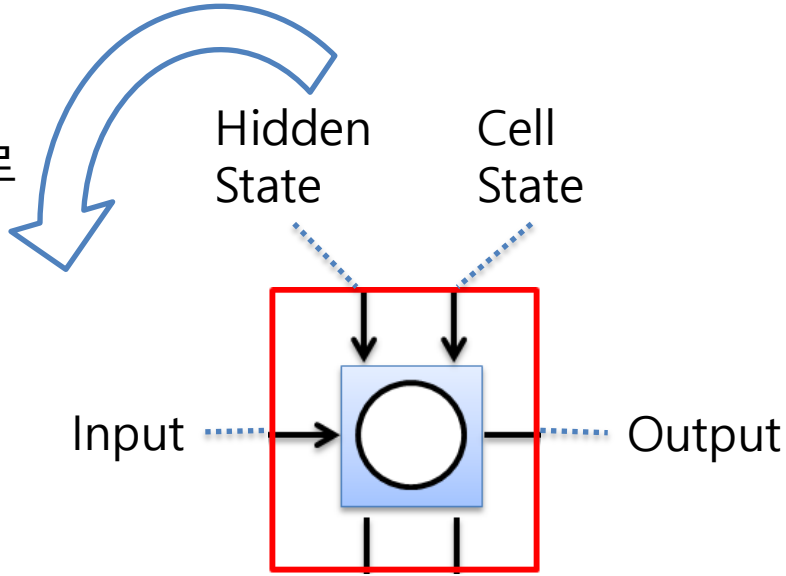


LSTM & GRU

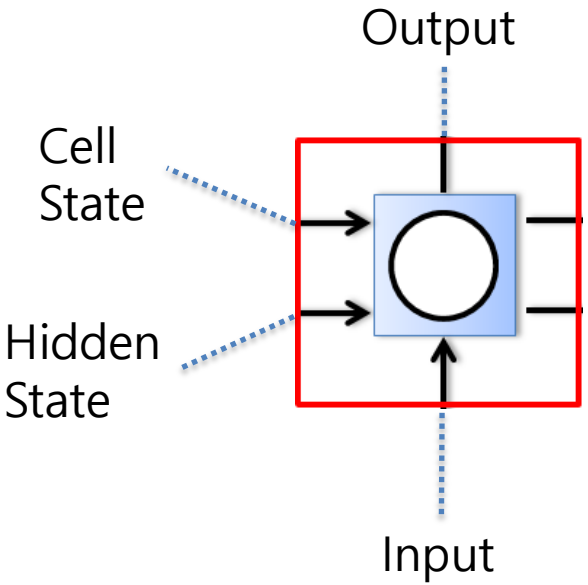


LSTM & GRU

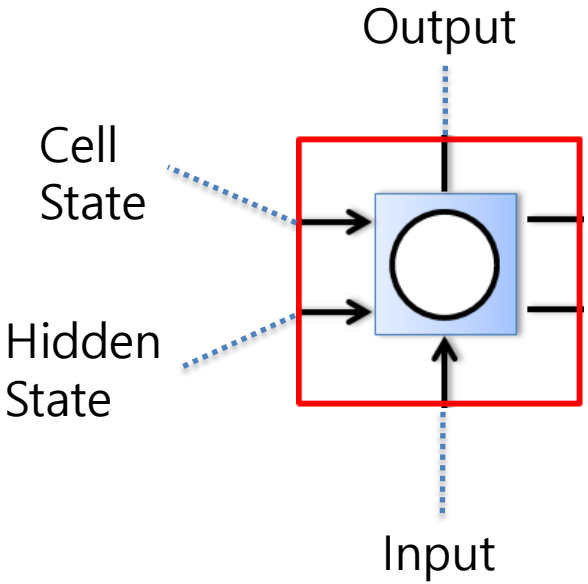
시계 반대 방향으로
90도 회전!



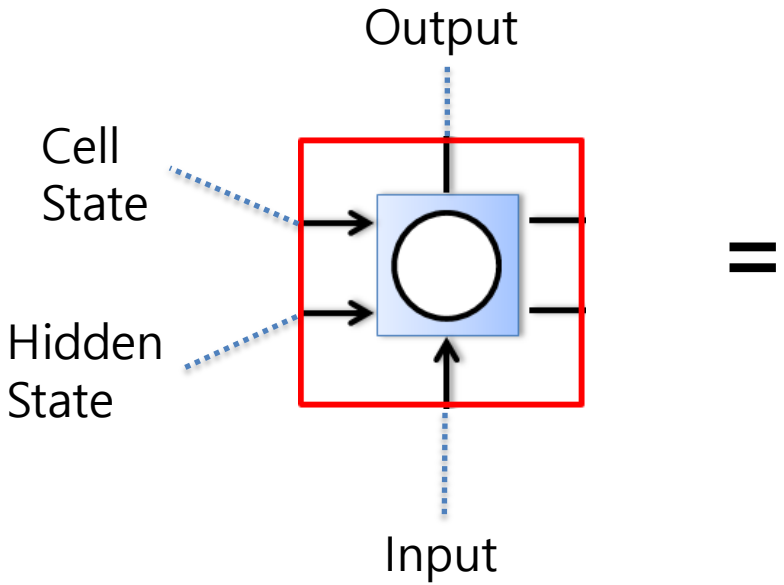
LSTM & GRU



LSTM & GRU

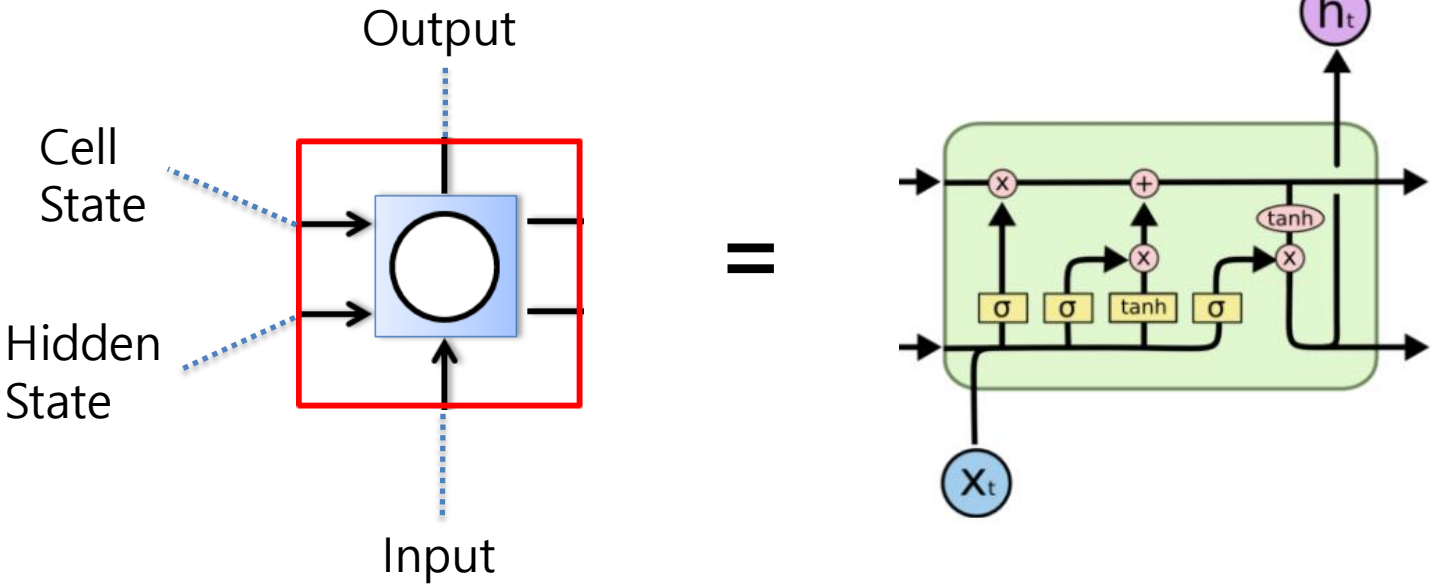


LSTM & GRU



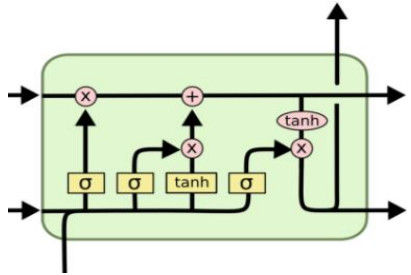
=

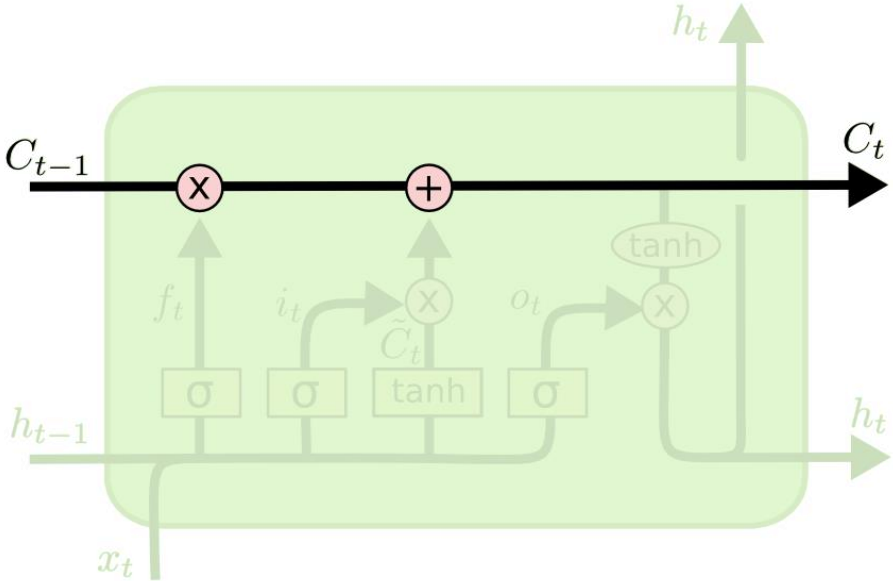
LSTM & GRU



(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU

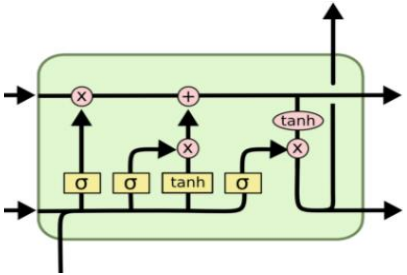


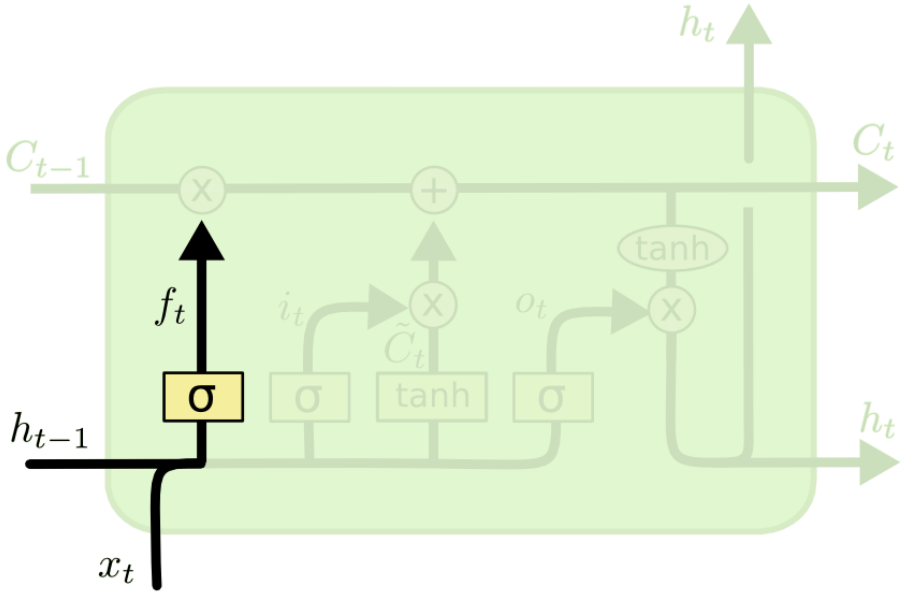


Cell state는 중요 정보를 그대로 쭉 전달하는 역할

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU



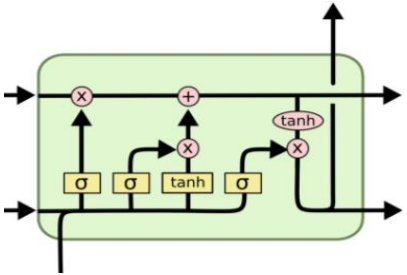


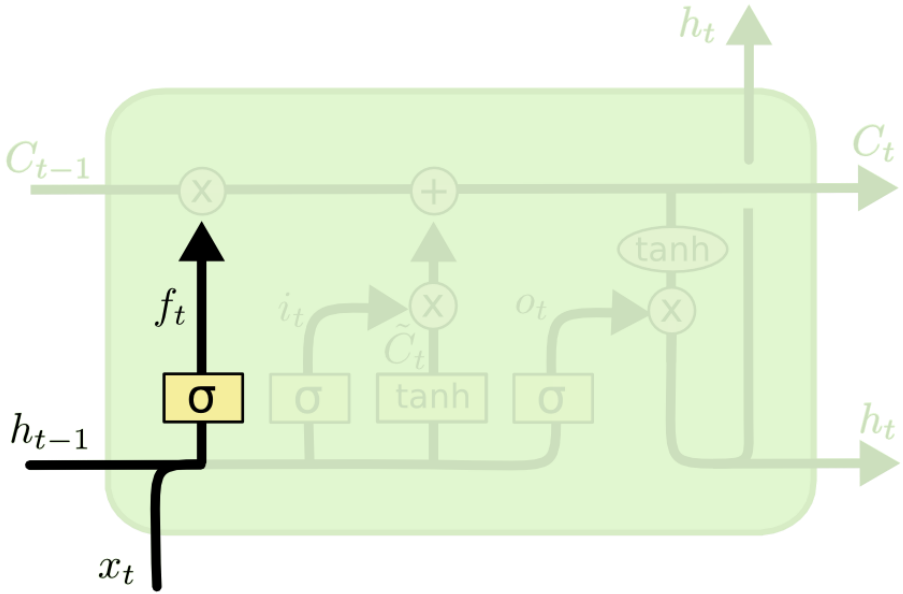
"Forget Gate Layer"

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU





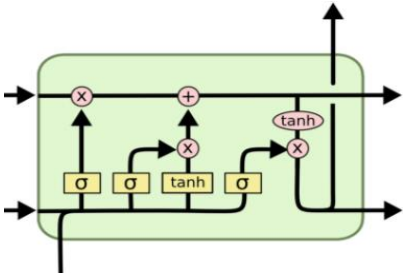
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

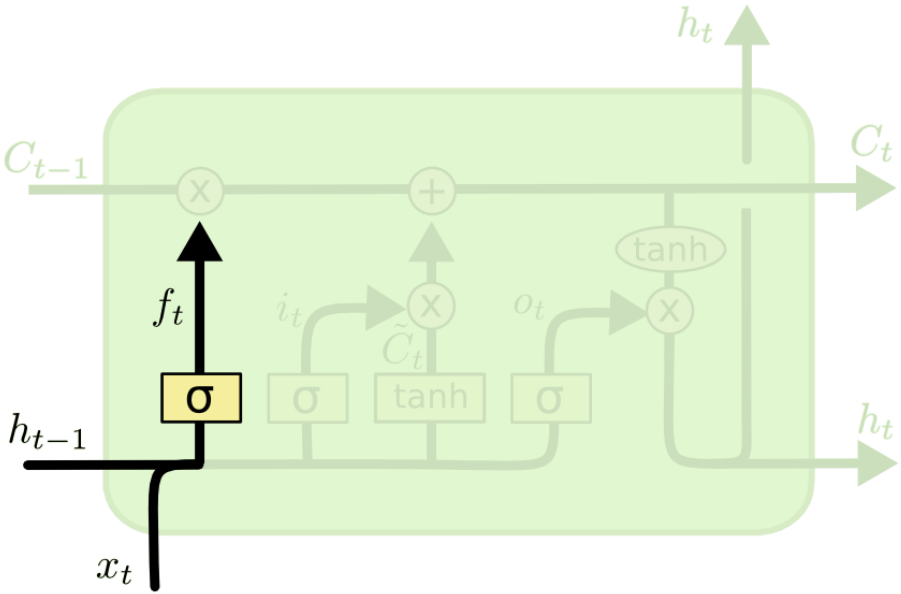
"Forget Gate Layer"

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU





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

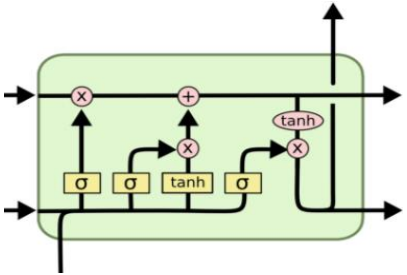
"Forget Gate Layer"

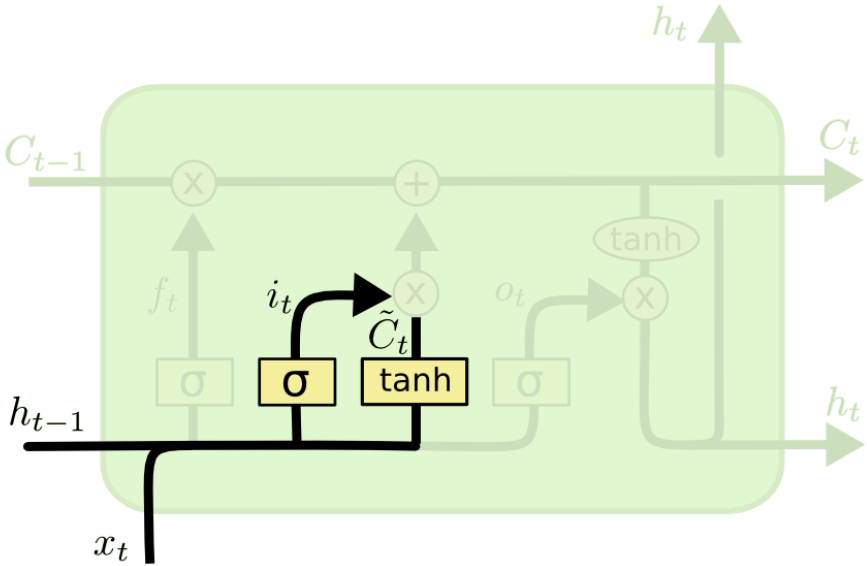
Input과 이전 hidden state를 기준으로 이전의 Cell State를 얼마나 계속 전달할 것인지 결정

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU



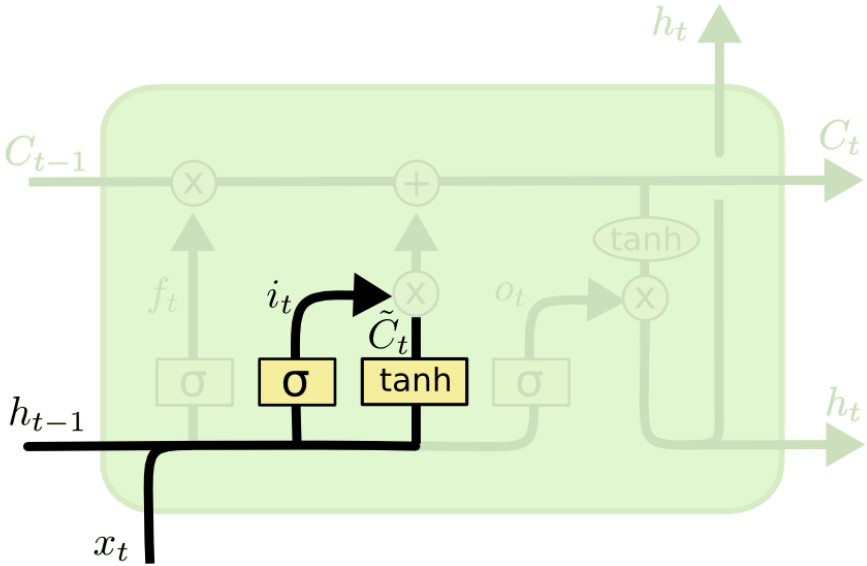


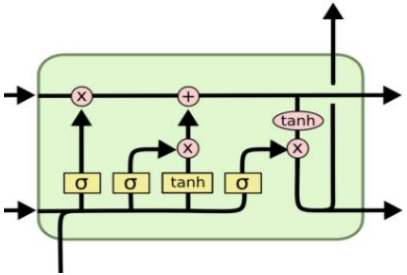
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU





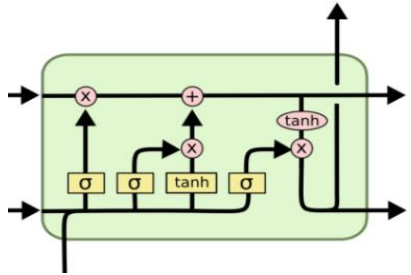
“Input Gate”

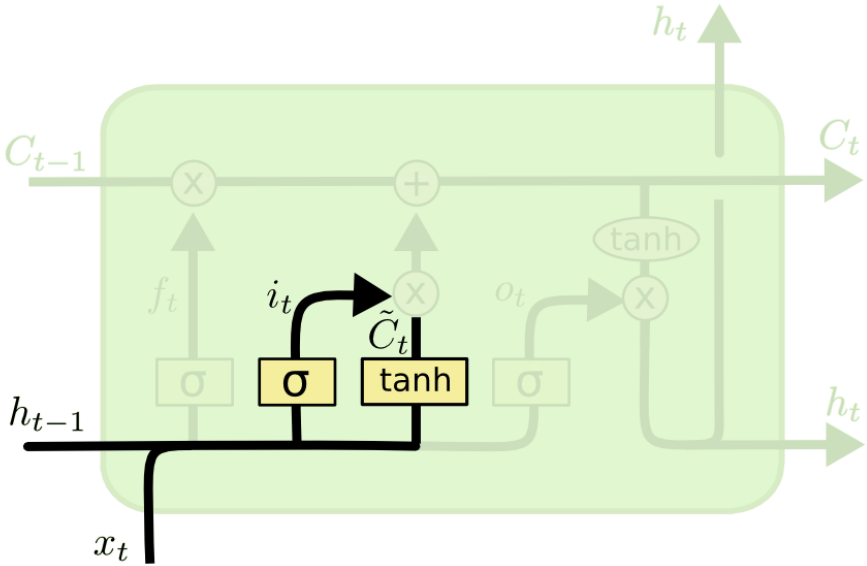
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU





"Input Gate"

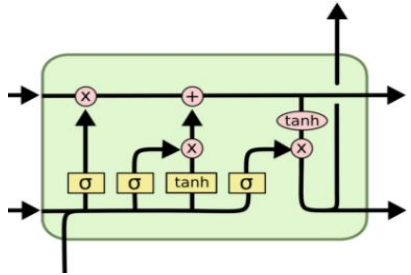
i_t는 hidden state와 input을 통해
Cell state update 비중을 정함

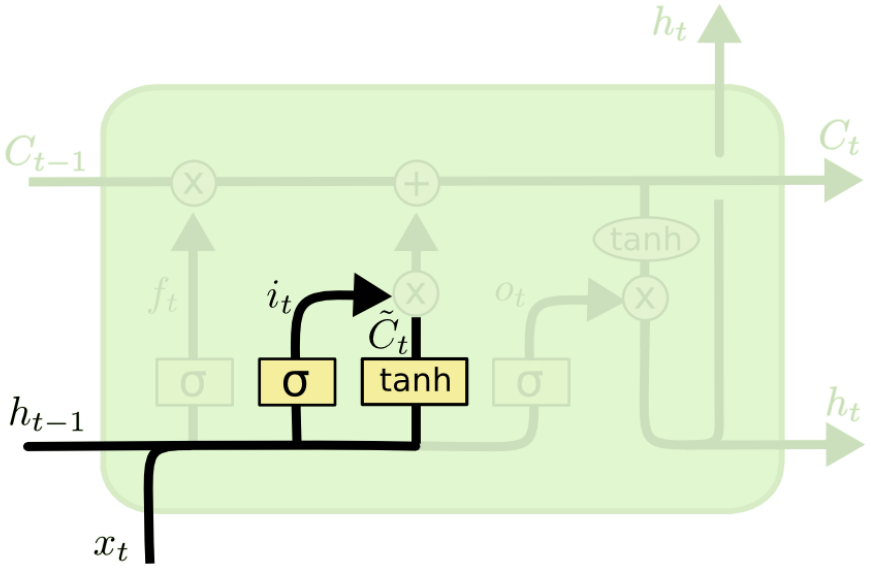
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU





"Input Gate"

i_t는 hidden state와 input을 통해 Cell state update 비중을 정함

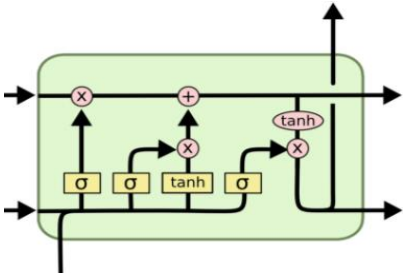
$\tilde{C}_t$는 cell state로 update될 정보를 생성

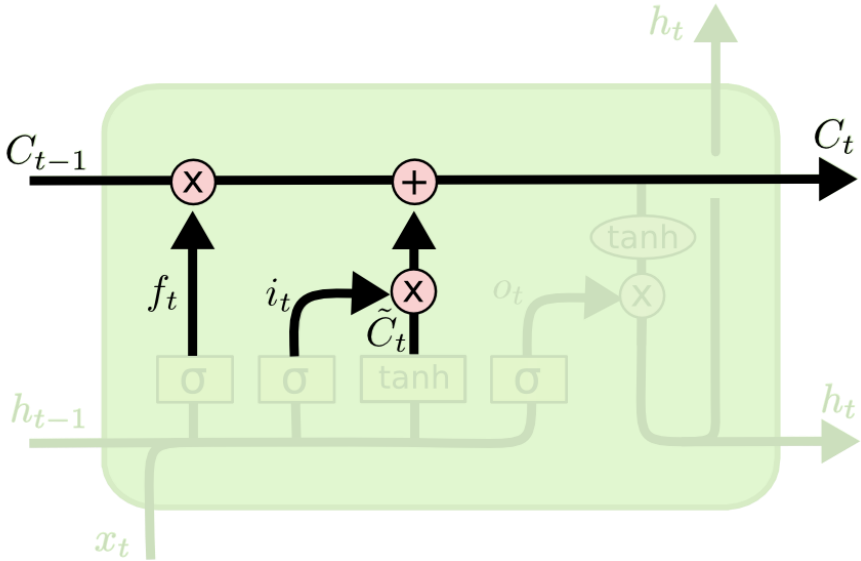
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU

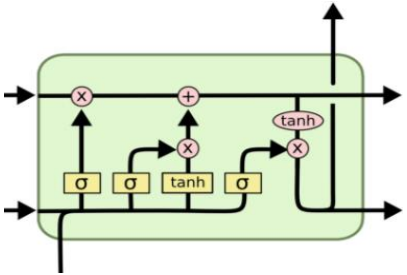


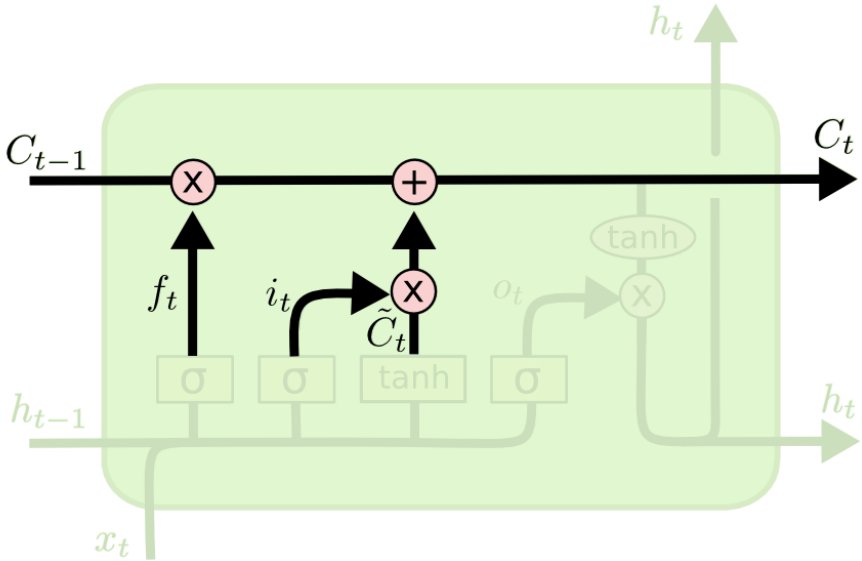


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU



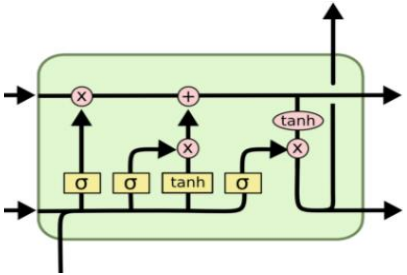


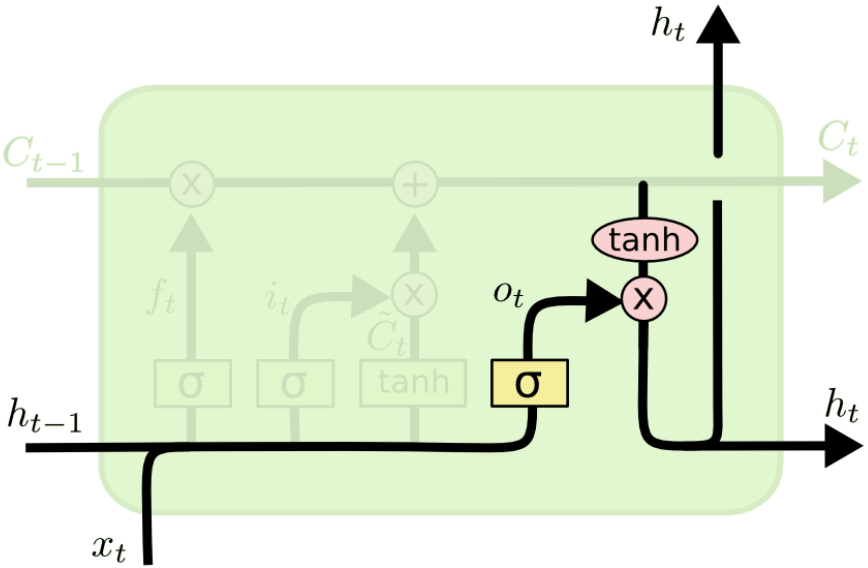
앞서 계산했던 얼마나 잊거나 기억할 것인지, 업데이트는 얼마의 비중으로 얼마나 업데이트 할 것인지 값들을 Cell state에 적용하는 단계

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU





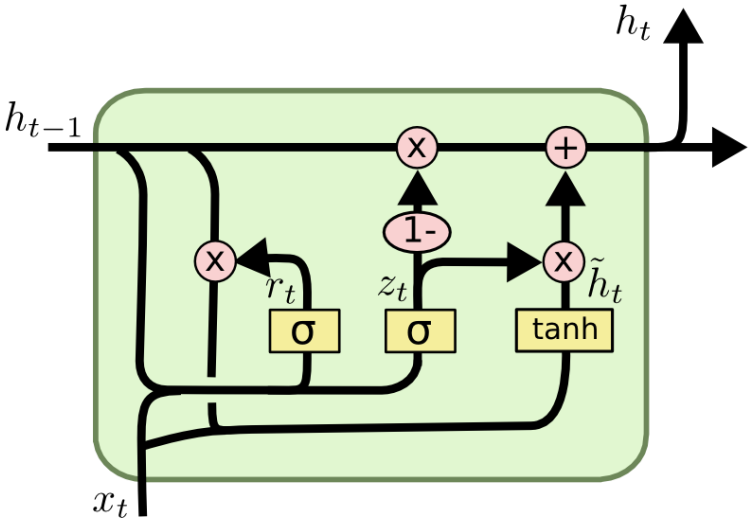
기존 hidden state, input으로 o_t를 정하고, 업데이트 된 cell state를 tanh에 통과시킨 비중을 곱해 새로운 hidden state를 생성

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

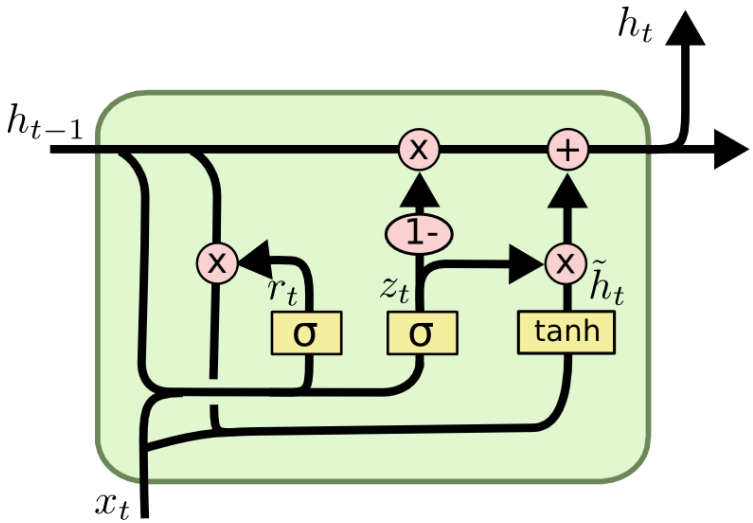
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

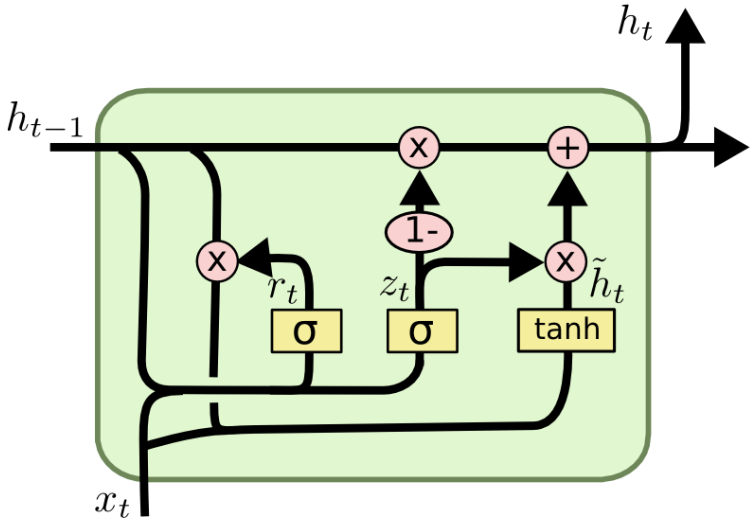
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

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

(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM & GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

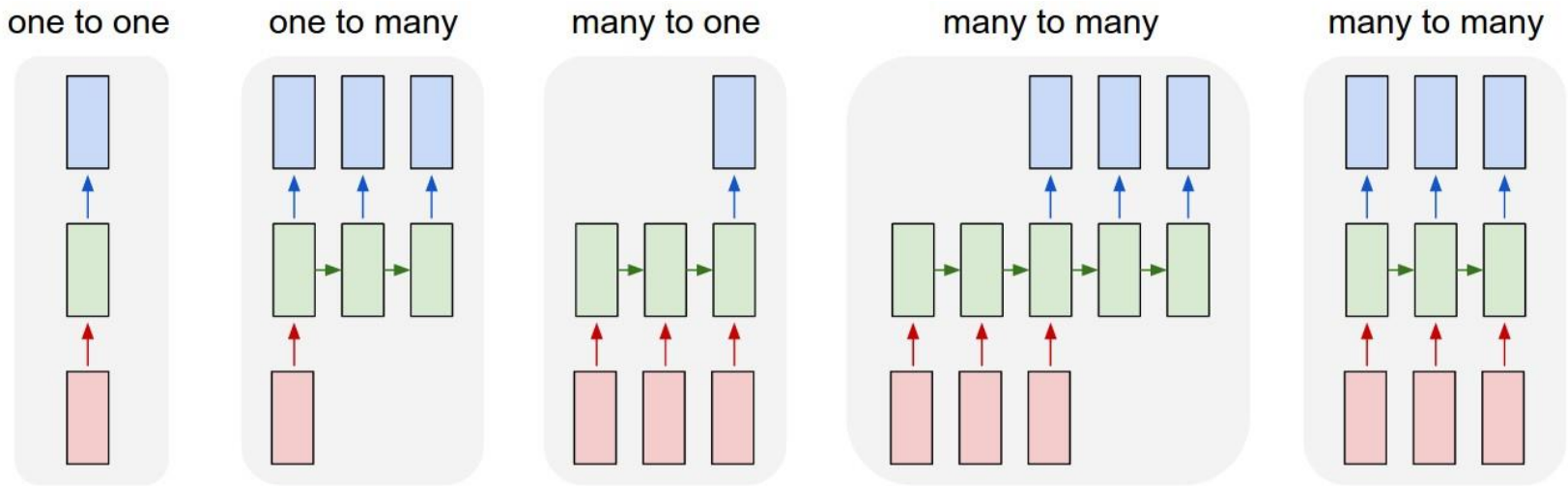
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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

더 간단하지만 성능 면에서 LSTM과 별 차이가 없어서
꽤 많이 사용됨

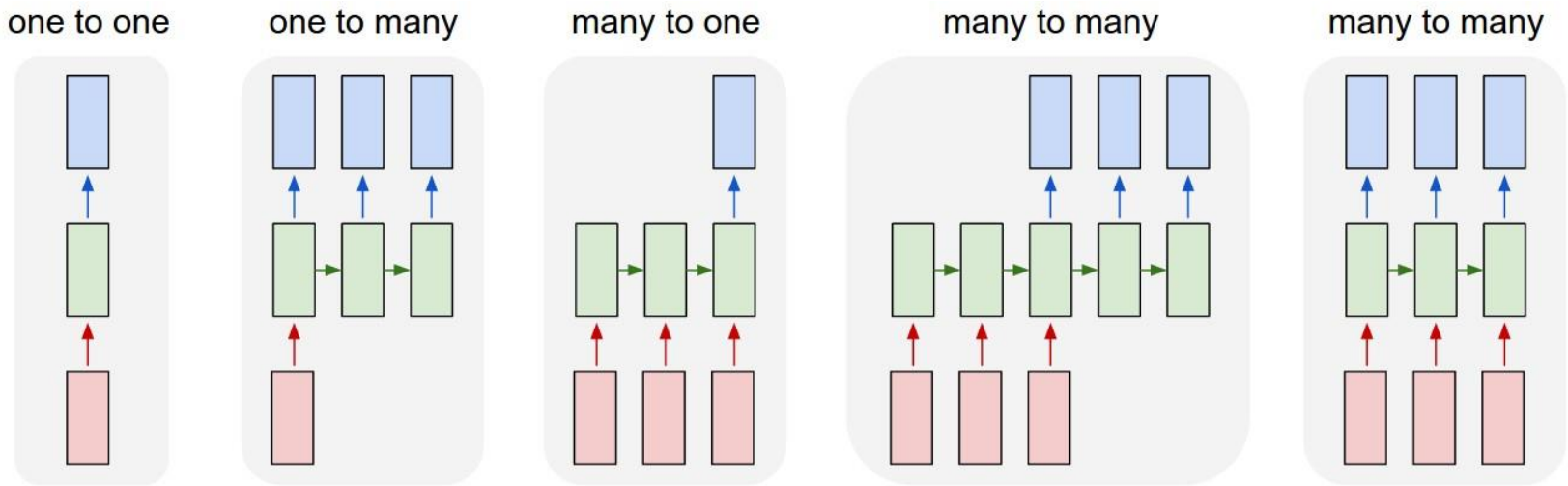
(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Character RNN



(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

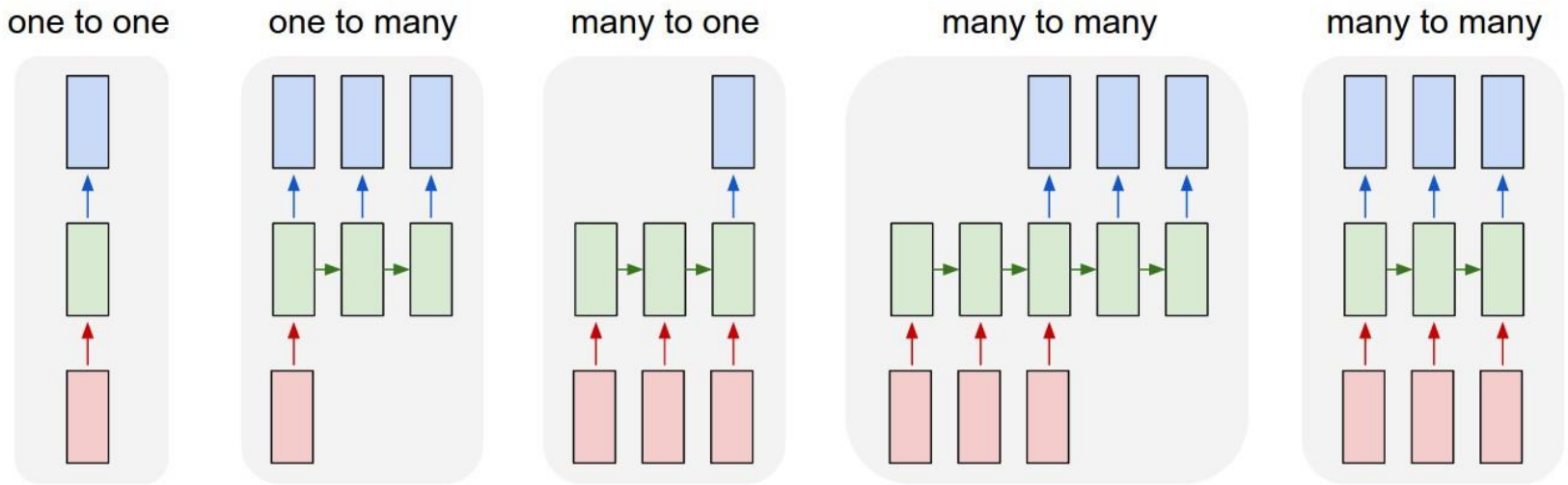
Character RNN



ex) vanilla RNN

(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

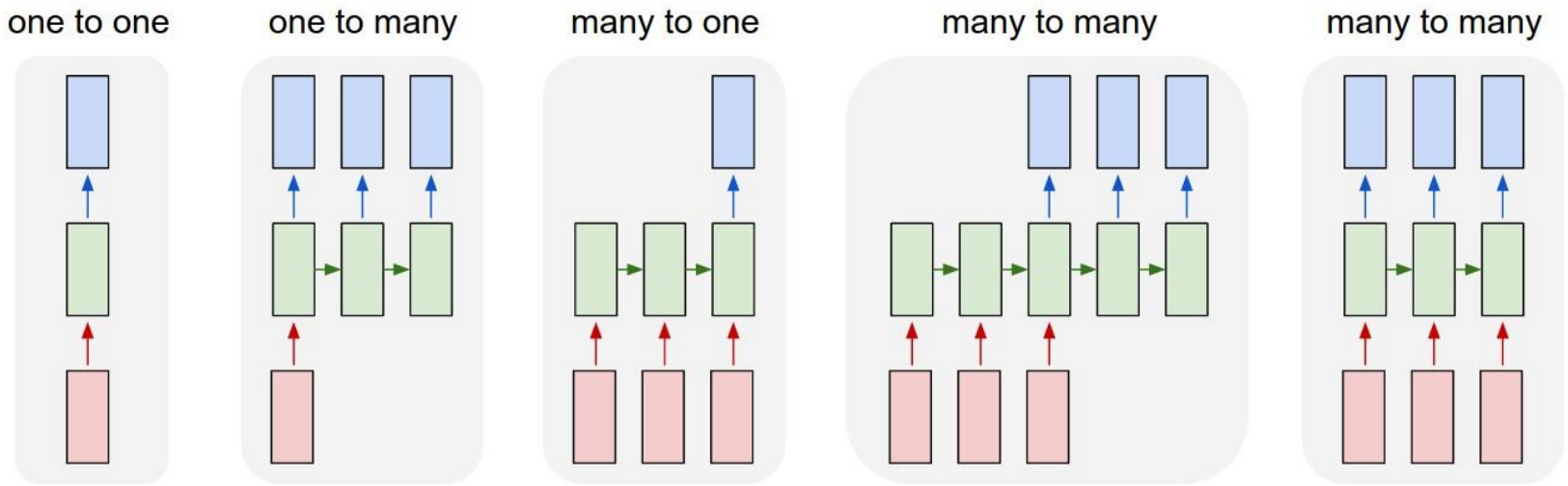
Character RNN



ex) vanilla RNN Image Captioning

(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

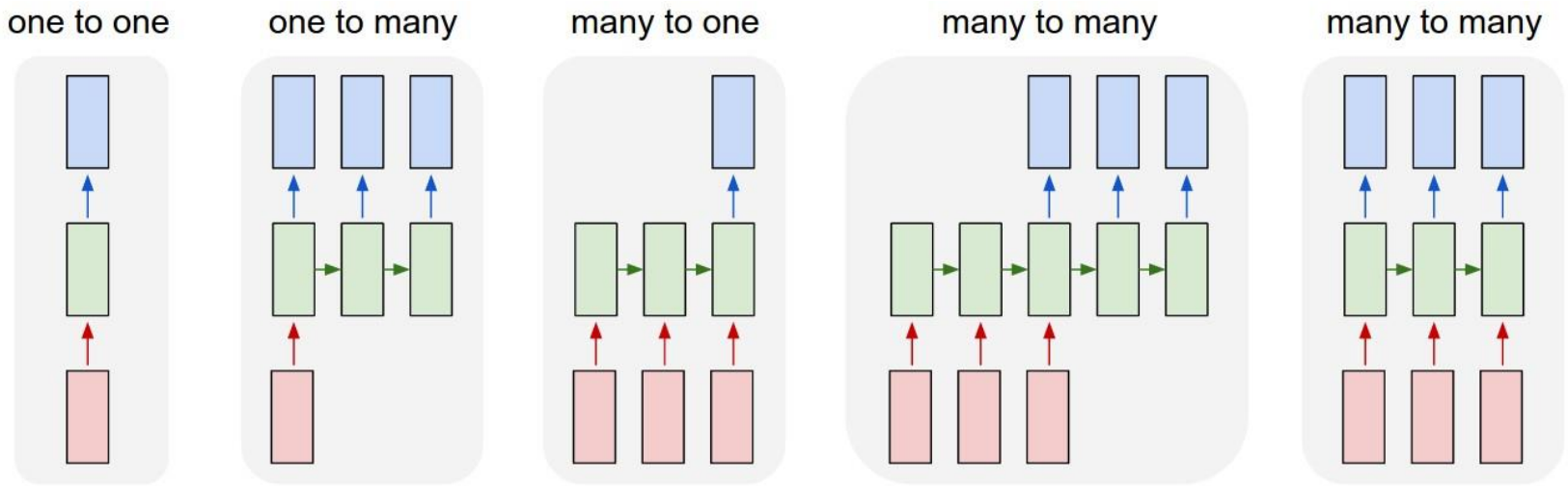
Character RNN



ex) vanilla RNN Image Captioning Sentiment Analysis

(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

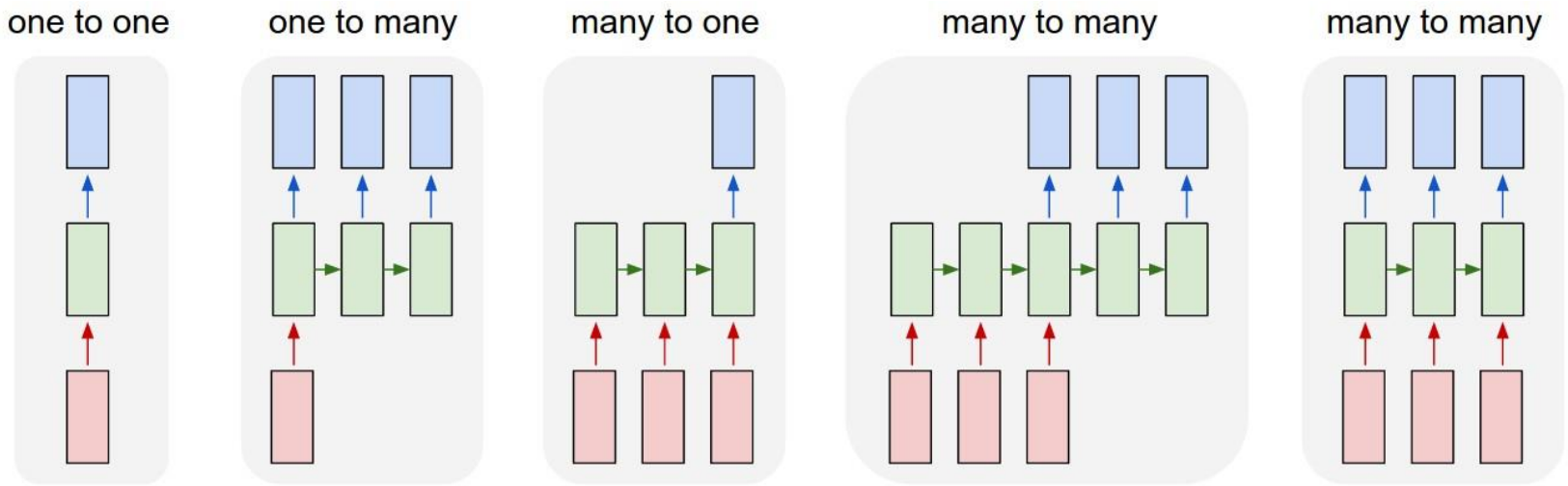
Character RNN



ex) vanilla RNN Image Captioning Sentiment Analysis Machine Translation

(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Character RNN



ex) vanilla RNN | Image Captioning | Sentiment Analysis | Machine Translation | Video Frame Classification

(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Character RNN

"hello pytorch"

Character RNN

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

Character RNN

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

"abcdefghijklmnopqrstuvwxyz "

Character RNN

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

"abcdefghijklmnopqrstuvwxyz " → one-hot vector

Character RNN

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

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

Character RNN

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

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

27

a =[1 0 0 0 ... 0]

b =[0 1 0 0 ... 0]

c =[0 0 1 0 ... 0]

⋮

Character RNN

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

"abcdefghijklmnopqrstuvwxyz "

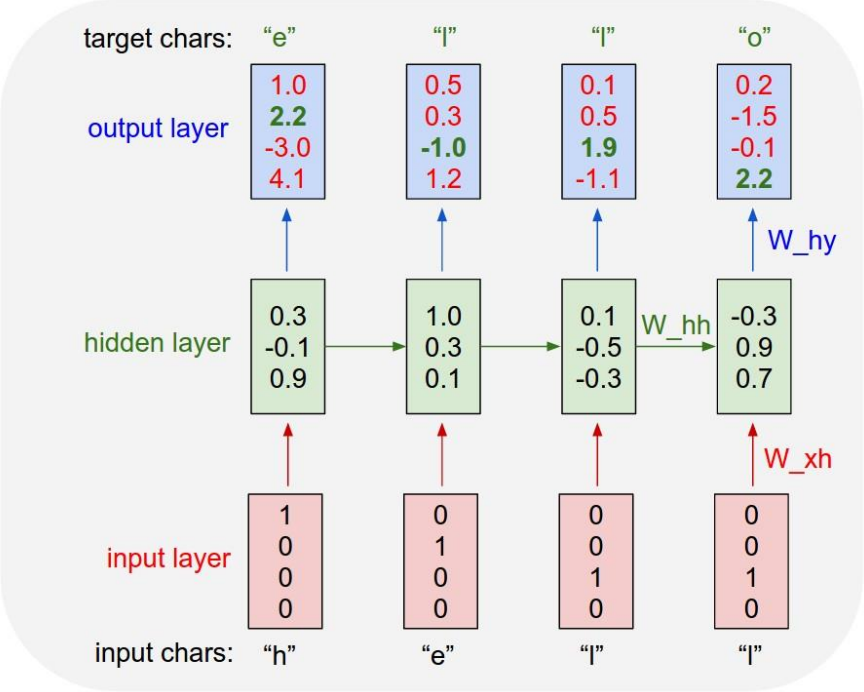
27

a =[1 0 0 0 ... 0]

b =[0 1 0 0 ... 0]

c =[0 0 1 0 ... 0]

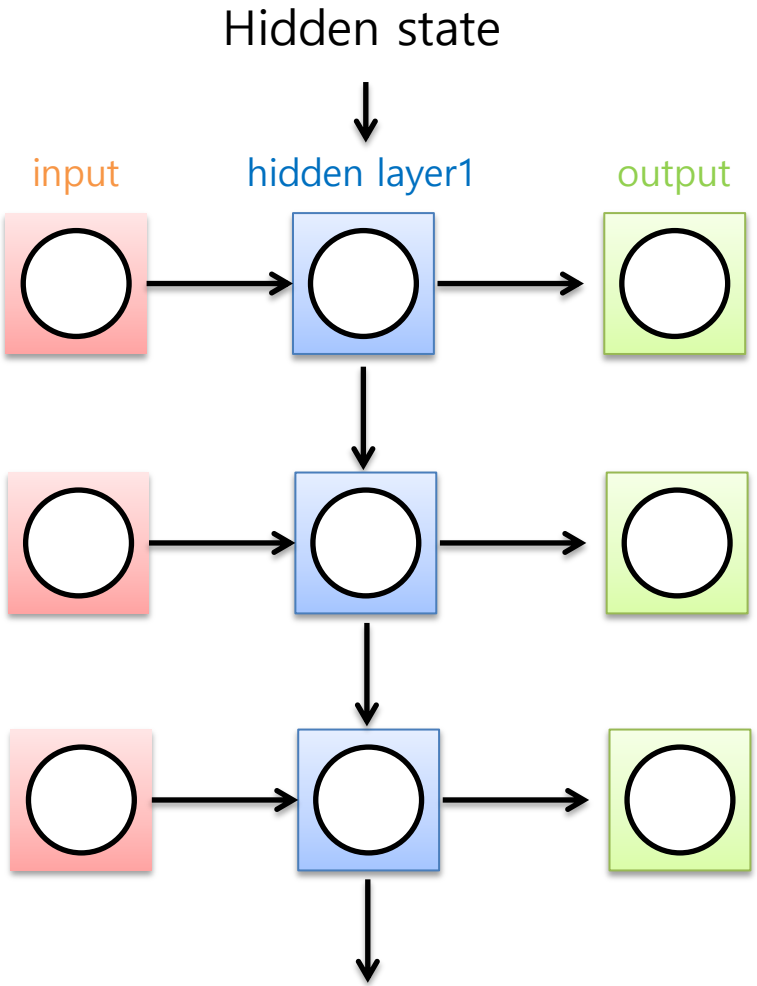
⋮



(출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

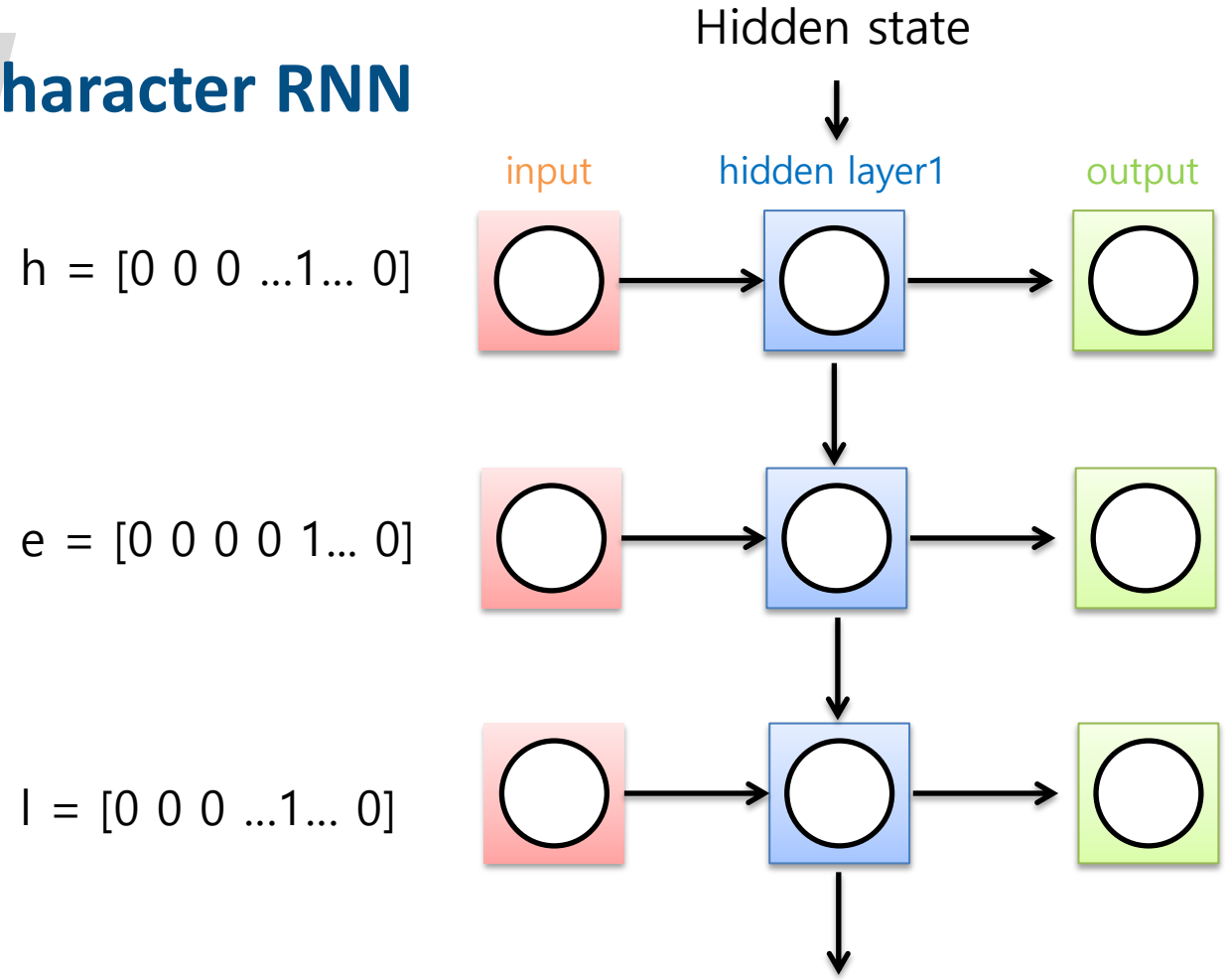
Character RNN

Training



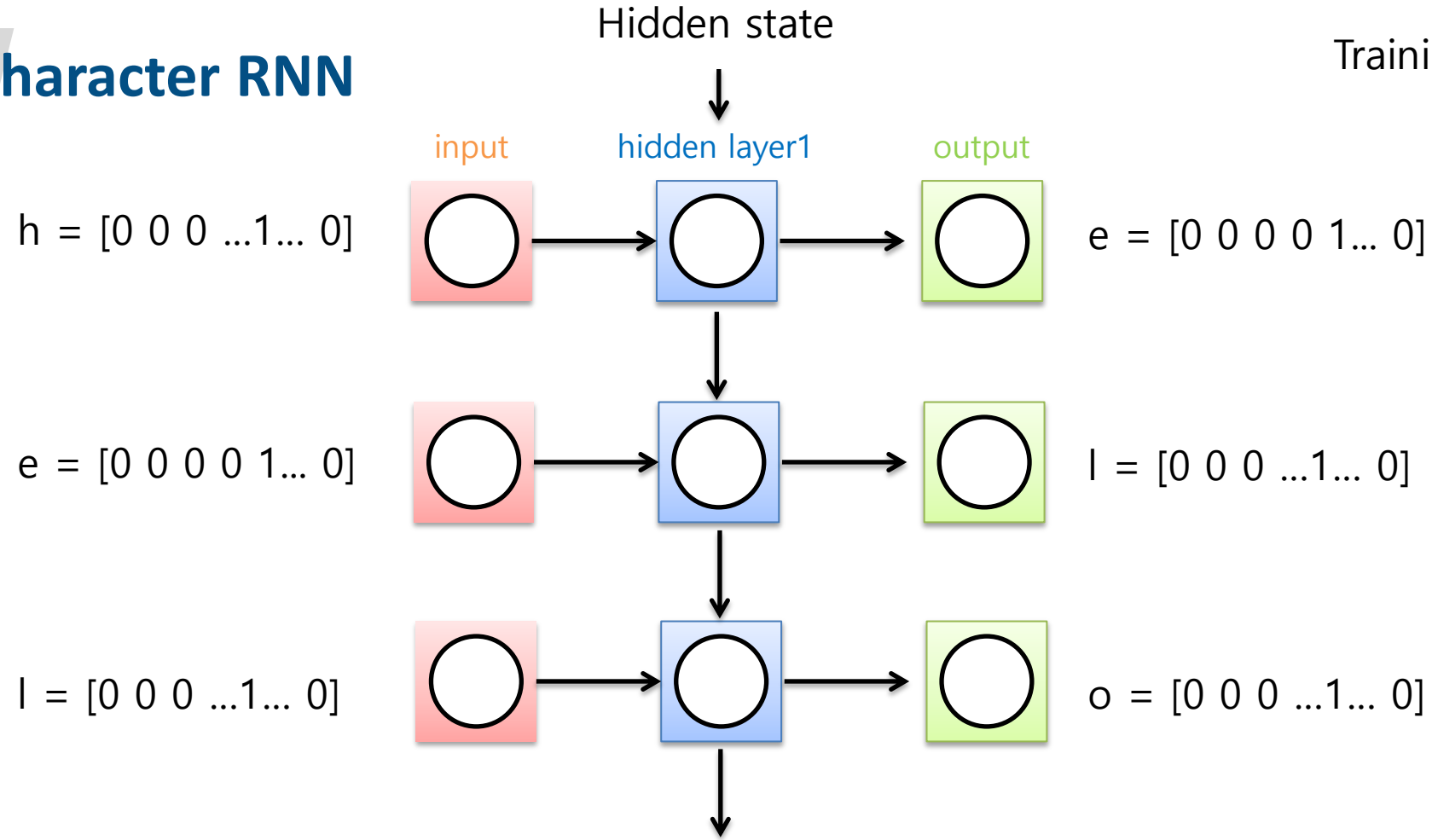
Character RNN

Training



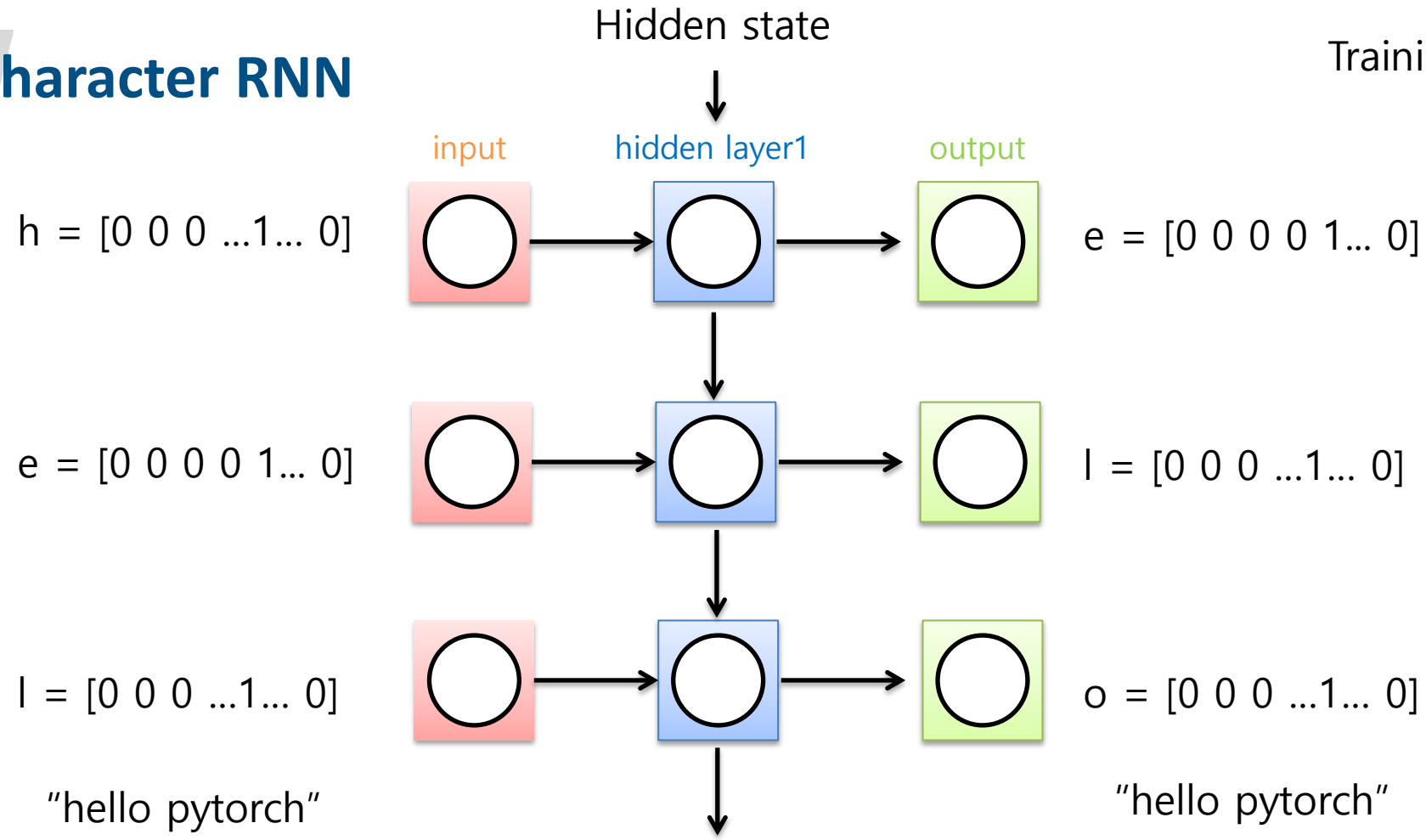
Character RNN

Training

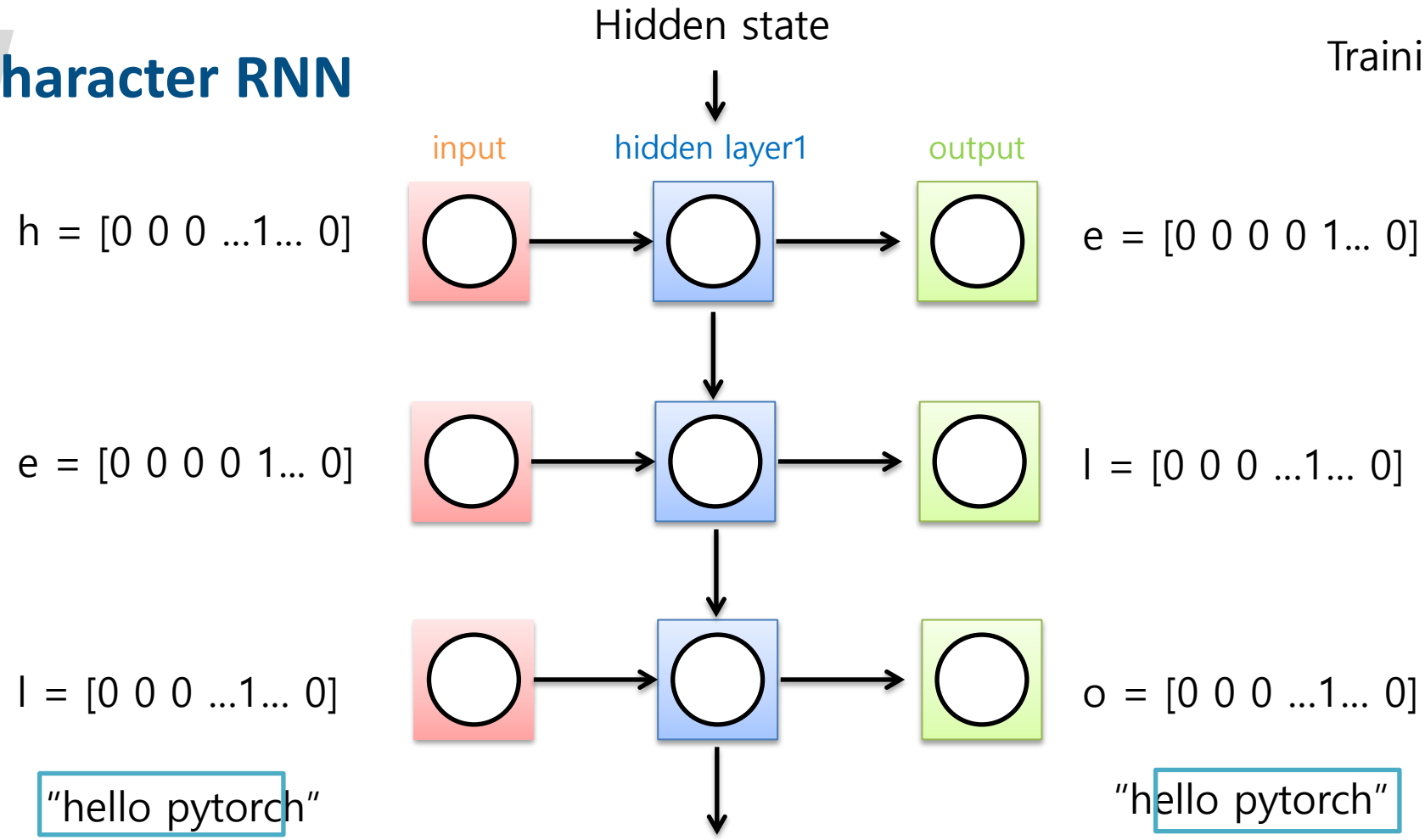


Character RNN

Training

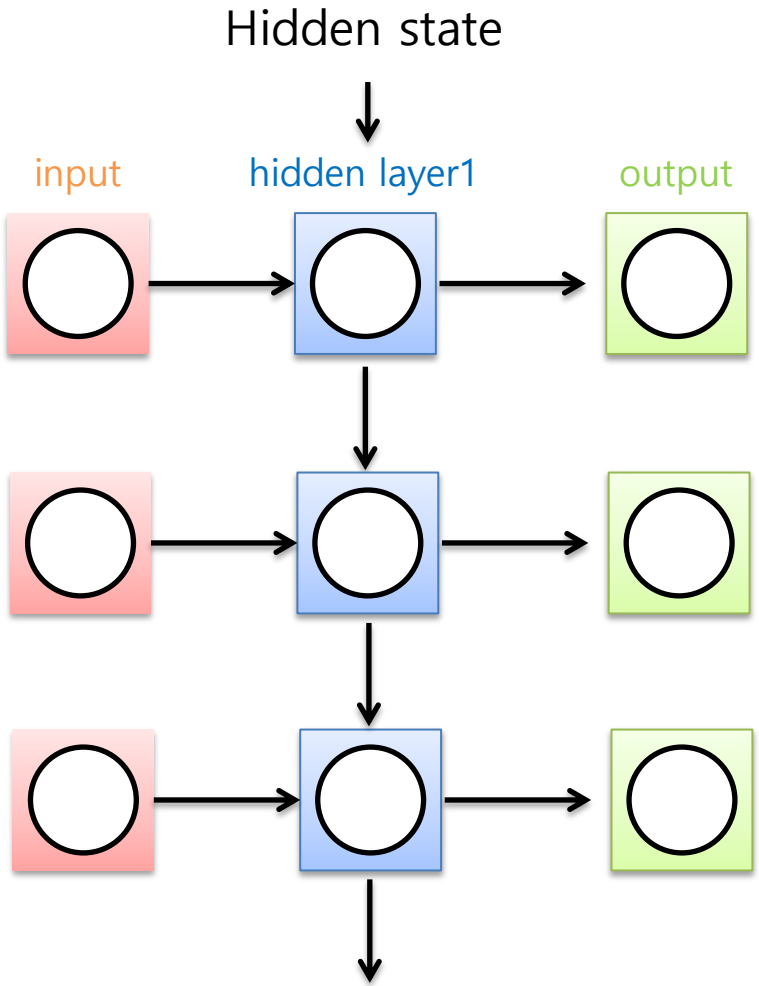


Character RNN



Character RNN

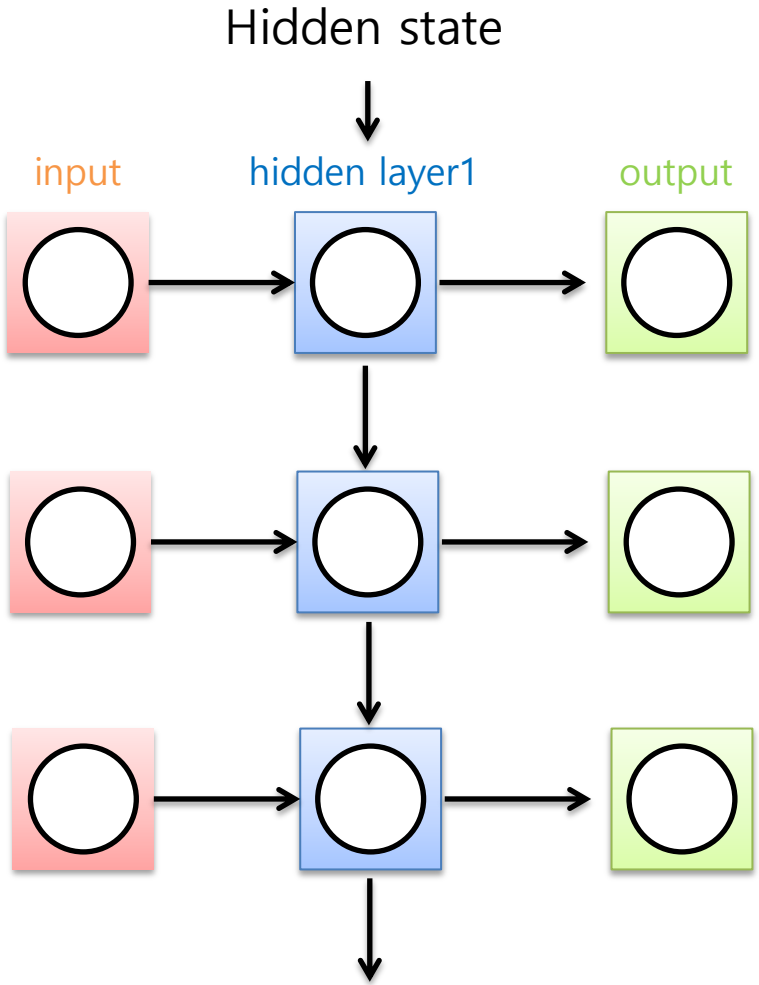
Test



Character RNN

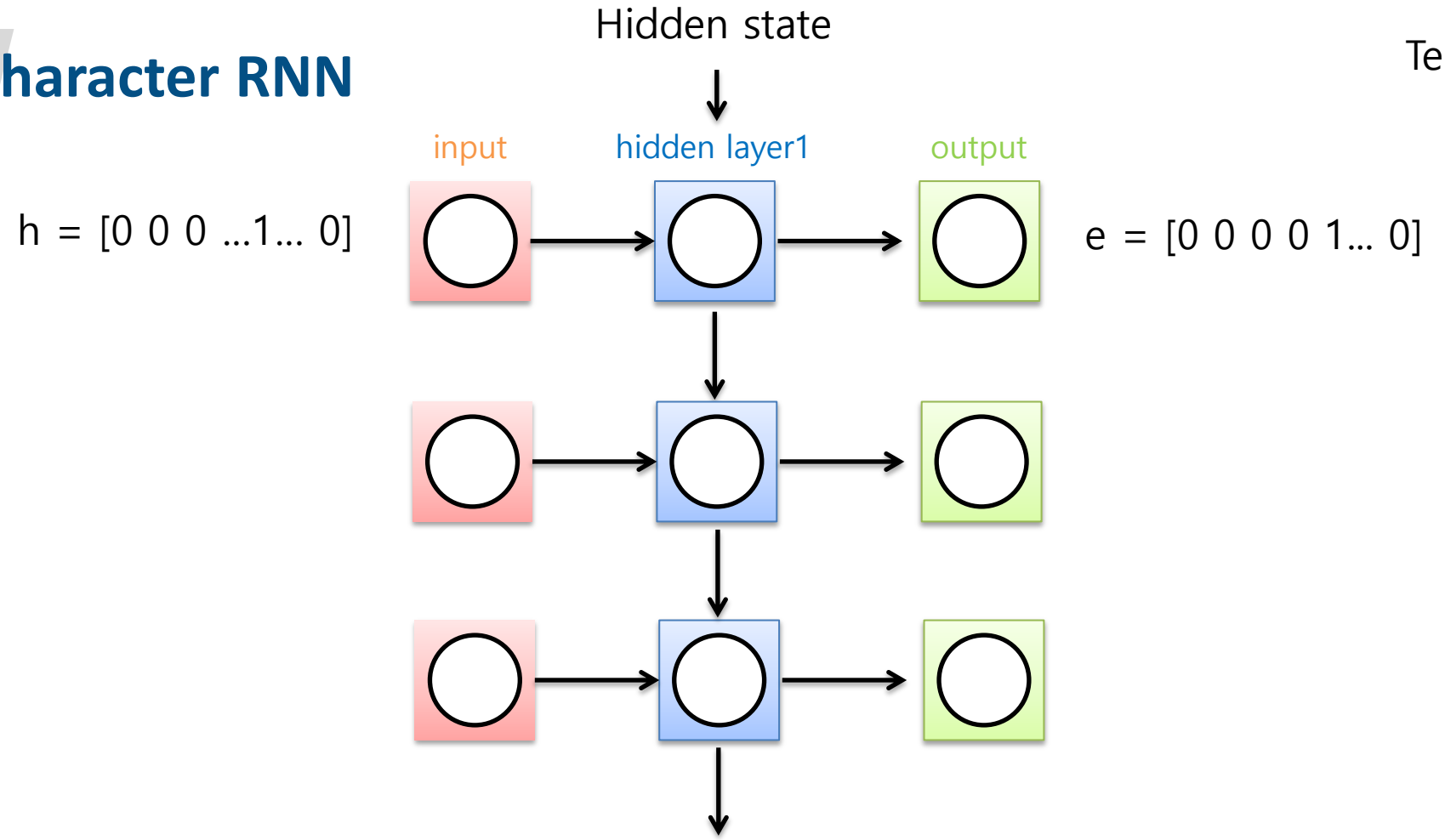
Test

h = [0 0 0 ...1... 0]



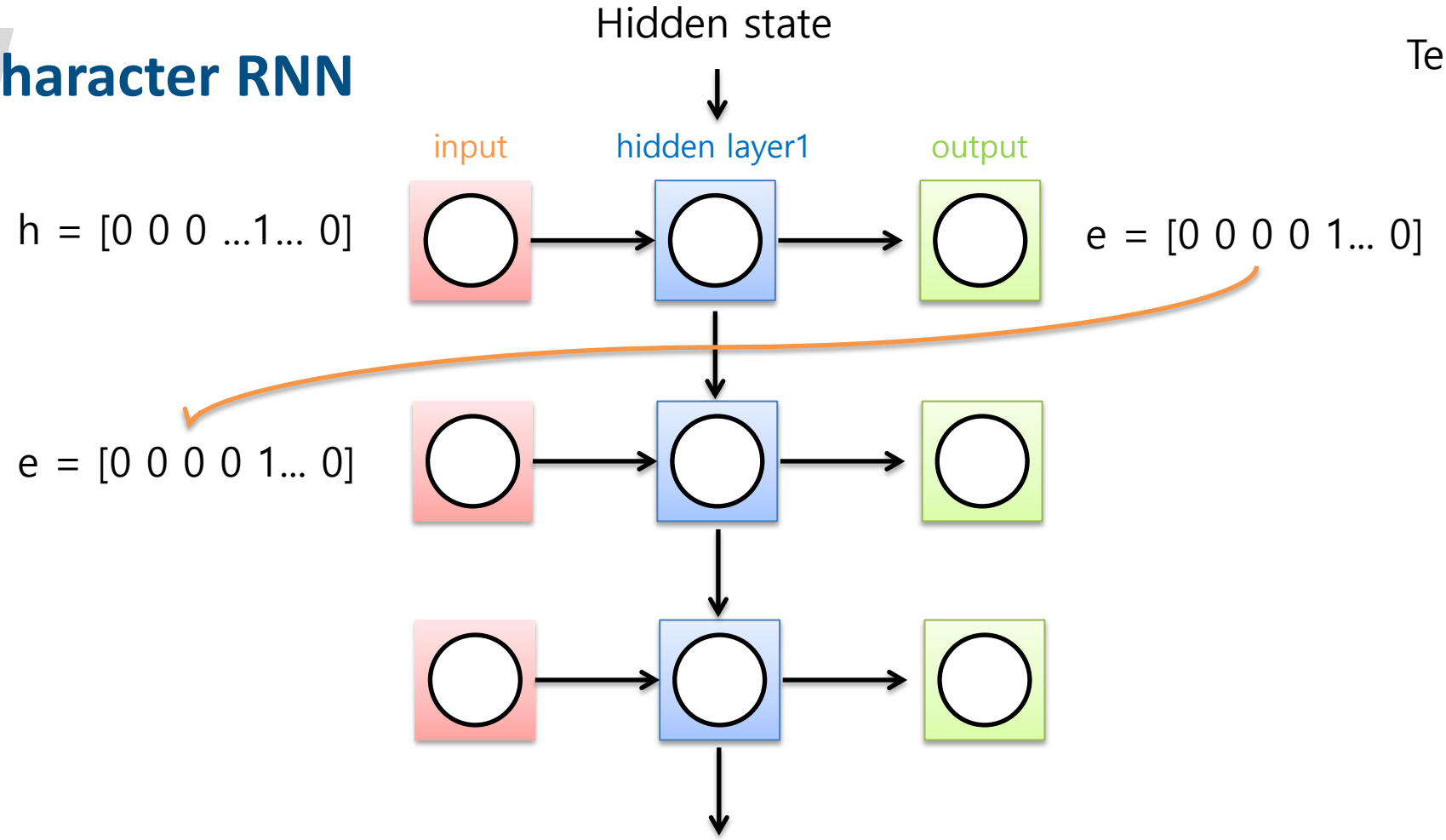
Character RNN

Test



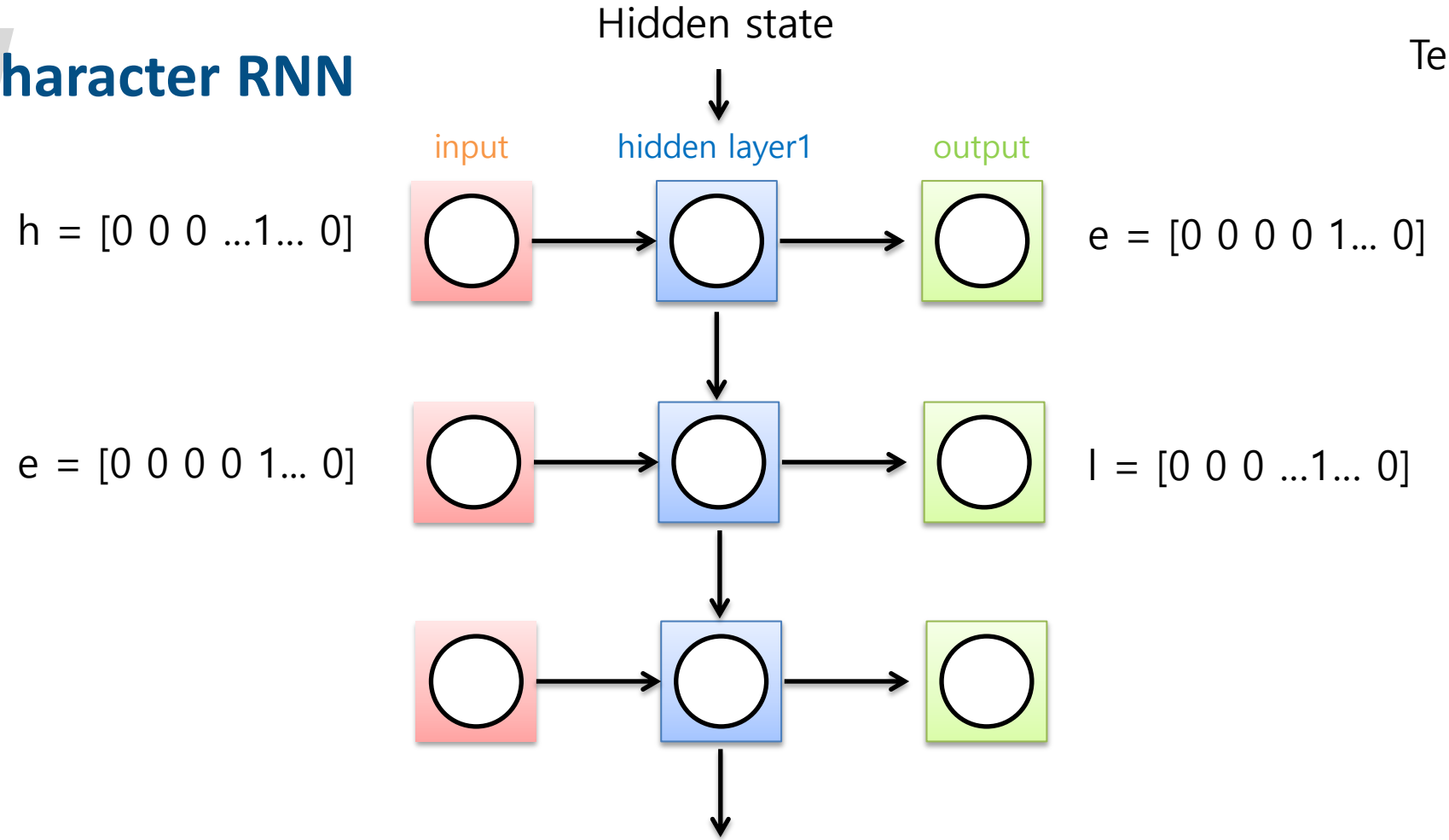
Character RNN

Test



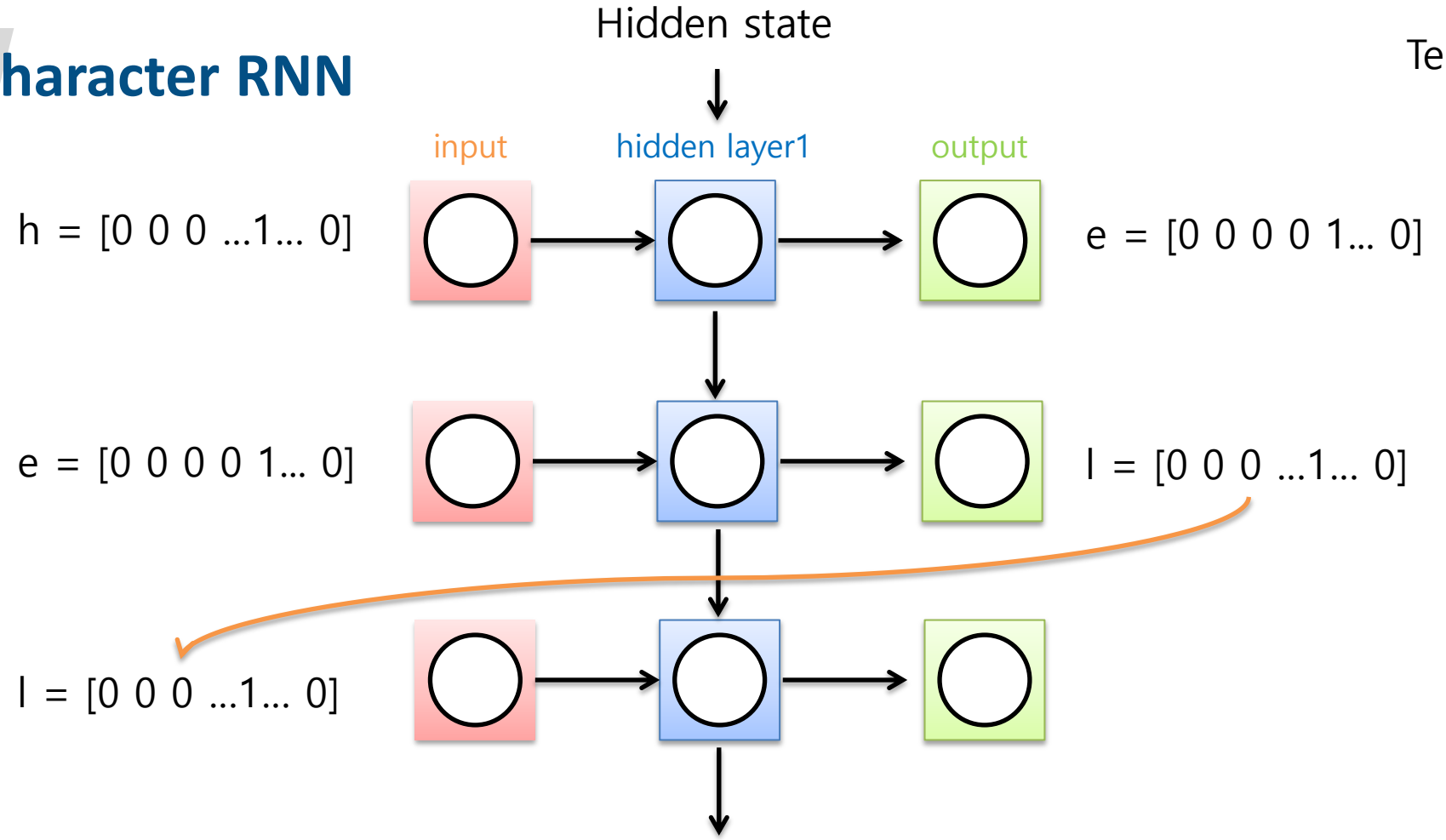
Character RNN

Test



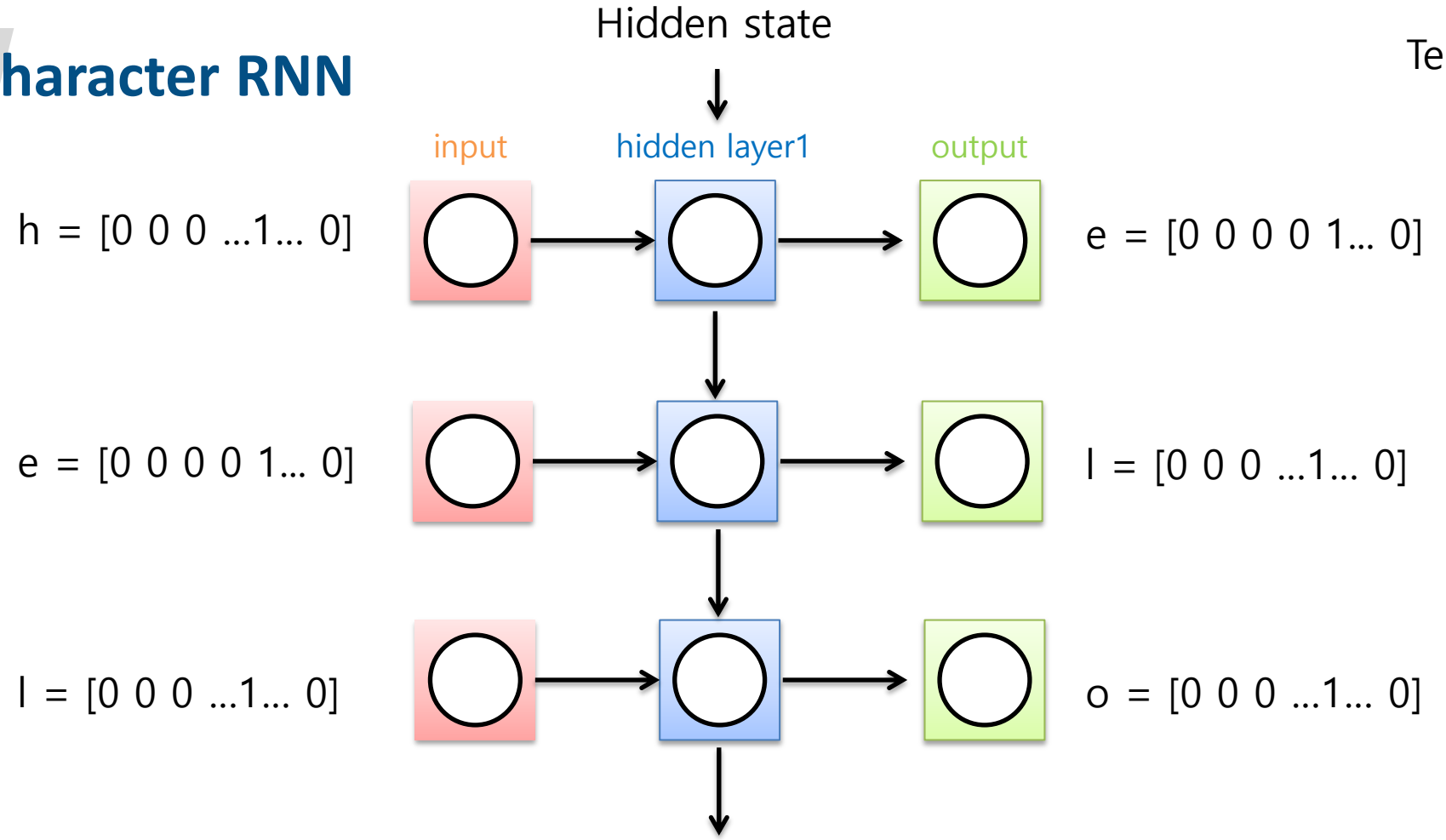
Character RNN

Test



Character RNN

Test



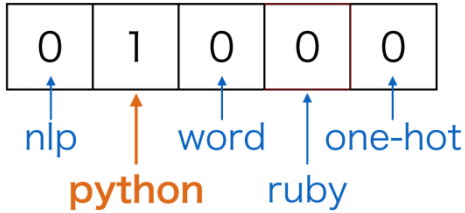
Character RNN

그런데 one-hot vector가 효율적인 방법일까?

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

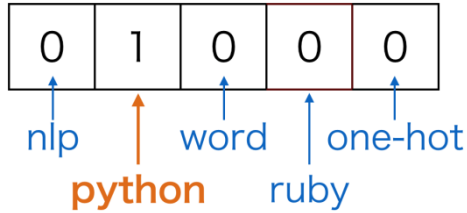
그런데 one-hot vector가 효율적인 방법일까?



(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

그런데 one-hot vector가 효율적인 방법일까?



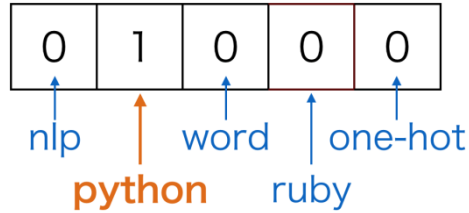
element-wise product is zero vector.
Thus, Inner product is zero.

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

그런데 one-hot vector가 효율적인 방법일까?

1. 의미를 제대로 담지 못함. 또한 벡터 간의 연산도 무의미



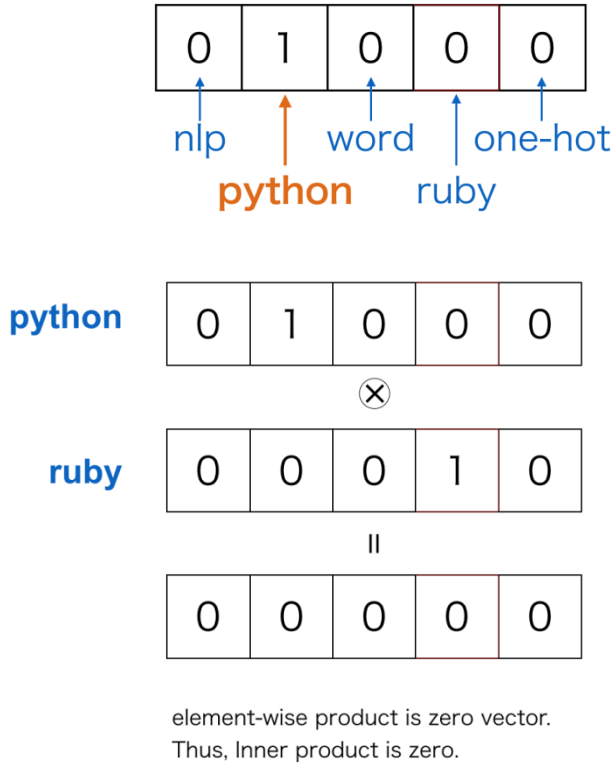
element-wise product is zero vector.
Thus, Inner product is zero.

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

그런데 one-hot vector가 효율적인 방법일까?

1. 의미를 제대로 담지 못함. 또한 벡터 간의 연산도 무의미
2. 만약 word를 one-hot으로 표현하려 한다면 길이도 무한정 늘어나고 새로운 단어가 들어오면 전체 길이가 늘어나서 기존의 모델이 무의미



(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

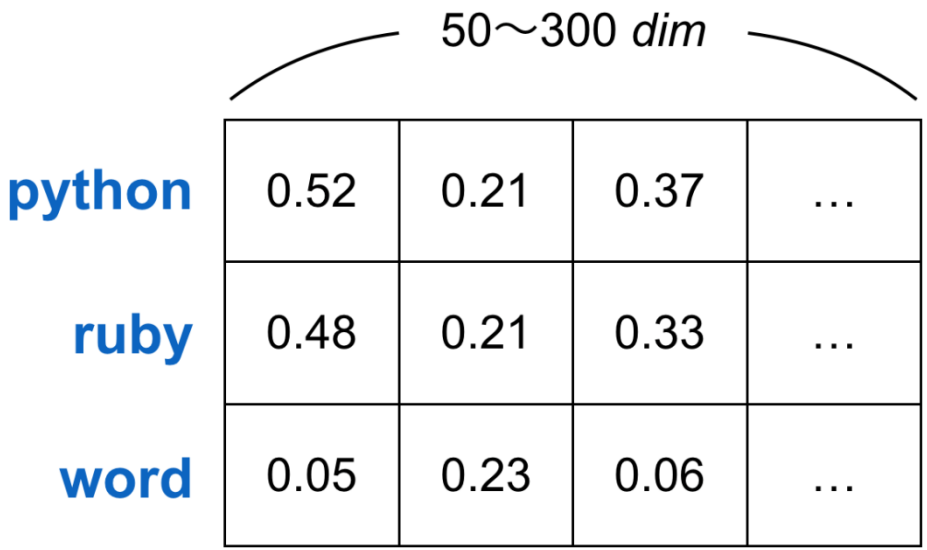
Character RNN

Word Embedding

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

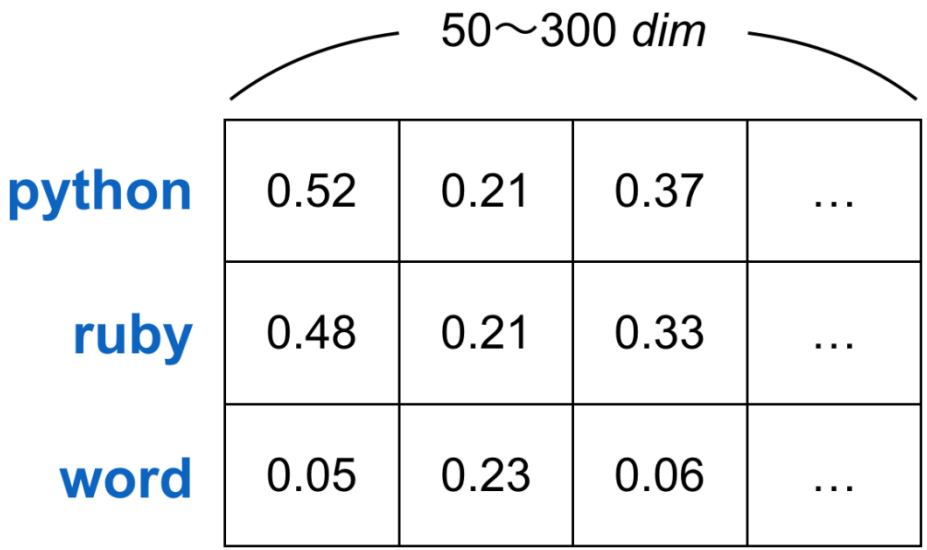
Word Embedding



(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

Word Embedding

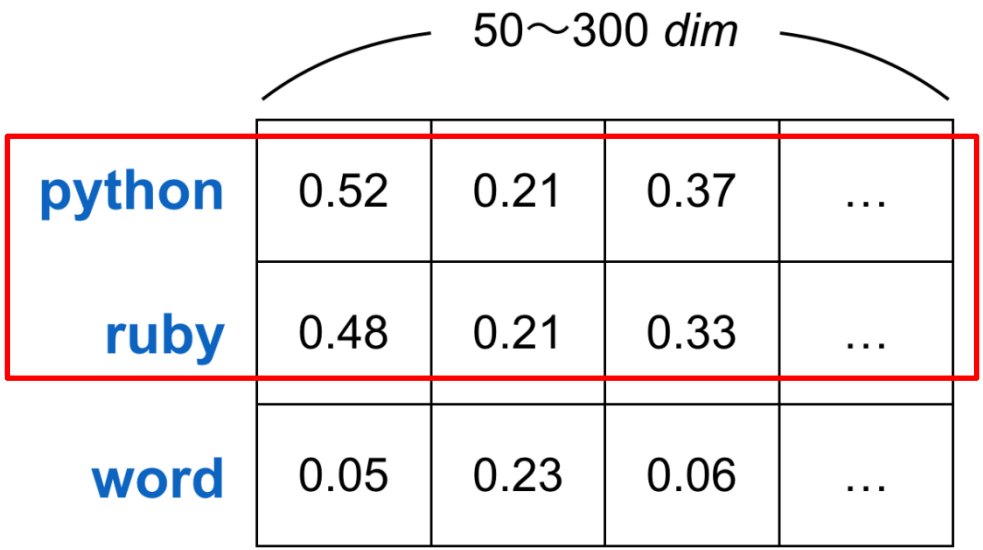


일정한 길이의 벡터에 continuous value로 단어들을 표현

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

Word Embedding

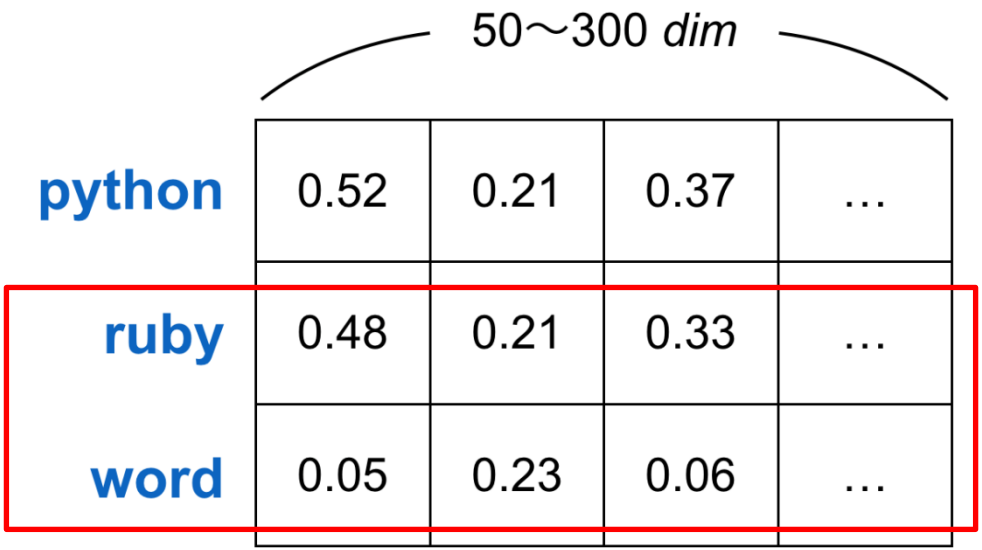


일정한 길이의 벡터에 continuous value로 단어들을 표현

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

Word Embedding



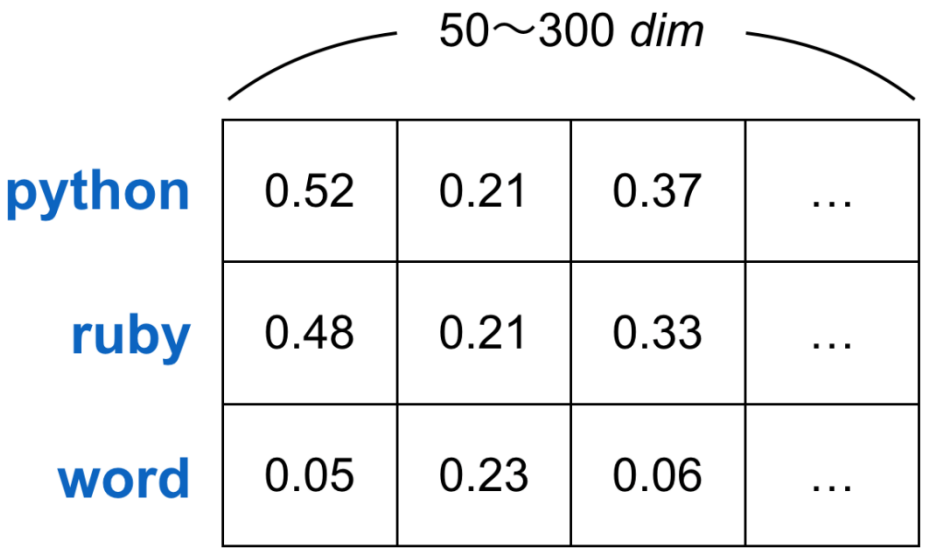
일정한 길이의 벡터에 continuous value로 단어들을 표현

semantic한 거리 차이가 있음

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

Word Embedding



일정한 길이의 벡터에 continuous value로 단어들을 표현

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

(출처: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/)

Character RNN

"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]

Character RNN

Word2Vec

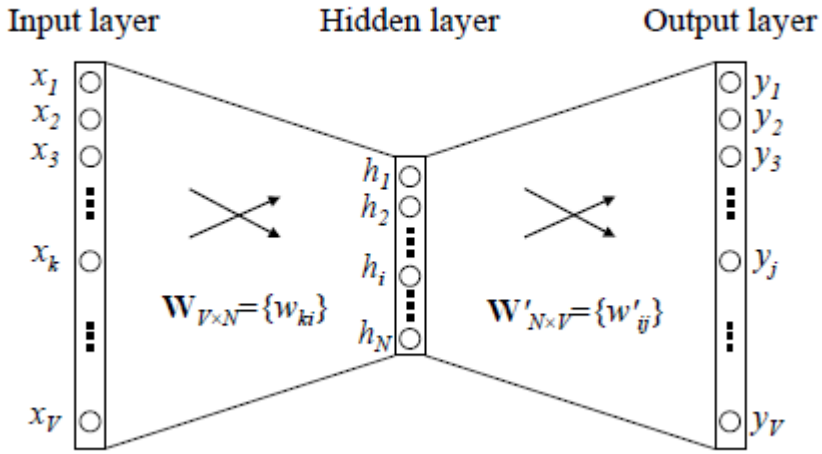
"The quick brown fox jumps over the lazy dog"

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]

Character RNN

Word2Vec

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]



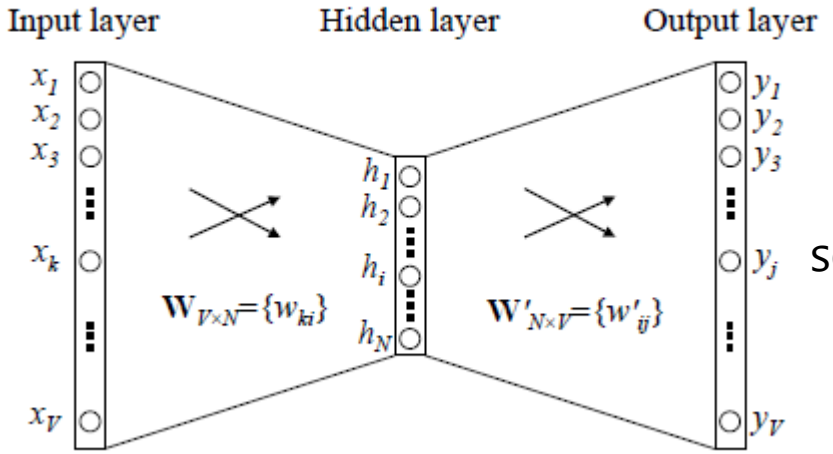
https://iksinc.wordpress.com/tag/skip-gram-model/

Character RNN

Word2Vec

"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]



softmax

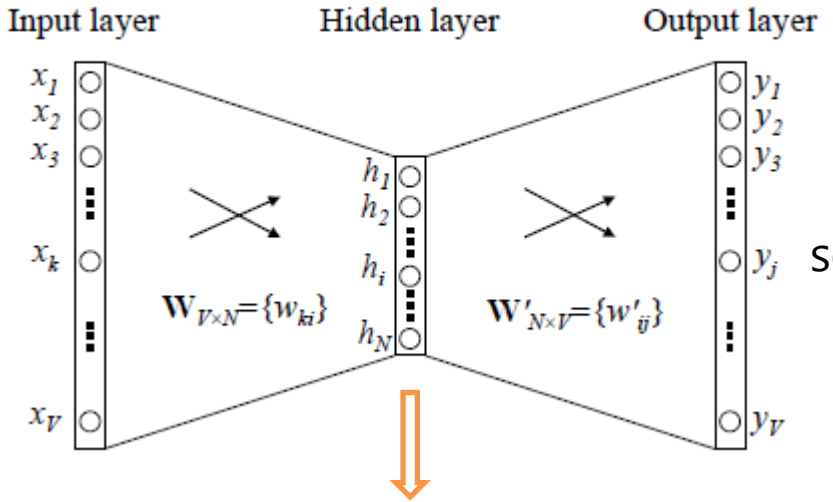
https://iksinc.wordpress.com/tag/skip-gram-model/

Character RNN

Word2Vec

"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]



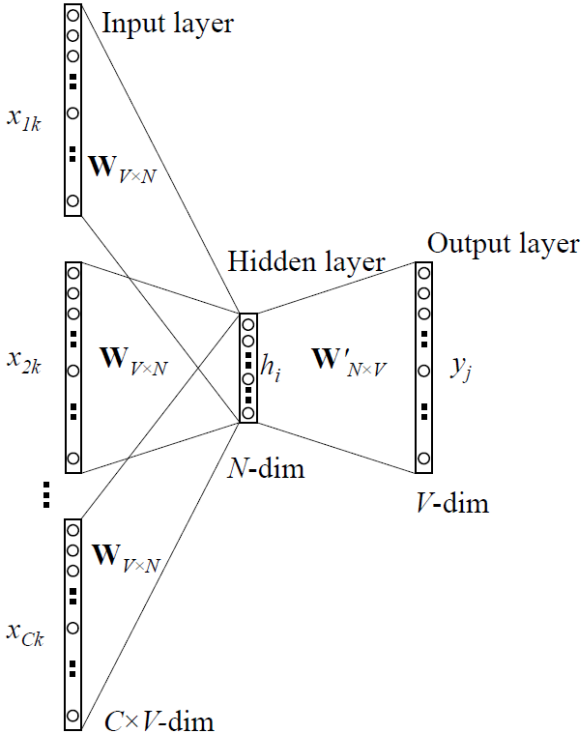
softmax

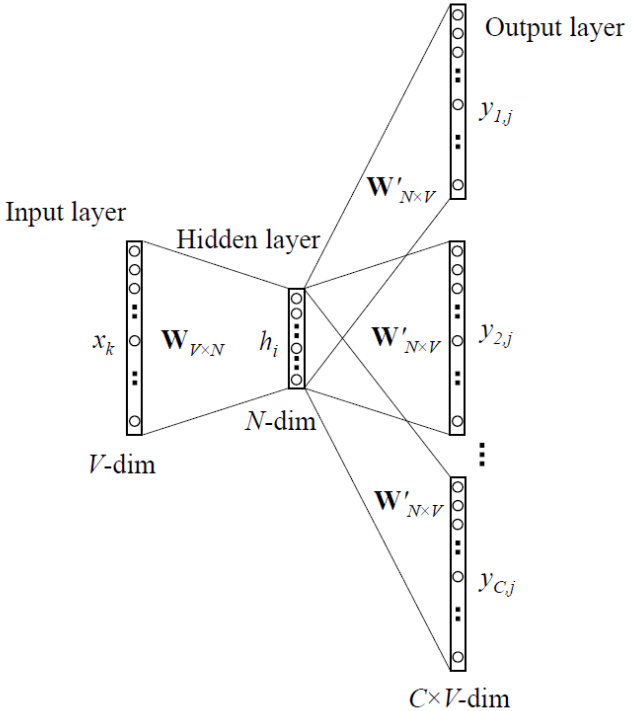
embedding

https://iksinc.wordpress.com/tag/skip-gram-model/

Character RNN

Word2Vec





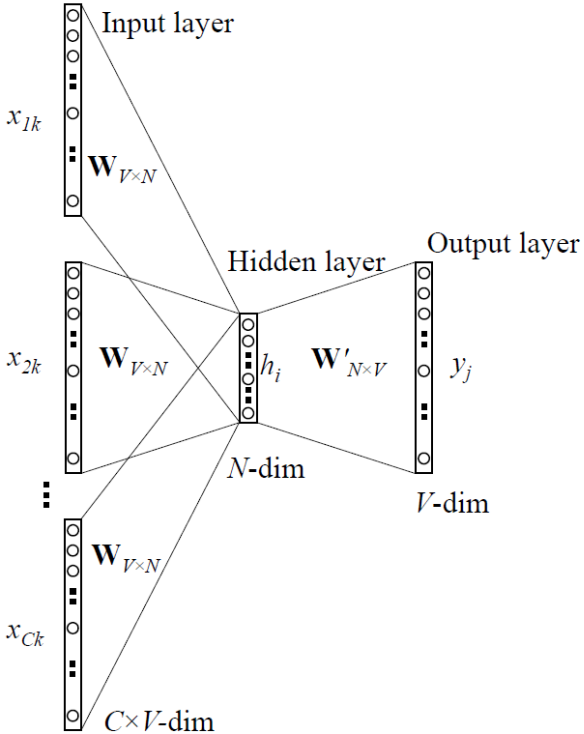
주변 단어로 중심 단어가 나오도록

중심 단어에서 주변 단어들이 나오도록

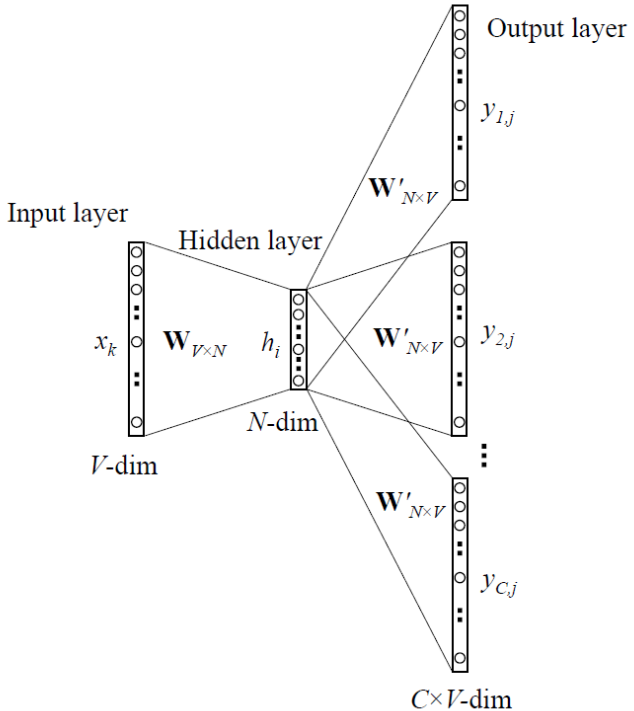
https://iksinc.wordpress.com/tag/skip-gram-model/

Character RNN

Word2Vec



Continuous Bag of Words (CBOW) Learning



Skip-Gram Model

https://iksinc.wordpress.com/tag/skip-gram-model/

Character RNN

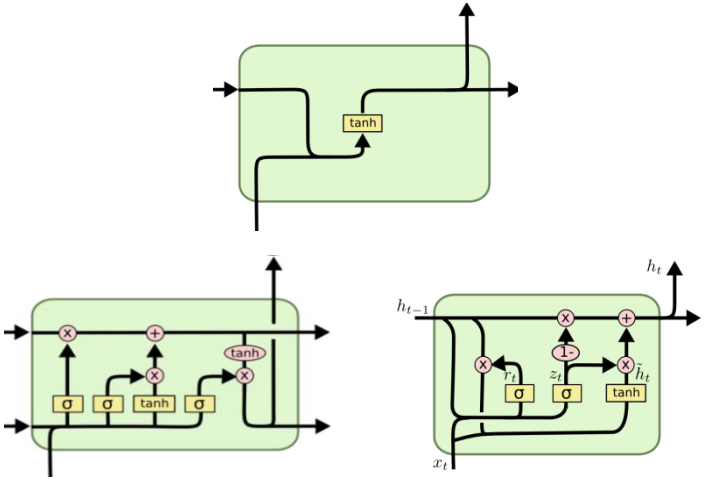
Shakespeare plays dataset

```
PANDARUS:
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.
```

Character RNN

Shakespeare plays dataset

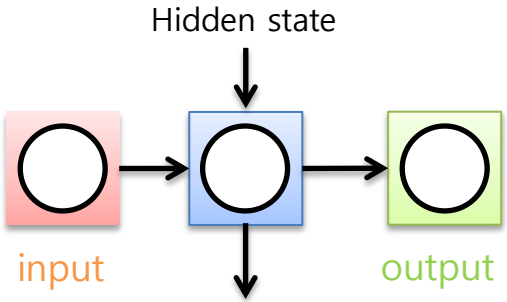
RNN, LSTM, GRU를 써서 얼마나 진짜처럼 모방할 수 있을까?




```
PANDARUS:
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.
```

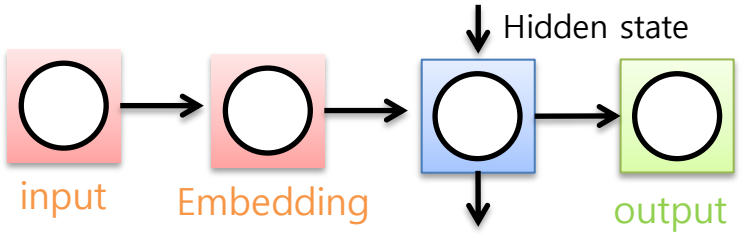
(출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Character RNN



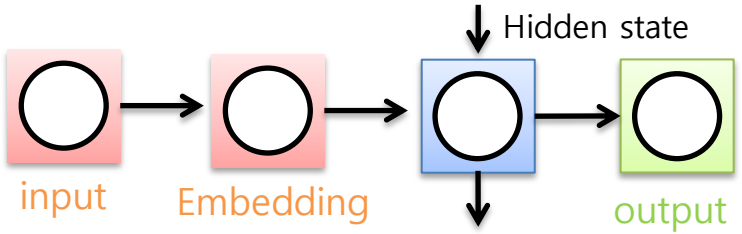

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers

        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()
```

Character RNN



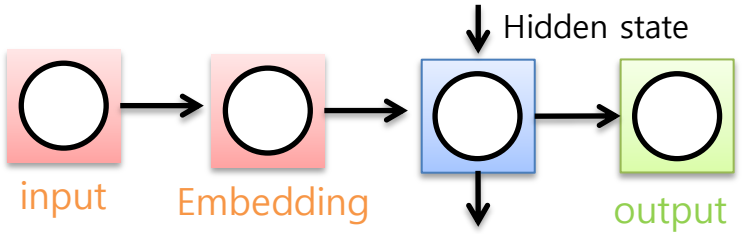

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers

        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()
```

Character RNN




```
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()
```

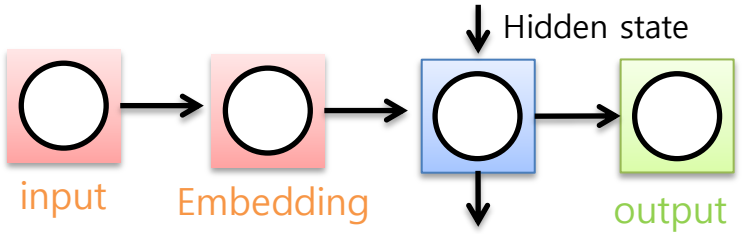
Character RNN



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(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()
```

Character RNN

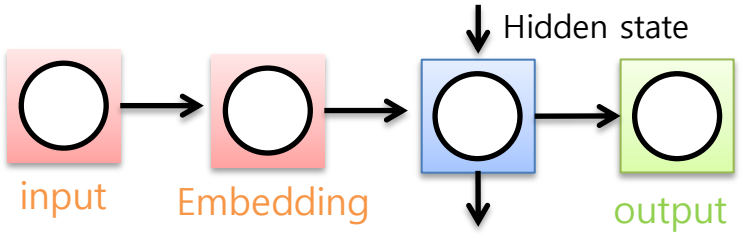


embedding할 가지 수
ex) 알파벳 소문자면 26

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(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()
```

Character RNN



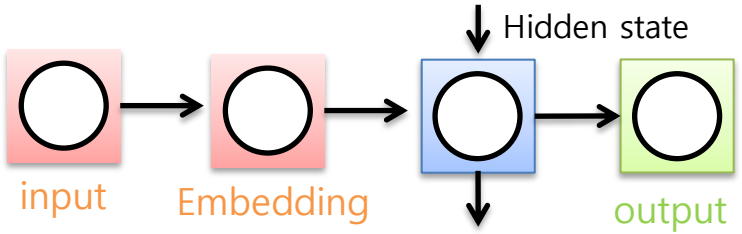
embedding할 가지 수
ex) 알파벳 소문자면 26

nn.Embedding(num_embeddings, embedding_dim)

목표 embedding 차원

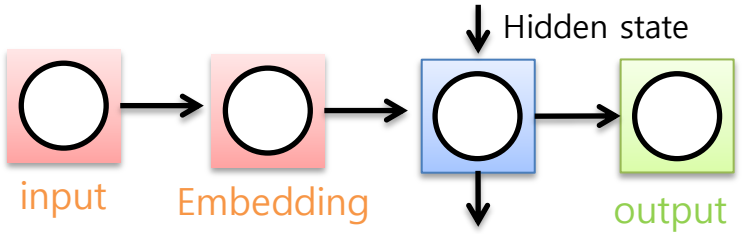
```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers

        self.encoder = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.RNN(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()
```

Character RNN




```
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()
```

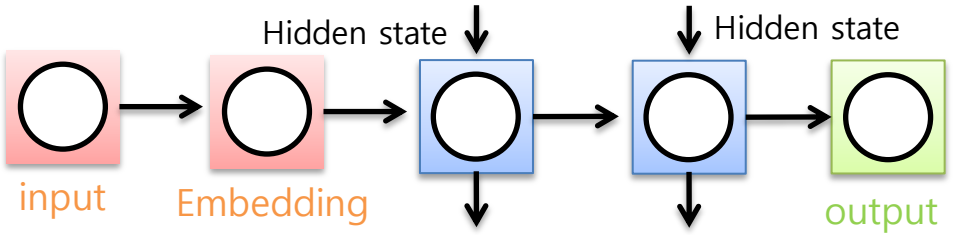
Character RNN



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()
```

Character RNN

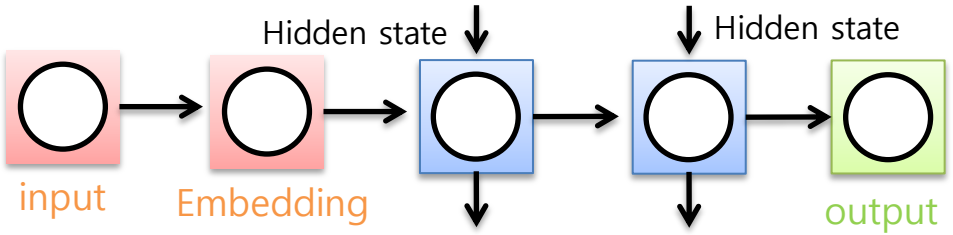


num_layers는 파란색 RNN Cell을 몇 개 쌓을 것 인지

ex) 2개 였다면?

```
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()
```

Character RNN



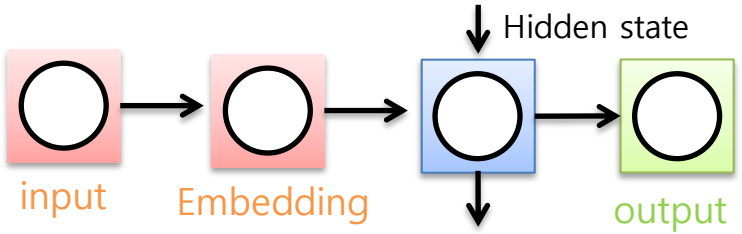
num_layers는 파란색 RNN Cell을 몇 개 쌓을 것 인지

ex) 2개 였다면?

GRU로 바꾸려면 nn.RNN을 nn.GRU로 바꿔주면 됨

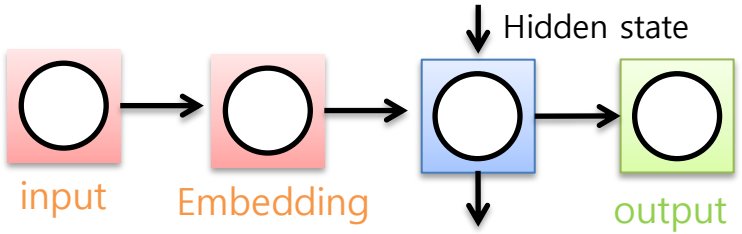
```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers

        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()
```

Character RNN




```
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()
```

Character RNN

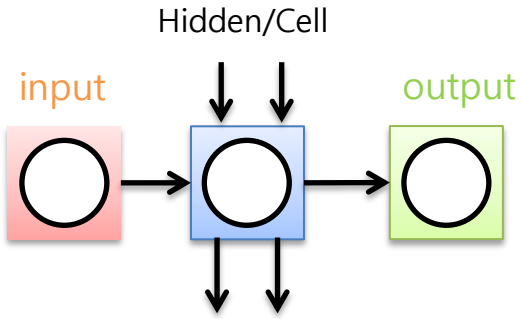


hidden state size로 나온 결과값을 원하는 모양으로 맞춰주는 부분

```
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_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()
```

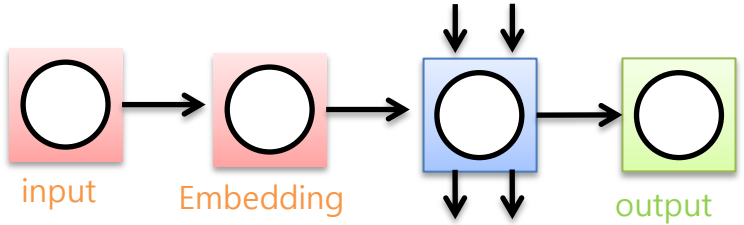
Character RNN

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()
```

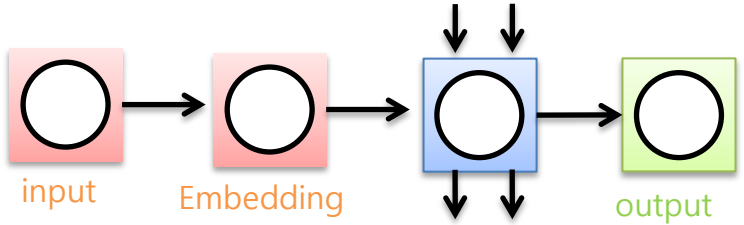
Character RNN




```
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()
```

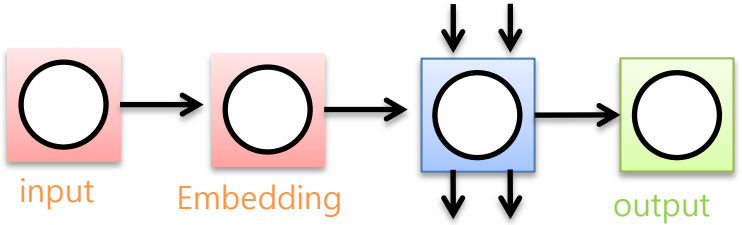
Character RNN

구현적인 측면에서는 hidden state에다가 cell state를 추가한 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.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()
```

Character RNN



구현적인 측면에서는 hidden state에다가 cell state를 추가한 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.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()
```

Q&A