RNN

분석 16기 이지혜

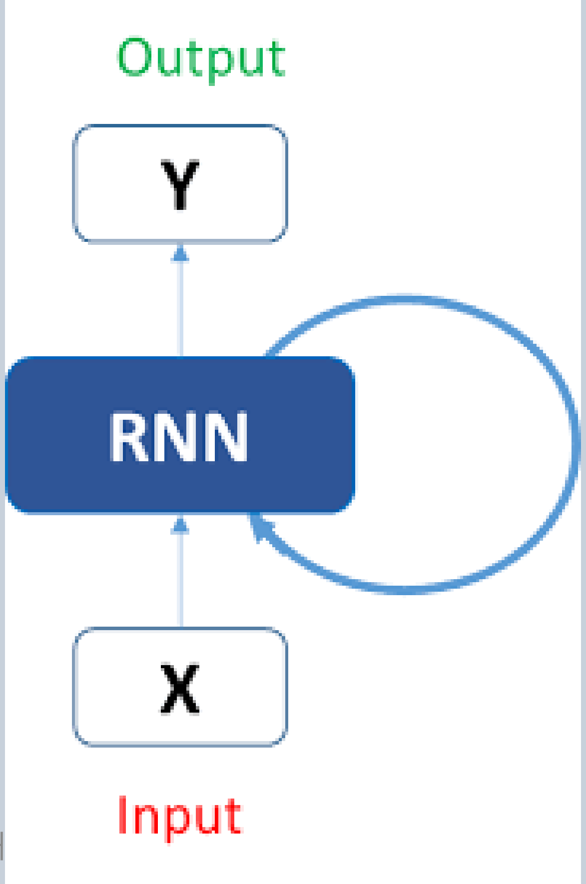
Contents

Objective

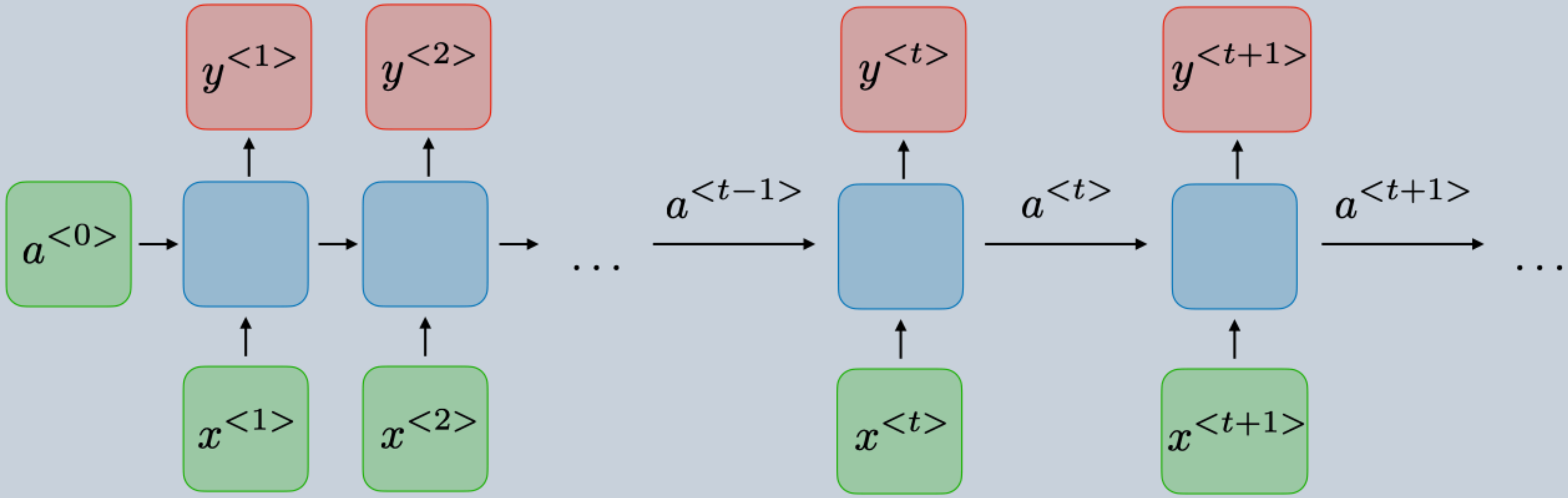
Feed Forward Models CANNOT find the RELATIONSHIP between
THE INPUT and WHAT WE PREDICT later on in the sequence

- Feed Forward Models
  : Deep Neural Network
  : Convolution Neural Network

- OUTPUT = f (INPUT, PAST MEMORY)

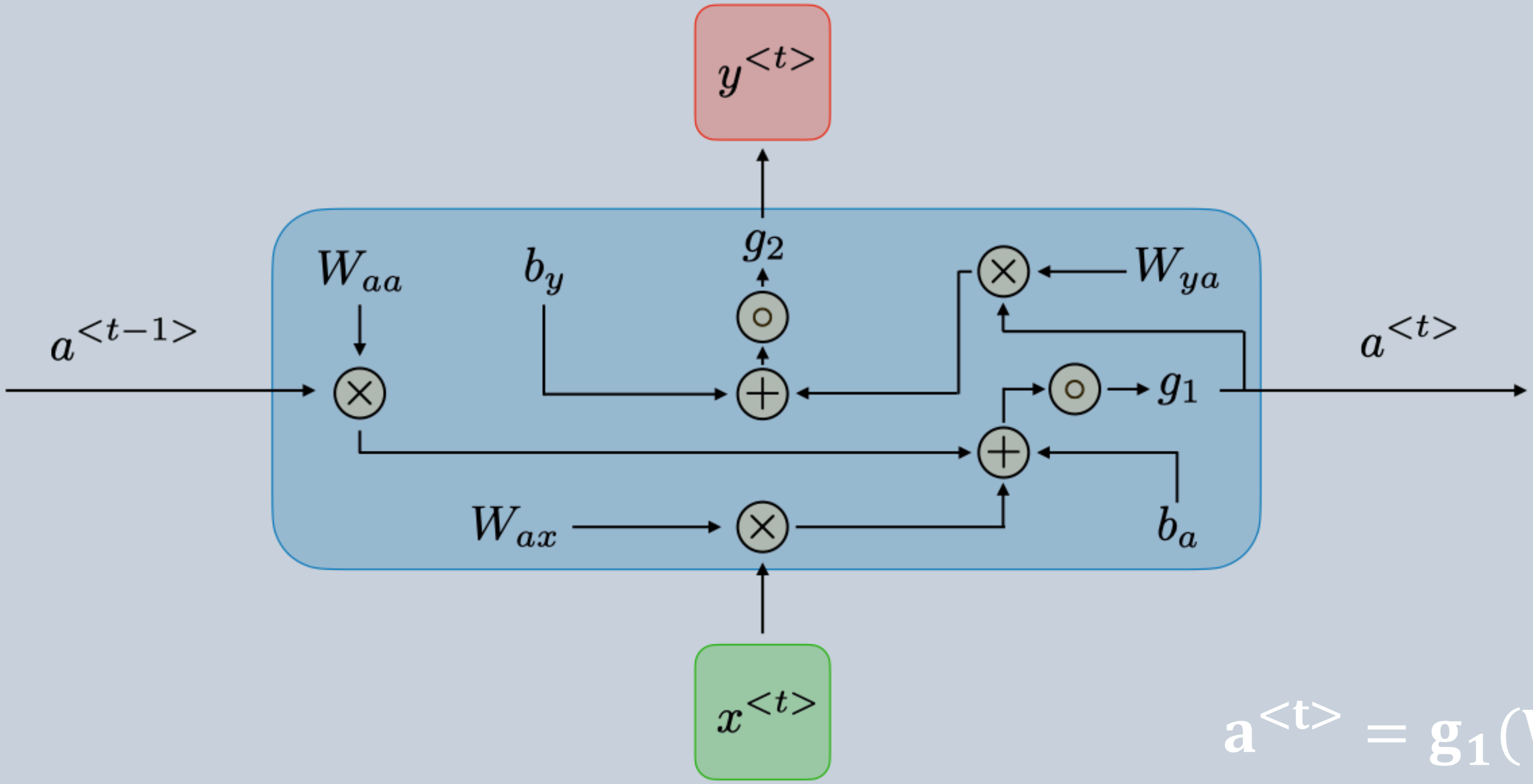


1. Traditional RNN



- Timestep t
- Hidden Data $a^{<t>}$
- Output $y^{<t>}$

1. Traditional RNN



$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

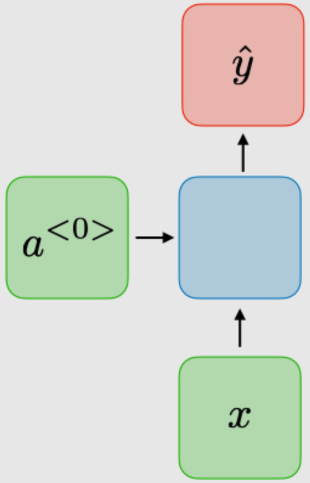
$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

1. Traditional RNN

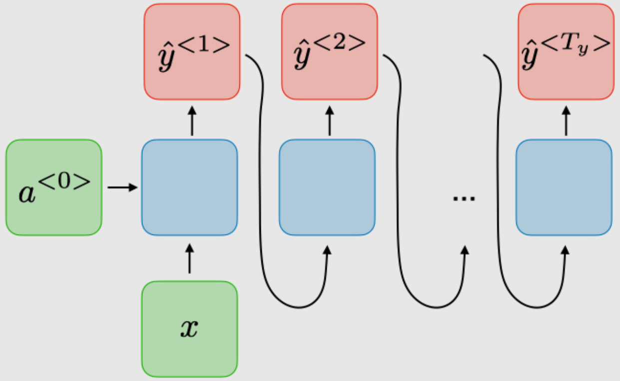
장점	단점
1. 길이의 제한이 없음 2. 입력 크기에 따라서 model의 크기가 따라서 커지지 않음 3. 시간의 흐름에 따라 가중치 공유	1. 계산 속도가 매우 느림 2. 단기의 기억만이 적용 가능 3. 미래의 출력만 예측할 뿐, 미래의 입력은 예측 불가

2. Applications of RNN

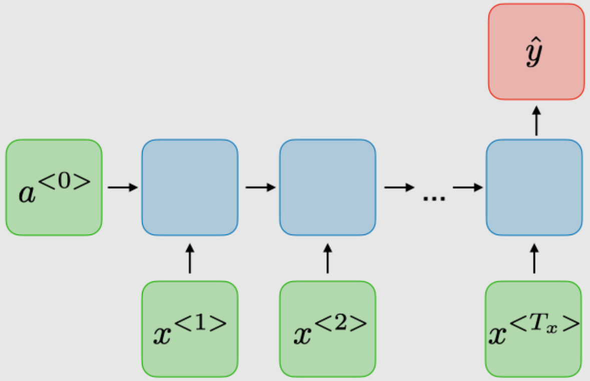
One - To - One	One - To - Many	Many - To - One
$T_x = T_y = 1$	$T_x = 1, T_y > 1$	$T_x > 1, T_y = 1$



Traditional Neural Network



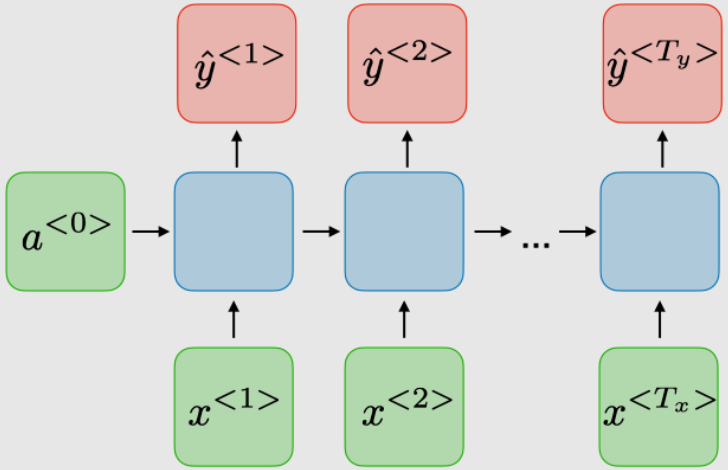
Music Generation



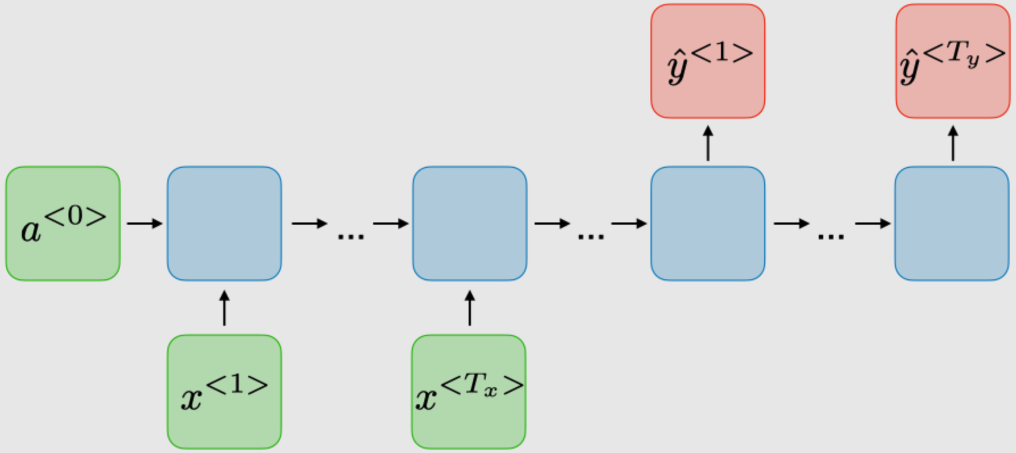
Sentiment Classification

2. Applications of RNN

Many - To - Many	Many - To - Many
$T_x = T_y$	$T_x \neq T_y$



Name Entity Recognition



Machine Translation

2. Applications of RNN

2-2. Loss Function

- 모든 timestep t에 대해 아래와 같이 loss function을 적용

$$\mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{\mathbf{t}=1}^{\mathbf{T_y}} \mathbf{L}(\hat{\mathbf{y}}^{<\mathbf{t}>}, \mathbf{y}^{<\mathbf{t}>})$$

3. Architectures of RNN

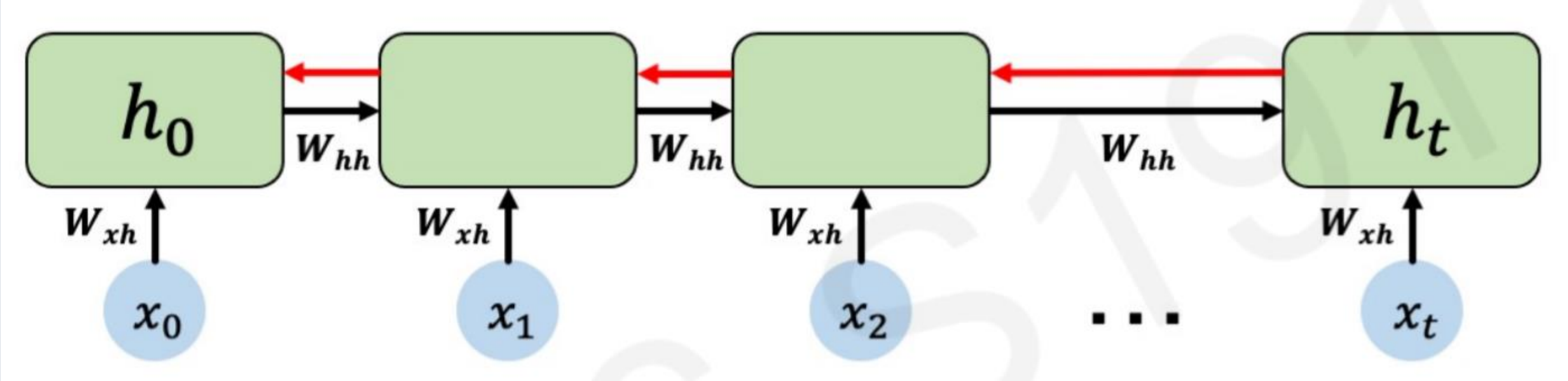
3-1. Backpropagation in Feed Forward Models

1. 모든 loss에 대해서 각각의 parameter의 편미분 값을 계산

2. loss를 최대한으로 줄이는 방향으로 parameter 값을 갱신/변형

3. Architectures of RNN

3-2. Backpropagation in RNN



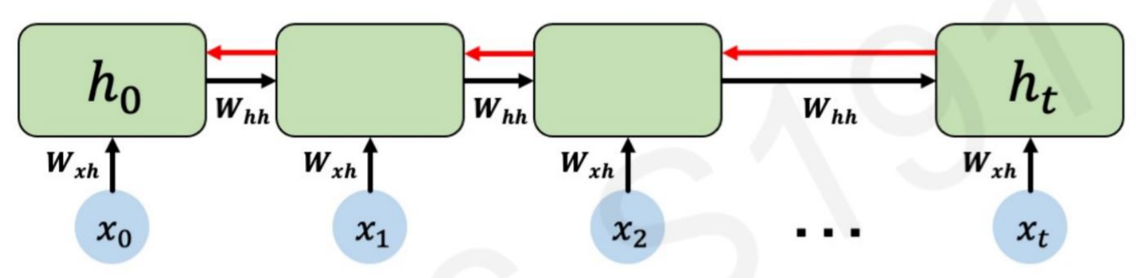
$$\frac{\partial L^{(t)}}{\partial W} = \sum_{t=1}^{T} \frac{\partial L^{(t)}}{\partial W}\Bigg|$$

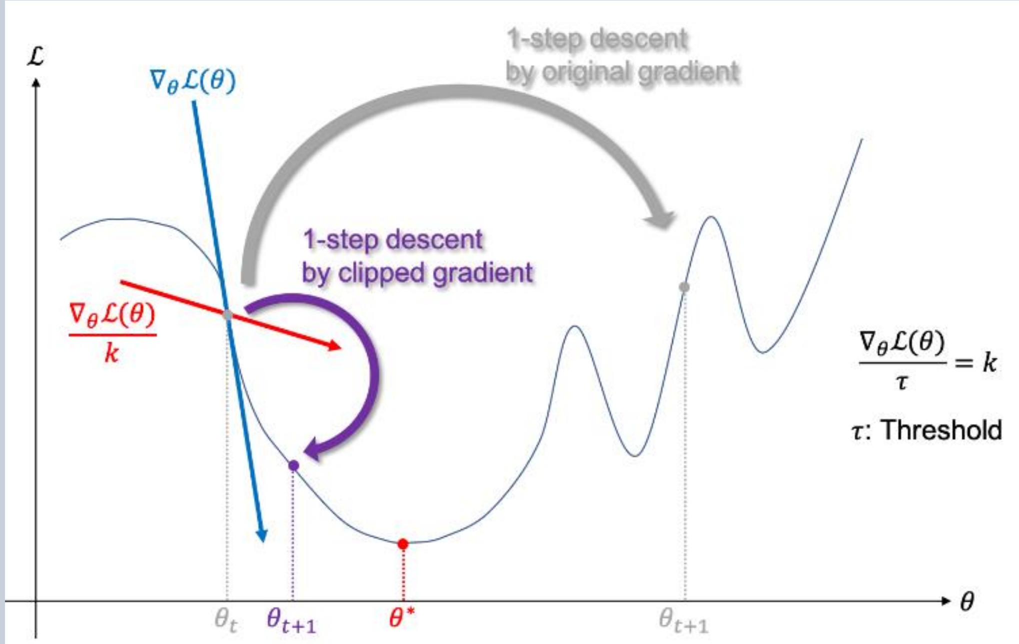
새로운 문제

- 긴 sequence로 RNN을 훈련을 하기 위해 많은 time step을 사용해야 함
- RNN 또한 이렇게 되면 펼쳤을 때 긴 네트워크가 됨
- 이런 경우 보통 심층 신경망처럼 불안정한 gradient 문제가 발생
- 뿐만 아니라 단기 기억 문제를 해결해야 하는 필요성도 생김

4. Handling Long Term Dependencies (= 장기 의존성 문제)

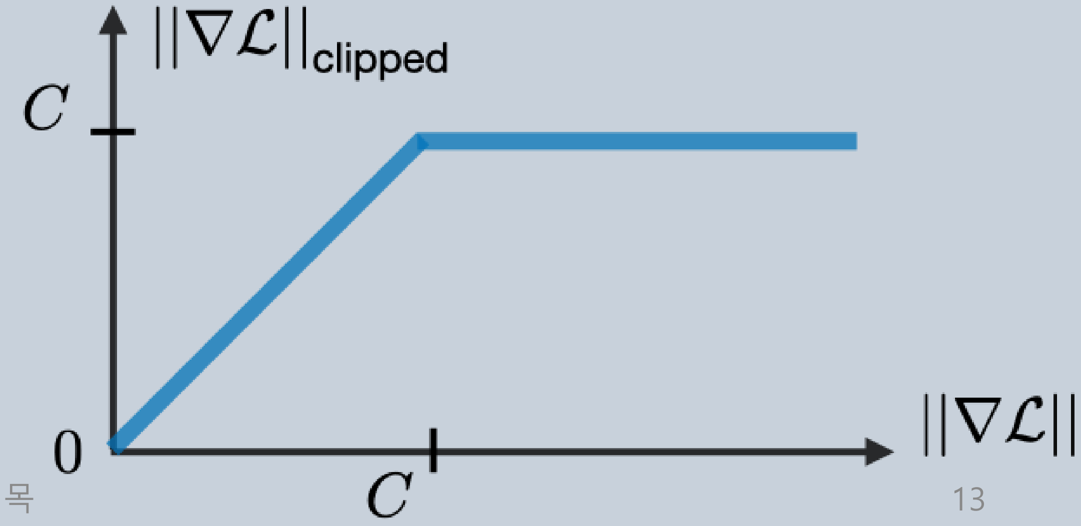
4-1. Exploding Gradients





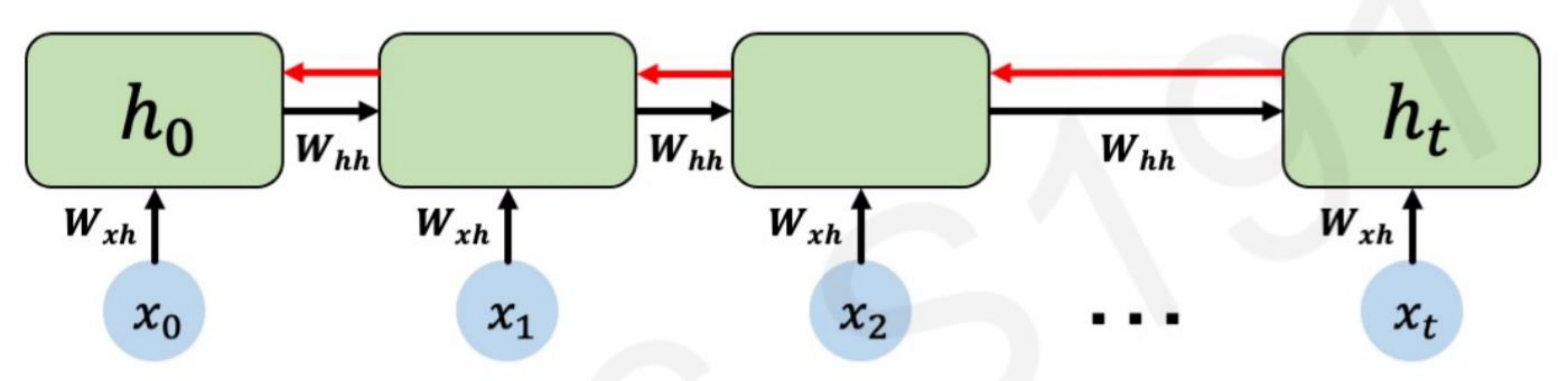
가중치 matrix 들이 1 이상일 때
-> Exploding Gradient

-> Gradient Clipping으로 해결



4. Handling Long Term Dependencies

4-2. Vanishing Gradients

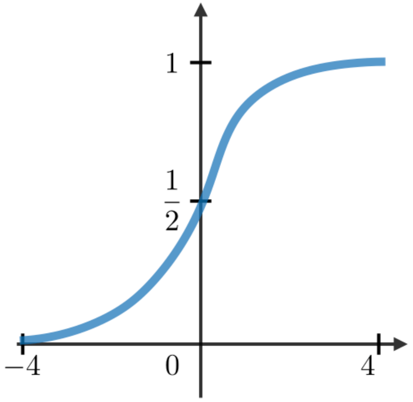
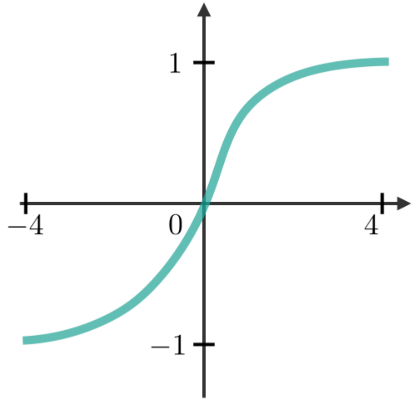
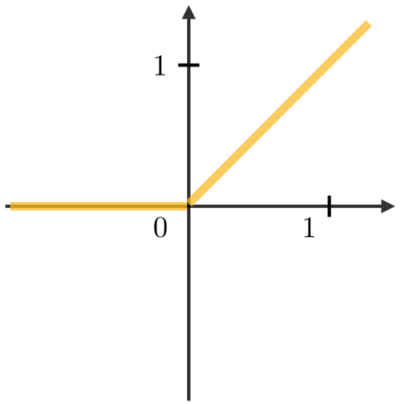


가중치 matrix들이 1 미만일 때
-> Vanishing Gradient

-> 해결책
1. ReLU Activation Function 사용
2. Weight Initialization
3. Gated Cells

4. Handling Long Term Dependencies

4-3. Activation Functions

Sigmoid	Tanh	ReLU
$g(z) = \dfrac{1}{1+e^{-x}}$	$g(z) = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

ReLU를 사용하면 f', 즉 gradient 값이 $x > 0$ 일 때에 작아지는 것을 다른 두 함수보다 방지해 줌

4. Handling Long Term Dependencies

4-4. Weight Initialization

1. 편향 값을 0으로 바꾸어 줌

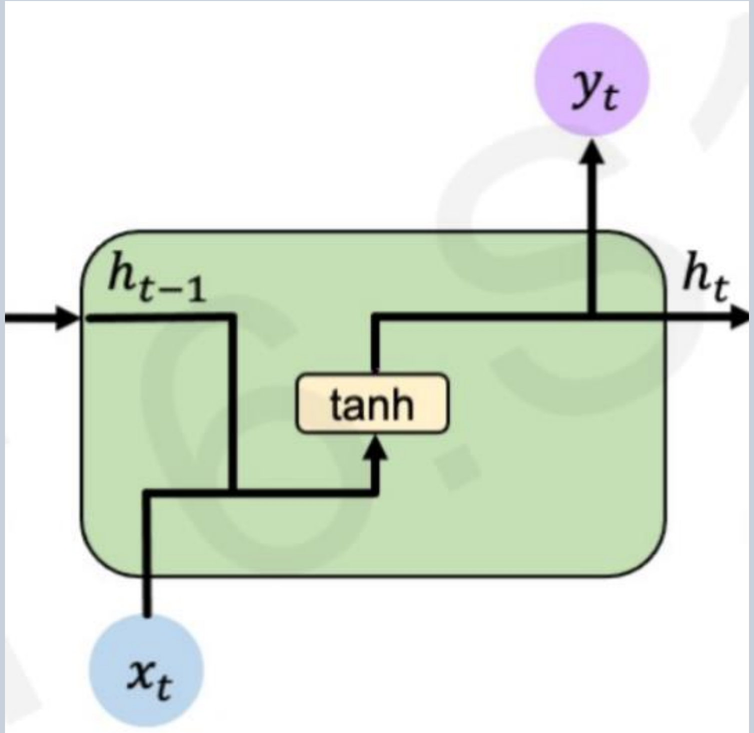
2. 가중치를 아래 행렬과 같은 identity matrix로 바꾸어 줌

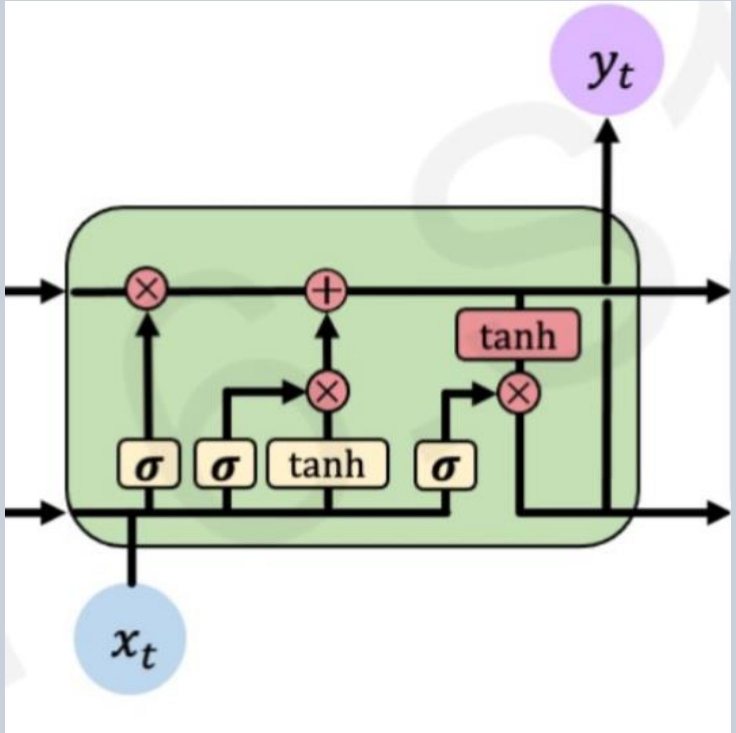
3. 층 정규화 (Layer Normalization)

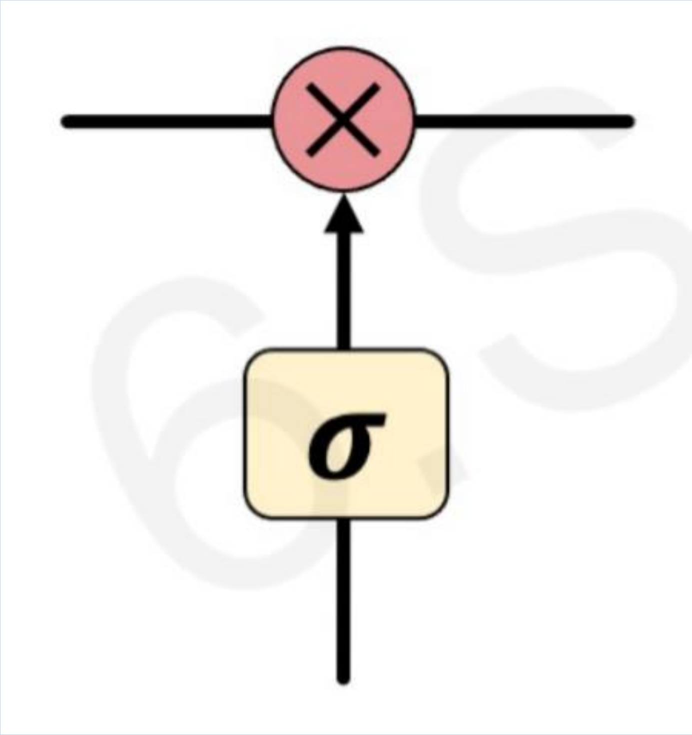
$$\mathbf{I_n} = \begin{bmatrix} \mathbf{1} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{1} \end{bmatrix}$$

4. Handling Long Term Dependencies

4-5. LSTM (Long Short-Term Memory)





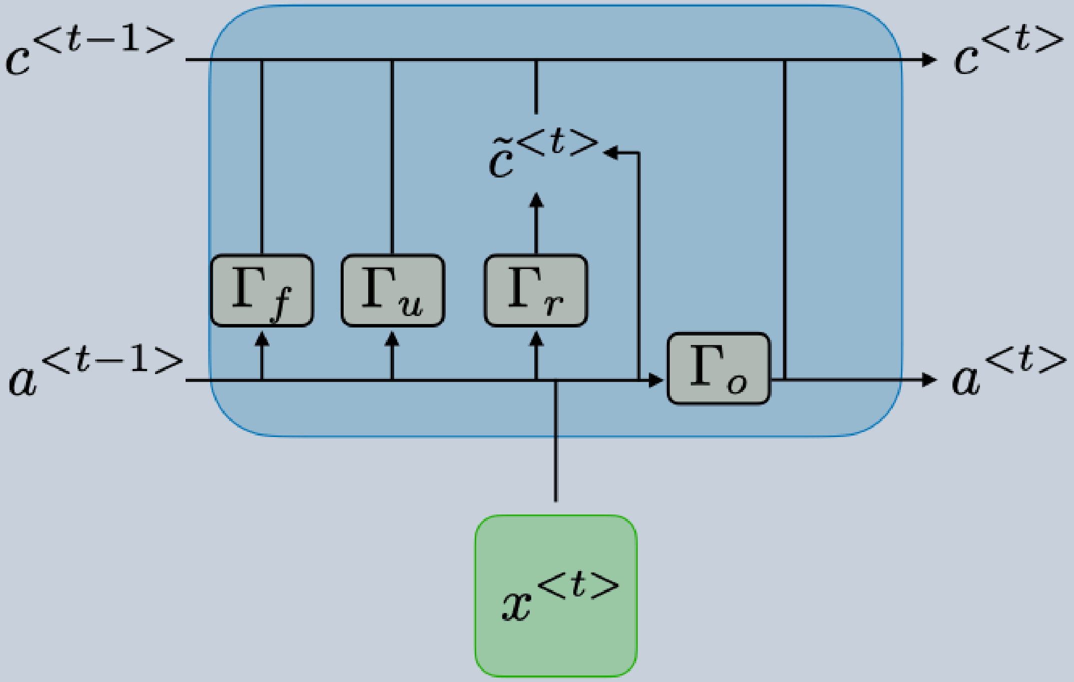


LSTM은 일반적인 RNN이 단순 계산만을 각각의 input에 대해 적용하는 것과 달리 정보의 흐름을 반영해서 정보를 처리하는 computation block을 사용

Gate를 이용해서
정보를 제거하고 추가

4. Handling Long Term Dependencies

4-5. LSTM (Long Short-Term Memory)



$$a^{<t>} = \tau_o \cdot c^{<t>}$$

$$c^{<t>} = \tau_u \cdot \tilde{c}^{<t>} + \tau_f \cdot c^{<t-1>}$$

$$\tilde{c}^{<t>} = \tanh\left(W_c\left(\tau_r \cdot a^{<t-1>}, x^t\right) + b_c\right)$$

4. Handling Long Term Dependencies

4-6. LSTM (Long Short-Term Memory)

과거의 불필요한/관련 없는 부분을 삭제

Forget

Store

새로운 관련된 정보를 cell state에 추가

내부 cell state value를 갱신

Update

Output

다음 time step에 정보 전달

4. Handling Long Term Dependencies

4-5. Gates

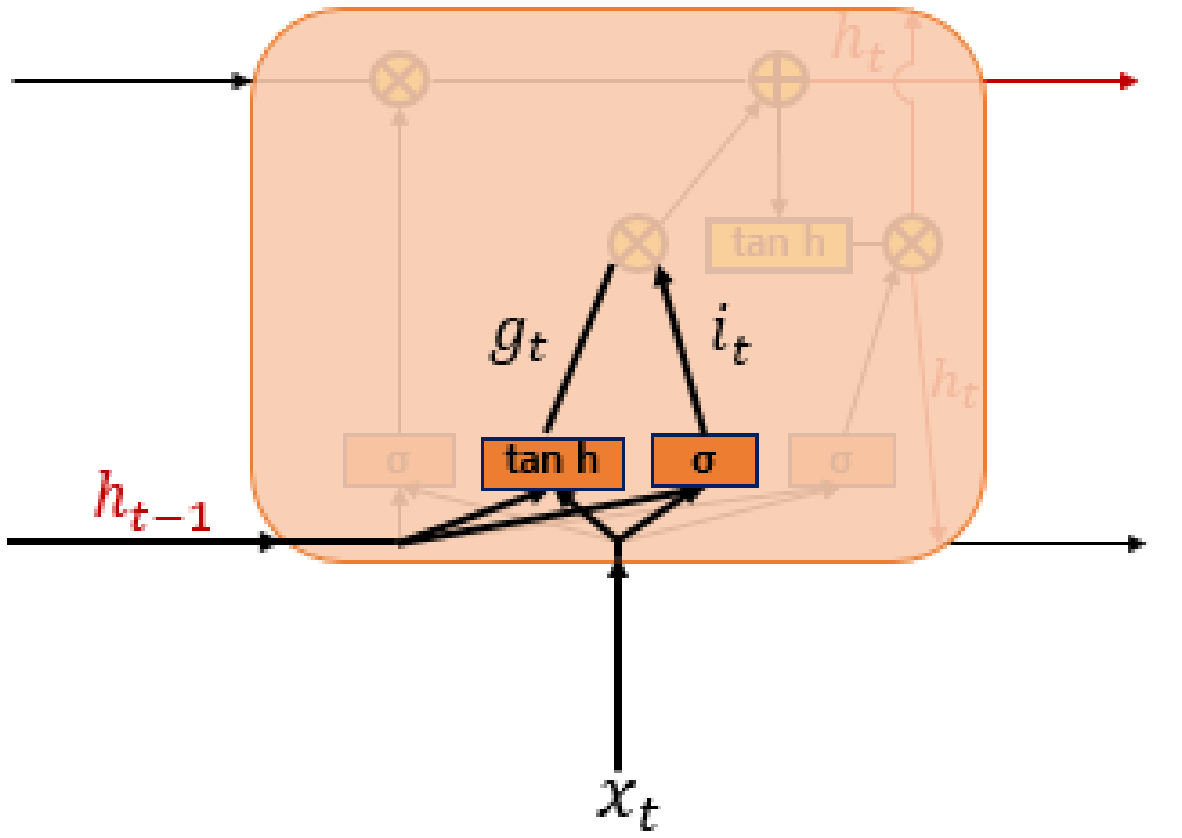
$$\boxed{\tau = \sigma(W_{x^{<t>}} + U_{a^{<t-1>}}) + b}$$

- 앞서 언급한 문제점인 vanishing gradient를 해결하기 위해 사용

Gate	역할	Unit
Update Gate (τ_u)	과거의 데이터의 얼마나 적용할지 결정	GRU, LSTM
Relevance Gate (τ_r)	과거의 어떤 정보를 지울지 말지	GRU, LSTM
Forget Gate (τ_f) (= 삭제 게이트)	특정 cell을 지울지 말지	LSTM
Output Gate (τ_o) (= 출력 게이트)	특정 cell을 얼마나 공개할지	LSTM

4. Handling Long Term Dependencies

4-6-1. Input Gate

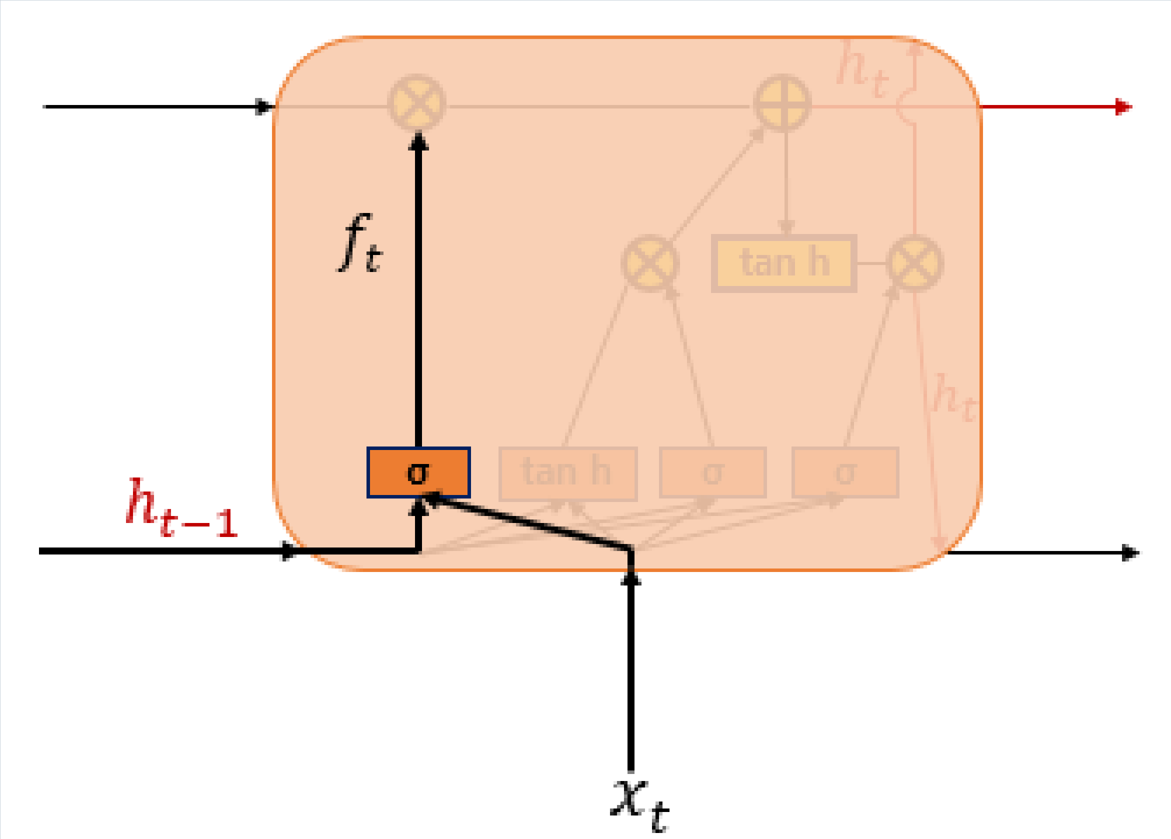


$$\mathbf{i}_{(\mathbf{t})} = \sigma(\mathbf{W}_{\mathbf{xi}}^{\mathbf{T}}\mathbf{x}_{(\mathbf{t})} + \mathbf{W}_{\mathbf{hi}}^{\mathbf{T}}\mathbf{h}_{(\mathbf{t}-\mathbf{1})} + \mathbf{b}_{\mathbf{i}})$$

$$g_{(\mathbf{t})} = \mathbf{tanh}(\mathbf{W}_{\mathbf{x}g}^{\mathbf{T}}\mathbf{x}_{(\mathbf{t})} + \mathbf{W}_{\mathbf{h}g}^{\mathbf{T}}\mathbf{h}_{(\mathbf{t}-\mathbf{1})} + \mathbf{b}_{g})$$

4. Handling Long Term Dependencies

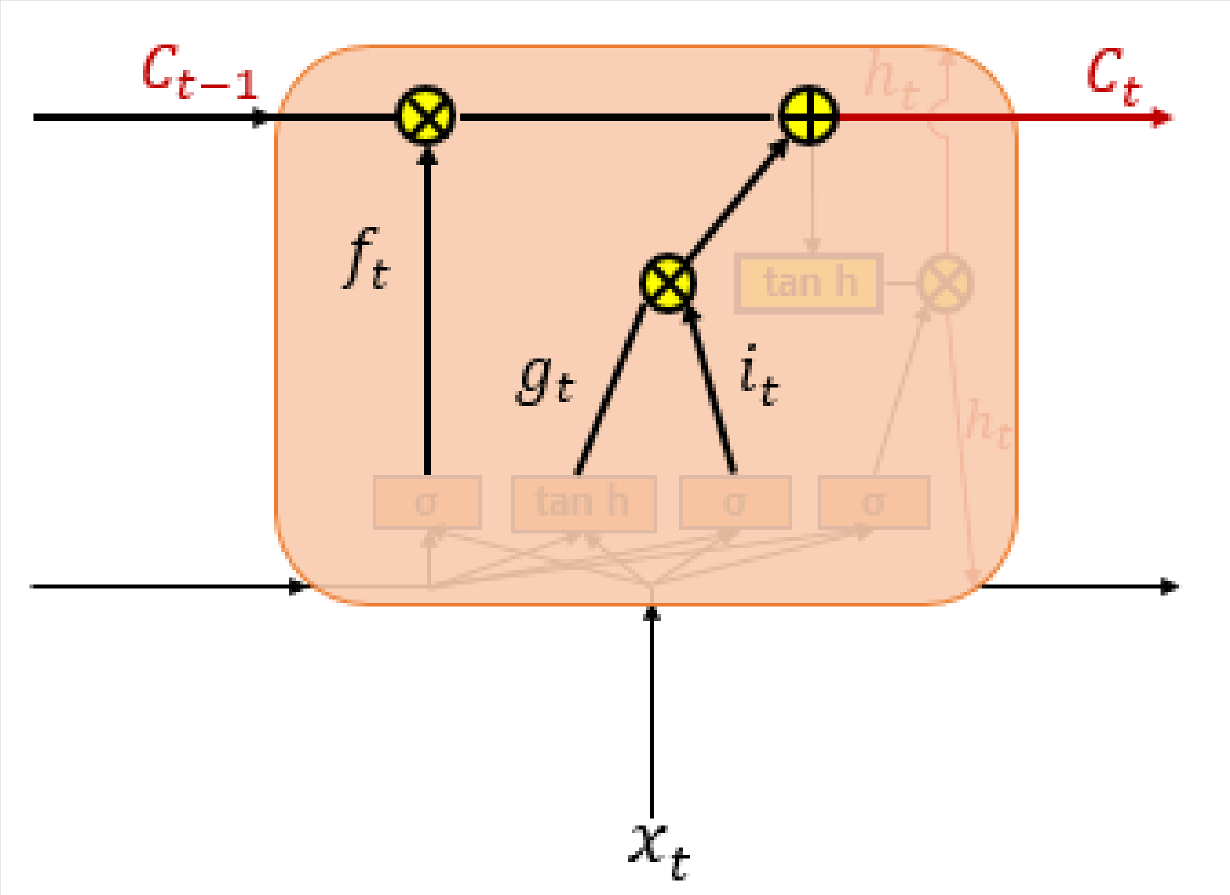
4-6-2. Reduce Gate



$$f_{(t)} = \sigma(\mathbf{W}_{xf}\mathbf{x}_{(t)} + \mathbf{W}_{hf}\mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

4. Handling Long Term Dependencies

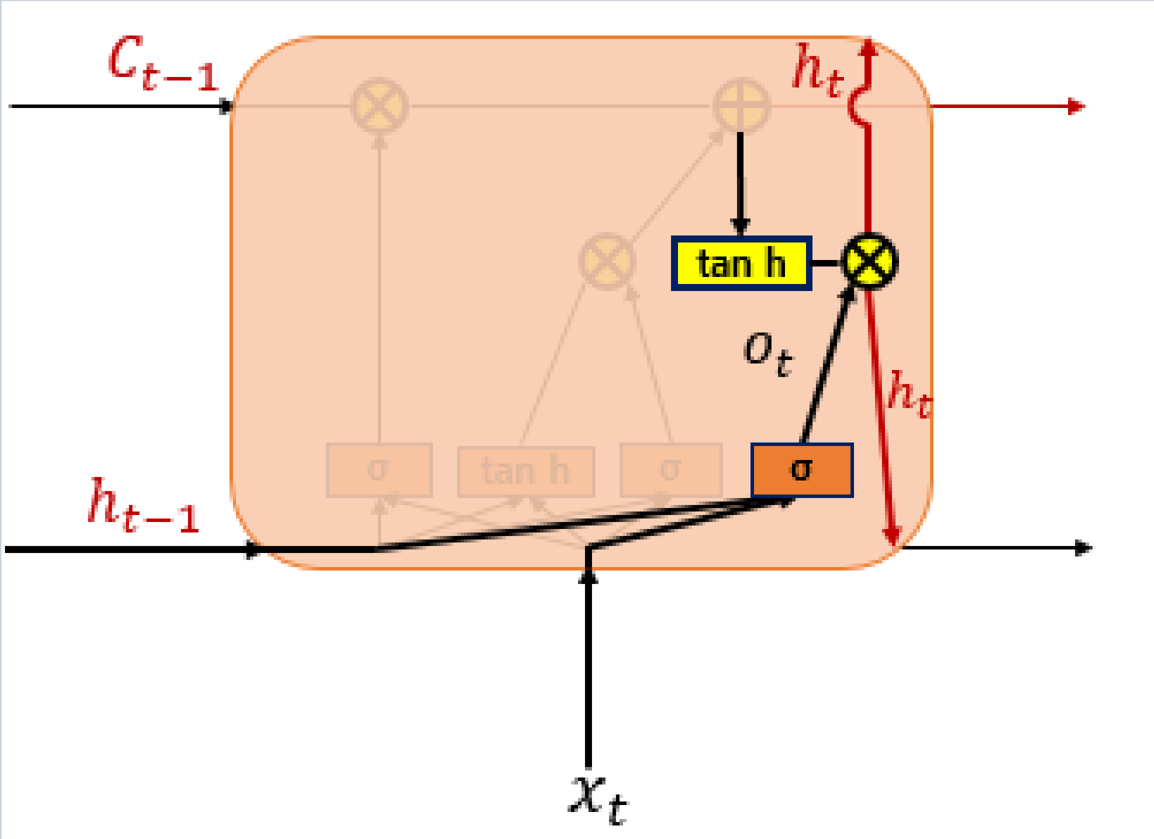
4-6. Cell 상태 (장기 상태)



$$C_{(t)} = f_t \cdot C_{t-1} + i_t \cdot g_t$$

4. Handling Long Term Dependencies

4-6. Output Gate + 은닉 상태 (단기 상태)

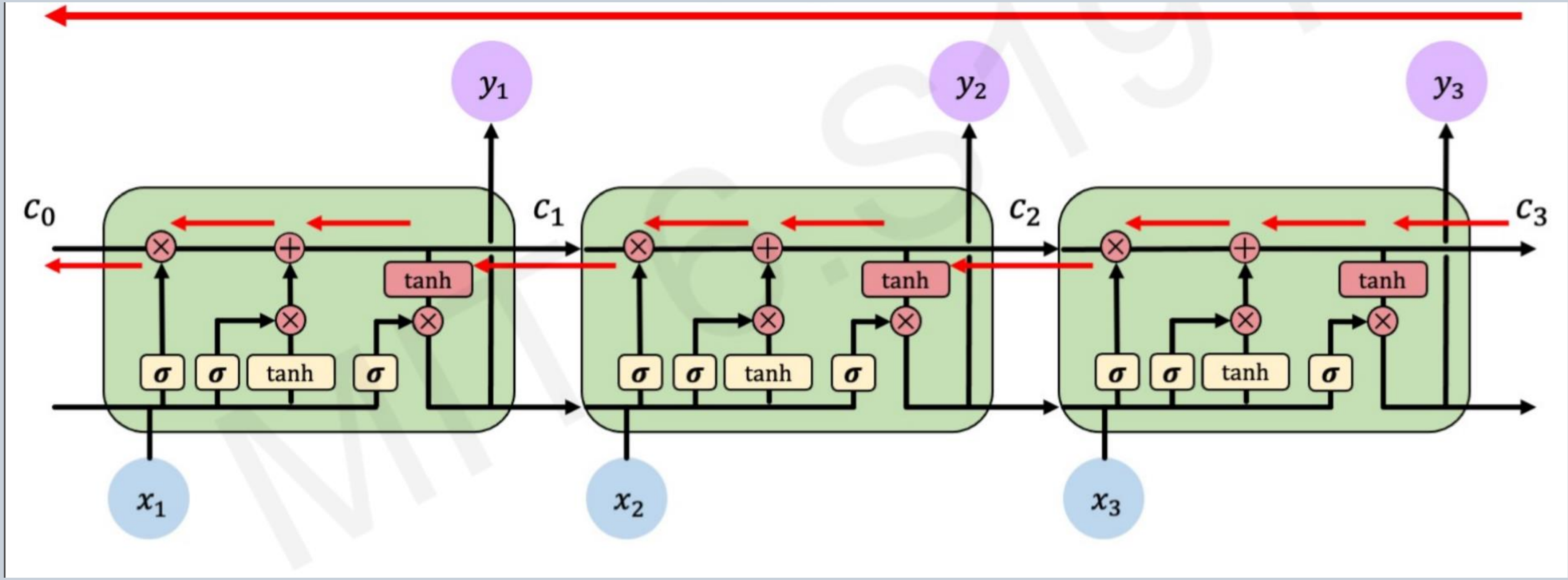


$$O_{(t)} = \sigma(W_{xo}x_{(t)} + W_{ho}h_{(t-1)} + b_o)$$

$$h_{(t)} = O_t \cdot tanh_{(c_t)}$$

4. Handling Long Term Dependencies

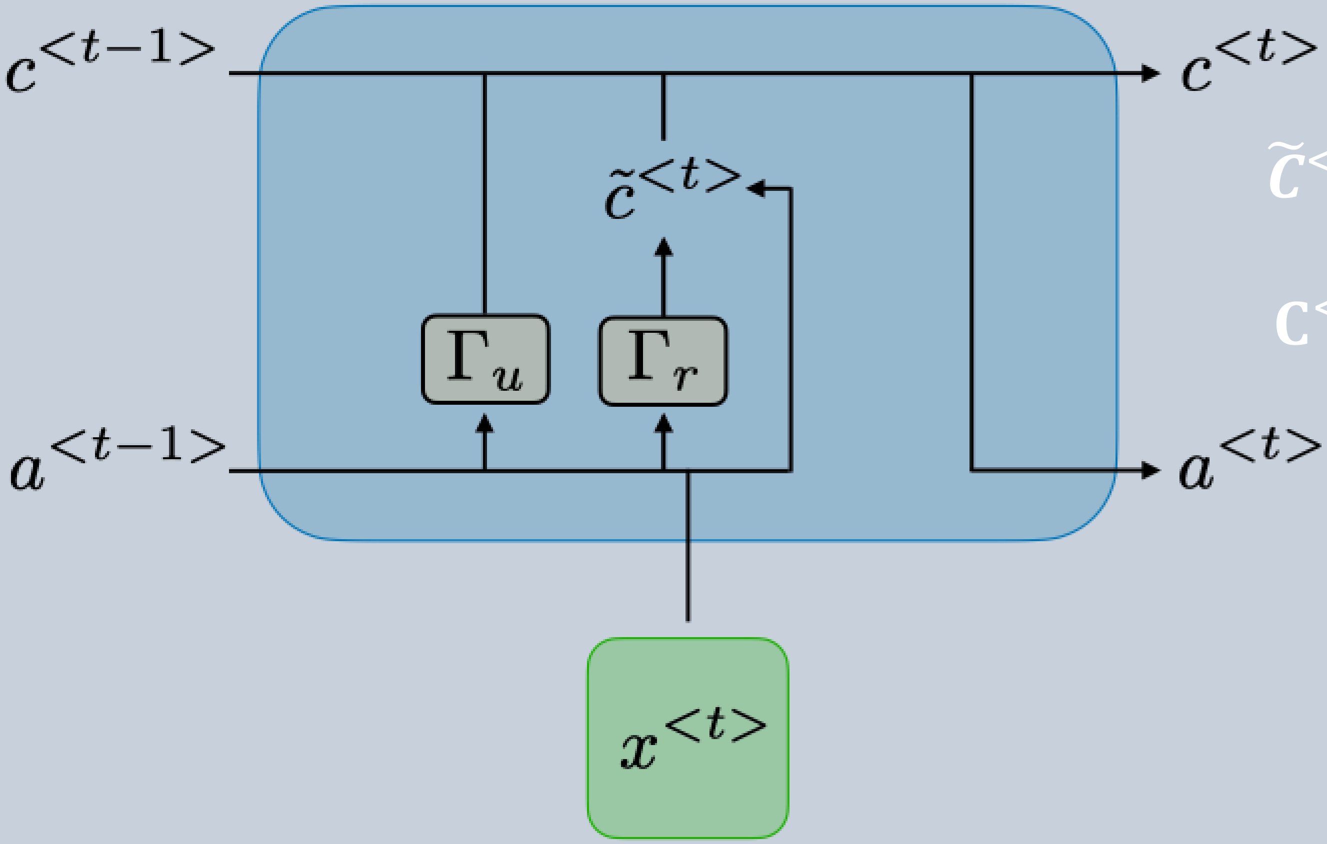
4-6. LSTM (Long Short-Term Memory)



Gradient의 흐름이 방해를 받지 않음

4. Handling Long Term Dependencies

4-6. GRU (Gates Recurrent Unit)



$$\tilde{C}^{<t>} = \tanh\left(W_c\left(\tau_r \cdot a^{<t-1>}, x^t\right) + b_c\right)$$

$$c^{<t>} = \tau_u \cdot \tilde{C}^{<t>} + (1 - \tau_u) \cdot c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

감사합니다