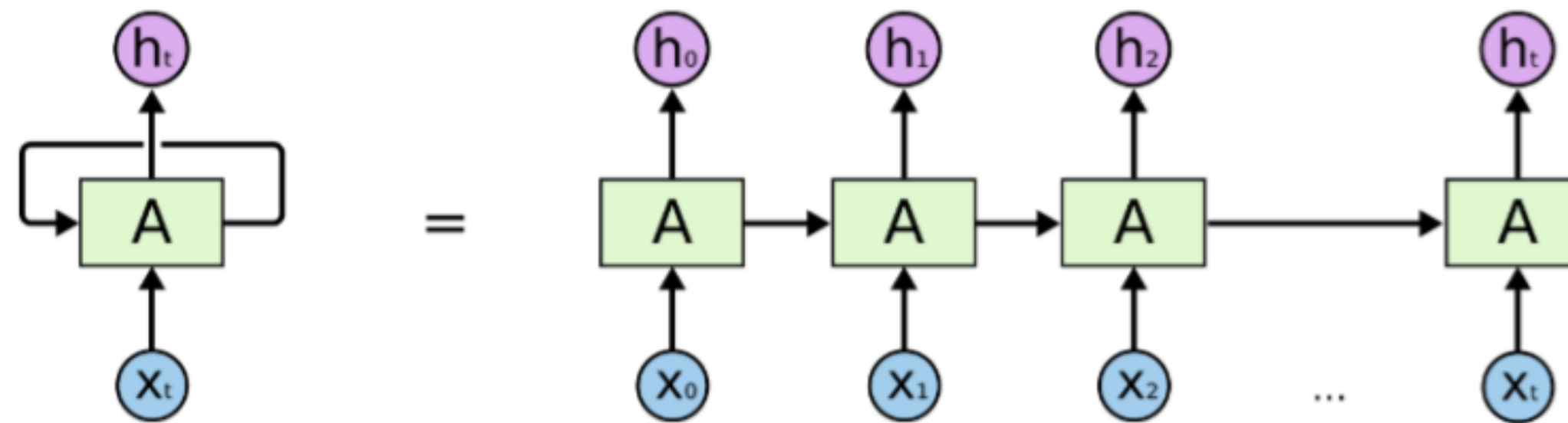


자연어처리와 딥러닝

Table of Contents

- I RNN
- II Change-level Language Model
- III BackPropagation through Time & Long-Term-Dependency
- IV LSTM
- V QUIZE

I RNN Structure



An unrolled recurrent neural network.

시간적 순서가 있는 데이터를 처리 할 때 사용하는 대표적인 모델

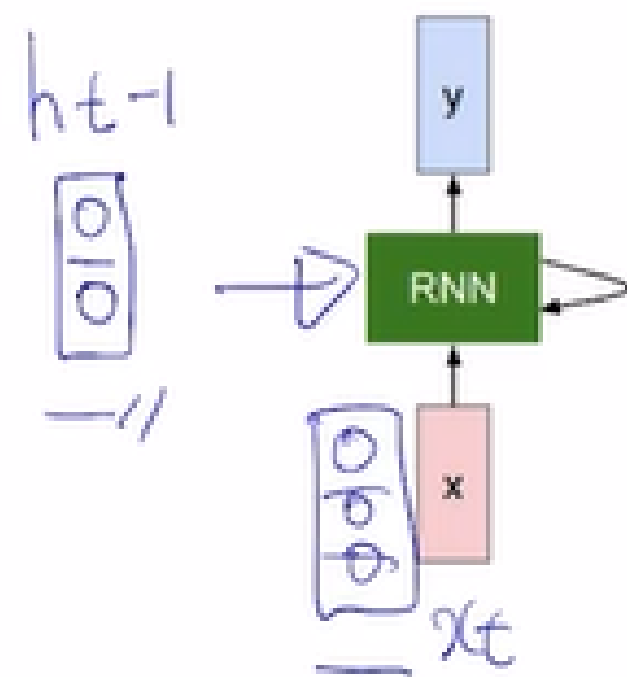
RNN의 구조는 위의 사진처럼 x_0 으로 만들어진 h_0 의 output이 x_1 의 학습에 사용되는 구조를 가짐
여기서 중요한 것은 다음 각 Time Step에서의 Weight parameter A 는 공유된다는 점이다.

I RNN의 학습 과정

Recurrent Neural Network

Vanilla RNN

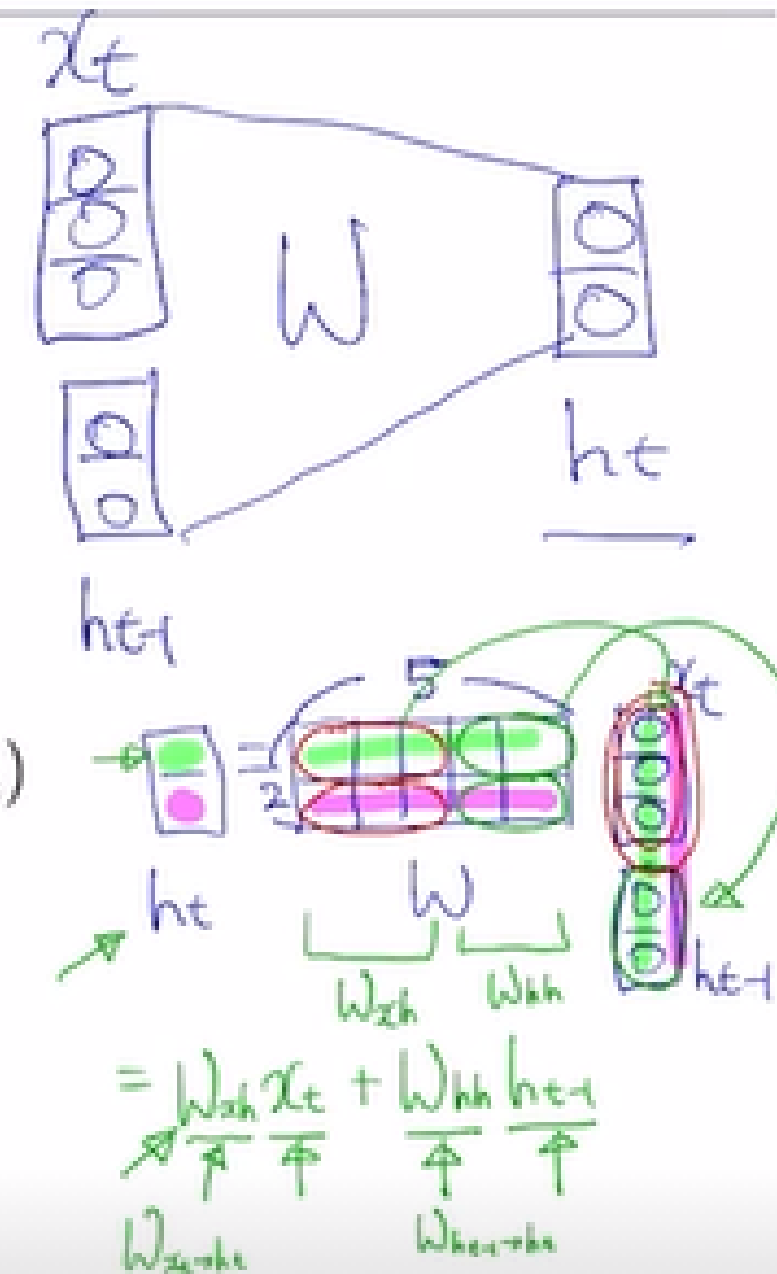
- How to calculate the hidden state of RNNs
 - The state consists of a single "hidden" vector \mathbf{h}



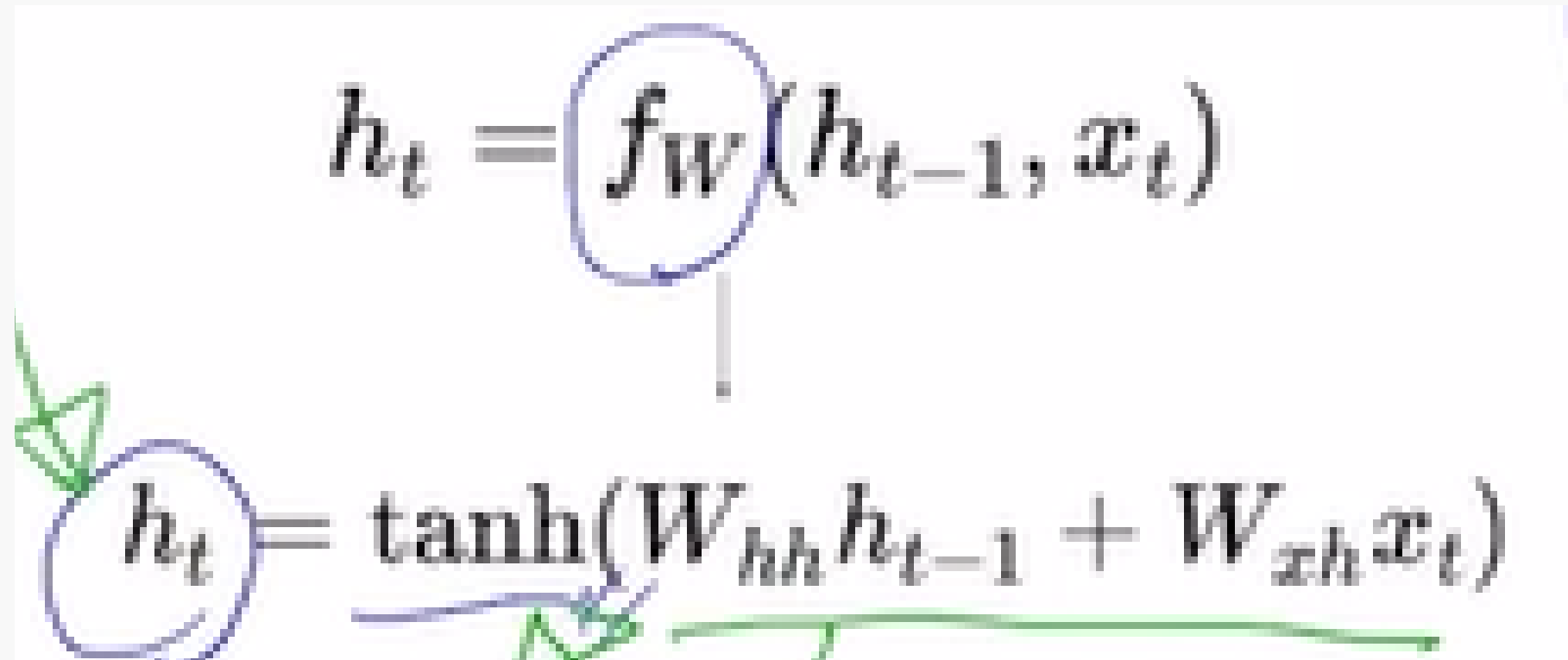
$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$



I RNN의 학습 과정



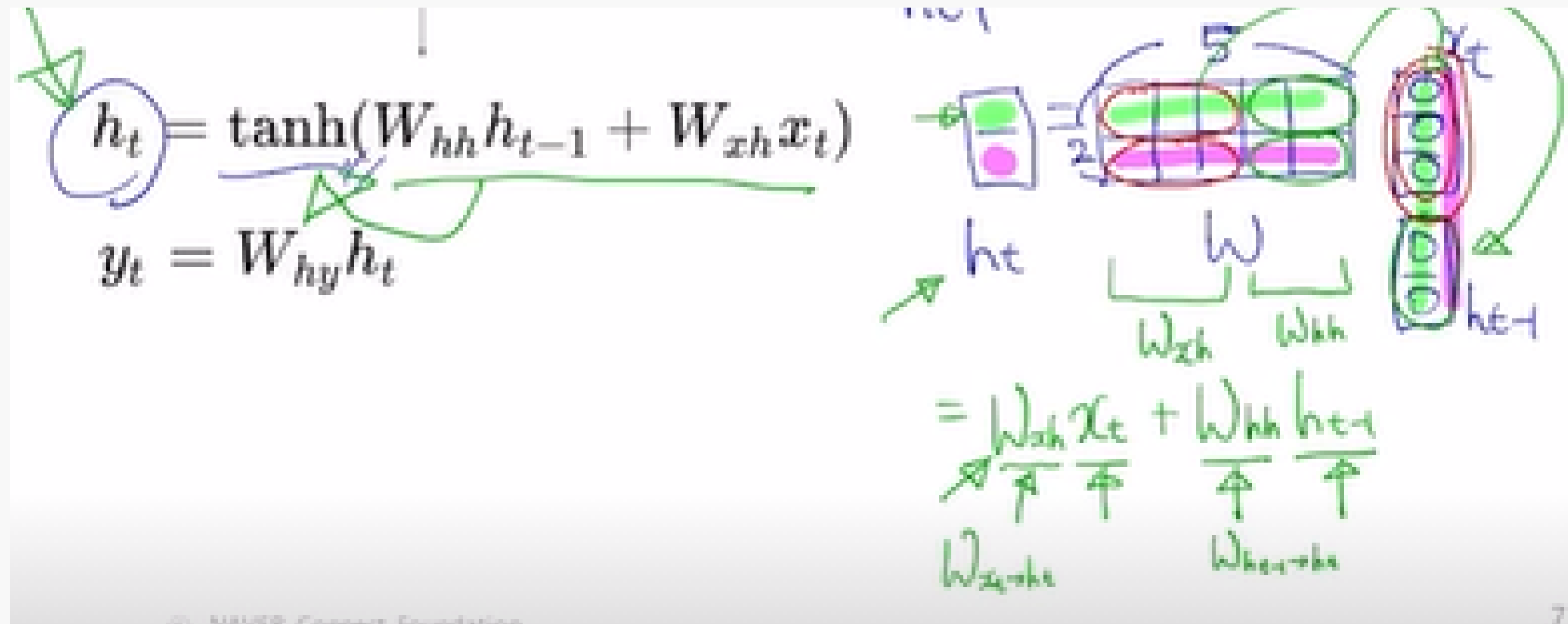
The image shows two handwritten equations. The top equation is $h_t = f_W(h_{t-1}, x_t)$, where the function f_W is circled in blue. A vertical line points from this circle down to the bottom equation. The bottom equation is $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$. In this equation, the h_t on the left is circled in blue. A green arrow points from the left towards this circle. The entire right-hand side of the equation, $\tanh(W_{hh}h_{t-1} + W_{xh}x_t)$, is underlined in green. Additionally, a green squiggly line is drawn under the h_{t-1} term within the parentheses.

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

현재 time의 hidden layer를 알기 위해

우리는 이전 hidden layer인 h_{t-1} 과 현재 time의 input인 x 를 활성화함수 $f_W(\tanh)$ 에 넣는다

I RNN의 학습 과정

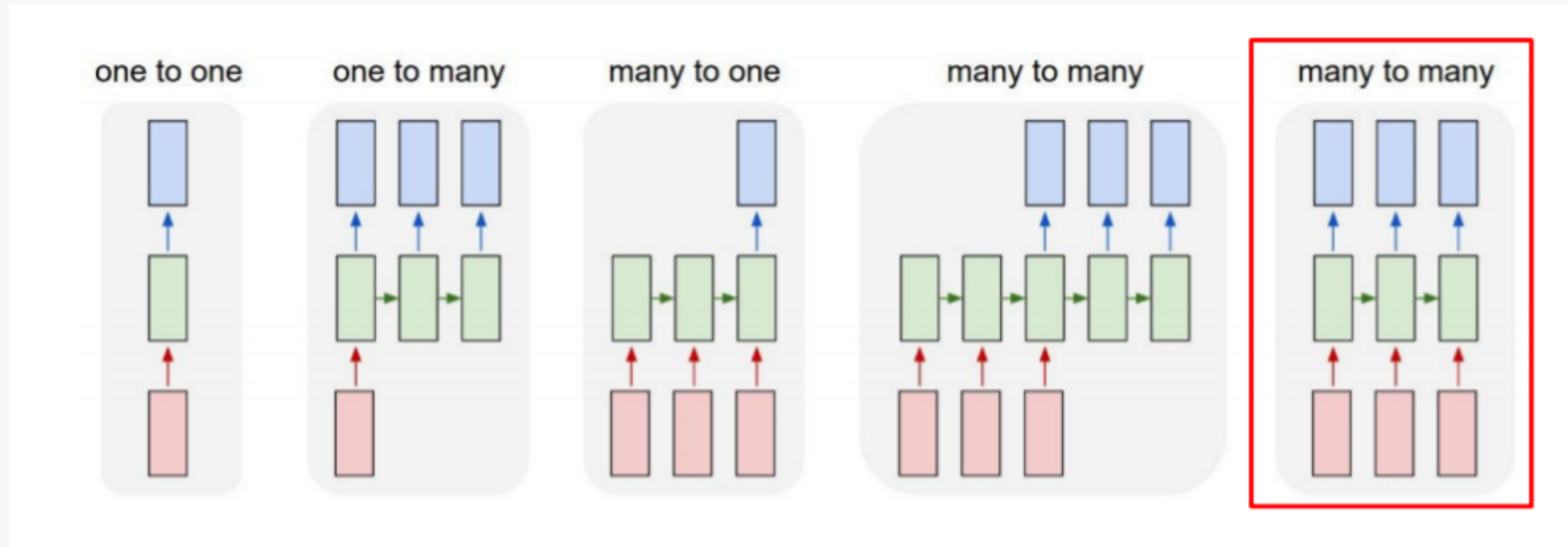


이 식을 학습하기 위해서 W 가 추가 되게 되는데 그 식은 앞 그림의 오른쪽에 쓰여진 식과 같다.

그림의 오른쪽 식과 같이 x_t 와 h_{t-1} 을 concat한 vector가 w 와 곱해지면서
현재 time의 h 가 구해지게 된다. 여기서 W_{xh} 는 x_t 를 h_t 로 만드는 부분이고
 W_{hh} 는 h_{t-1} 를 h_t 로 만드는 파라미터가 된다.

결국 RNN은 이 W 를 공유하면서 각 Time step의 결과값인 h_t 를 구하는 과정이 된다.

I RNN 종류

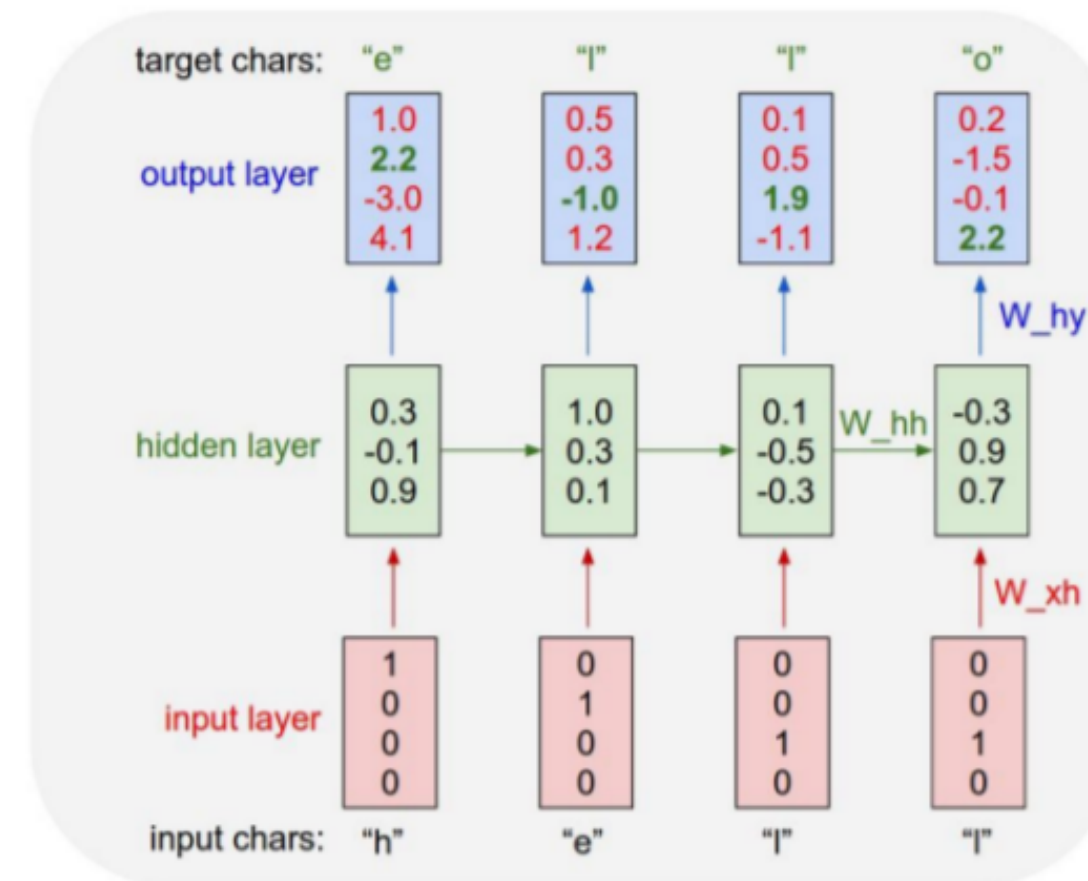


RNN의 종류는 위 사진과 같이
각 time step의 input의 갯수와 output의 갯수에 따라 나뉜다.

II Change-level Language Model

- Example of training sequence "hello"

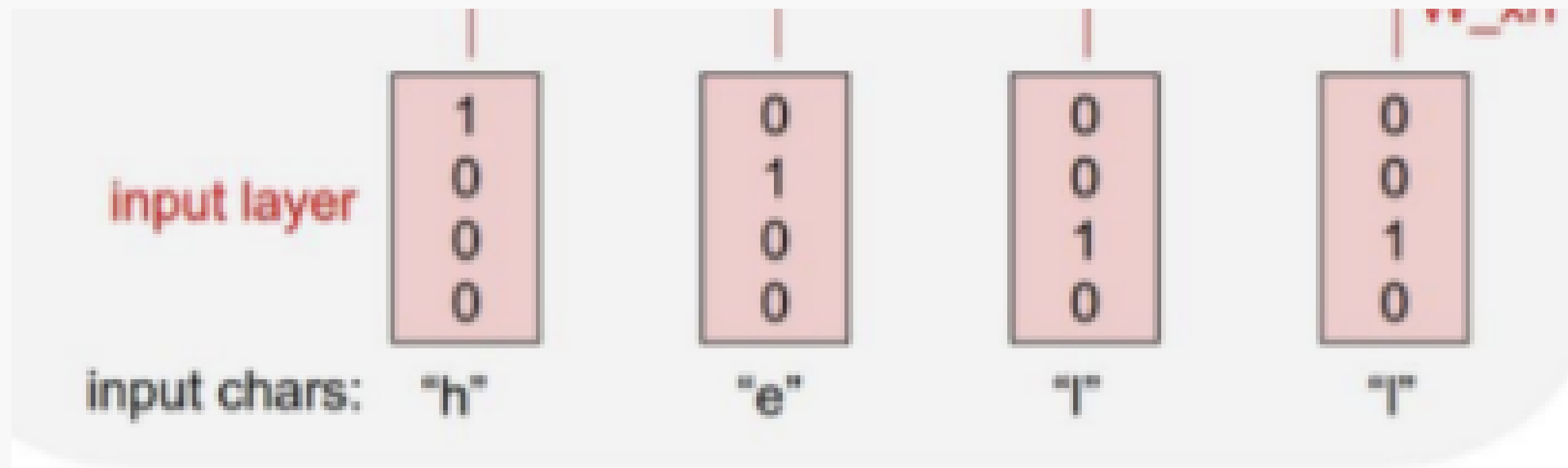
- $\text{Logit} = W_{hy}h_t + b$



위의 그림은 "hello"라는 sequence data를 주고, 정답을 찾을 수 있는 지 확인하는 예시

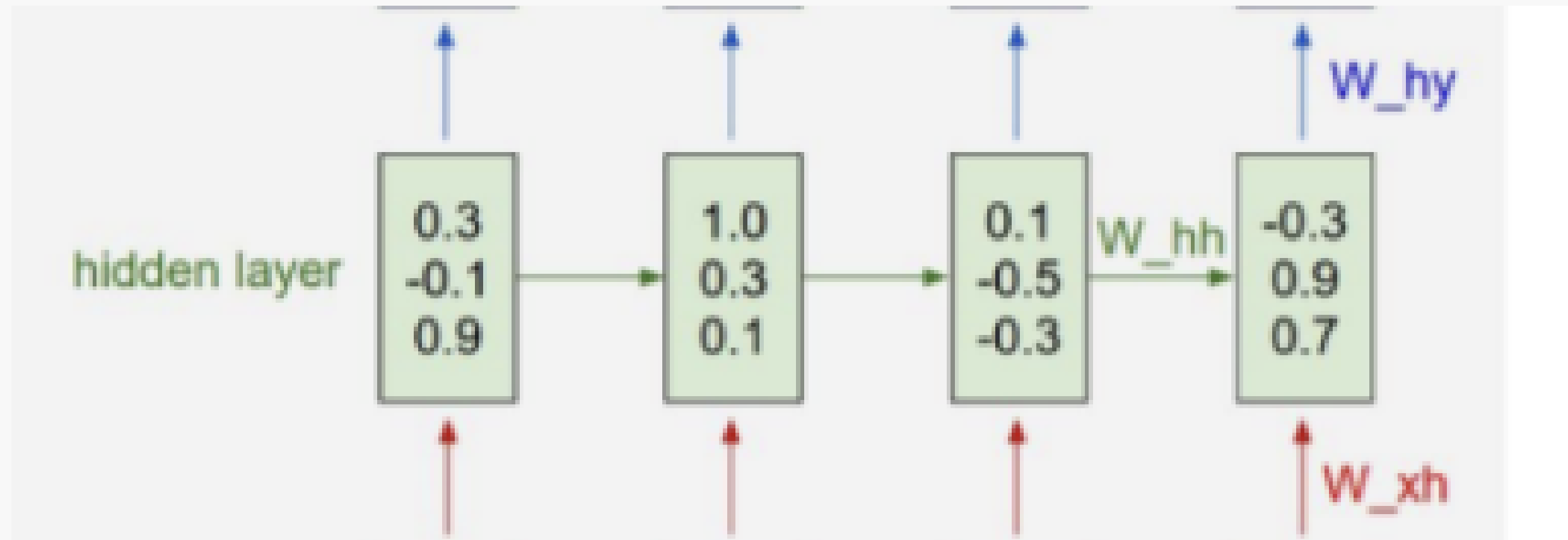
앞에서 정리한 것과 같이 현재 단계의 H를 예측하기 위해 H_{t-1} 와 X_t 를 W 와 연산한 후, 활성화 함수를 거쳐야 한다

II Change-level Language Model



앞 예시에서 사용한 알파벳은 4개이므로, 4개를 각각 원-핫 인코딩을 통해 벡터화해서 **Vocabulary**에 저장
이때, **input**으로는 "h"를 넣었을 때에는 "e"가 출력되고, "e"를 입력으로 넣었을 때에는 "l"이 출력되도록 설정
이와 같은 방법으로 학습하게 되는데 "h"를 넣었을 때 "e"값이 출력되는 확률이 높아지도록 학습을 진행

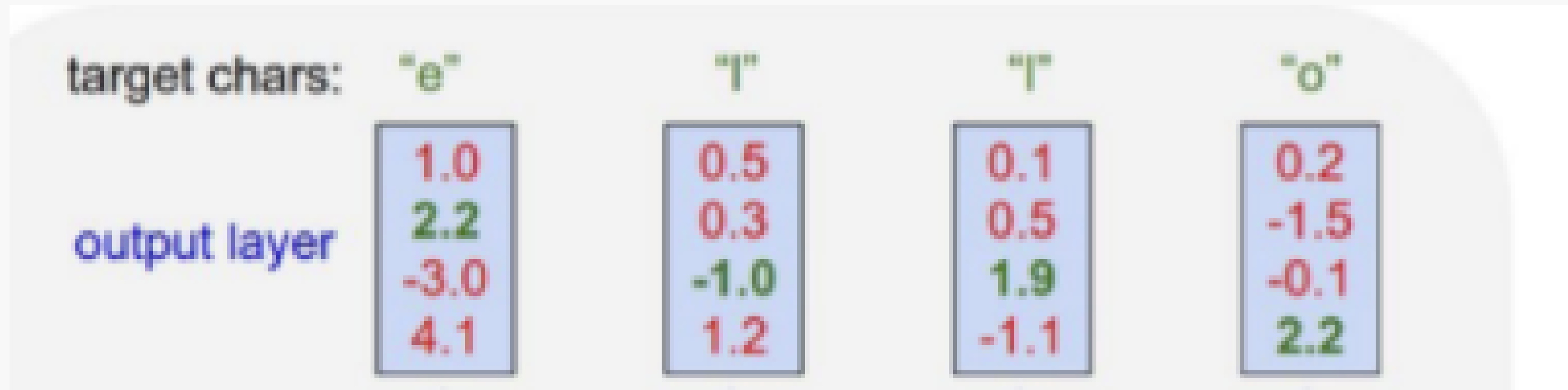
II Change-level Language Model



이때, input으로는 "h"를 넣었을 때에는 "e"가 출력되고, "e"를 입력으로 넣었을 때에는 "l"이 출력되도록 설정
이와 같은 방법으로 학습하게 되는데 "h"를 넣었을 때 "e"값이 출력되는 확률이 높아지도록 학습을 진행

여기서 눈여겨 볼 점은 첫번째 "l"은 다음 char로 l을 예측해야하고 두번째 "l"은다음의 char는 o를 예측해야한다는 점
이렇게 예측하기 위해 w_{hh} 에 이전 h,e,l 까지의 정보들을 담고, w_{xh} 에는 현재 input에 대한 정보를 담아서 활용

II Change-level Language Model

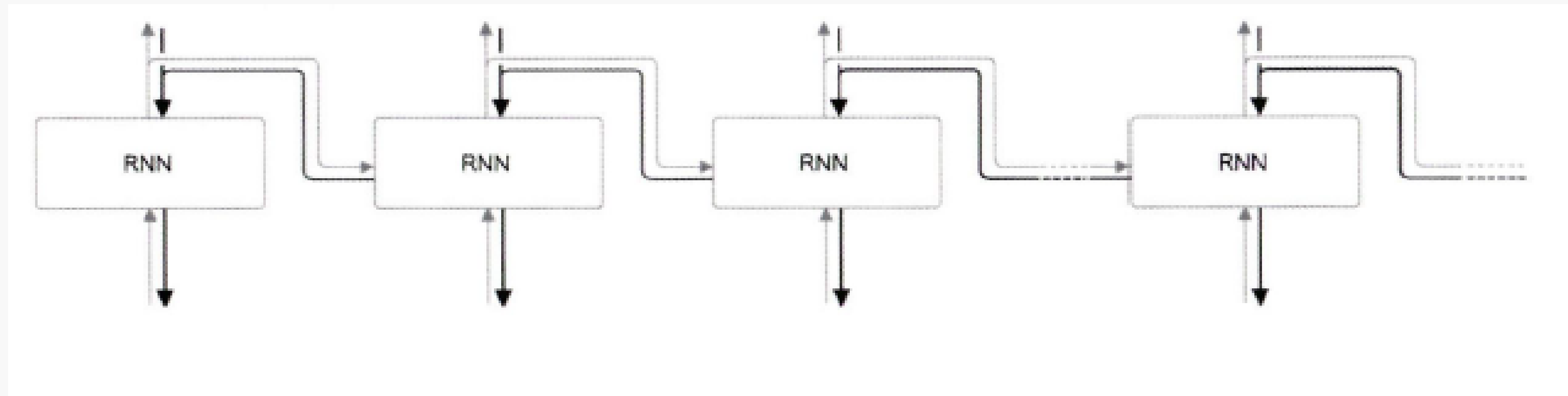


최종 결과 쪽을 보면 이상한 것을 알 수 있음

"e"는 $[0, 1, 0, 0]$ 이지만, output layer에서는 $[1, 2.2, -3, 4.1]$ 로 4.1이 가장 큰 값! (logit)
하지만, 가장 큰 값을 출력하는 것이 아닌 샘플링 (softmax layer)을 통해서 값을 출력

III BPTT

BPTBACKPROPAGATION THROUGH TIME의 줄임말로
RNN에서 타임스텝마다 계산된 WEIGHT를 BACKWARD PROPAGATION을 통해 학습하는 방식을 의미

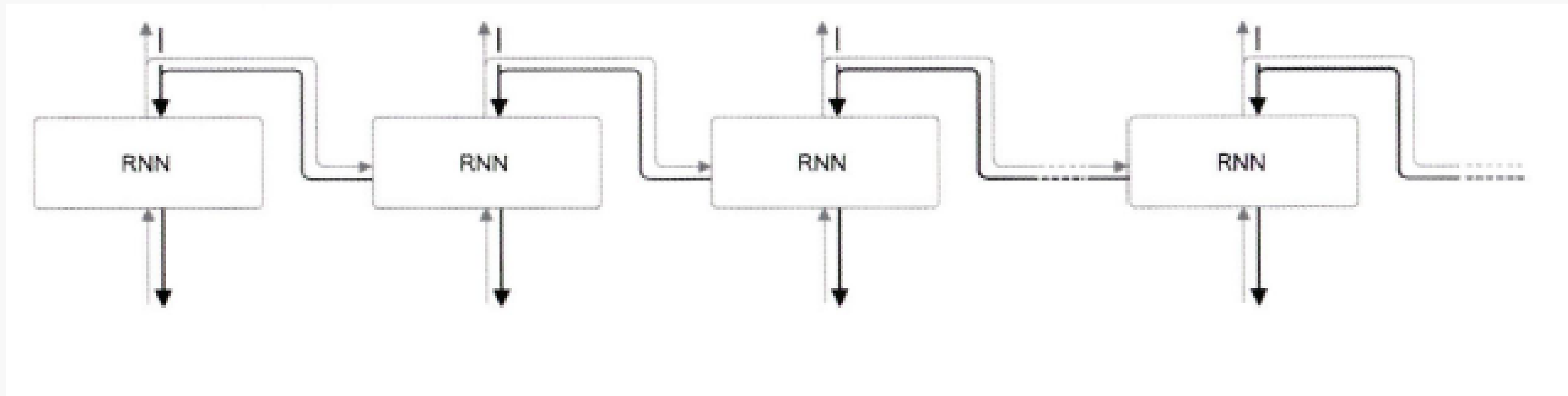


RNN은 일반적인 FULLY CONNECTED LAYER 모델과는 달리
각 파라미터들이 네트워크의 매 TIME STEP마다 공유된다.

즉, 각 TIME STEP의 출력에서의 GRADIENT는
현재 시간 스텝에서의 계산에만 의존하는 것이 아니라 이전 시간 스텝에도 의존한다.
그래서 원래의 BACKPROPAGATION과 비슷하지만
차이점은 매 TIME STEP마다 파라미터 W 에 관한 GRADIENT를 더해주는 것이다.

III BPTT 문제점

RNN은 LONG TERM DEPENDENCY를 잘 모델링 하지 못한다.



이전 모델과 같이 순전파를 수행하고, 이어서 역전파를 수행하여 원하는 기울기를 구할 수 있다. 여기서 사용하는 오차역전파 법이 BPTT이다. 하지만 긴 시계열 데이터 학습할 때 문제가 생긴다.

바로 역전파 과정에서 상당한 컴퓨팅 메모리를 사용한다는 것과, 너무 SENSITIVE한 GRADIENT를 얻는다는 것

III BPTT 문제점

Toy Example

- $h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1} + b), t = 1, 2, 3$
- For $w_{hh} = 3, w_{xh} = 2, b = 1$

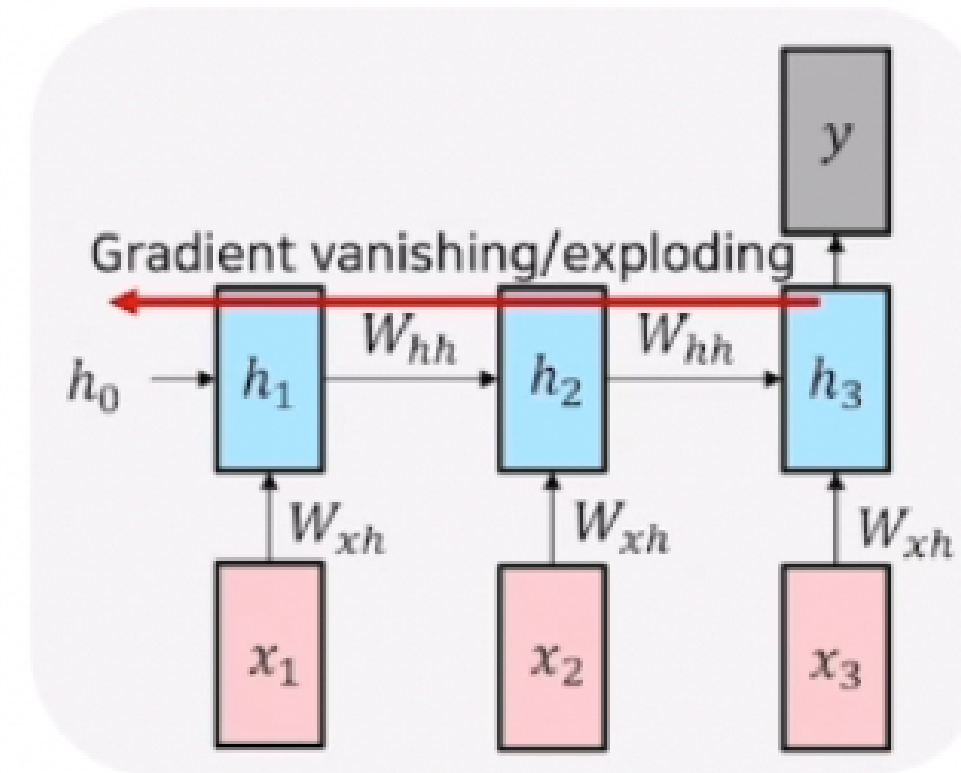
$$h_3 = \tanh(2x_3 + 3h_2 + 1)$$

$$h_2 = \tanh(2x_2 + 3h_1 + 1)$$

$$h_1 = \tanh(2x_1 + 3h_0 + 1)$$

...

$$h_3 = \tanh(2x_3 + 3 \tanh(2x_2 + 3h_1 + 1) + 1)$$



과거에 얻어진 정보를 다 취합하고 이를 미래에 고려해야 하는데, 먼 과거의 정보가 많이 반영되지 못함
순차 데이터 길이가 길어질수록 BPTT를 통한 미분 값이 너무 커지거나 작아질 수 있어 불안정해지기 쉬움

III BPTT 문제점

$$H_1 = \sigma(W_H H_0 + W_X X_1 + b)$$

$$H_2 = \sigma(W_H \sigma(W_H H_0 + W_X X_1 + b) + W_X X_2 + b)$$

TIME STEP에서의 HIDDEN STATE 관점에서 매 TIME STEP마다
동일한 파라미터를 마치 등비수열처럼 곱하게 되므로
과거의 TIME STEP에서의 정보가 HIDDEN STATE에 잘 반영되지 않거나 너무 크게 반영되는 현상 발생

그래서 BACKPROPAGATION을 통해 과거의 GRADIENT를 구할 때
이 GRADIENT의 크기가 소멸될 정도로 매우 작아지거나 폭발적으로 커지는 문제가 발생할 수 있음.
‣ 점점 HIDDEN STATE에서 VANISHING(SIGMOID, TANH) 또는 EXPLODING(RELU) 문제 발생
‣ LONG-TERM-DEPENDENCY 발생

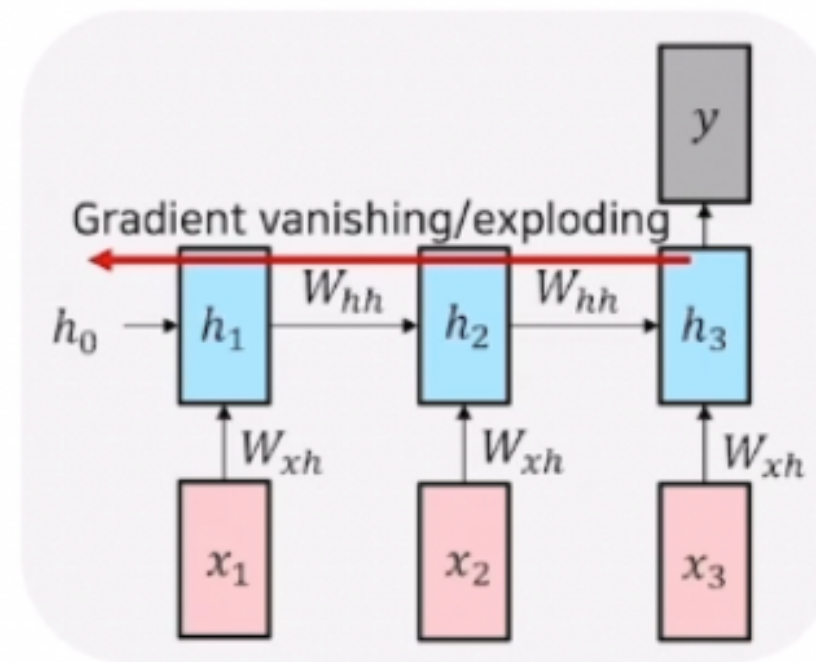
III BPTT 문제점

Toy Example

- $h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1} + b), t = 1, 2, 3$
- For $w_{hh} = 3, w_{xh} = 2, b = 1$

$$\begin{aligned} h_3 &= \tanh(2x_3 + 3h_2 + 1) \\ h_2 &= \tanh(2x_2 + 3h_1 + 1) \\ h_1 &= \tanh(2x_1 + 3h_0 + 1) \\ &\dots \end{aligned}$$

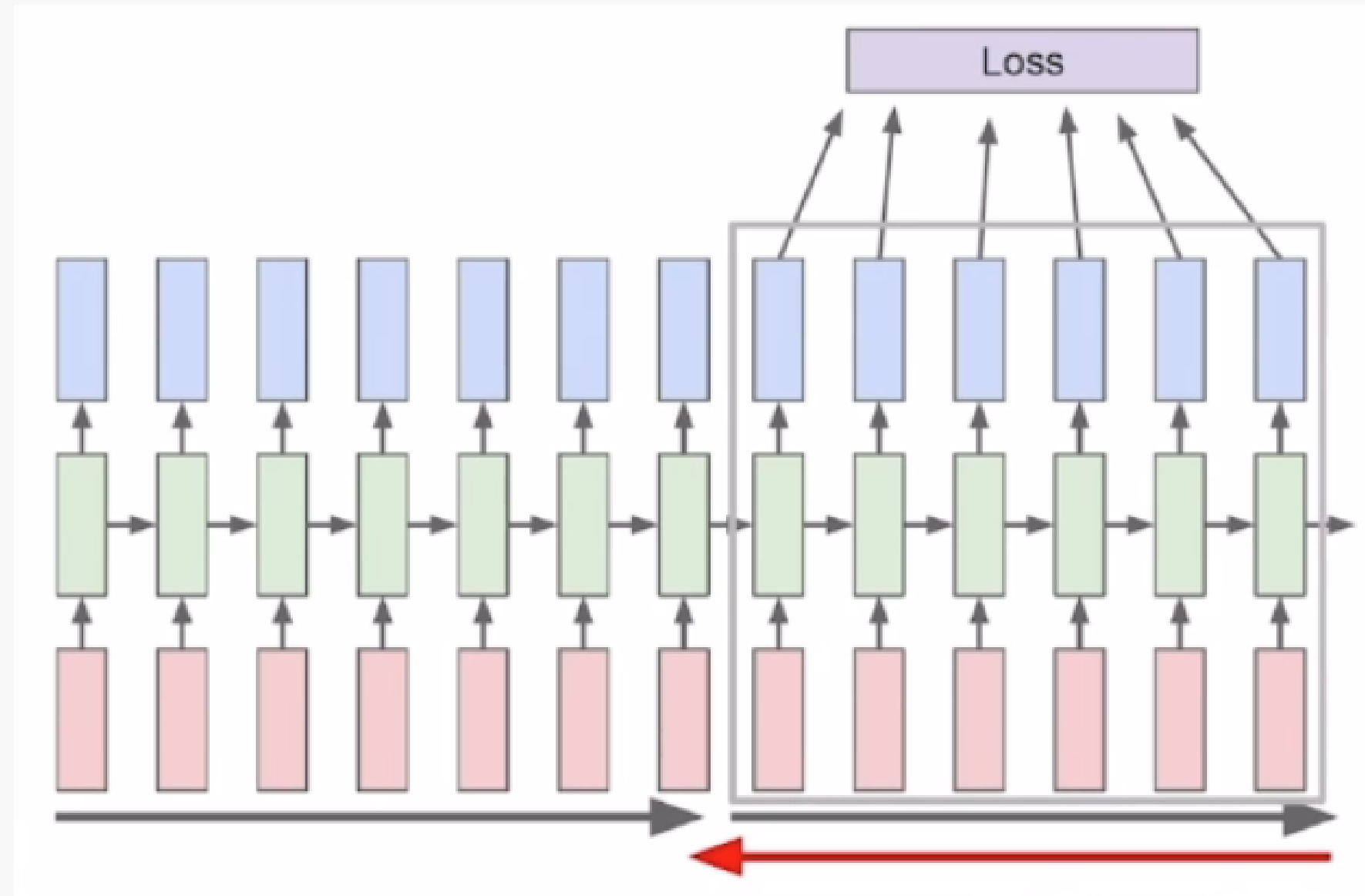
$$h_3 = \tanh(2x_3 + 3 \tanh(2x_2 + 3 \tanh(2x_1 + 1) + 1) + 1)$$



예를 들어, RNN의 각 파라미터를 **LOSS**를 줄이는 방향으로 업데이트 할 때 첫 **TIME STEP**에서의 **HIDDEN STATE H1**의 **GRADIENT**를 계산해야 하는데, **OUTPUT Y**를 구하는 과정에서 **HIDDEN STATE H1**이 계속 RNN 모듈을 통과하며 파라미터 **W_HH**가 곱해지고
매번 활성화 함수를 적용하는 단계를 반복한다.

그러면 $\partial Y / \partial H1$ 를 구하는 과정에서 **SEQUENCE**의 길이가 길어질수록 활성화 함수를 미분한 값과 파라미터 **W_HH**를 **SEQUENCE**의 길이만큼 곱하면서(**CHAIN RULE**)이 편미분 값이 매우 작아지거나 커질 가능성이 있을 수 있다.

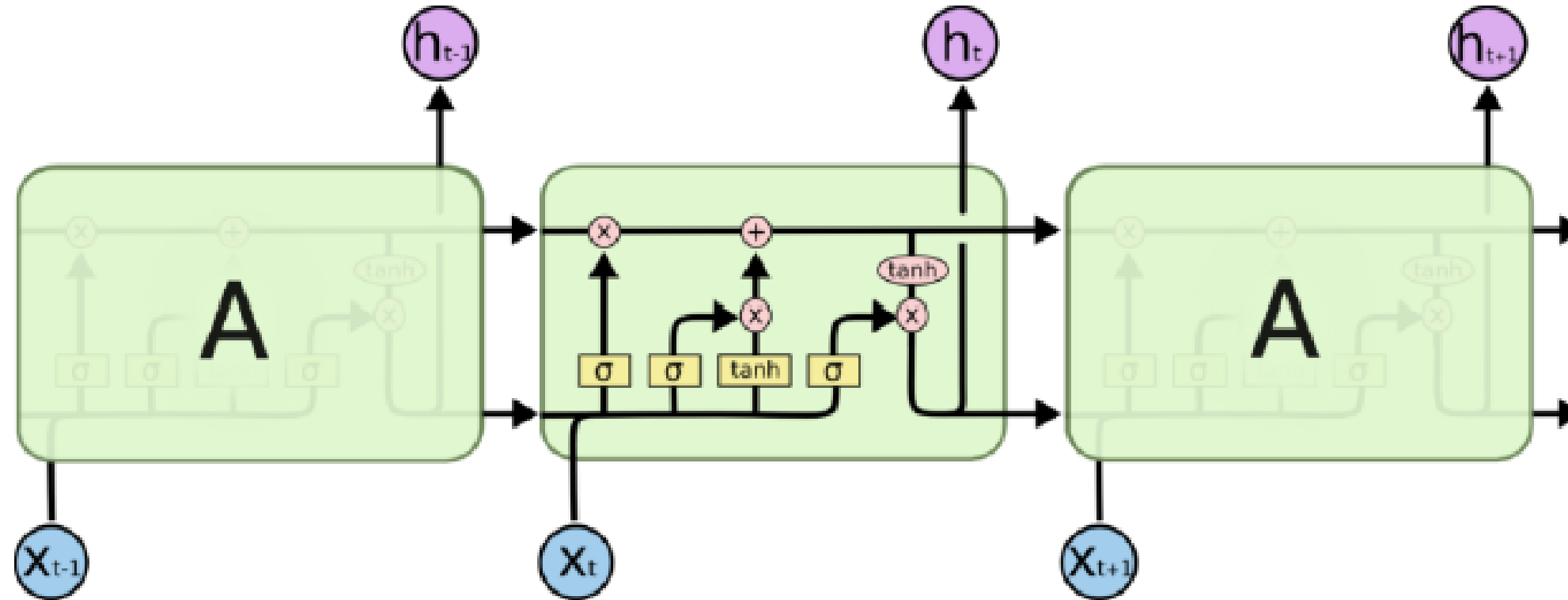
III Truncated BPTT



그런데 현실적으로 학습 데이터로 매우 긴 길이의 **SEQUENCE**가 입력으로 주어질 때, 각 **TIME STEP**의 **OUTPUT**을 계산하여 **GT**와의 차이를 줄이는 방향으로 학습하기에는 한정된 **GPU**와 메모리 자원으로 한꺼번에 처리하는 데 무리가 있다.

그래서 이를 **TRUNCATION**하여 제한된 길이의 **SEQUENCE**만으로 학습을 진행하는 방식을 택할 수 있다. 즉, 한번에 학습할 수 있는 **SEQUENCE**의 길이를 제한하여 **RNN**의 파라미터를 학습하는 것

IV LSTM



The repeating module in an LSTM contains four interacting layers.

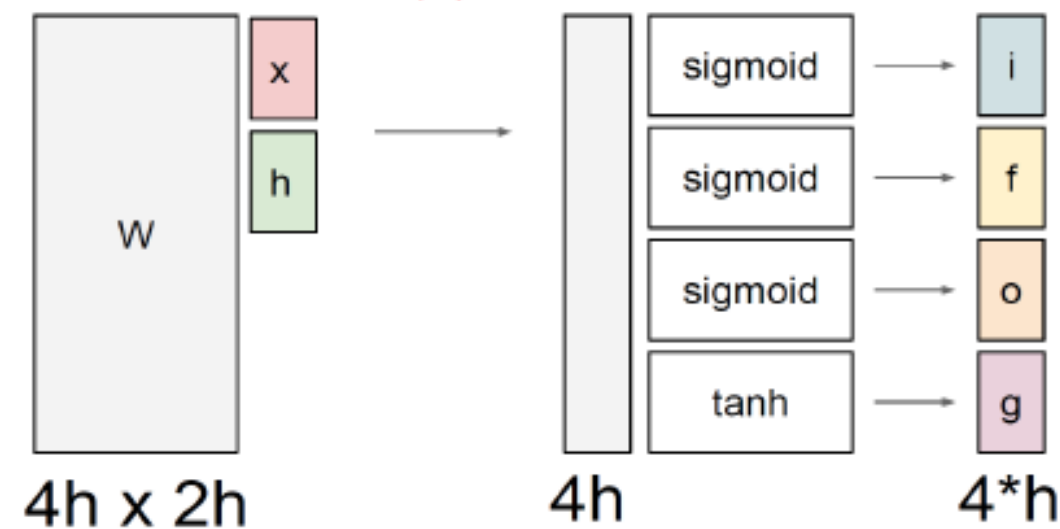
RNN에서 발생하는 Long term dependency를 해결할 수 있는 Model

Cell state에는 핵심 정보들을 모두 담아두고, 필요할 때마다 Hidden state를 가공해 time step에 필요한 정보만 노출하는 형태로 정보가 전파

이러한 방법으로 인해 타임 스텝이 멀어도 정보 전달이 잘 되기 때문에 학습이 잘 된다.

IV LSTM 구조

- Long short-term memory
 - i: Input gate, Whether to write to cell
 - f: Forget gate, Whether to erase cell
 - o: Output gate, How much to reveal cell
 - g: Gate gate, How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

Long short-term memory, Neural computation'97

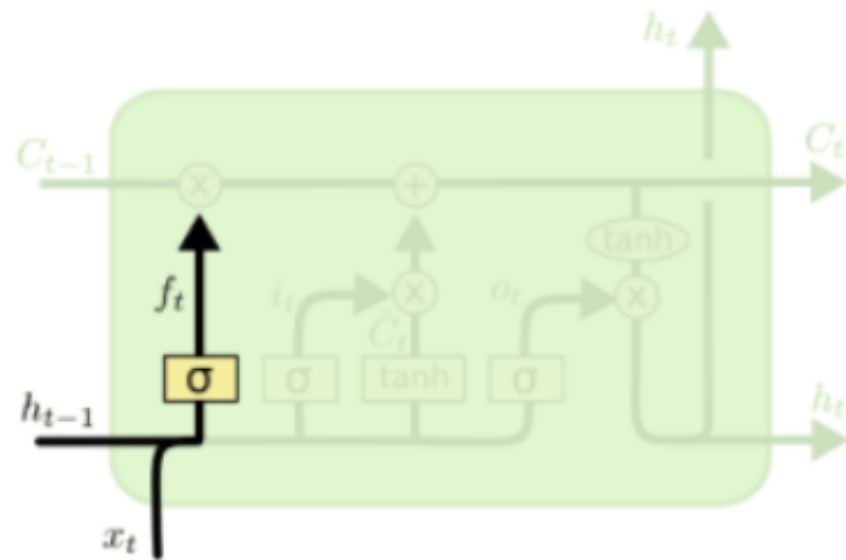
LSTM은 어떻게 RNN 구조에서 가지는 근본적인 문제인 Long term dependency를 해결 할 수 있을까?
그 정답은 위 사진에 보이는 4개의 Gate들이다

IV LSTM 구조 -Forget Gate

이전 정보들 중 어떤 것을 잊어야하고, 어떤 것을 기억해야 할지 판단 하는 **Gate**.

- Forget gate

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



현재 Time Step의 input인 x_t 와 이전 Time Step의 결과값인 h_{t-1} 을 Weight parameter W_f 와 연산한다. 이 후 활성화함수인 시그모이드를 거쳐 나오게된 결과 vector와 이전 timestep의 cell state c_{t-1} 과 element-wise로 곱해져서 이전 timestep에서 넘어온 정보를 얼마나 유지할지 결정한다. (f vector 값이 0.7이면 30 퍼센트만 유지한다는 뜻 \rightarrow sigmoid가 0에서 1 사이 값으로 만들어주기 때문)

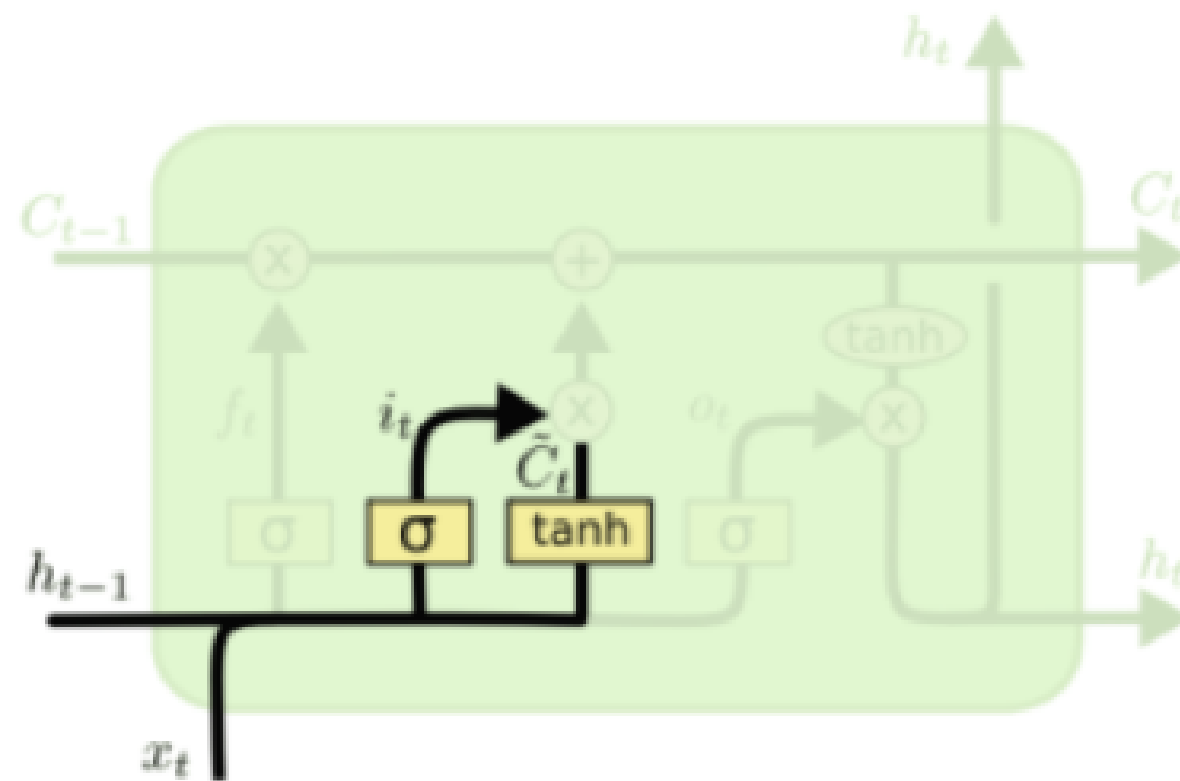
IV LSTM 구조 -Input gate & Gate gate

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

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

이전 HIDDEN STATE인 h_{t-1} 와 현재 TIMES STEP의 INPUT인 x_t 를 가지고 이들 중 어떤 정보를 현재 STATE로 저장할지를 정한다. 그래서 INPUT GATE가 위치한 영역에서는 새로운 CELL STATE를 만들기 위한 임시 상태인 $C \sim T$ 와 이를 얼마만큼의 비율로 반영할지를 결정하는 i_t 를 계산한다. 참고로 이 $C \sim T$ 를 만들 때도 TANH를 통해 입력과 파라미터를 가지고 만든 벡터의 각 원소를 얼마만큼 반영할지를 반영하게 된다.

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

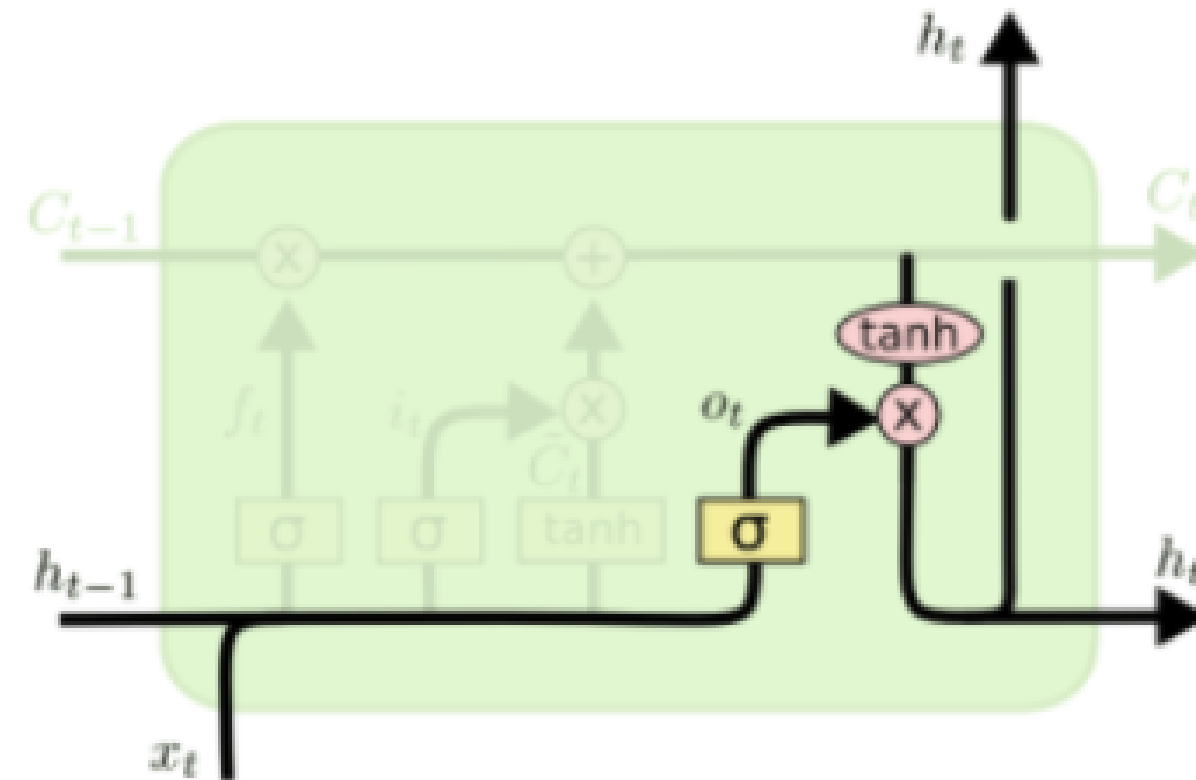


FORGET GATE에서 구한 반영 비율 결과와 INPUT GATE 영역에서 구한 반영 비율 결과와 임시 상태인 $C \sim T$ 를 가지고 현재 TIME STEP의 STATE를 반영한다.

IV LSTM 구조 -Output gate

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

$$h_t = o_t \cdot \tanh(C_t)$$



마찬가지로 OUTPUT GATE에서도 이전 HIDDEN STATE인 h_{t-1} 와 현재 TIME STEP의 INPUT인 x_t 를 가지고 SIGMOID를 통과시켜서 C_t 에서 얼마만큼의 비율을 반영할지를 계산한다.

이때, 현재 TIME STEP T에서의 HIDDEN STATE를 계산하기 위해 C_t 에 TANH를 적용한 결과를 앞서 구한 OUTPUT GATE에서 구한 비율과 곱한다.

OUTPUT GATE의 과정은 이제까지의 좀 더 완전하고 많은 정보를 갖고 있는 C_t 에서 일부 정보만을 필터링하여 HIDDEN STATE가 현재 TIME STEP에 직접적으로 필요한 정보만을 갖도록 하는 것으로 이해할 수 있다.

IV GRU

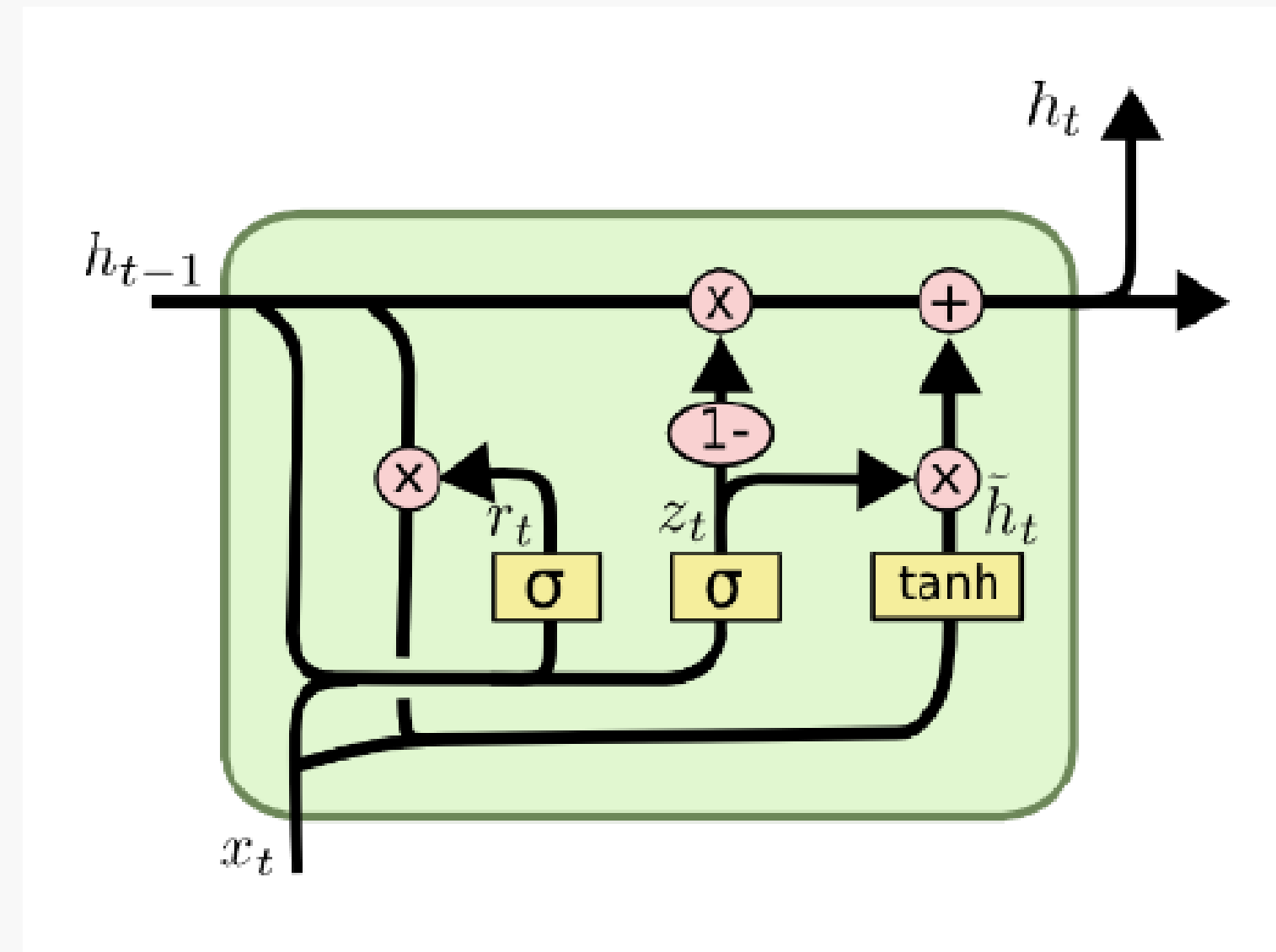
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

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

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

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

c.f) $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$
in LSTM



전체적인 동작은 LSTM과 비슷하지만, LSTM과의 가장 큰 차이점은
GRU에는 LSTM에서 입력 데이터로 사용되던 CELL STATE와 HIDDEN STATE 두 개가 아닌
이를 일원화한 HIDDEN STATE만 사용한다는 것이 특징이다.

즉, GRU에서의 HIDDEN STATE는 LSTM에서의 CELL STATE와 유사한 역할을 한다고 볼 수 있다.

IV LSTM vs GRU

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

$$\text{c.f) } C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

in LSTM

앞서 LSTM에서는 현재 TIME STEP의 CELL STATE인 CT를 구할 때,
이전 CELL STATE인 CT-1에 FORGET GATE를 통과한 결과를 곱하고,
임시 현재 CELL STATE인 C~T에 INPUT GATE를 통과한 결과를 곱해서 더하는 과정을 거친다.

GRU에서는 UPDATE GATE의 결과인 ZT만 가지고 임시 HIDDEN STATE인 H~T에 그대로 곱하고
이전 HIDDEN STATE인 HT-1에는 마치 FORGET GATE를 적용하는 것처럼 (1-ZT)를 곱해서 이를 더하는 식을 사용하는데,
이는 H~T와 HT-1의 가중 평균을 구하는 것으로 볼 수 있다.

LSTM에서는 INPUT GATE와 FORGET GATE의 독립적인 두 개의 GATE 결과를 가지고 CELL STATE를 업데이트했다면,
GRU에서는 하나의 GATE에서 HIDDEN STATE를 연산하는 것을 볼 수 있다.
이로 인해 구조적으로 GRU는 LSTM에 비해 경량화된 모델로 볼 수 있는 것이다.

✓ 3. RNN의 오차역전법(Backpropagation through time, BPTT)에 대한 설 1/1
명으로 옳지 않은 것은?

- ☐ 손실함수를 제한하기 위해 전체 시퀀스에 대해 앞으로 이동한 다음, 전체 시퀀스에 대해 다시 뒤로 기울기를 계산한다.
- ☐ 같은 가중치 행렬을 매 타임 스텝마다 곱하기 때문에 기울기 소실 또는 폭주 문제가 발생할 수 있다.
- ☐ 은닉 상태를 시간에 따라 계속 앞으로 나르지만, 일부의 스텝에 대해서만 오차역전이 이루어진다.
- ☒ 학습 과정에서 모든 타임 스텝의 정보를 안정적으로 반영할 수 있다. ✓

✕ 6. Character-level Language Model에 대한 설명으로 옳지 않은 보기를 골라주세요 0/1

- ☐ 언어 모델 중 하나로, 다음에 올 문자를 예측하는 태스크를 수행한다.
- ☐ Hidden state를 사용하여 이전 문자열의 정보를 기반으로 다음 문자를 예측한다.
- ☒ Output layer를 통해 원-핫 벡터 형태의 출력값이 나오게 된다. ✕
- ☐ LSTM과 같은 모델을 사용하여 Long-Term-Dependency 문제를 극복한다.

정답

- ☒ LSTM과 같은 모델을 사용하여 Long-Term-Dependency 문제를 극복한다.

V quiz

✕ 10. 아래 사진에 대해 틀리게 설명한 보기를 모두 골라주세요

0/1

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *  
    siginfo_t *info)  
{  
    int sig = next_signal(pending, mask);  
    if (sig) {  
        if (current->notifier) {  
            if (sigismember(current->notifier_mask, sig)) {  
                if (!current->notifier(current->notifier_data)) {  
                    clear_thread_flag(TIF_SIGPENDING);  
                    return 0;  
                }  
            }  
        }  
        collect_signal(sig, pending, info);  
    }  
    return sig;  
}
```

☐ 특정 hidden state 벡터의 dimension을 시각화한 결과이다.

☐ Vanila RNN으로 학습했을 때 흔히 볼 수 있다.

☒ true/false를 detection하는 태스크를 수행한 결과이다. ✓

☒ Character-level Language Model의 한 cell을 거쳤을 때는 볼 수 없는 결과이다. ✓

정답

☒ Vanila RNN으로 학습했을 때 흔히 볼 수 있다.

☒ true/false를 detection하는 태스크를 수행한 결과이다.

☒ Character-level Language Model의 한 cell을 거쳤을 때는 볼 수 없는 결과이다.

하지만, 기존의 VANILLA RNN으로는 위와 같이 학습될 수 X
그 이유는 GRADIENT가 전파되면서 소실되거나 증폭되면서
멀리까지 학습정보를 잘 전달하지 못하는 LONG-TERM-DEPENDENCY가 발생하기 때문

› RNN의 LONG-TERM-DEPENDENCY 문제를 보완한 LSTM 모델로 학습한 결과이기 때문

**Thank you
for listening!**