Chapter 4

Recurrent Neural Networks

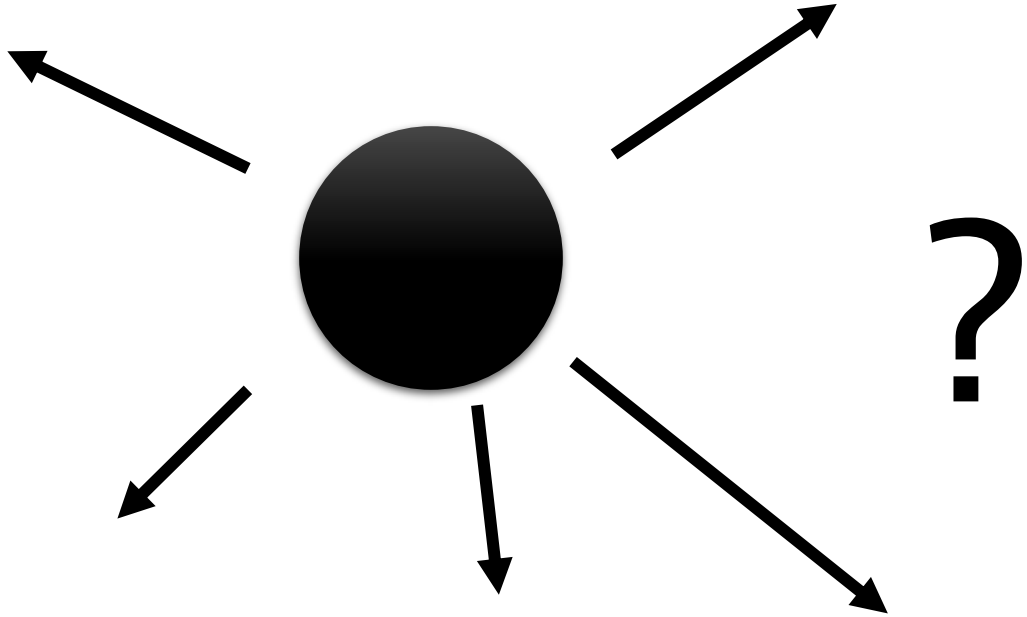
김지환

서강대학교 컴퓨터공학과

Table of contents

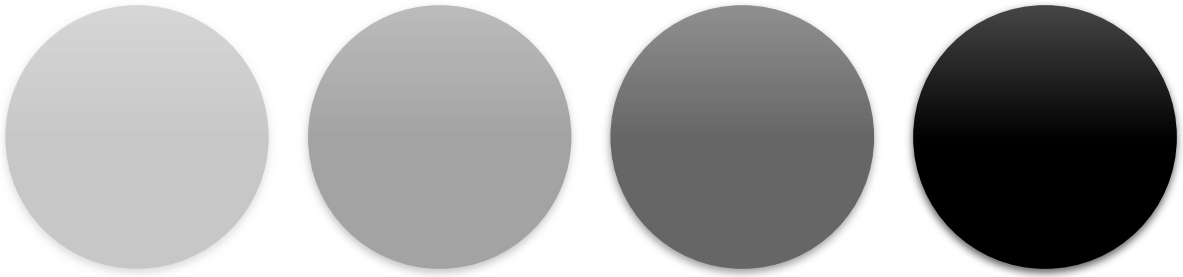
4.1 Introduction

- 아래와 같은 공이 있을 때 다음 위치를 예측 할 수 있을까?

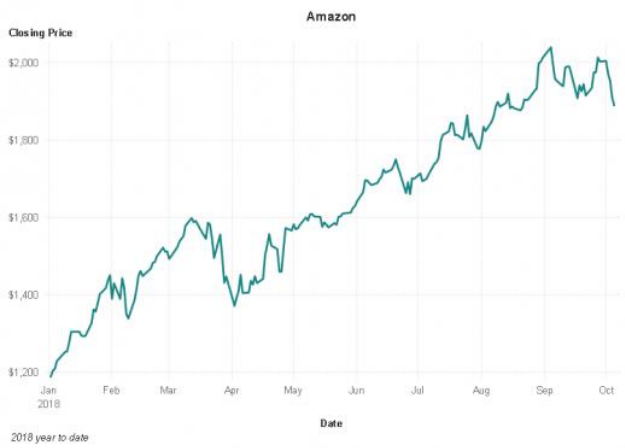


4.1 Introduction

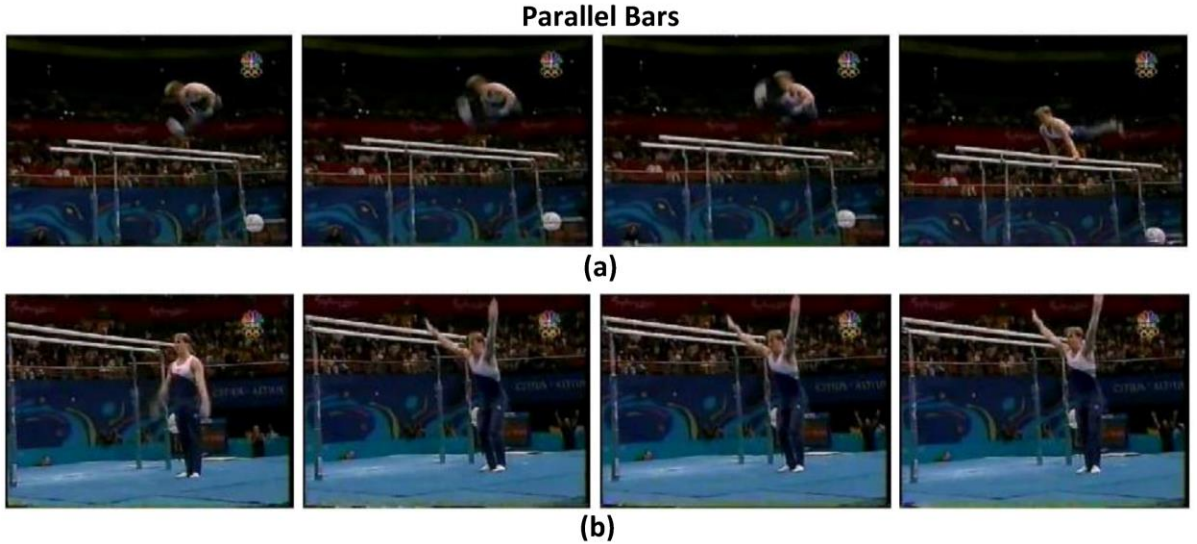
- 이전 시점들의 정보가 주어진다면?



4.1.1 Examples of Sequence Data



Time series



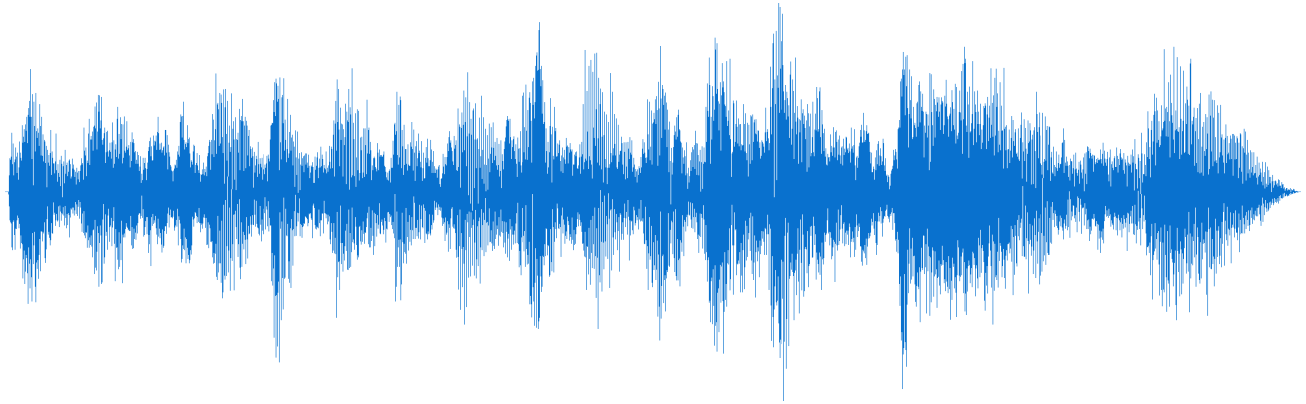
Video

오후 3시에 학교에 가야지

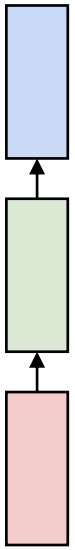
Text

4.1.1 Examples of Sequence Data

Audio

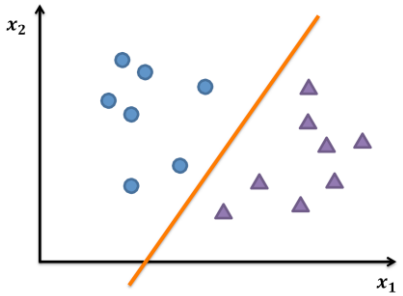


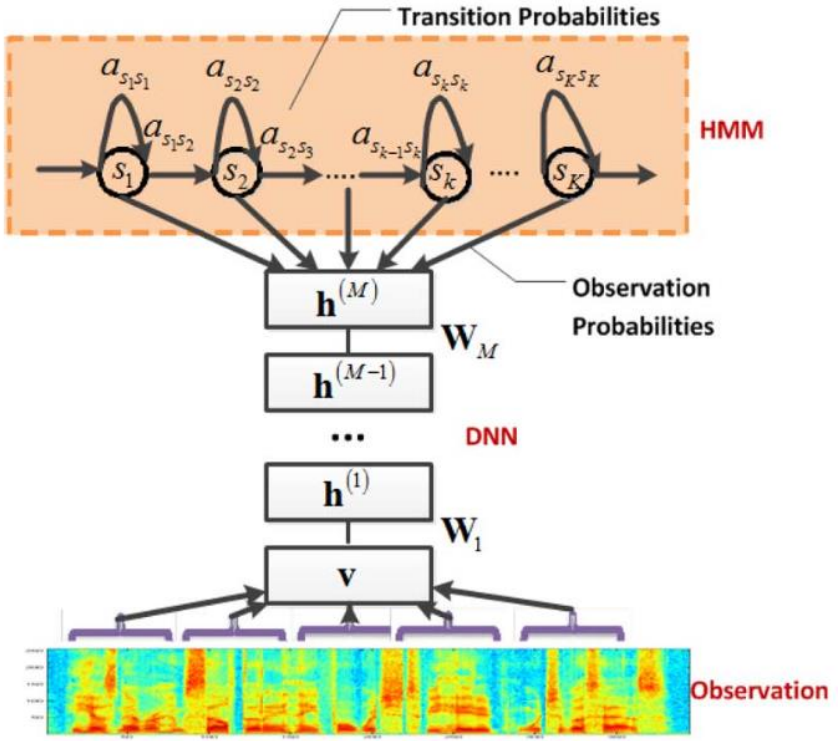
4.1.2 Sequence Modeling Applications



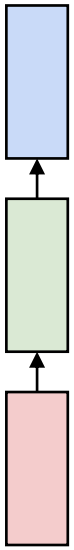
One-to-one

이진 분류



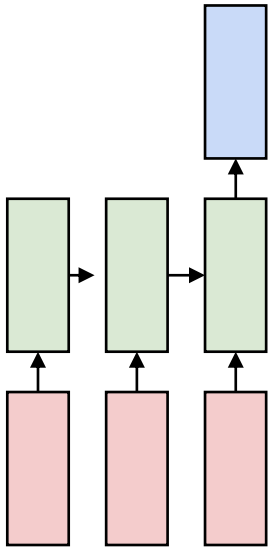


4.1.2 Sequence Modeling Applications



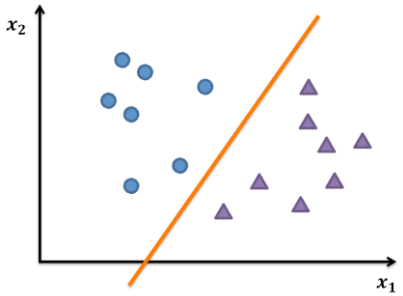
One-to-one

이진 분류



Many-to-one

감정 분류

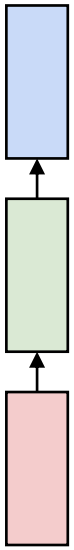


"교도소 이야기구먼,
솔직히 재미는 없다…"

Positive

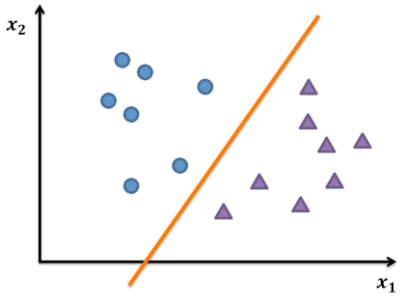
Negative

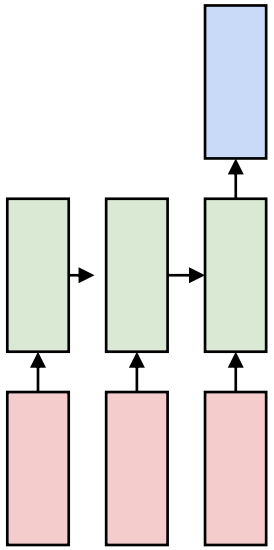
4.1.2 Sequence Modeling Applications



One-to-one

이진 분류





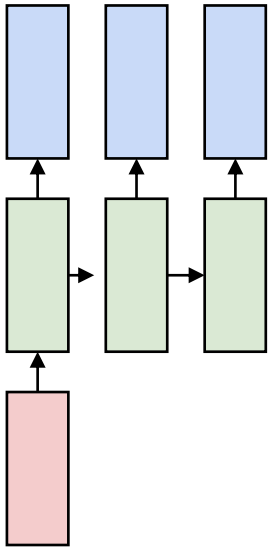
Many-to-one

감정 분류

"교도소 이야기구먼,
솔직히 재미는 없다…"

Positive

Negative

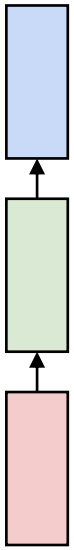


One-to-many

이미지 캡션 생성

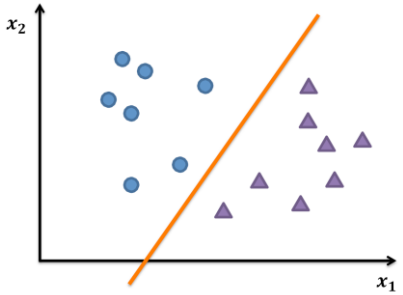
"빨간 헬멧을 쓴 남자가 작은
오토바이를 타고 비포장 도로에 있다"

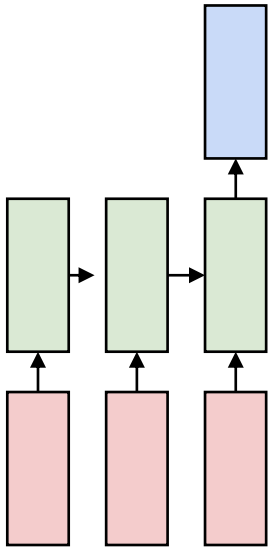
4.1.2 Sequence Modeling Applications



One-to-one

이진 분류





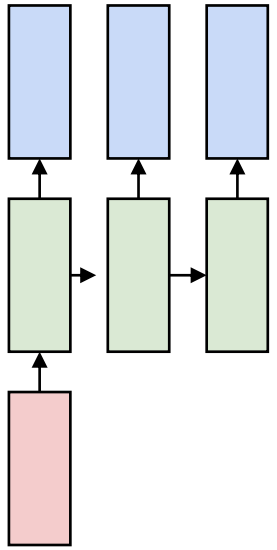
Many-to-one

감정 분류

"교도소 이야기구먼,
솔직히 재미는 없다…"

Positive

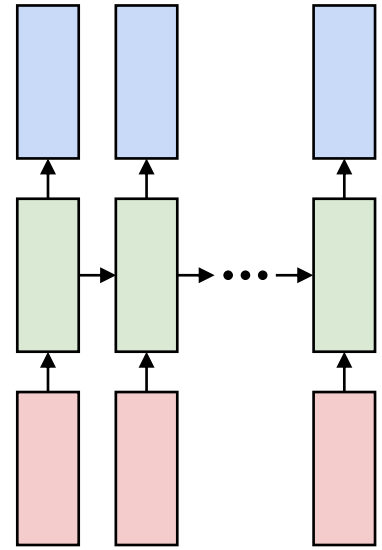
Negative



One-to-many

이미지 캡션 생성

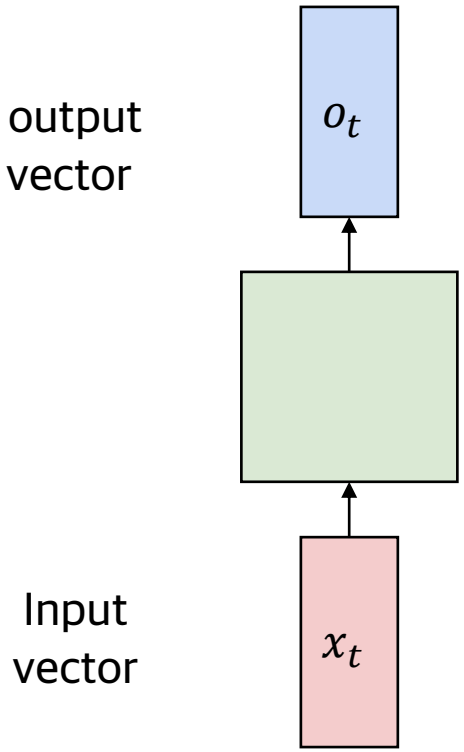
"빨간 헬멧을 쓴 남자가 작은
오토바이를 타고 비포장 도로에 있다"



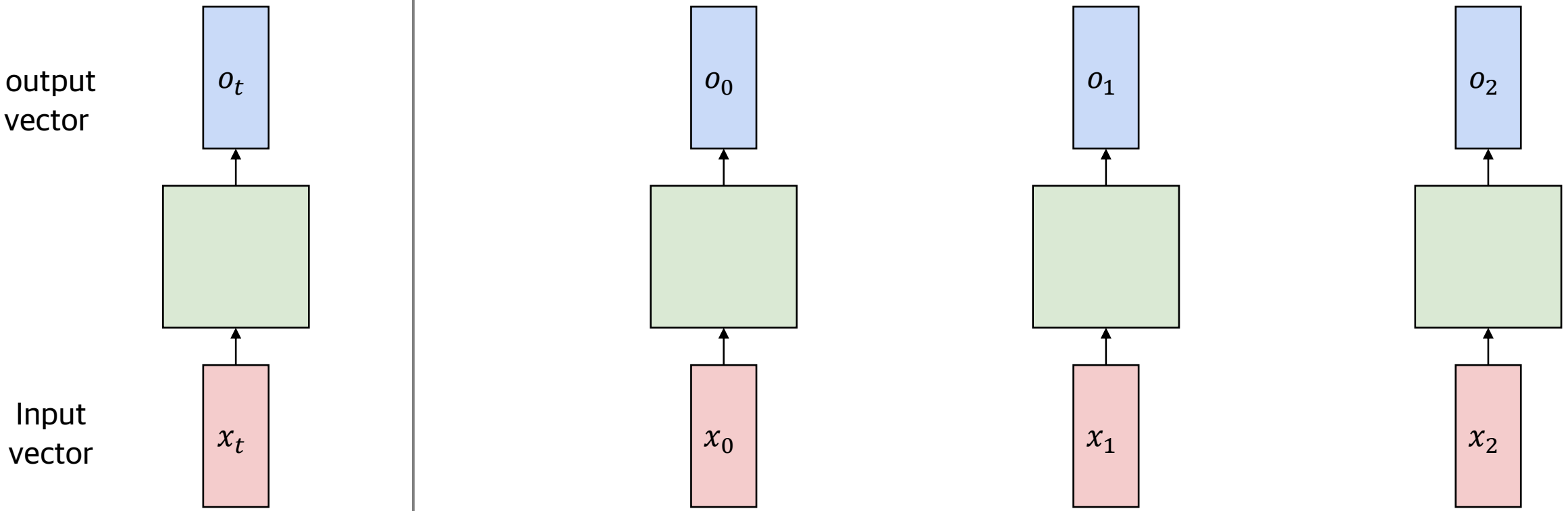
Many-to-many

번역, 음성인식

4.1.2 Handling Individual Time steps



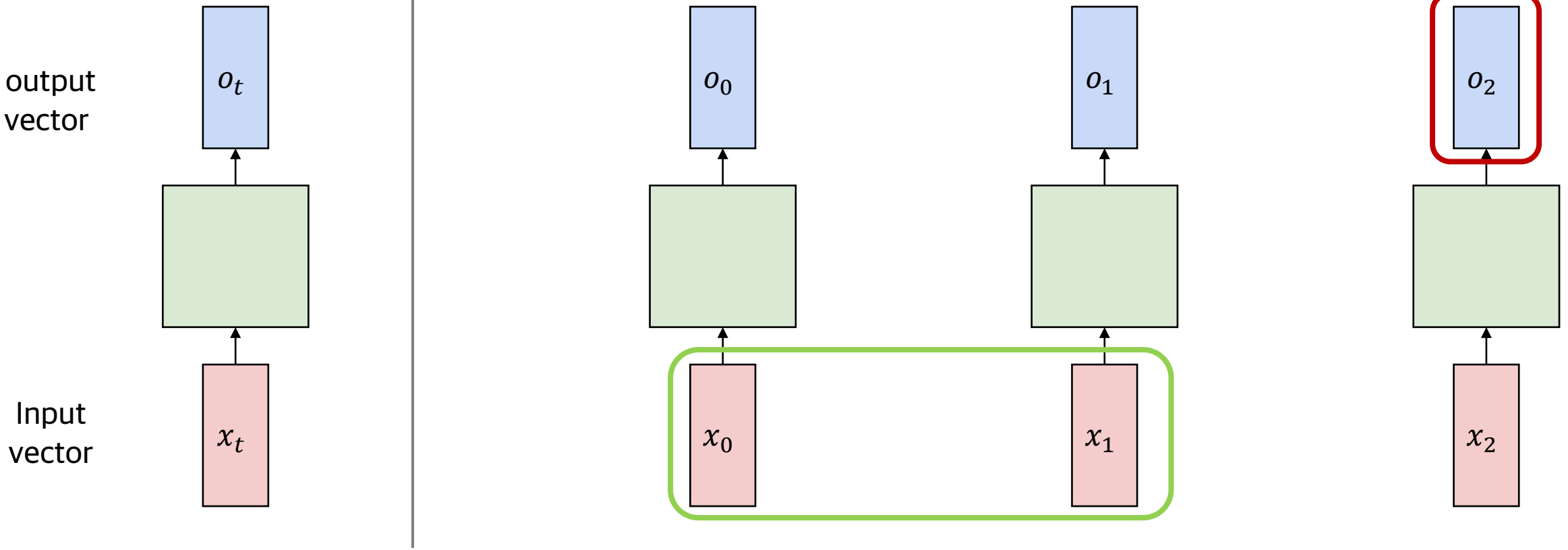
4.1.2 Handling Individual Time steps



$$o_t = f(x_t)$$

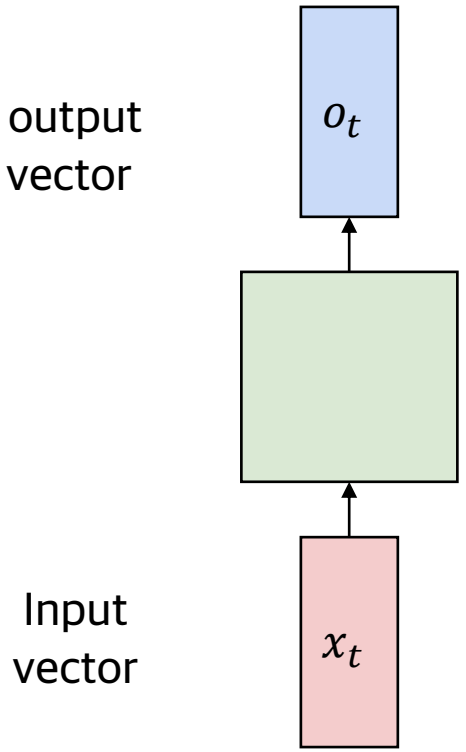
4.1.2 Handling Individual Time steps

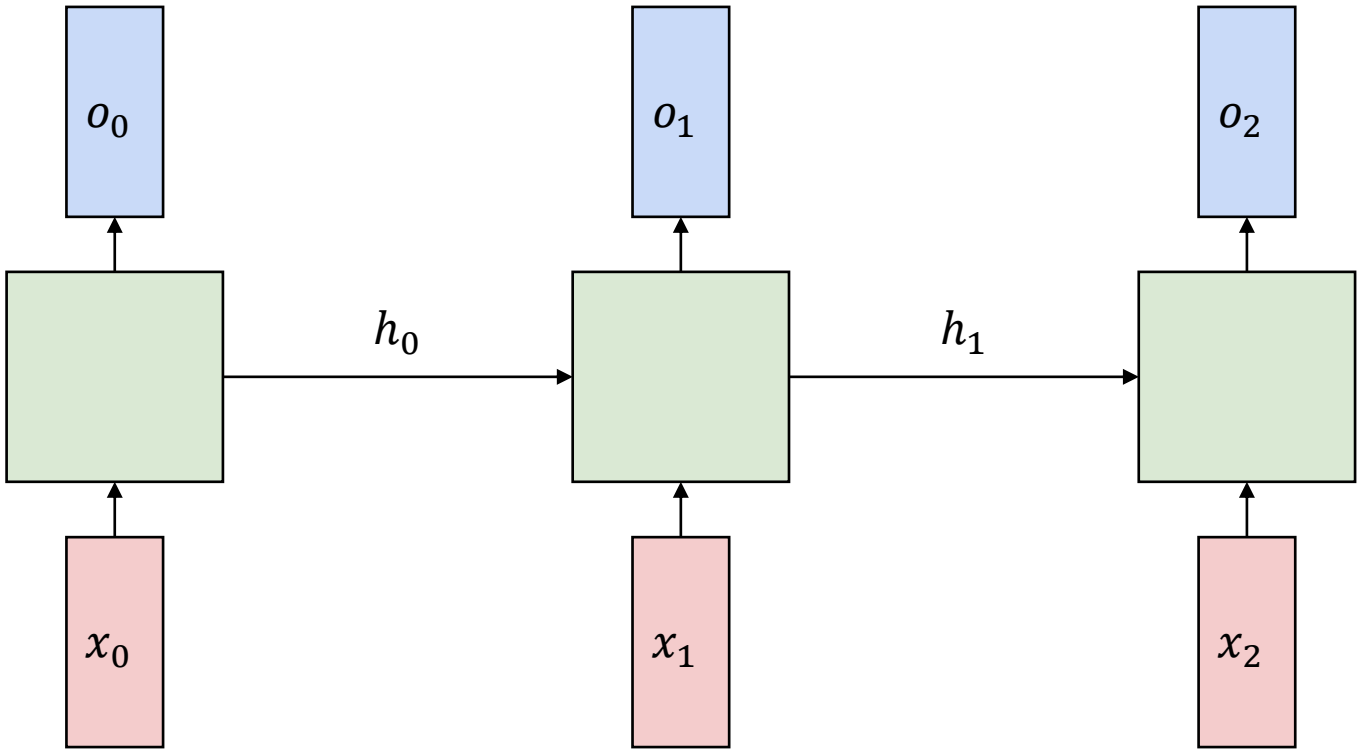
이전 시점들과는 독립적으로
현재 시점의 입력에 따라 출력을 계산



$$o_t = f(x_t)$$

4.1.2 Handling Individual Time steps

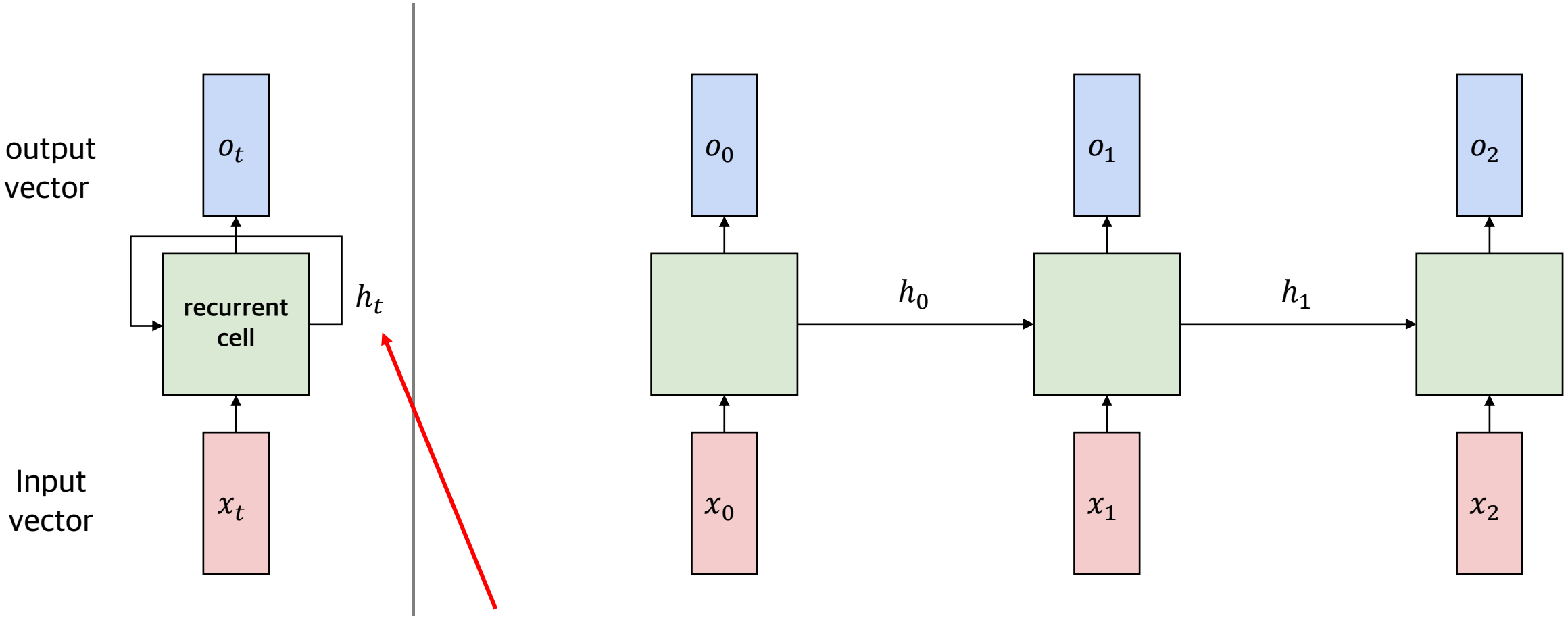




$$o_t = f(x_t, h_{t-1})$$

output input past memory

4.2 Neurons with Recurrence

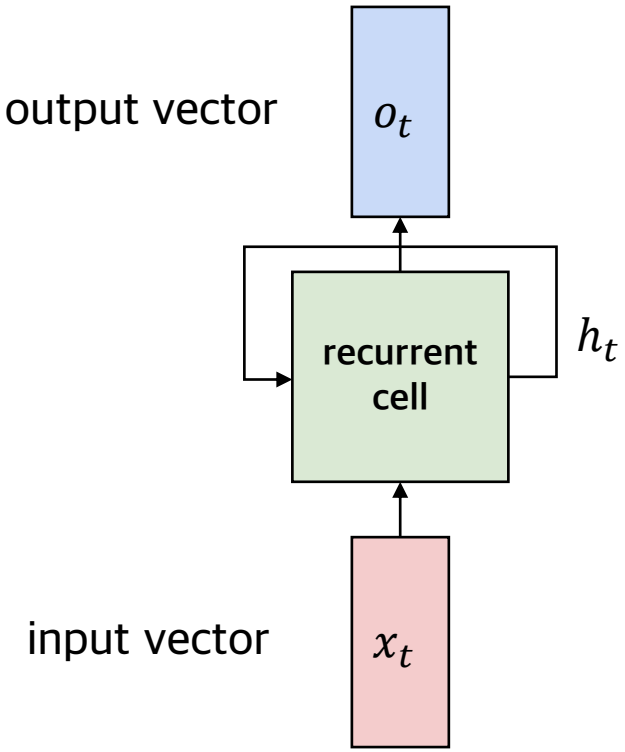


RNN에는 sequence를 처리할 때마다 바뀌는 hidden state가 존재

$$o_t = f(x_t, h_{t-1})$$

output input past memory

4.2.1 Recurrent Neural Networks (RNNs)



Hidden state

h_t는 시간 t에서 입력 벡터 x_t와 이전 상태 h_{t-1}로 계산

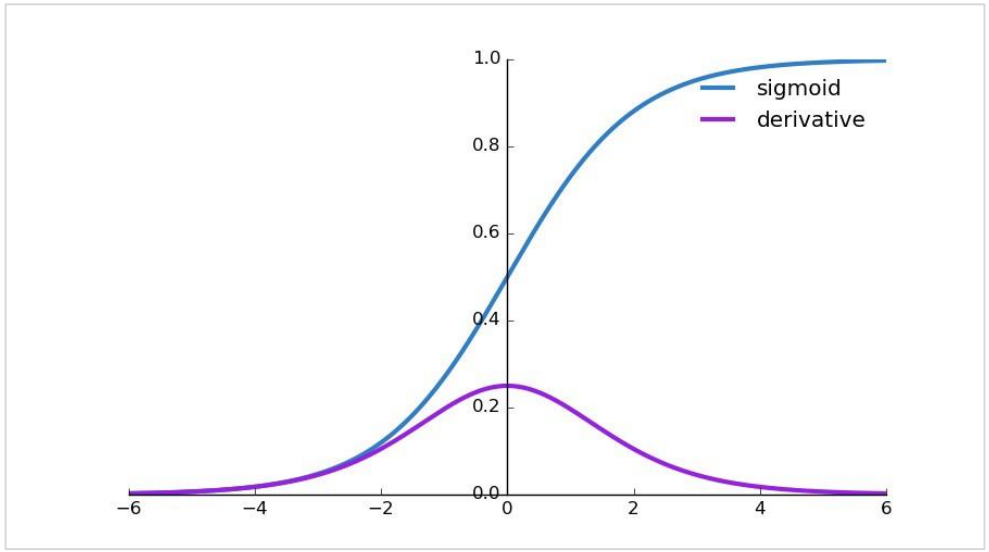
함수 f는tanh나 $sigmoid$와 같은 비선형 함수

$$h_t = f_W(x_t, h_{t-1})$$

hidden state — Function with weight W — input — previous time step's hidden state

image from : https://medium.com/@omkar.nallagoni

4.2.2 Activation Functions



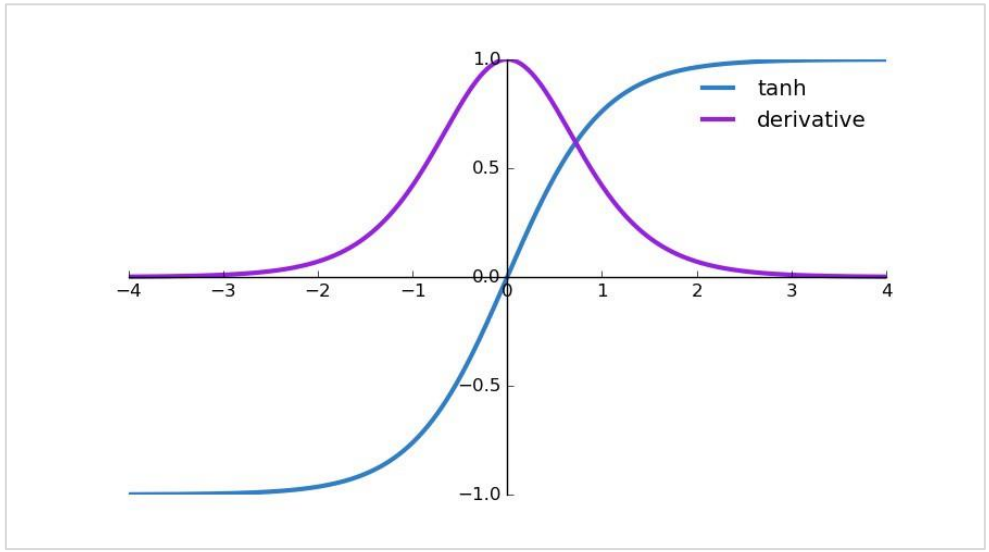
sigmoid

x의 값을 [0,1] 사이의 값으로 변환

not zero-centered

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial}{\partial x}\sigma(x) = \sigma(x)(1-\sigma(x))$$



tanh

x의 값을 [-1,1] 사이의 값으로 변환

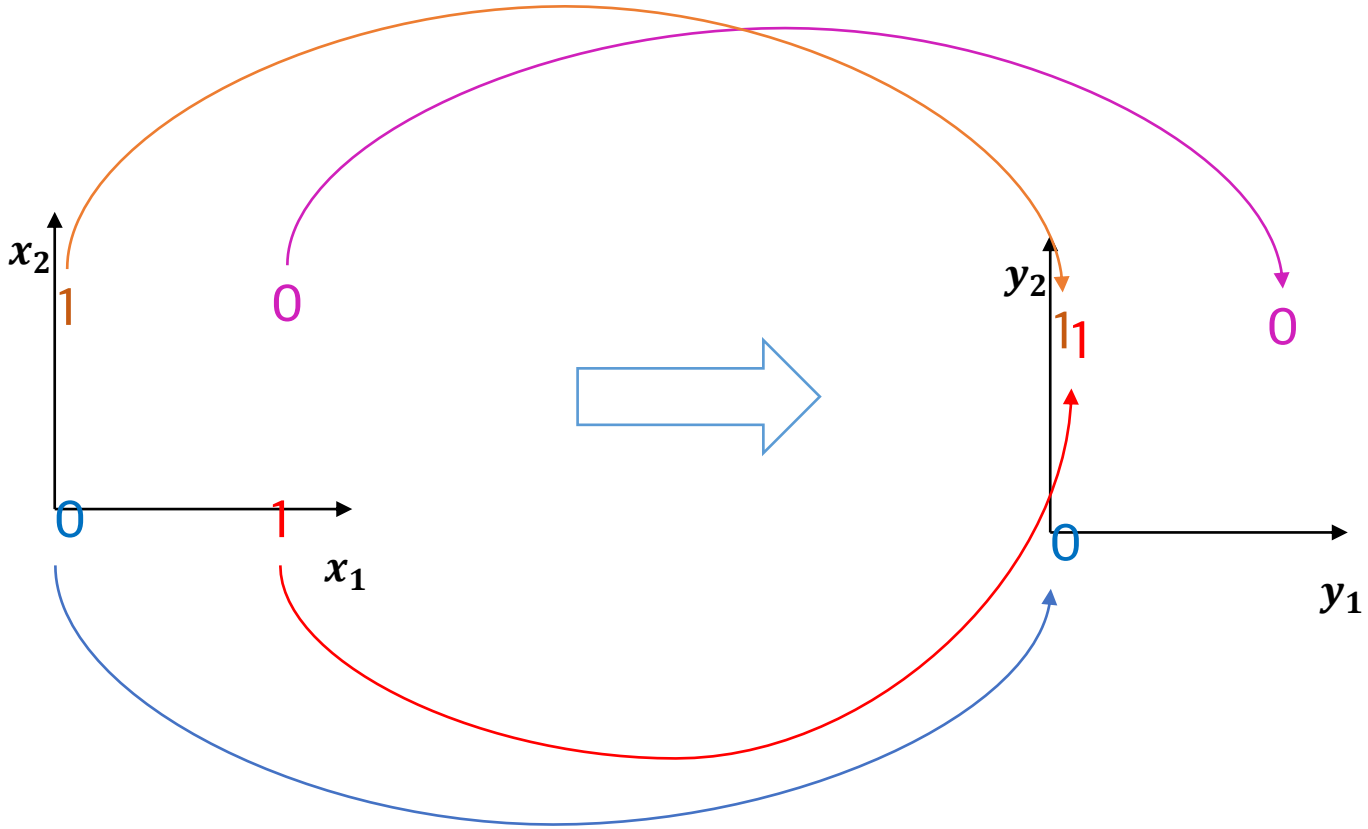
zero-centered

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

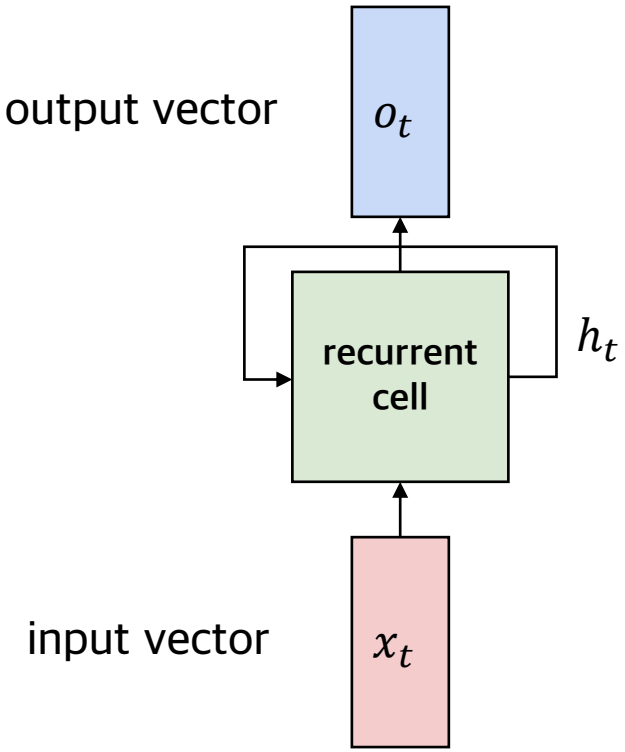
$$\frac{d}{dx}\tanh(x) = 1-\tanh(x)^2$$

4.2.3 The Role of the Hidden Layer

The role of the hidden layer is to map samples from the input space to the hidden layer space.



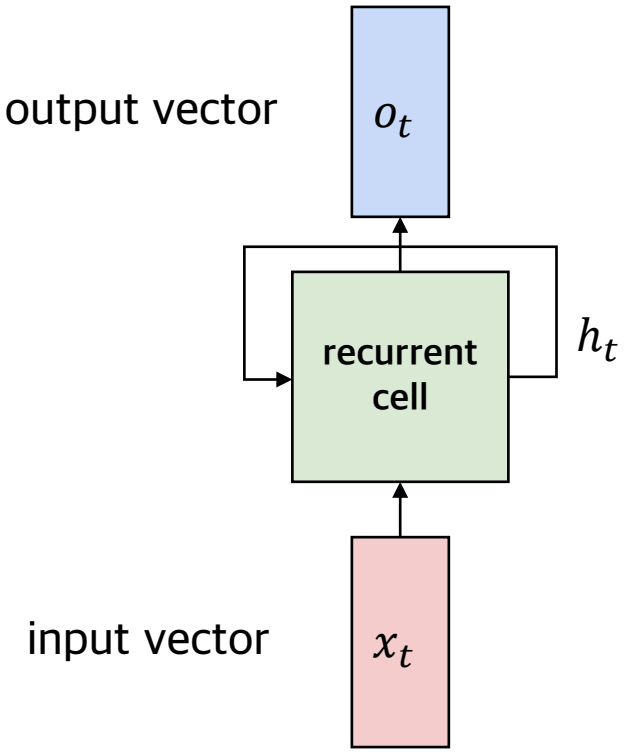
4.2.4 RNN State Update and Output



Input Vector

$$x_t$$

4.2.4 RNN State Update and Output



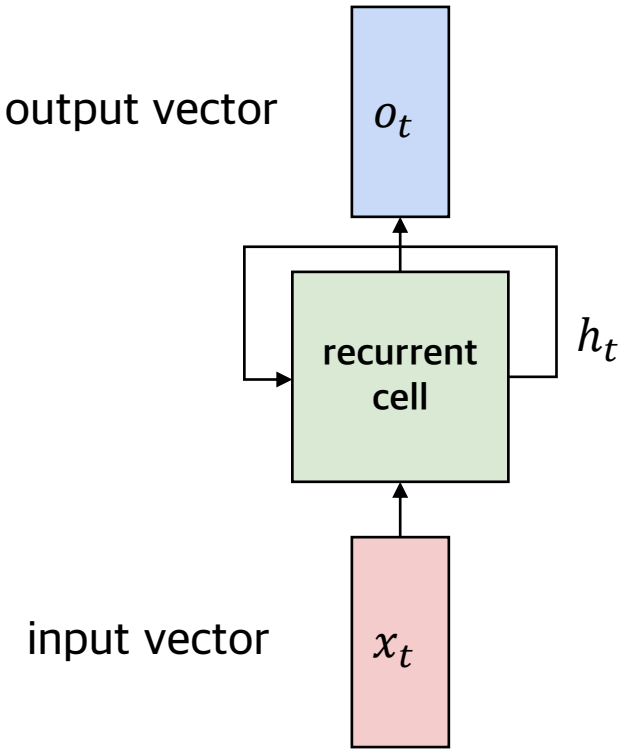
Update Hidden State

$$h_t = \tanh(h_{t-1}W_{hh} + x_t W_{xh})$$

Input Vector

$$x_t$$

4.2.4 RNN State Update and Output



Output Vector

$$o_t = h_t W_{ho}$$

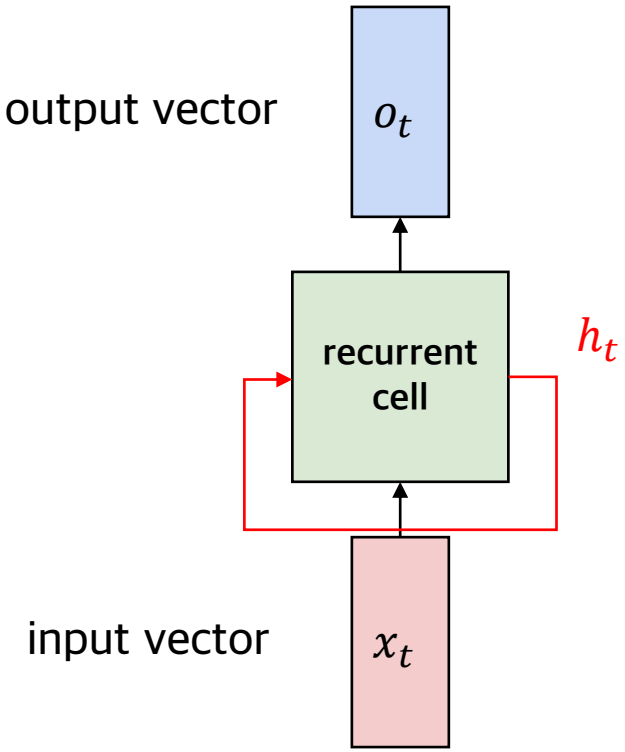
Update Hidden State

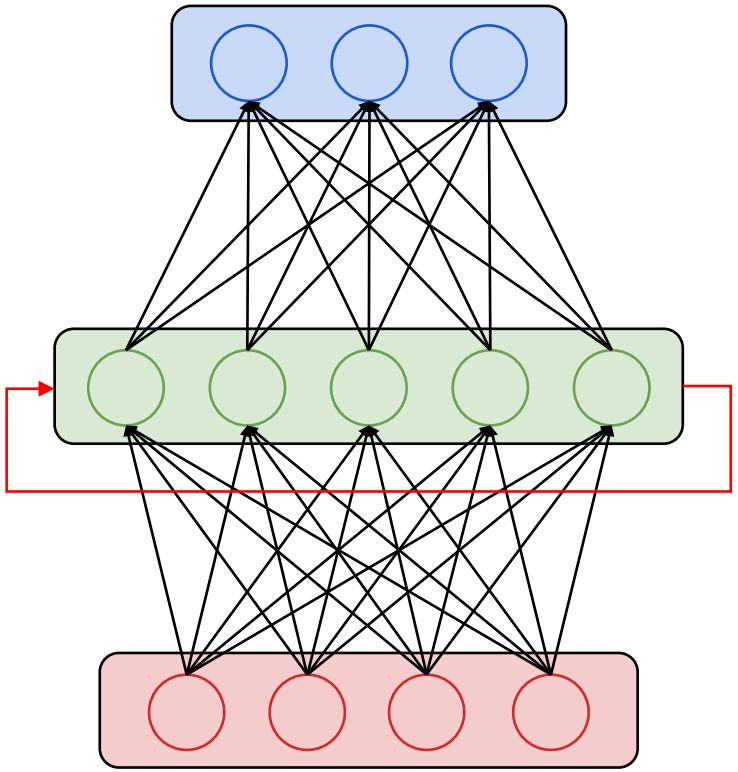
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

Input Vector

$$x_t$$

4.2.5 RNN : Graph Representation



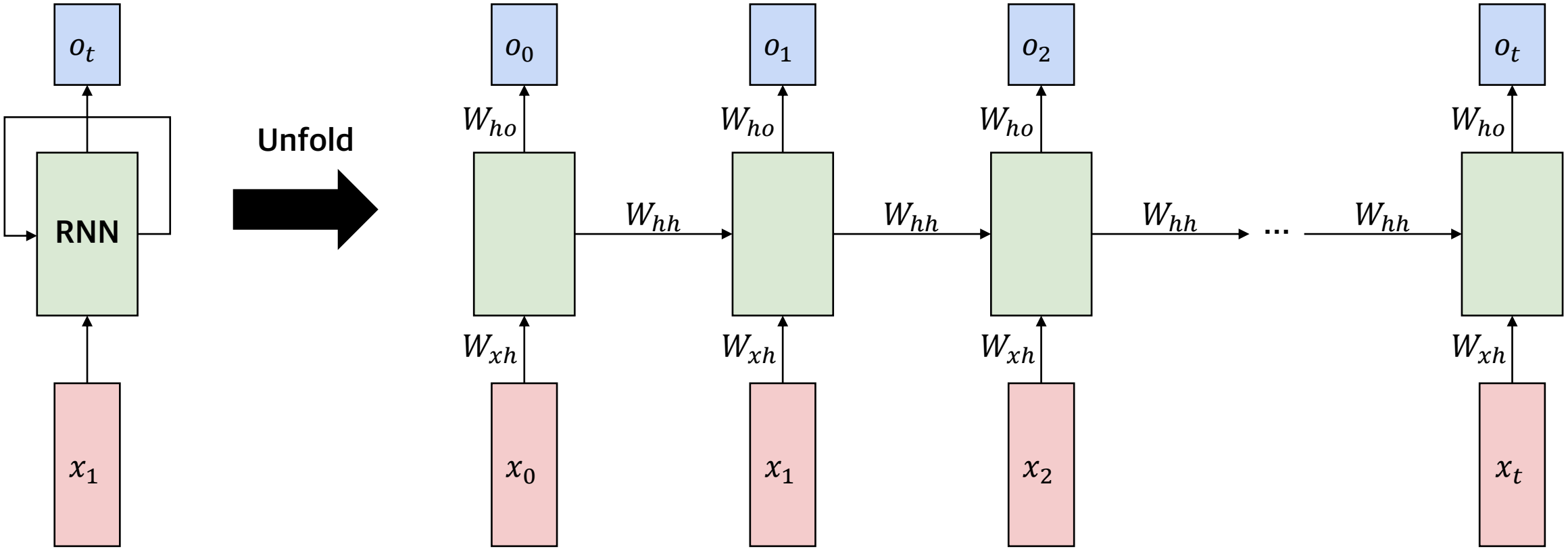


4.2.6 RNNs : Computational Graph Across Time

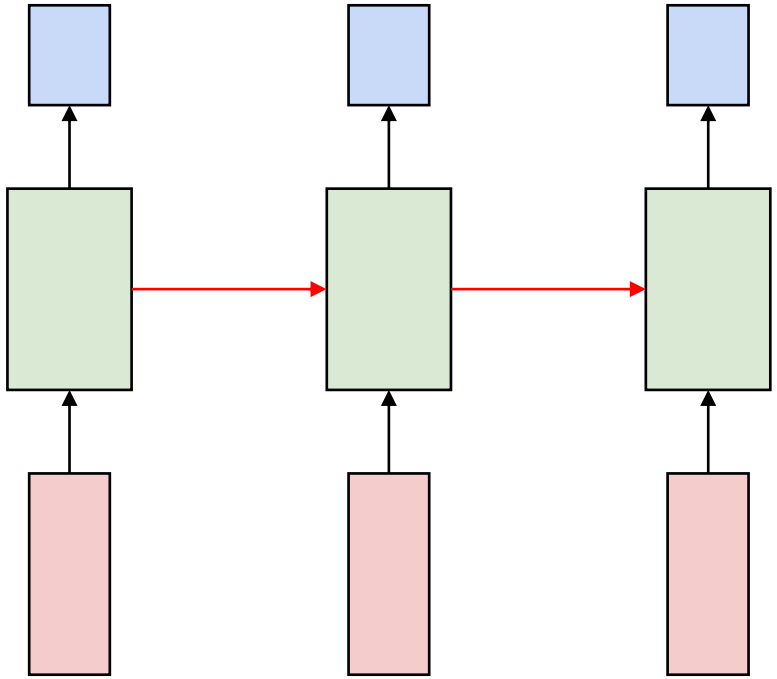
매 시간마다

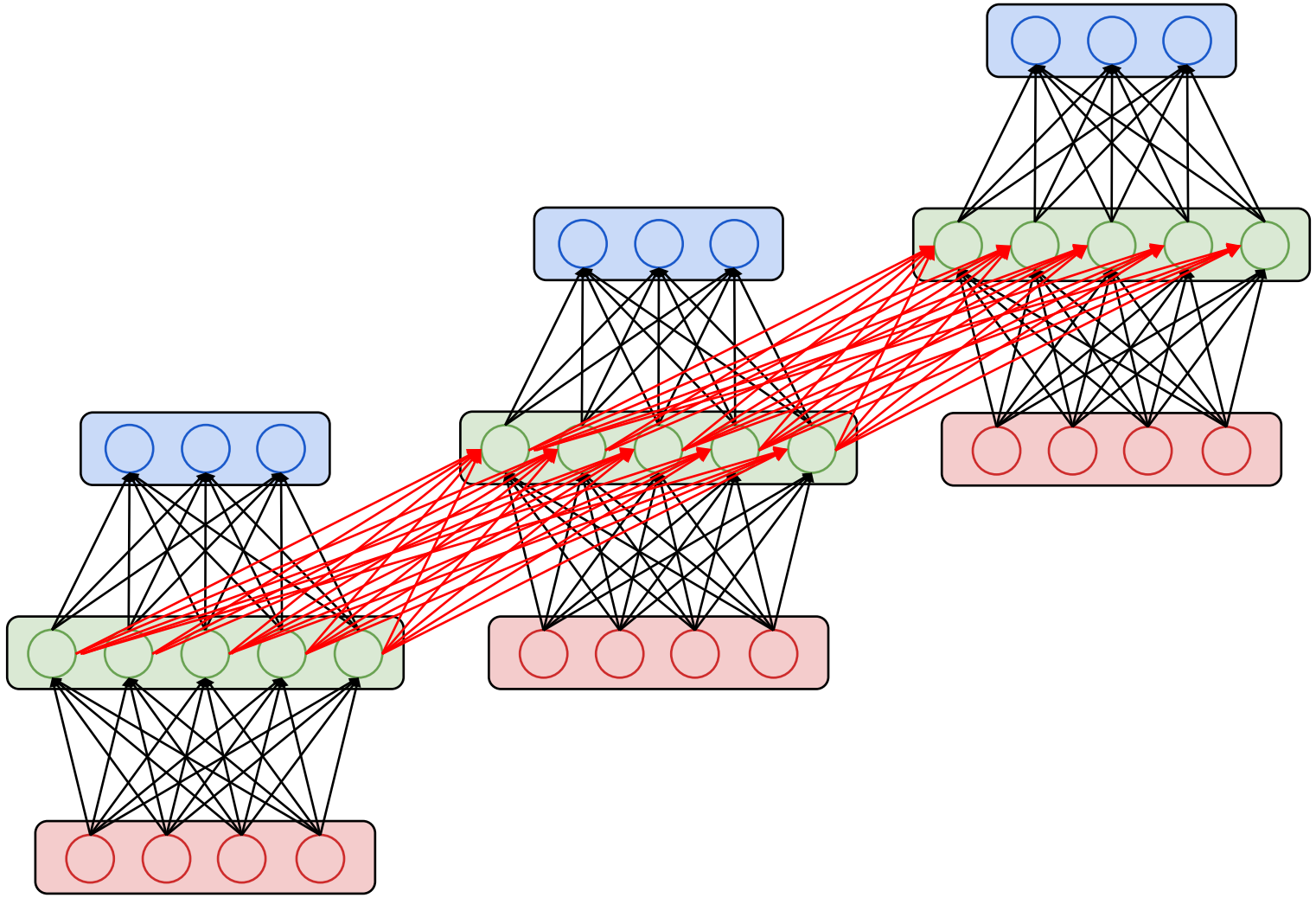
같은 weight matrices를

재사용



4.2.6 RNNs : Computational Graph Across Time



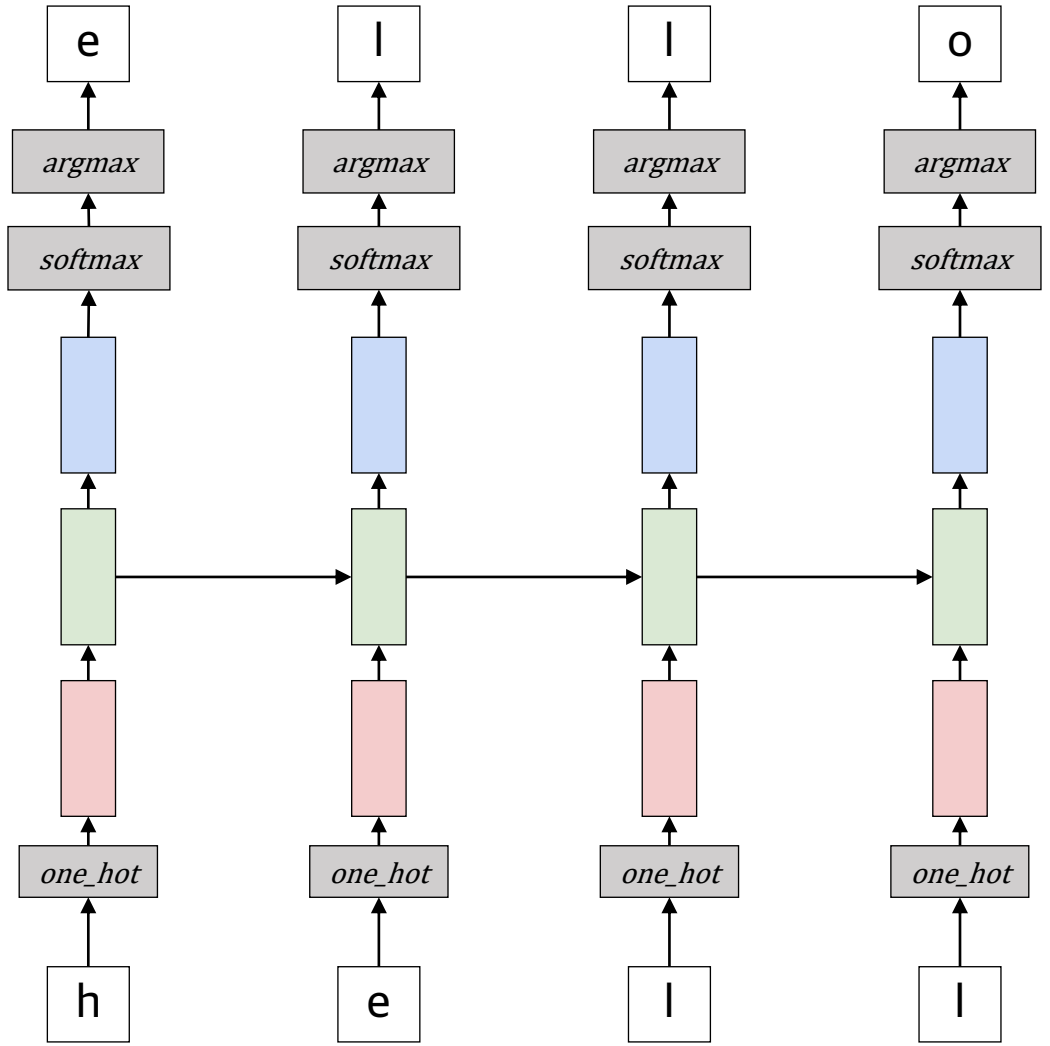


4.2.7 Example : Predict Sequence of Character

- 다음 문자를 예측하는 네트워크를 설계
  - “hello”라는 단어 기준
  - h,e,l,l이 입력으로 들어왔을 때 o를 예측
    - {‘h’:0, ‘e’:1, ‘l’:2, ‘o’:3}
  - one-hot encoding

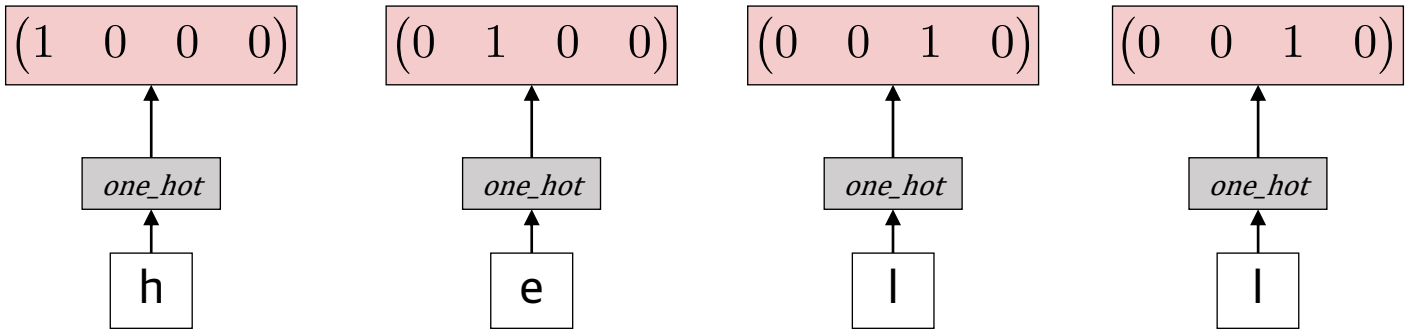
h	[1,0,0,0]
e	[0,1,0,0]
l	[0,0,1,0]
o	[0,0,0,1]

  - hidden layer는 1층의 RNN을 사용



4.2.7 Example : Predict Sequence of Character

- 다음과 같은 one-hot encoding을 입력으로 받음



- RNN model :

$$o_t = h_t W_{ho}$$

$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

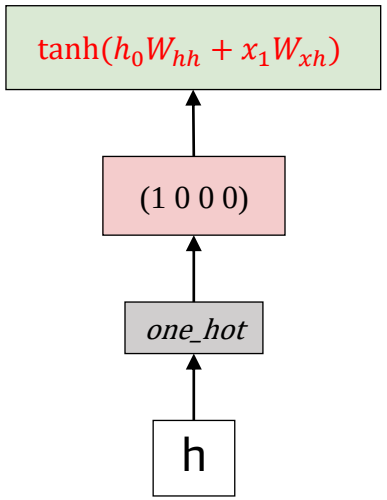
4.2.7 Example : Predict Sequence of Character

- W_{xh}는 다음과 같은 0에서 1사이의 값으로 random initialized 4×3 행렬

$$W_{xh} = \begin{pmatrix} 0.2973 & 0.2766 & 0.7974 \\ 0.3869 & 0.9170 & 0.4125 \\ 0.5538 & 0.5646 & 0.5026 \\ 0.2206 & 0.6801 & 0.3880 \end{pmatrix}$$

- $x_1 W_{xh}$는 다음과 같이 계산

$$x_1 W_{xh} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0.2973 & 0.2766 & 0.7974 \\ 0.3869 & 0.9170 & 0.4125 \\ 0.5538 & 0.5646 & 0.5026 \\ 0.2206 & 0.6801 & 0.3880 \end{pmatrix} = \begin{pmatrix} 0.2973 & 0.2766 & 0.7974 \end{pmatrix}$$



$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

4.2.7 Example : Predict Sequence of Character

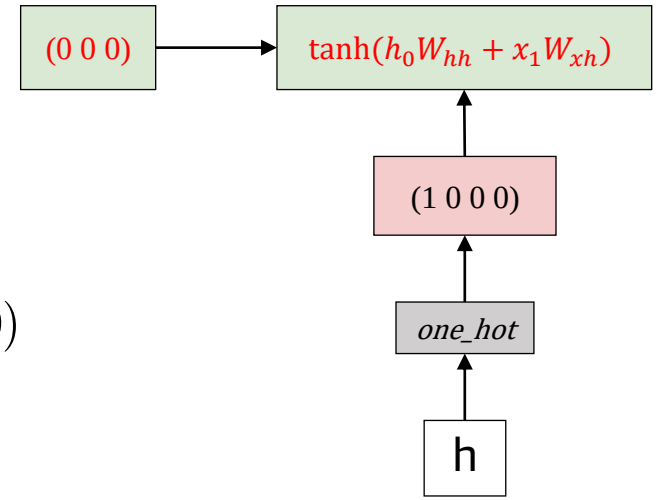
- W_{hh}는 다음과 같은 0에서 1사이의 값으로 random initialized 3×3 행렬

$$W_{hh} = \begin{pmatrix} 0.3152 & 0.5083 & 0.9454 \\ 0.3008 & 0.6058 & 0.2999 \\ 0.9741 & 0.3419 & 0.9133 \end{pmatrix}$$

- $h_{t-1}W_{hh}$를 계산하기 위해서는 initial hidden state h_0가 필요
  - 일반적으로 initial hidden state로는 0를 사용(영행렬)

$$h_0 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

$$h_0 W_{hh} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0.3152 & 0.5083 & 0.9454 \\ 0.3008 & 0.6058 & 0.2999 \\ 0.9741 & 0.3419 & 0.9133 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

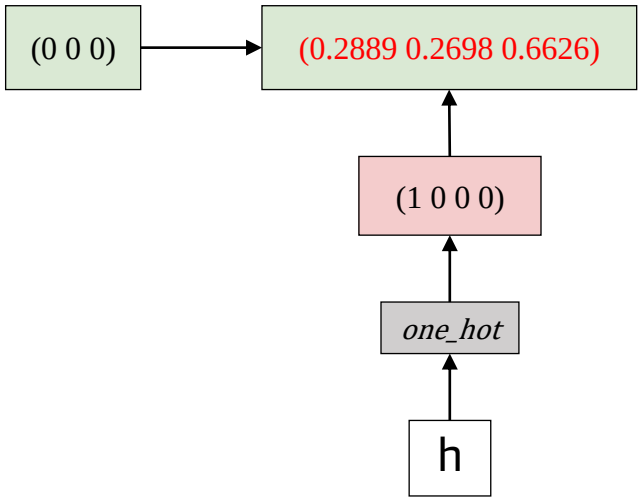


$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1}W_{hh} + x_t W_{xh})$$

4.2.7 Example : Predict Sequence of Character

- h_1은 다음과 같이 update

$$
\begin{aligned}
h_1 &= \tanh\left(h_0 W_{hh} + x_1 W_{xh}\right) \\
&= \tanh\left(\begin{pmatrix}0 & 0 & 0\end{pmatrix} + \begin{pmatrix}0.2973 & 0.2766 & 0.7974\end{pmatrix}\right) \\
&= \tanh\left(\begin{pmatrix}0.2973 & 0.2766 & 0.7974\end{pmatrix}\right) \\
&= \begin{pmatrix}0.2889 & 0.2698 & 0.6626\end{pmatrix}
\end{aligned}
$$



$$
\begin{aligned}
o_t &= h_t W_{ho} \\
h_t &= \tanh(h_{t-1} W_{hh} + x_t W_{xh})
\end{aligned}
$$

4.2.7 Example : Predict Sequence of Character

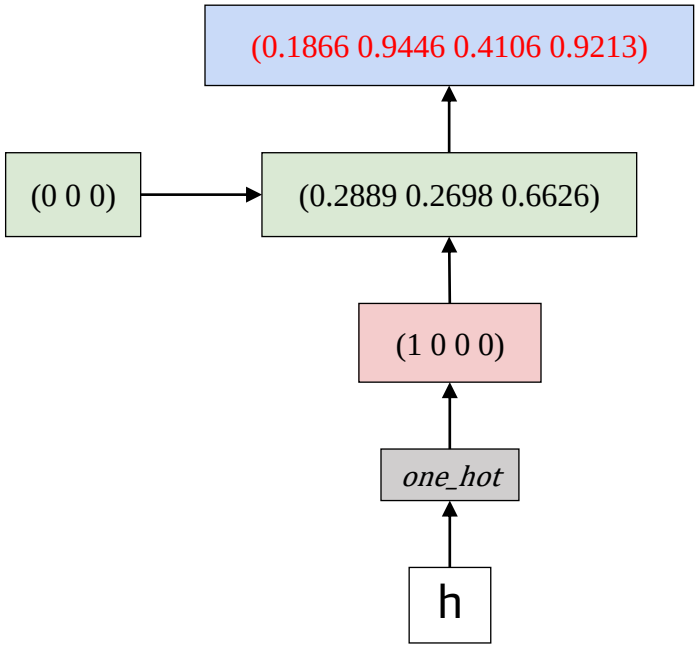
- W_{hy}는 같은 0에서 1사이의 값으로 random initialized 3×4 행렬

$$W_{ho} = \begin{pmatrix} 0.3189 & 0.6216 & 0.5164 & 0.7501 \\ 0.0260 & 0.6230 & 0.0179 & 0.6123 \\ 0.1320 & 0.9009 & 0.3873 & 0.8142 \end{pmatrix}$$

$$o_1 = h_1 W_{ho}$$

$$= \begin{pmatrix} 0.2889 & 0.2698 & 0.6626 \end{pmatrix} \begin{pmatrix} 0.3189 & 0.6216 & 0.5164 & 0.7501 \\ 0.0260 & 0.6230 & 0.0179 & 0.6123 \\ 0.1320 & 0.9009 & 0.3873 & 0.8142 \end{pmatrix}$$

$$= \begin{pmatrix} 0.1866 & 0.9446 & 0.4106 & 0.9213 \end{pmatrix}$$



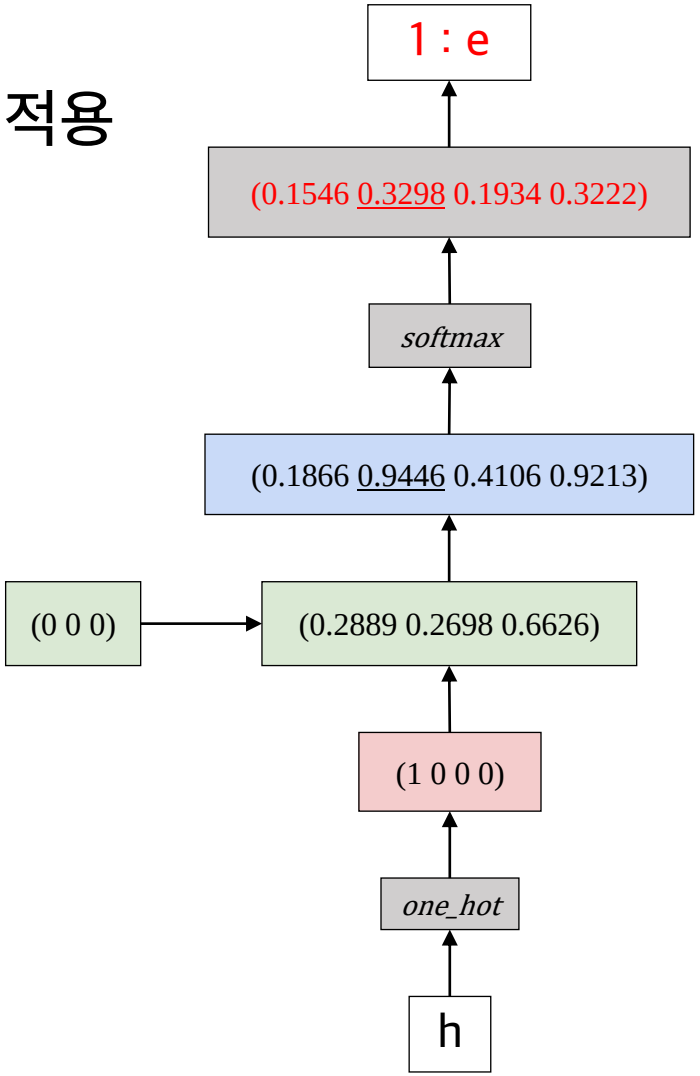
$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

4.2.7 Example : Predict Sequence of Character

- o_1의 softmax 값과 이중 가장 큰 값의 index를 추출하는 argmax를 적용

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

$$\begin{aligned}\text{softmax}(o_1) &= \text{softmax}(\begin{pmatrix}0.1866 & 0.9446 & 0.4106 & 0.9213\end{pmatrix}) \\ &= \begin{pmatrix}0.1546 & 0.3298 & 0.1934 & 0.3222\end{pmatrix})\end{aligned}$$



$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

4.2.7 Example : Predict Sequence of Character

- W_{xh}, W_{hh}는 동일한 weight를 사용
- $x_2 W_{xh}$과 $h_1 W_{hh}$는 다음과 같이 계산

모든 시점에서 동일한 weight를 사용

$$x_2 W_{xh} = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0.2973 & 0.2766 & 0.7974 \\ 0.3869 & 0.9170 & 0.4125 \\ 0.5538 & 0.5646 & 0.5026 \\ 0.2206 & 0.6801 & 0.3880 \end{pmatrix} = \begin{pmatrix} 0.3869 & 0.9170 & 0.4125 \end{pmatrix}$$

$$h_1 W_{hh} = \begin{pmatrix} 0.2889 & 0.2698 & 0.6626 \end{pmatrix} \begin{pmatrix} 0.3152 & 0.5083 & 0.9454 \\ 0.3008 & 0.6058 & 0.2999 \\ 0.9741 & 0.3419 & 0.9133 \end{pmatrix} = \begin{pmatrix} 0.8176 & 0.5368 & 0.9591 \end{pmatrix}$$

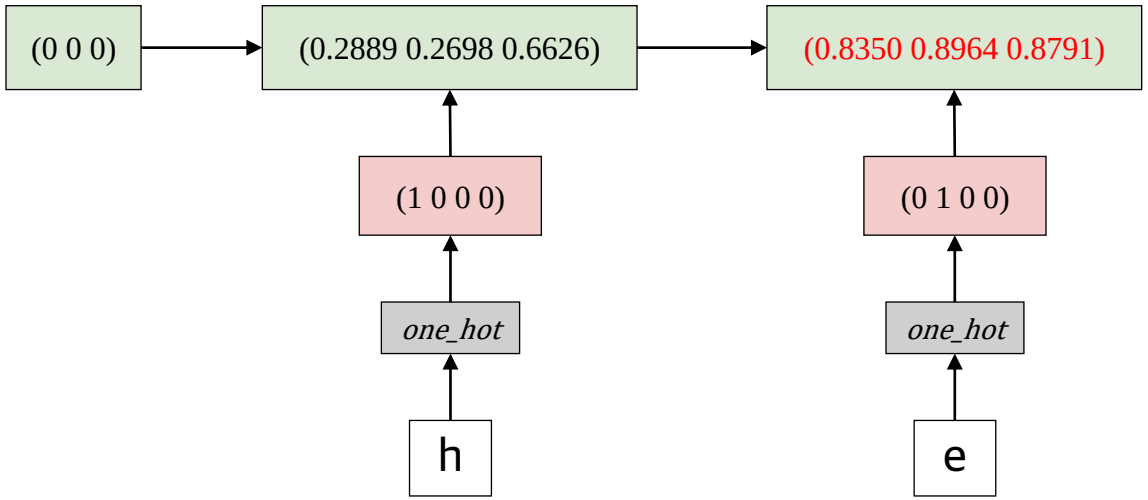
이전 시점에 update된 hidden state의 값을 사용

$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

4.2.7 Example : Predict Sequence of Character

- h_2는 다음과 같이 update

$$
\begin{aligned}
h_2 &= \tanh\left(h_1 W_{hh} + x_2 W_{xh}\right) \\
&= \tanh\left(\begin{pmatrix} 0.8176 & 0.5368 & 0.9591 \end{pmatrix} + \begin{pmatrix} 0.3869 & 0.9170 & 0.4125 \end{pmatrix}\right) \\
&= \tanh\left(\begin{pmatrix} 1.2045 & 1.4538 & 1.3716 \end{pmatrix}\right) \\
&= \begin{pmatrix} 0.8350 & 0.8964 & 0.8791 \end{pmatrix}
\end{aligned}
$$



$$
\begin{aligned}
o_t &= h_t W_{ho} \\
h_t &= \tanh(h_{t-1} W_{hh} + x_t W_{xh})
\end{aligned}
$$

4.2.7 Example : Predict Sequence of Character

- o_2는 시점 1에서와 동일한 방법으로 계산 가능

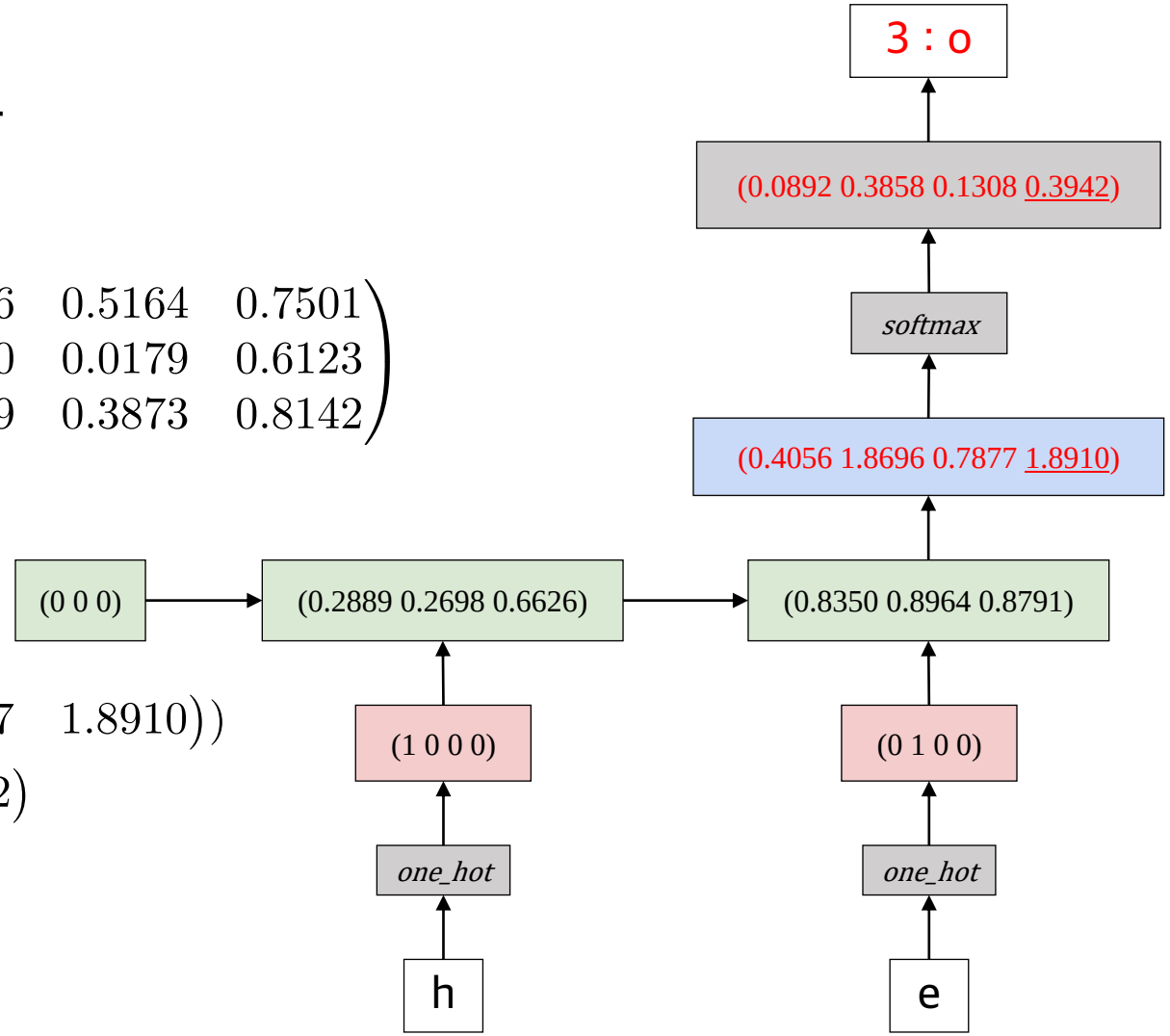
$$o_2 = h_2 W_{ho}$$

$$= \begin{pmatrix} 0.8350 & 0.8946 & 0.8791 \end{pmatrix} \begin{pmatrix} 0.3189 & 0.6216 & 0.5164 & 0.7501 \\ 0.0260 & 0.6230 & 0.0179 & 0.6123 \\ 0.1320 & 0.9009 & 0.3873 & 0.8142 \end{pmatrix}$$

$$= \begin{pmatrix} 0.4056 & 1.8696 & 0.7877 & 1.8910 \end{pmatrix}$$

$$\text{softmax}(o_2) = \text{softmax}(\begin{pmatrix} 0.4056 & 1.8696 & 0.7877 & 1.8910 \end{pmatrix})$$

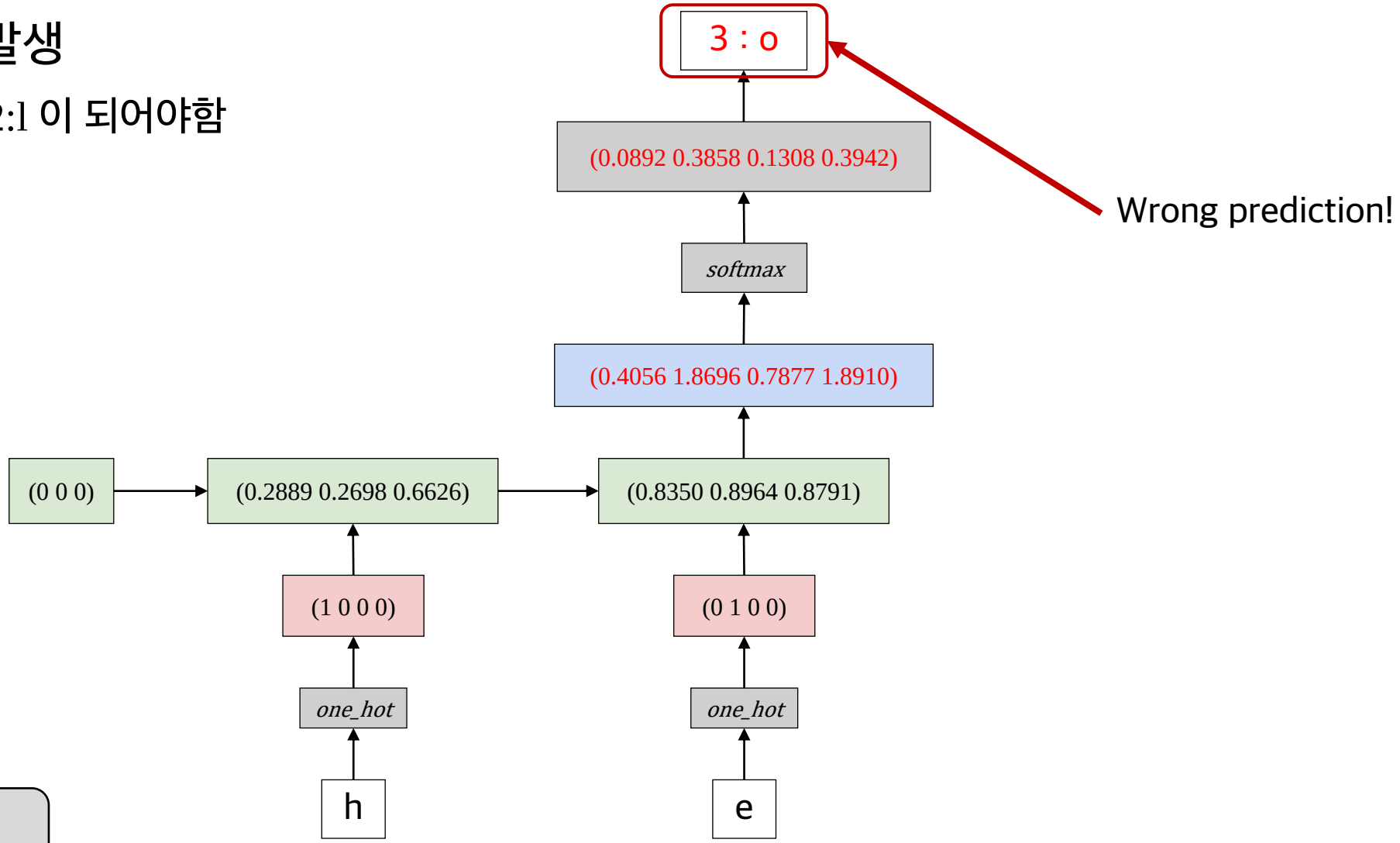
$$= \begin{pmatrix} 0.0892 & 0.3858 & 0.1308 & 0.3942 \end{pmatrix}$$



$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

4.2.7 Example : Predict Sequence of Character

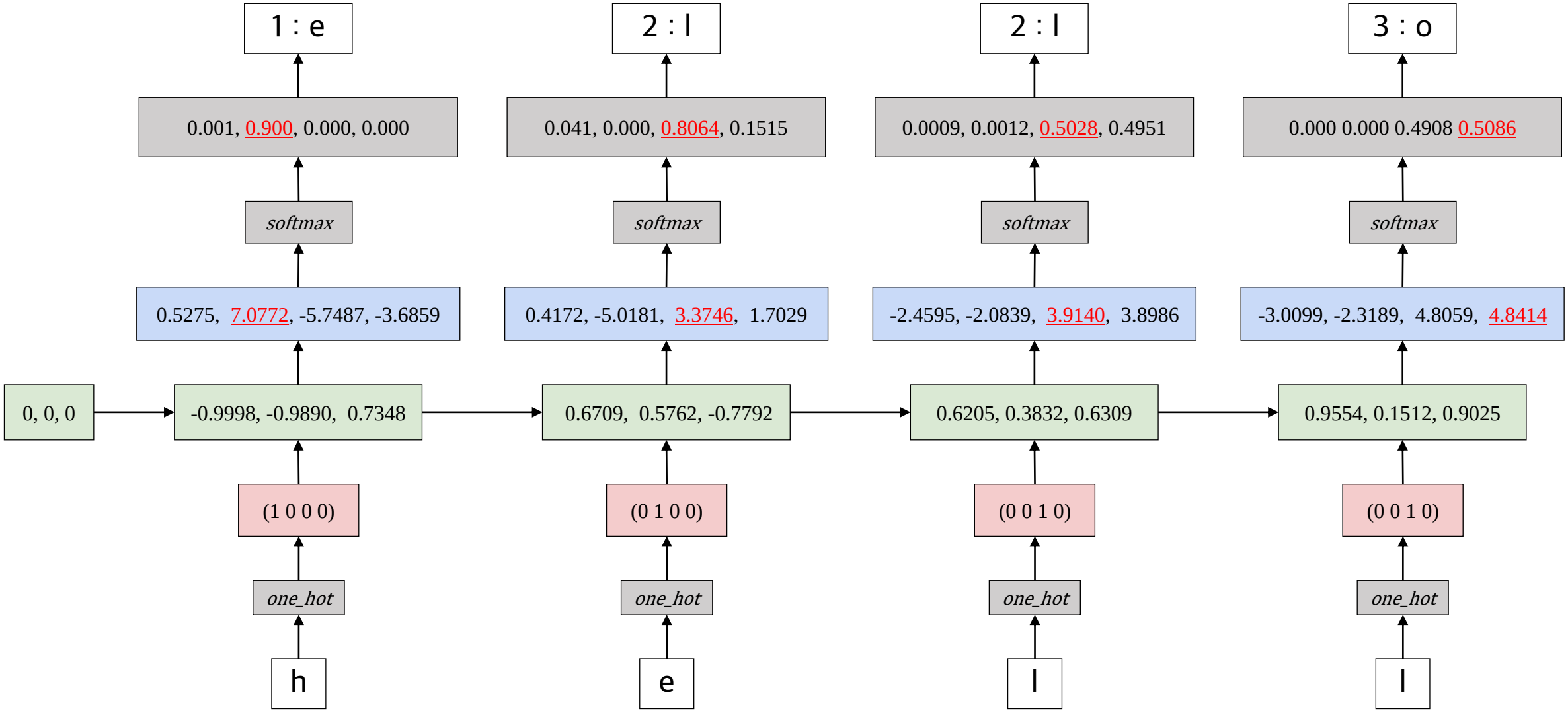
- 잘못된 예측이 발생
  - 올바른 예측은 2:1 이 되어야함



$$o_t = h_t W_{ho}$$
$$h_t = \tanh(h_{t-1} W_{hh} + x_t W_{xh})$$

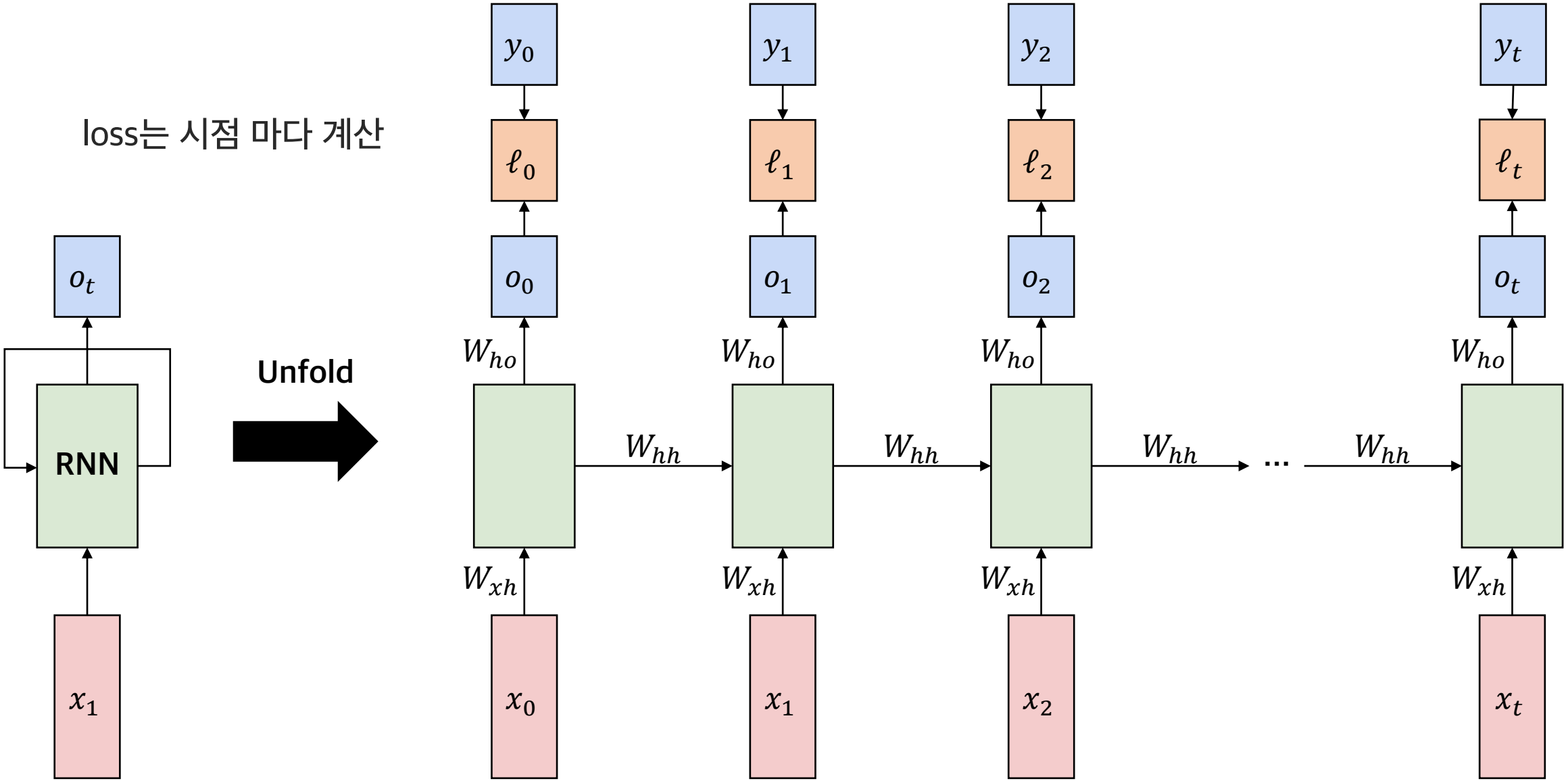
4.2.7 Example : Predict Sequence of Character

- 올바르게 학습된 RNN Model



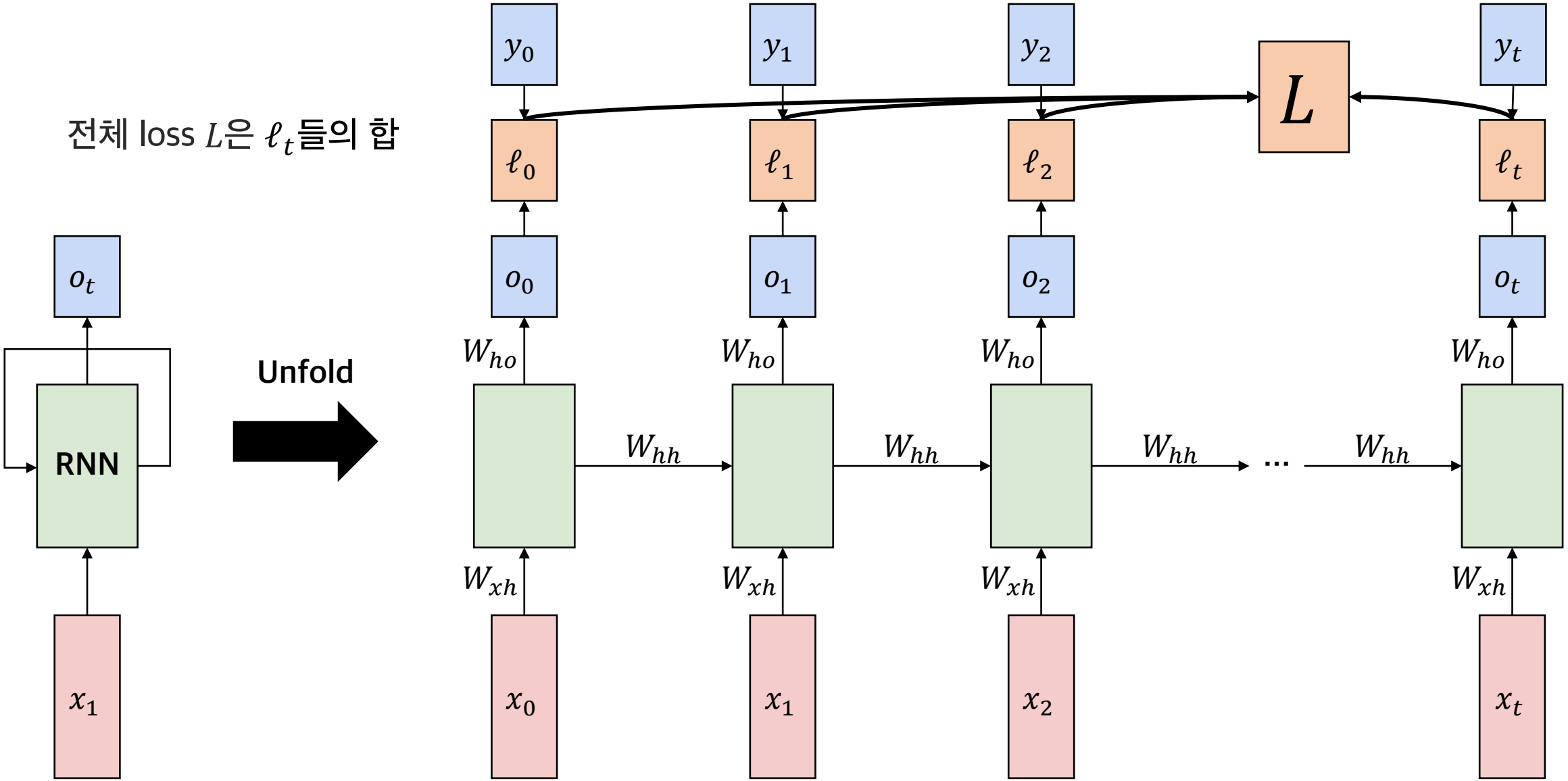
4.3 RNNs : Computational Graph Across Time

loss는 시점 마다 계산



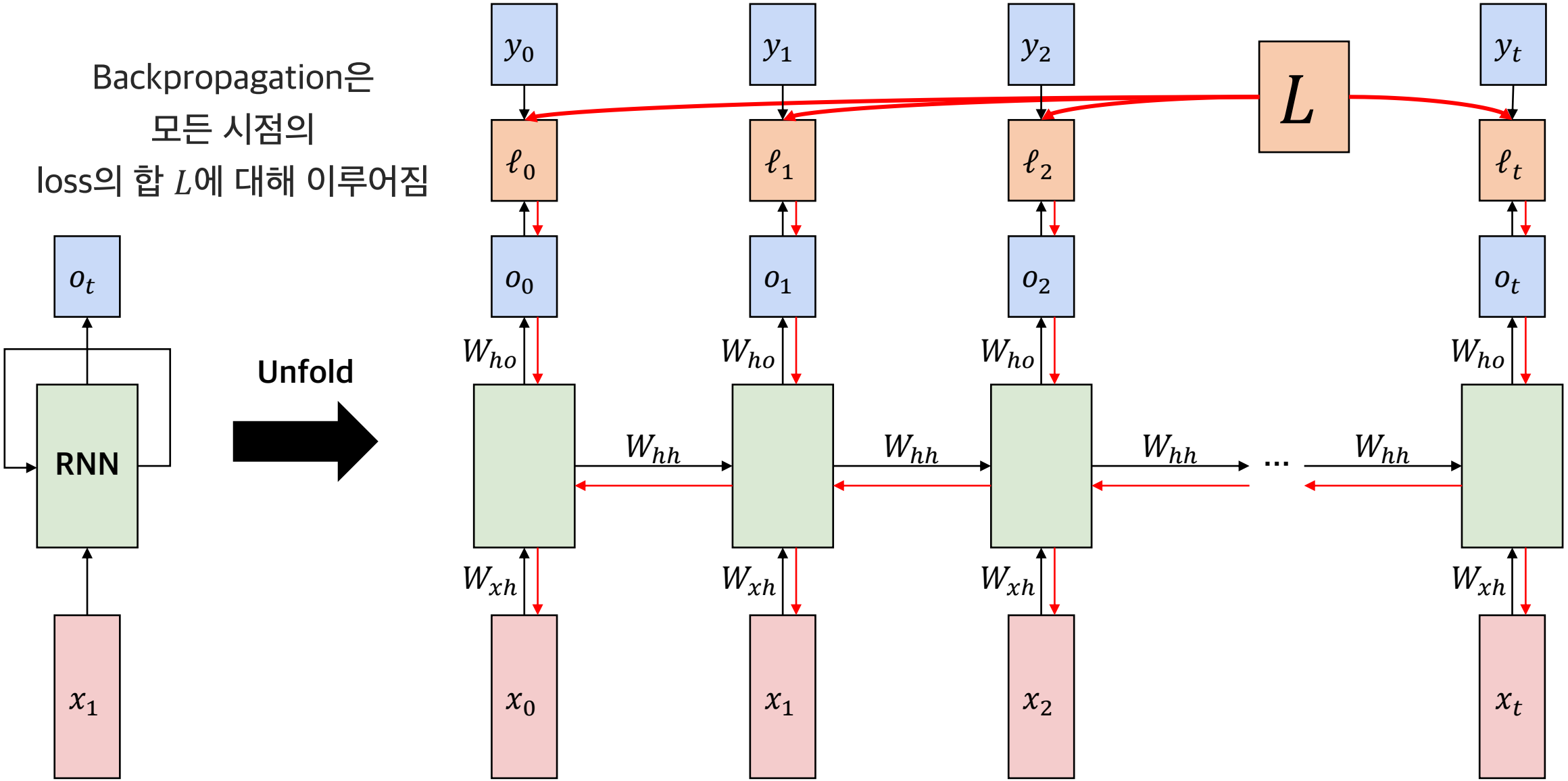
4.3.1 RNNs : Computational Graph Across Time

전체 loss L은 ℓ_t들의 합



4.3.1 RNNs : Computational Graph Across Time

Backpropagation은
모든 시점의
loss의 합 L에 대해 이루어짐



4.3.2 Backpropagation Through Time(BPTT)

Backpropagation through time을 계산 하기 위하여 다음과 같은 RNN 모델을 정의한다.

$$h_t = f\left(X_t \cdot W_{xh} + h_{t-1} \cdot W_{hh}\right) \ \ (f \text{ is activation function})$$
$$o_t = h_t \cdot W_{yh}$$
$$\hat{y}_t = \text{softmax}\left(o_t\right)$$

이때, o_t는 출력 logit이고$\hat{y}_t$는 여기에 SoftMax를 취한 값이다.
C개의 Class를 갖는 출력에 대한 cross entropy를 이용할 때
시점 t에서의 정답 y_t에 대한 loss ℓ_t를 이용하여 길이 T의 sequence에 대한 총 loss function 은 다음과 같이 주어진다.

$$L(\hat{y}, y) = \sum_{t=1}^{T} \ell_t\left(\hat{y}_t, y_t\right)$$
$$= -\sum_{t=1}^{T}\sum_{j=1}^{C} y_t^{(j)} \log \hat{y}_t^{(j)}$$

4.3.2 Backpropagation Through Time(BPTT)

Hidden state를 출력과 연결하는 weight와 관련된 gradient를, 즉 W_{ho}에 속하는 가중치 w_{ij}에 대하여 계산
w_{ij}는 hidden state의 i번째 unit을 j번째 출력 unit에 연결하는 weight이다.

$$\frac{\partial \ell_t}{\partial w_{ij}} = \frac{\partial \ell_t}{\partial \hat{y}_t^{(j)}} \frac{\partial \hat{y}_t^{(j)}}{\partial o_t^{(j)}} \frac{\partial o_t^{(j)}}{\partial w_{ij}}$$

$$\frac{\partial o_t^{(j)}}{\partial w_{ij}} = h_t^{(i)} \quad (\because o_t = h_t \cdot W_{yh})$$

w_{ij}에 대한 loss function L에 관한 식을 계산 하기 위해서는 각 시점에서의 gradient를 합산 해야한다.
그러므로 $\frac{\partial L}{\partial w_{ij}}$에 대한 식은 다음과 같아진다.

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \hat{y}_t^{(j)}} \frac{\partial \hat{y}_t^{(j)}}{\partial o_t^{(j)}} h_t^{(i)}$$

4.3.2 Backpropagation Through Time(BPTT)

$\frac{\partial L}{\partial w_{ij}}$에 대한 식에서 각 항들의 값들은 다음과 같이 구할 수 있다.

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \hat{y}_t^{(j)}} \frac{\partial \hat{y}_t^{(j)}}{\partial o_t^{(j)}} h_t^{(i)}$$

$$\frac{\partial \ell_t}{\partial \hat{y}_t^{(j)}} = -\frac{y_t^{(j)}}{\hat{y}_t^{(j)}} \quad \text{where } \ell_t = -\sum_{j=1}^{C} y_t^{(j)} \log \hat{y}_t^{(j)}$$

$$\frac{\partial \hat{y}_t^{(j)}}{\partial o_t^{(j)}} = \hat{y}_t^{(j)}(1 - \hat{y}_t^{(j)}) \quad \text{where } \hat{y}_t = \frac{\exp(o_t^{(j)})}{\sum_{k=1}^{C} \exp(o_t^{(k)})}$$

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial w_{ij}} = -\sum_{t=1}^{T} y_t^{(j)}(1 - \hat{y}_t^{(j)}) h_t^{(j)}$$

4.3.2 Backpropagation Through Time(BPTT)

$T-1$ 시점에서의 hidden state와 T 시점의 hidden state의 weight, 즉 W_{hh}에 속하는 weight u_{ki} 에 대해 계산

u_{ki}는 hidden state 내의 k번째 unit과 i번째 unit을 연결하는 weight

hidden state는 이전 시점의 값에 따라 변경되기 때문에 이를 이해하기 위해 시점 t에서 i번째 hidden state unit, 즉 $h_t^{(i)}$에 대해

$$h_t^{(i)} = f\left(\sum_{l=1}^{N} u_{li} h_{t-1}^{(l)} + \sum_{m=1}^{D} v_{mi} x_t^{(m)} + b_{hi}\right)$$

weight u_{ki}에 대한 t 시점에서의 loss function에 대한 gradient는 다음과 같다.

$$\frac{\partial \ell_t}{\partial u_{ki}} = \frac{\partial \ell_t}{\partial h_t^{(i)}} \frac{\partial h_t^{(i)}}{\partial u_{ki}}$$

4.3.2 Backpropagation Through Time(BPTT)

gradient를 계산하기 위해 $h_t^{(i)}$를 u_{ki}에 관하여 표현해야 하므로, 식을 다음과 같이 정리한다.

$$h_t^{(i)} = f\left(u_{ki}h_{t-1}^{(k)} + u_{ii}h_{t-1}^{(i)} + \sum_{\substack{l=1 \\ l\neq k,i}}^{N} u_{li}h_{t-1}^{(l)} + \sum_{m=1}^{D} v_{mi}x_t^{(m)}\right)$$
$$= f\left(u_{ki}h_{t-1}^{(k)} + u_{ii}h_{t-1}^{(i)} + c_t^{(i)}\right)$$

여기서의 $c_t^{(i)}$는 다음과 같다.

$$c_t^{(i)} = \sum_{l=1}^{N} u_{li}h_{t-1}^{(l)} + \sum_{m=1}^{D} v_{mi}x_t^{(m)}$$

$h_{t-1}^{(i)}$는 점화식에 의해 $f(u_{ki}h_{t-2}^{(k)} + u_{ii}h_{t-2}^{(i)} + c_{t-1}^{(i)})$로 표현 가능하므로 계산하려는 weight u_{ki}와 $h_{t-1}^{(i)}$ 에 관한 항으로 정리

4.3.2 Backpropagation Through Time(BPTT)

시점 t에서의 점화식은 첫번째 시점까지 계속 될 것이므로, t=t에서 부터 t=1까지의 모든 gradient의 합을 고려해야한다.

weight u_{ki}에 관한 $h_t^{(i)}$의 gradient가 점화식에 의해 정의 되므로, u_{ki}에 대한$h_t^{(i)}$편미분은 다음과 같이 정리할 수 있다.

$$h_t^{(i)} = f\left(u_{ki}h_{t-1}^{(k)} + u_{ii}h_{t-1}^{(i)} + c_t^{(i)}\right)$$

$$\frac{\partial h_t^{(i)}}{\partial u_{ki}} = \sum_{t'=1}^{t} \frac{\partial h_t^{(i)}}{\partial h_{t'}^{(i)}} \frac{\bar{\partial} h_{t'}^{(i)}}{\partial u_{ki}}$$

$\frac{\bar{\partial} h_{t'}^{(i)}}{\partial u_{ki}}$는 $h_{t'-1}^{(i)}$ 를 상수로 유지한 채 계산한 u_{ki}에 대한 $h_t^{(i)}$의 local gradient를 나타낸다.

$$\frac{\partial \ell_t}{\partial u_{ki}} = \sum_{t'=1}^{t} \frac{\partial \ell_t}{\partial h_t^{(i)}} \frac{\partial h_t^{(i)}}{\partial h_{t'}^{(i)}} \frac{\bar{\partial} h_{t'}^{(i)}}{\partial u_{ki}}$$

4.3.2 Backpropagation Through Time(BPTT)

이전 식에서 시점 t에서의 loss function에 관한 gradient로 일반화를 하기 위해서는 모든 시점에서의 gradient의 합으로 표현

$$\frac{\partial L}{\partial u_{ki}} = \sum_{t=1}^{T}\sum_{t'=1}^{t} \frac{\partial \ell_t}{\partial h_t^{(i)}} \frac{\partial h_t^{(i)}}{\partial h_{t'}^{(i)}} \frac{\bar{\partial} h_{t'}^{(i)}}{\partial u_{ki}}$$

$\frac{\partial h_t^{(i)}}{\partial \mathrm{h}_{t'}^{(i)}}$는 다음과 같이 나타낼 수 있다.

$$\frac{\partial h_t^{(i)}}{\partial h_{t'}^{(i)}} = \prod_{g=t'+1}^{t} \frac{\partial h_g^{(i)}}{\partial h_{g-1}^{(i)}}$$

이를 위 식에 대입하면 다음과 같은 수식을 얻을 수 있다.

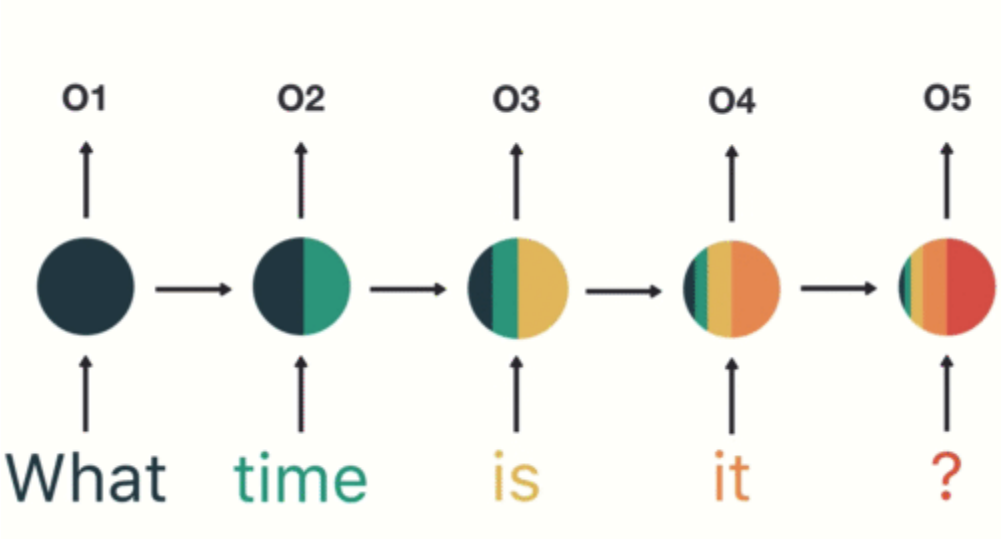
$$\frac{\partial L}{\partial u_{ki}} = \sum_{t=1}^{T}\sum_{t'=1}^{t} \frac{\partial \ell_t}{\partial h_t^{(i)}} \left(\prod_{g=t'+1}^{t} \frac{\partial h_g^{(i)}}{\partial h_{g-1}^{(i)}} \right) \frac{\bar{\partial} h_{t'}^{(i)}}{\partial u_{ki}}$$

W_{xh}에 속하는 weight에 대해서도 유사한 방법으로 계산 가능

image from : https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

4.3.3 Problem of Vanilla RNN

- Short term memory 문제



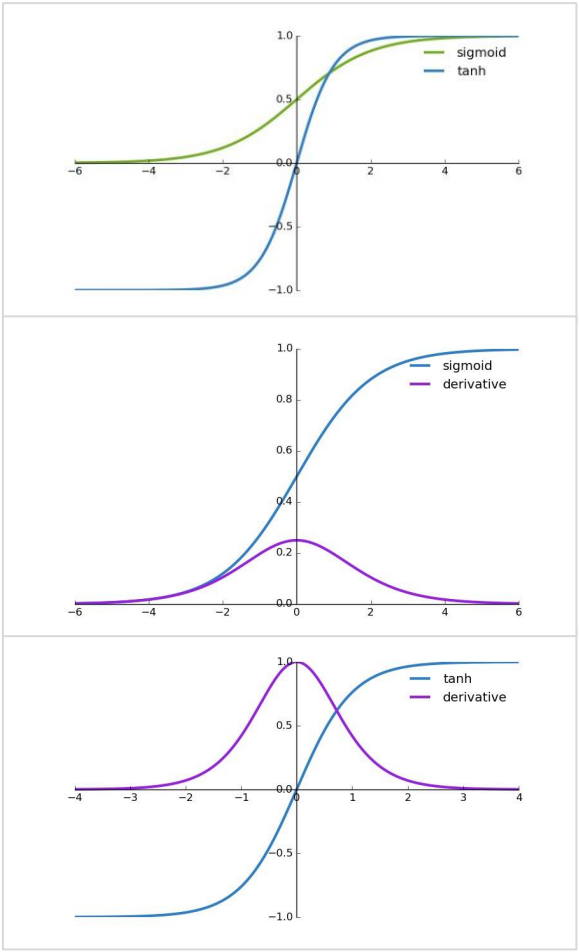
Vanilla RNN은 단기 메모리를 갖는다

http://cs231n.stanford.edu/slides/2021/lecture_10.pdf
image from : https://medium.com/@omkar.nallagoni

4.3.3 Problem of Vanilla RNN

- Non-linearity functions에 따른 문제
  - sigmoid보다 derivative의 폭이 넓은 $tanh$를 사용
  - 그럼에도 $tanh$의 값이 거의 항상 1보다 작음 → Vanishing gradient(기울기 소실)

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T}\tanh'\left(W_{hh}h_{t-1}+W_{xh}x_t\right)\right)W_{hh}^{T-1}\frac{\partial h_1}{\partial W}$$



- Non-linearity를 제거한다면,

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T}\frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}W_{hh}^{T-1}\frac{\partial h_1}{\partial W}$$

값이 1보다 크다면
→ Exploding gradient(기울기 폭발)

값이 1보다 크다면
→ Vanishing gradient(기울기 소실)

image from : http://dprogrammer.org/rnn-lstm-gru

4.3.4 Solution to The Problem of Vanilla RNN

- Gate를 추가하여 선택적으로 정보를 기억할 수 있도록 개선
  - Long Short Term Memory (LSTM) [Hochreiter, 1997], [Gers, 2000]
  - Gated Recurrent Unit (GRU) [Cho, 2014]