Recurrent Neural Network

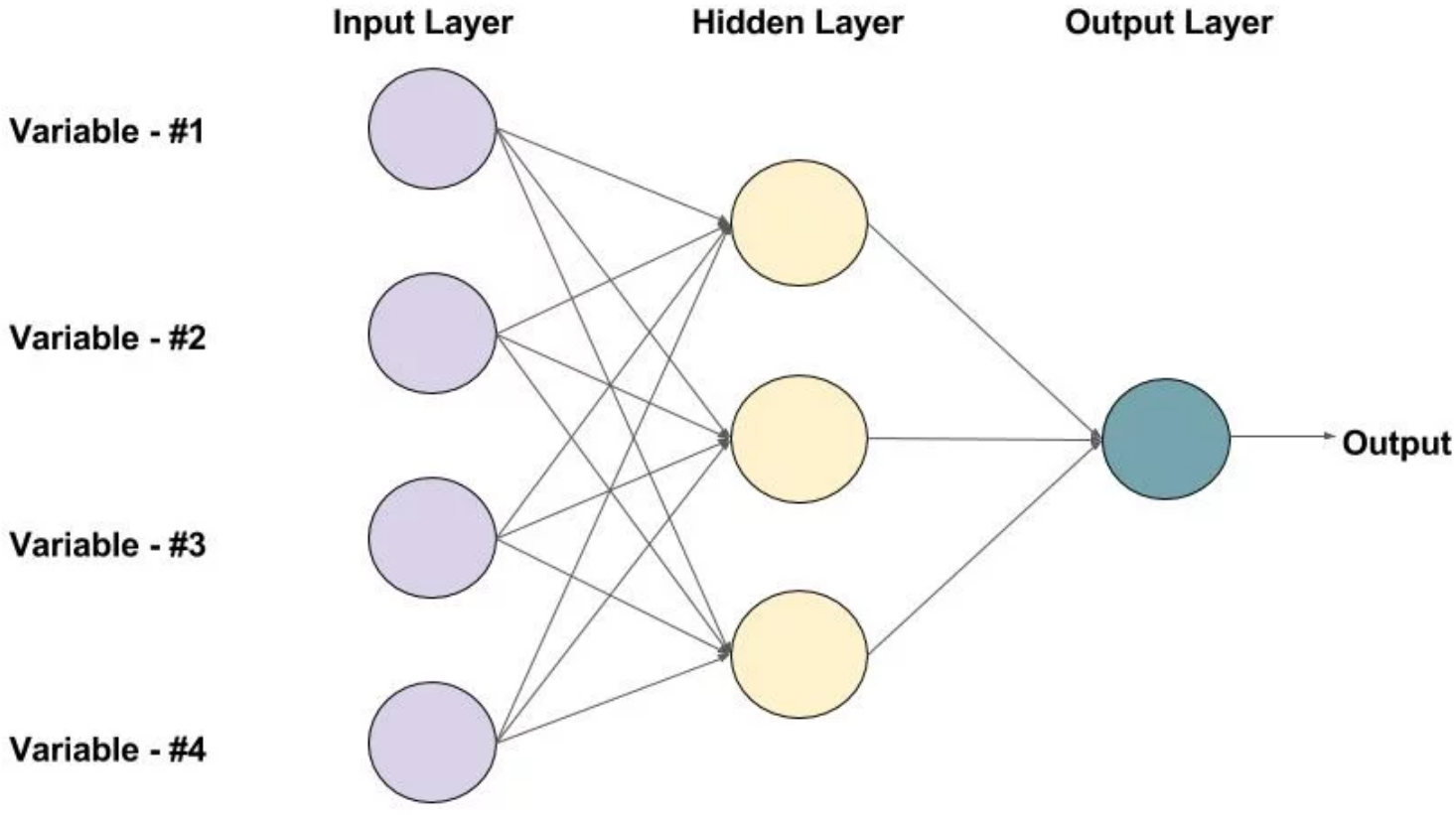
https://github.com/dlcjfgmlnasa/Tongmyong_College_Tensorflow_Tutorial

Created by **Choelhui lee**

Recurrent Neural Network (RNN)

- **Recurrent == 재귀적**
- 사전 의미 : **[명사] 원래의 자리로 되돌아가거나 되돌아옴.**

일반적인 Nerual Network



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

- **각각의 입력이 독립적이기 때문에 서로에게 영향을 안줌!!**

- **`그러나, 입력 데이터간 연관성이 있는 정보들을 무수히 많음`**
- *example*:
  - `Natural language` : 순차적으로 입력되는 정보간의 상관관계를 처리해야됨
  - `voice` : 순차적으로 입력되는 정보간의 상관관계를 처리해야됨

`RNN이 해결책!!`

RNN

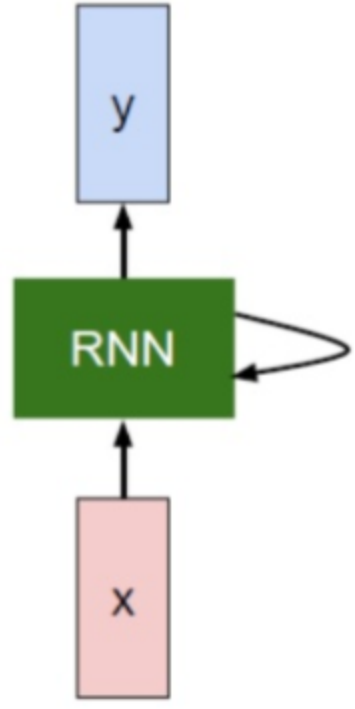
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

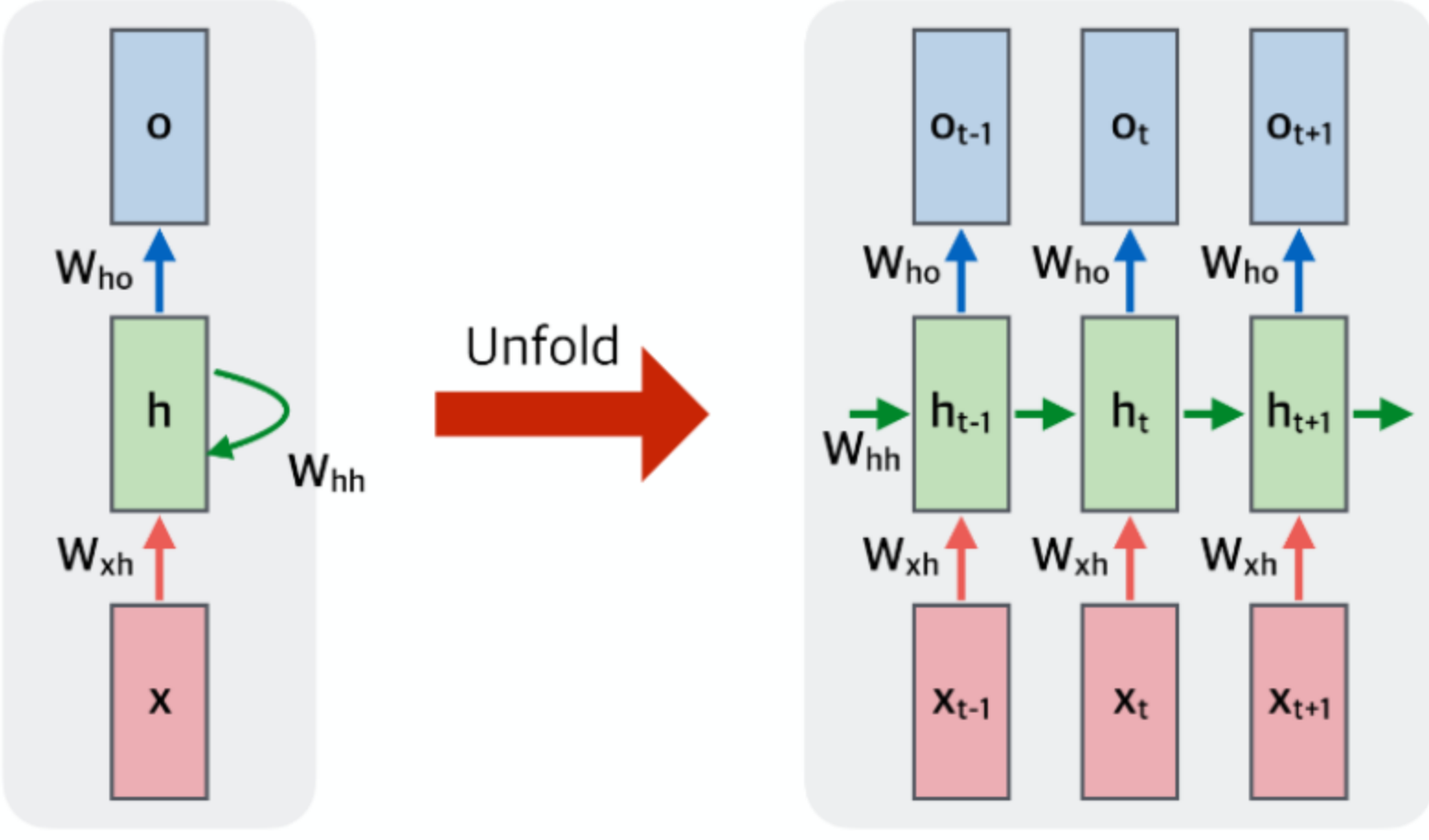
old state

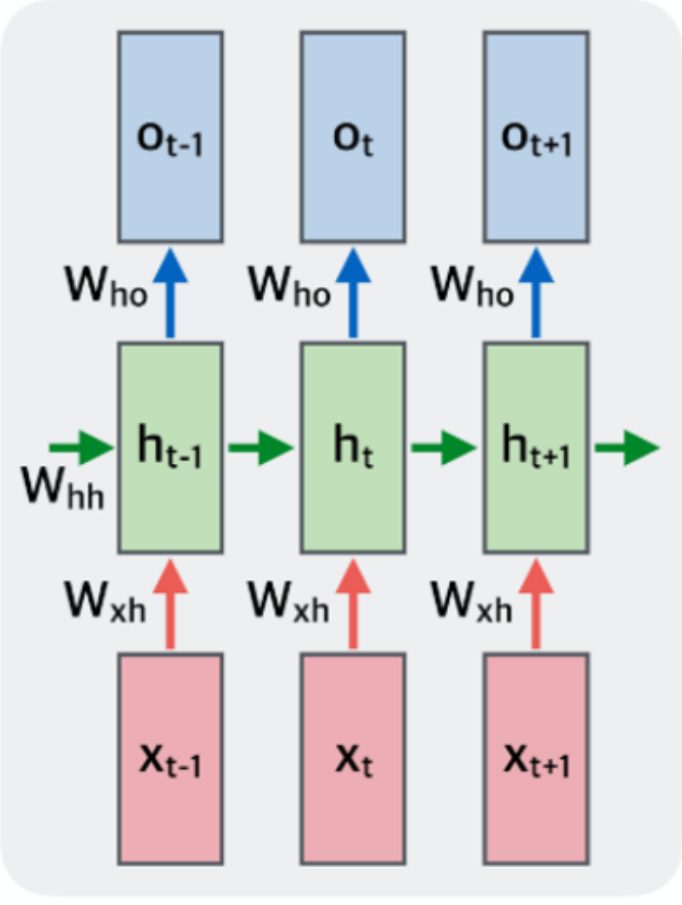
input vector at some time step



- **외부입력**과 **자신의 이전 상태**를 입력 받아 **자신의 상태를 갱신**
- 모든 시간대에서 동일한 매개변수(parameter)를 적용

- RNN을 Unfold 해보면...

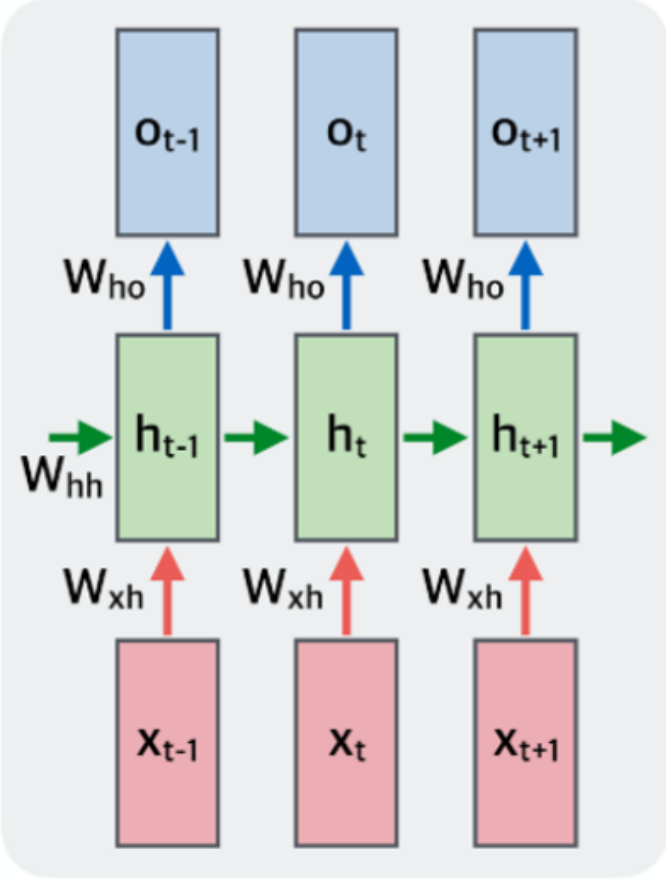




- x_t: 시간 스텝 t 에서의 입력값
- W_{hh}, W_{xh}, W_{ho}: **모든 스텝(시간)에서 공유됨**
- h_t: 시간 스텝 t 에서의 중간 값 (hidden state)

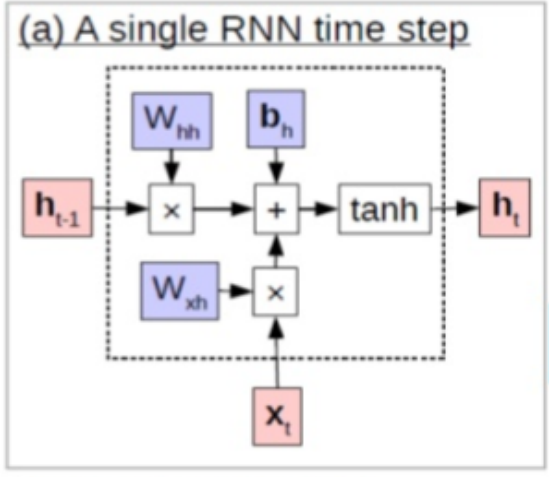
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

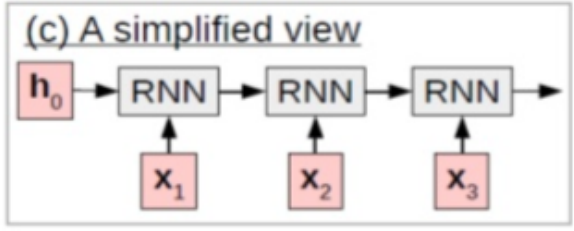
- 현재 상태 h_t는 이전 스텝의 상태 h_{t-1}와 현재 스텝의 입력값 x_t에 의해 계산됨
- o_t: 시간 스텝 t 에서의 출력값 (score)



RNN의 특징

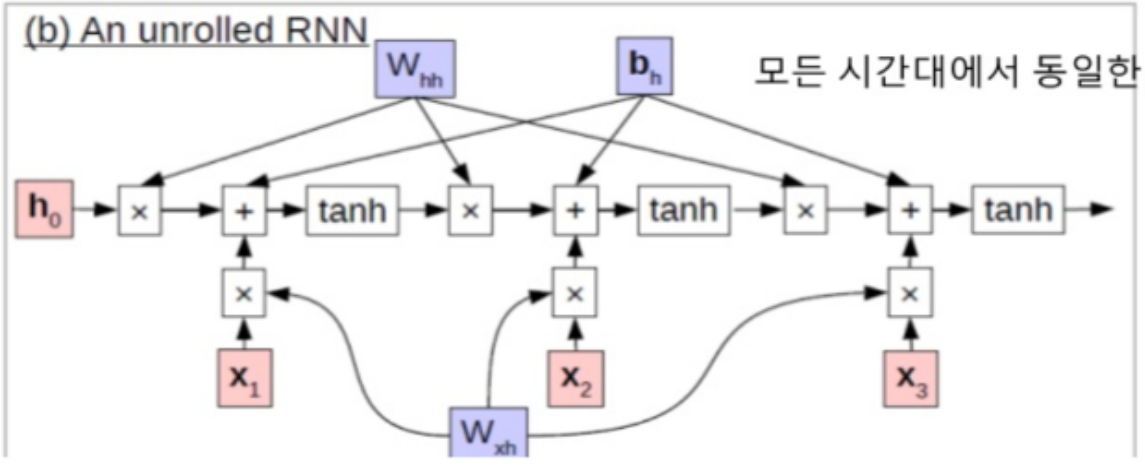
- 상태 h_t를 네트워크의 메모리라고 생각할 수 있음 (이전 시간 스텝에 영향을 받으며, 이에 대한 정보를 담고 있음)
- 출력값 o_t는 현재 상태 h_t에만 의존하고 있음
- 일반적인 뉴럴 네트워크와 달리 파라미터는 **모든 스텝에서 공유**되고 있음
- 학습해야 할 파라미터를 대폭 낮춰 줌 (공유 하지 않으면 시퀀스 길이가 긴 경우 엄청난 파라미터 개수..)





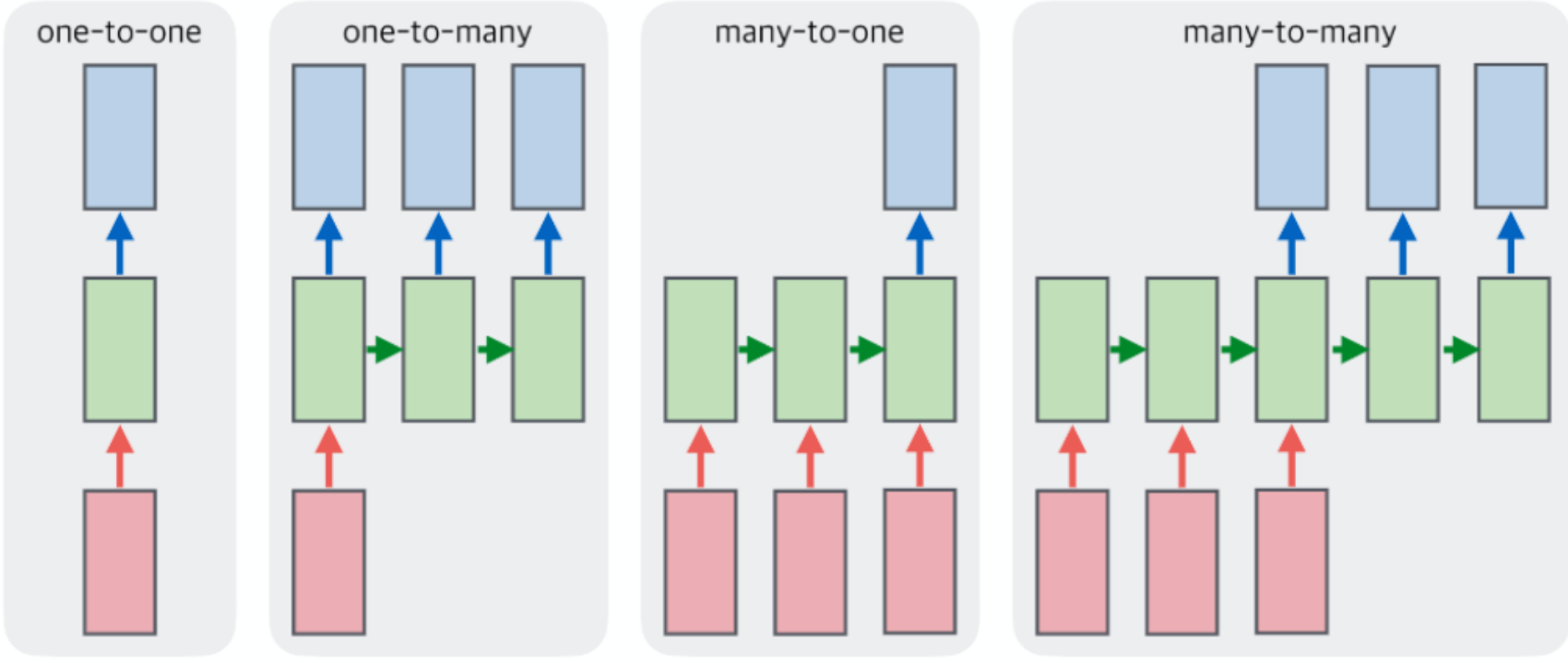
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

(하얀 노드: operation, 노드 간 연결: tensor)



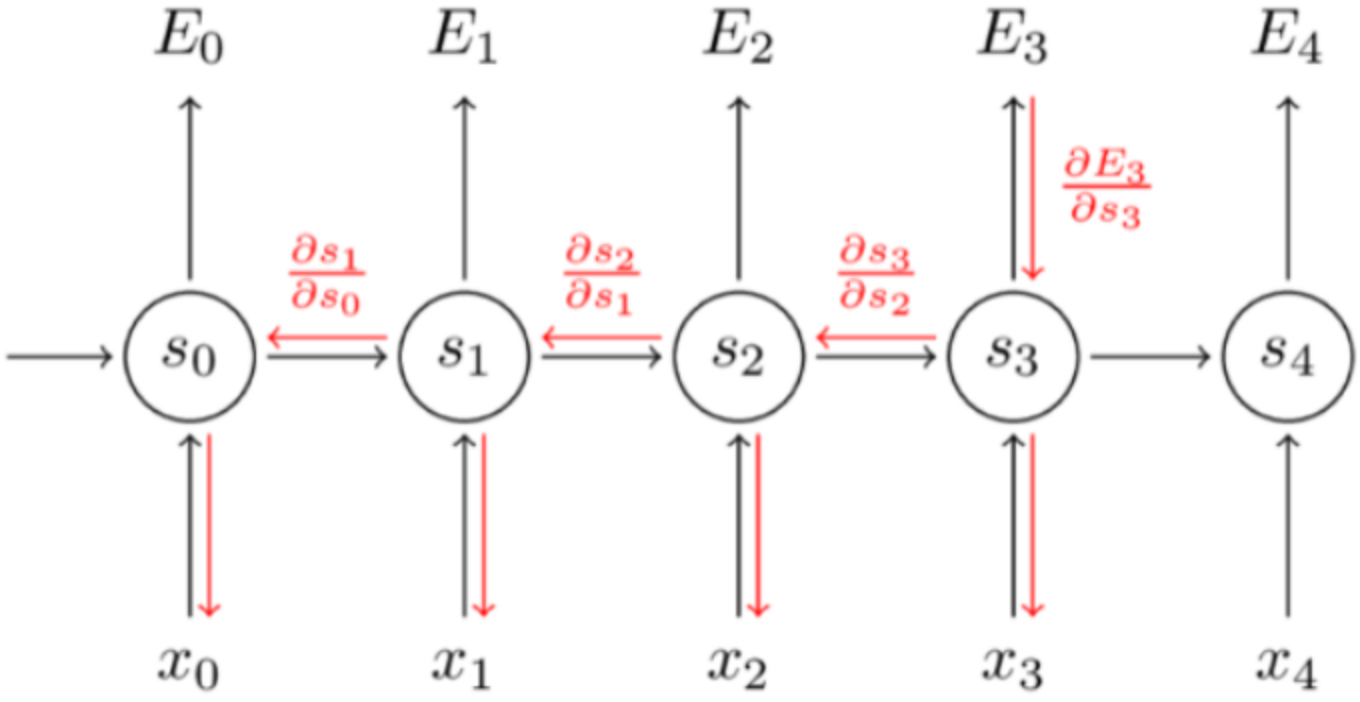
모든 시간대에서 동일한 매개변수를 적용: W_{hh}, W_{xh}, b_h

- 풀고자하는 문제에 따라 RNN을 이용해 다양한 구조를 직접 설계

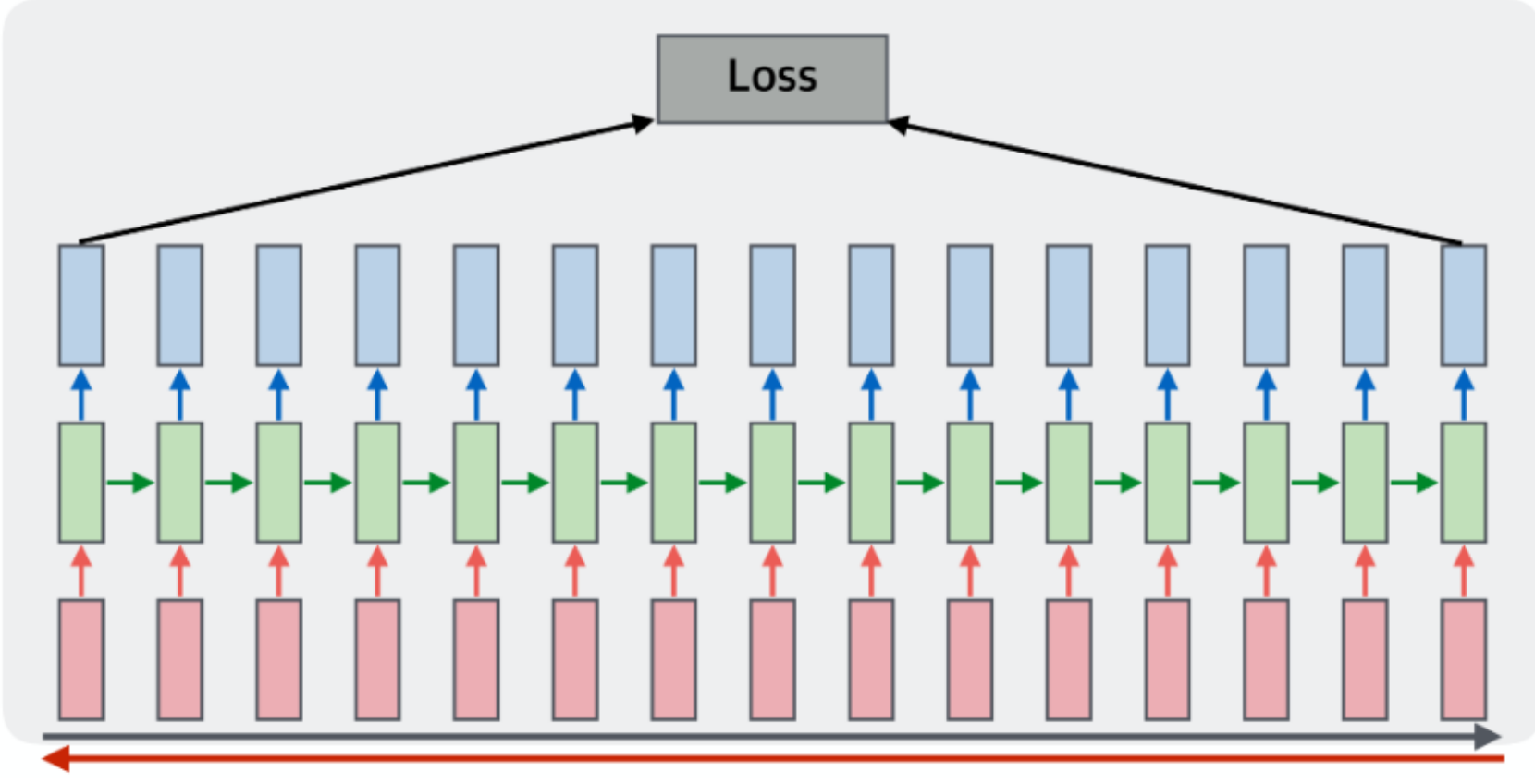


Backpropagation through time

- RNN은 문장과 같은 sequence를 입력으로 받기 때문에 학습 시 backpropagation을 **시간에 대해서도 수행**
- 매 스텝마다 loss를 계산하며(E0~E4), 총 loss는 각 스텝 loss의 합

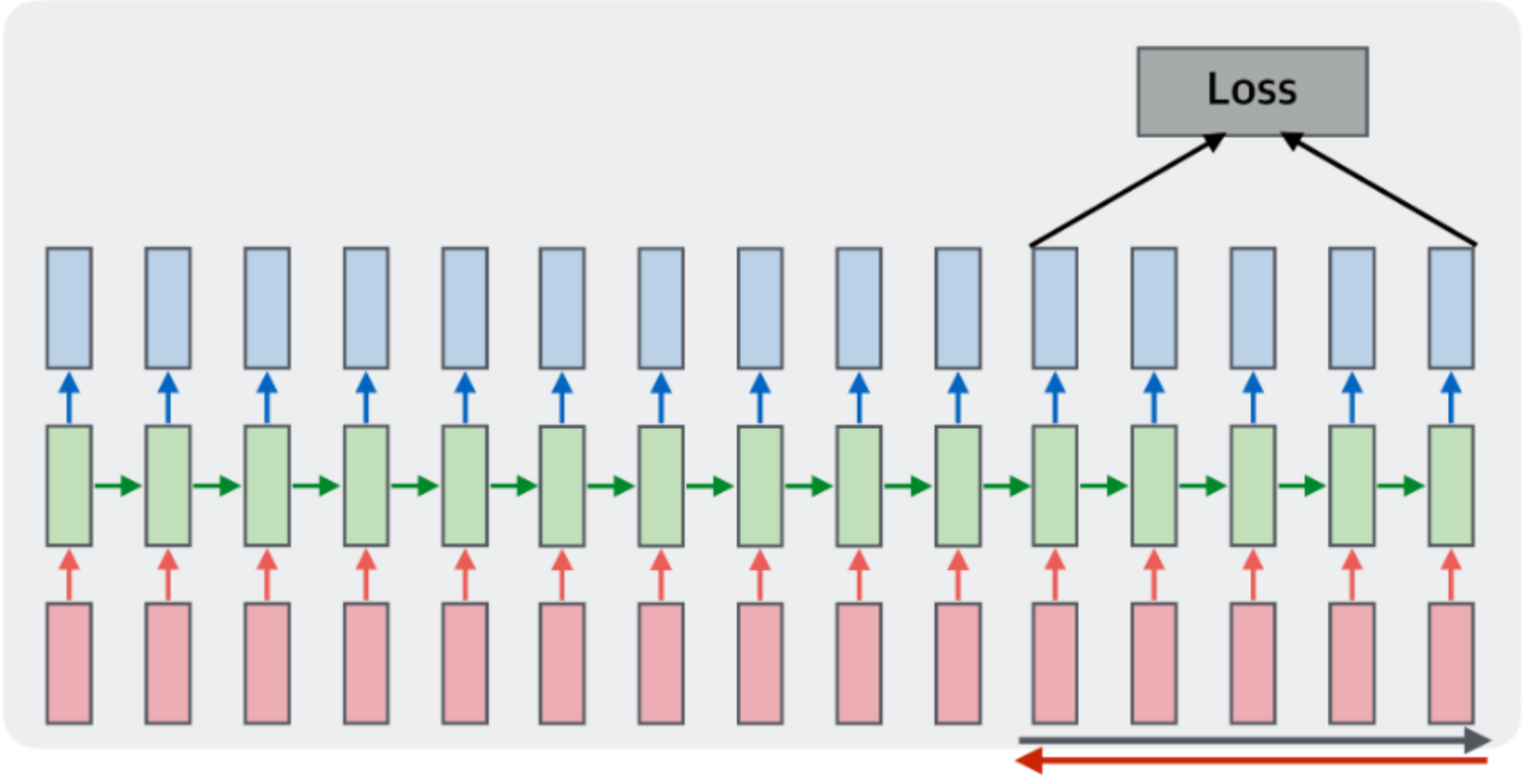


- **모든 시퀀스**에 대해 forwardprop을 진행하고 loss를 계산 한 뒤, backprop시 **모든 시퀀스**에 대해 그라디언트를 계산
- 시퀀스가 길면? -> 매우 느린 계산 속도



Truncated BPTT

- Forward와 backward를 전체 시퀀스에 대해 수행하지 않고 **부분 (chunk) 시퀀스**에 대해서만 수행



RNN Problem (Vanishing gradients)

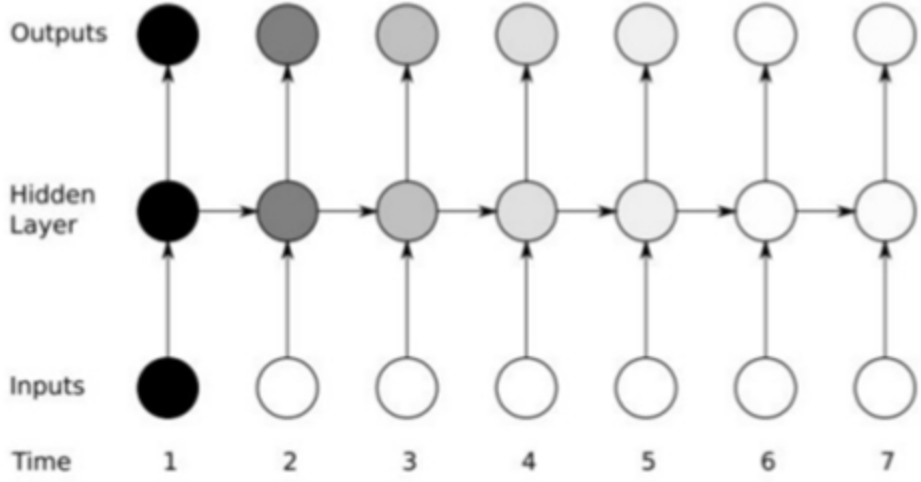
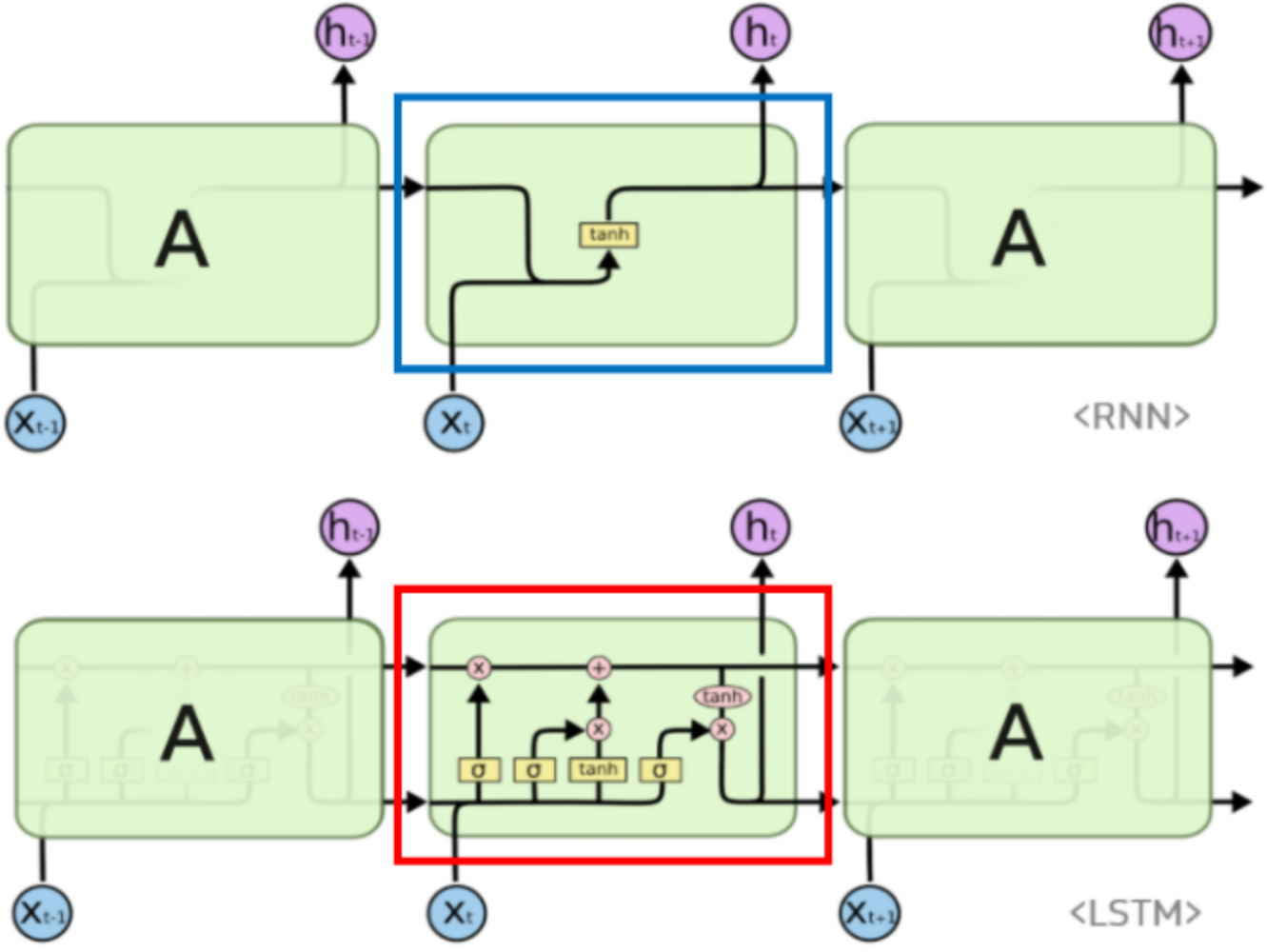


Figure 4.1: **The vanishing gradient problem for RNNs.** The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

- **RNN은 long-time sequence 를 잘 처리하지 못함**
  - 여러 스텝이 지난 경우 먼 step의 단어를 까먹음
  - **각 스텝마다 그라이언트를 곱하면서 vanishing이 일어나기 때문!!**

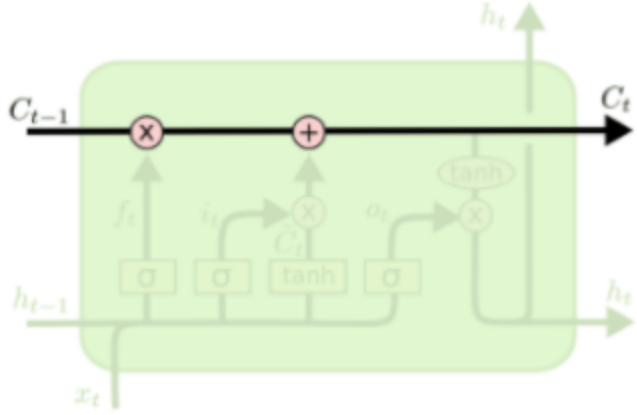
- **해결책**
  - RNN의 활성화 함수를 ReLU로 바꾸자 (하책)
  - **LSTM, GRU와 같은 RNN변형 모델을 사용해 보자 (상책)**

LSTM



기본 RNN보다 구조가 훨씬 복잡해보임

LSTM 의 핵심



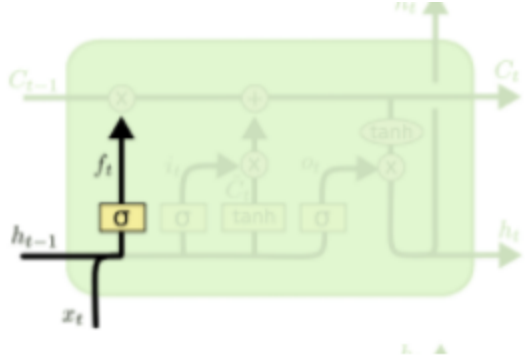
- `Cell State`
  - 약간의 선형 상호작용만 일으키며 지나간다.
  - `일종의 컨베이어 벨트 역할` = **`과거의 데이터를 쭉~~~~ 알 수 있다!!`**
    - LSTM은 cell state에 정보를 더하거나 지운다.
    - `gate` 들이 이 과정을 조절

Gate(게이트) 란?

- `sigmoid 함수에 의해 0~1로 제한`
- **`벡터와 게이트가 곱해진다면 이 벡터를 얼마나 통과시킬지 정해주는 역할`**

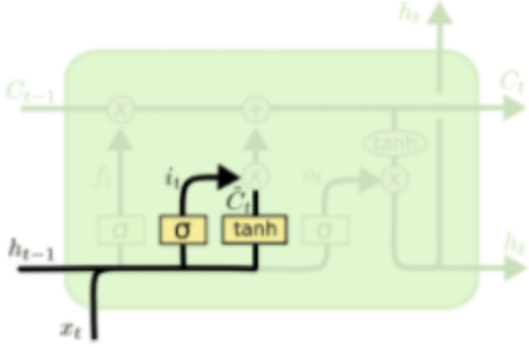
LSTM Decomposition

- **01. forget Gate**
  - cell의 어떤 정보를 버릴 것인지 결정
  - sigmoid로 결정



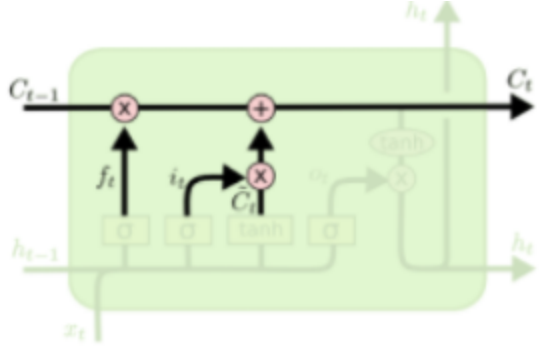
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- **02. Input Gate**
  - sigmoid층은 어떤 값을 갱신할지 결정
  - tanh층은 cell state에 더해질 새로운 후보 값들의 벡터 Ct를 만듦
  - 이 둘을 합침!!



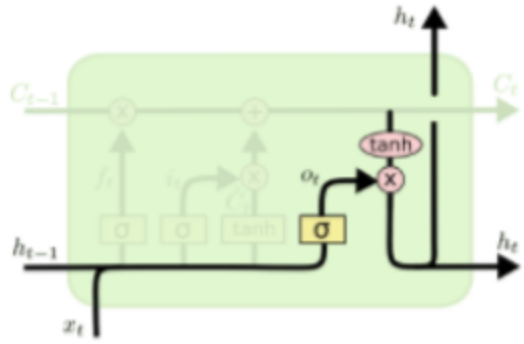
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- **03. 갱신**
  - 낡은 Ct-1을 Ct로 갱신



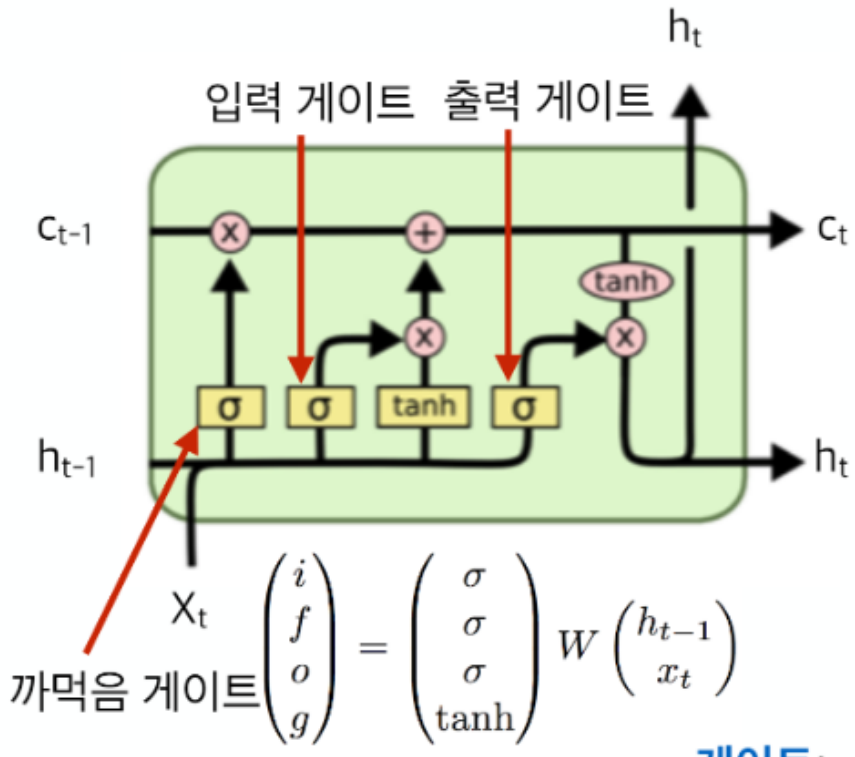
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **04. Output Gate**
  - cell state 여과 버전



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

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



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

i: 입력 게이트
상태 h_t을 계산 시 현재 입력값 x_t를
얼마나 사용할 지 결정

f: 까먹음 게이트
상태 h_t을 계산 시 이전 내부 상태값 c_{t-1}를
얼마나 기억하고 사용할 지 결정

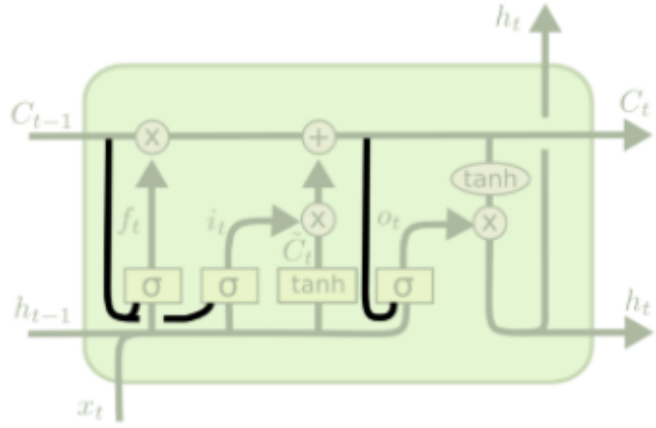
o: 출력 게이트
내부 상태값 c_t를 외부에 얼마나
출력할지 결정

g:
기존 RNN의 상태값 h_t와 비슷한 역할
(동일한 수식)

게이트:
시그모이드 함수에 의해 0~1로 제한되며,
벡터와 곱한다면 이 벡터를 얼마나 통과시킬 지 정해줌

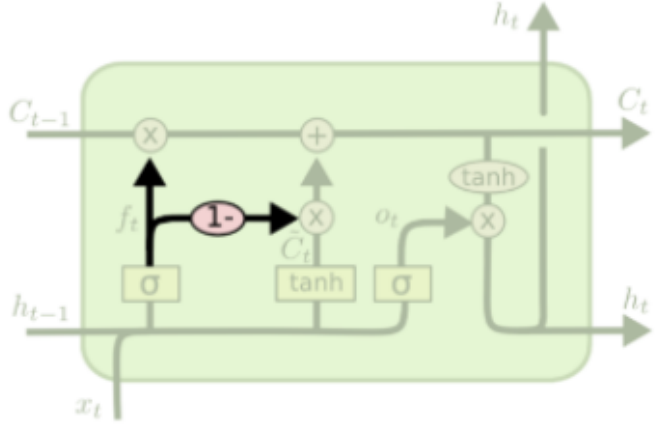
LSTM 파생형들

- Peephole LSTM



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

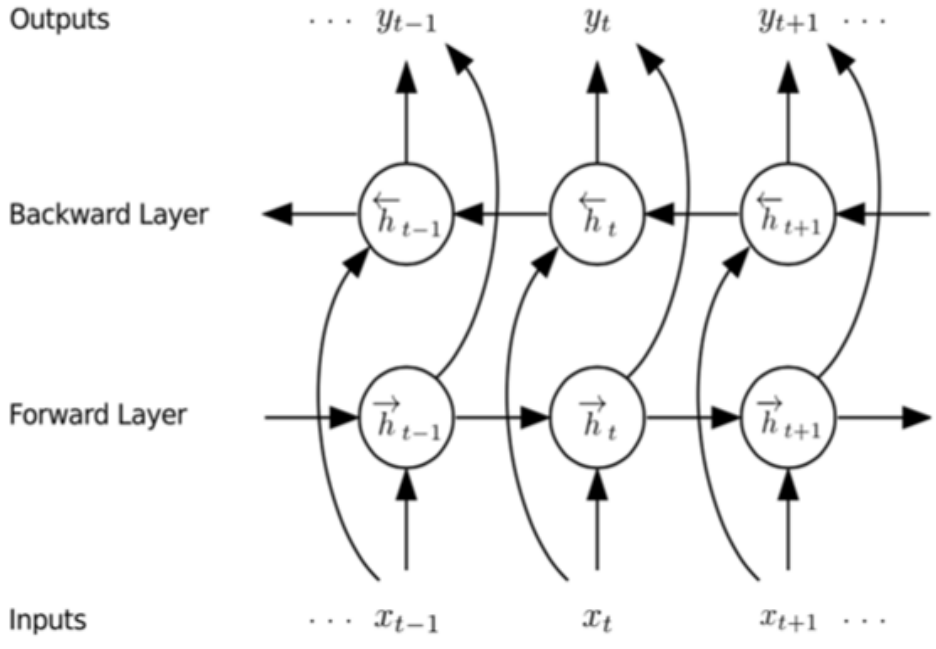
- **GRU**



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

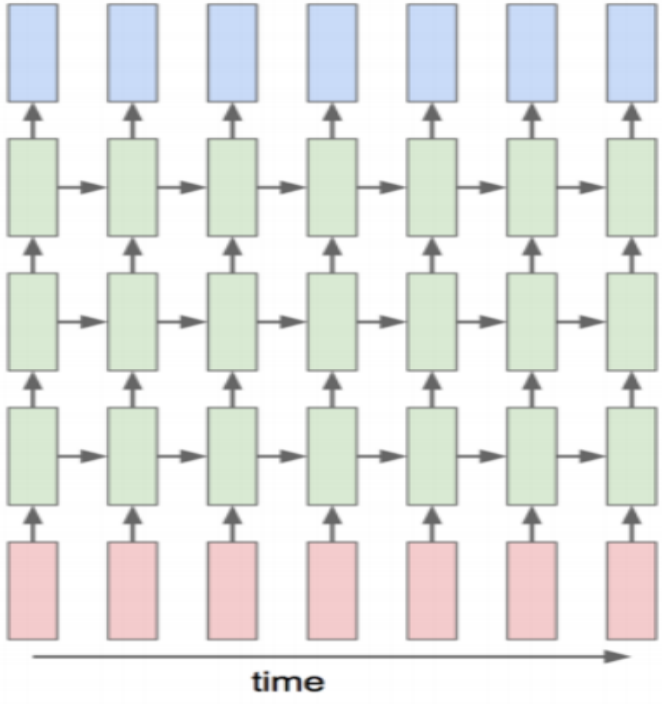
Bi-RNN

- **실제 우리가 글을 적거나 말을 할 때 앞의 나올 상황에서 뿐만 아니라 뒤에 상황도 고려**
- 한쪽 방향 뿐만 아니라 **양방향으로 학습 할 수 있게 설계**
- 일반적으로 RNN보다 성능이 더 좋다고 알려져 짐
- 기본 RNN에서는 잘 사용하지 않고 LSTM이나 GRU에서 많이 사용.



Multi-RNN

- RNN은 모든 sequence에 대해 계산하기 때문에 연산량이 NN, CNN보다 많아 CNN처럼 깊게 쌓기는 어려움
- 데이터가 많고 연산 속도가 빠른 환경이라면 **Multi-RNN도 좋은 선택지가 될 수 있음**
- 기본 RNN에서는 잘 사용하지 않고 LSTM이나 GRU에서 많이사용



RNN 정리

- 문장과 같은 시퀀스 데이터를 처리할 때는 RNN계열의 네트워크를 고려 (가변 길이의 데이터를 입력 받을 수 있음)
- 기본 RNN은 gradient vanishing / exploding 현상 우려
- LSTM, GRU는 기본 RNN의 장기 종속성 문제를 해결해 줌