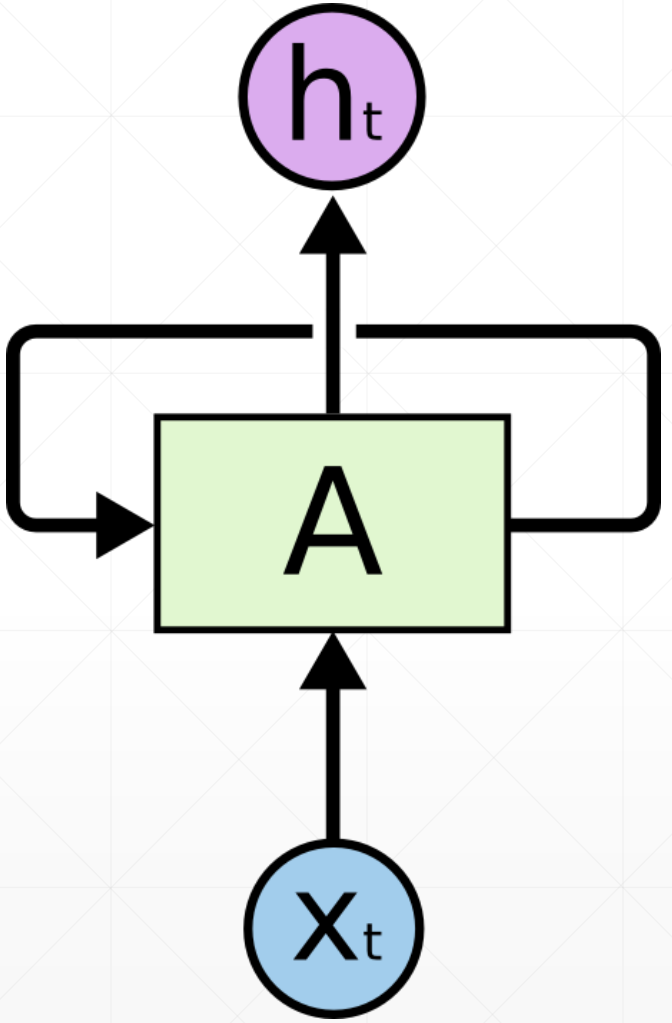
Recurrent Neural Networks (RNN, LSTM, GRU)

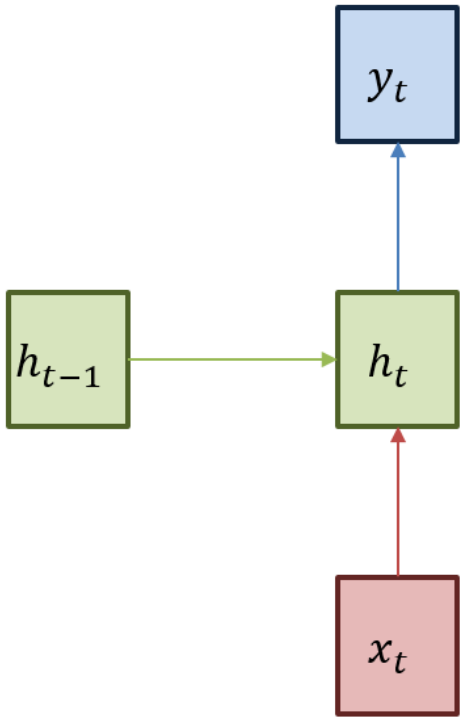
서상우

RNN

- RNN은 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류.
- 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델
- 시퀀스 길이에 관계없이 인풋과 아웃풋을 받아들일 수 있는 네트워크 구조이기 때문에 필요에 따라 다양하고 유연하게 구조



RNN : 수식



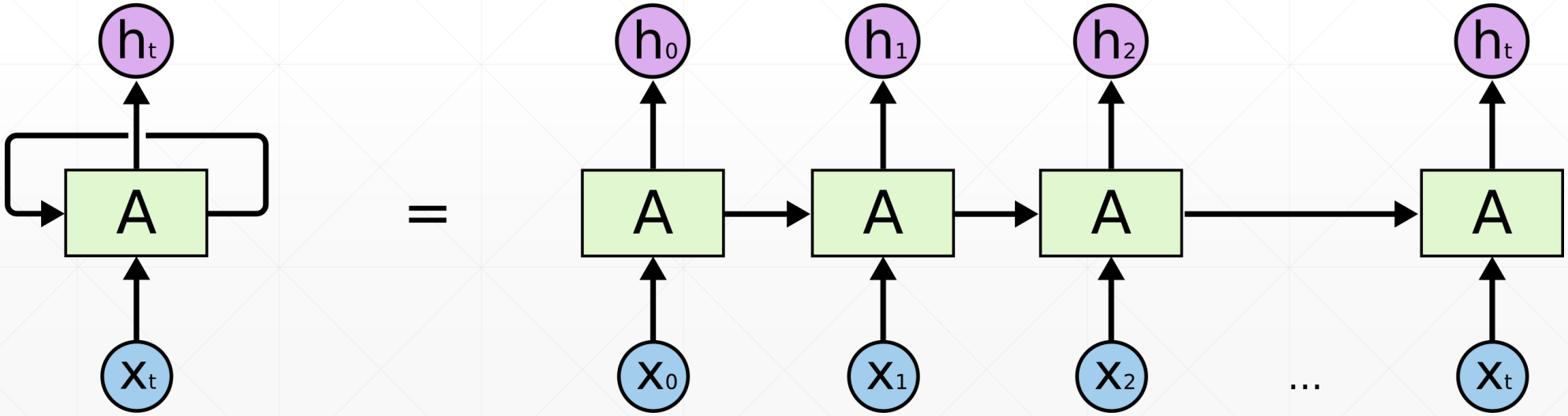
$$y_t = W_{hy} h_t + b_y$$

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

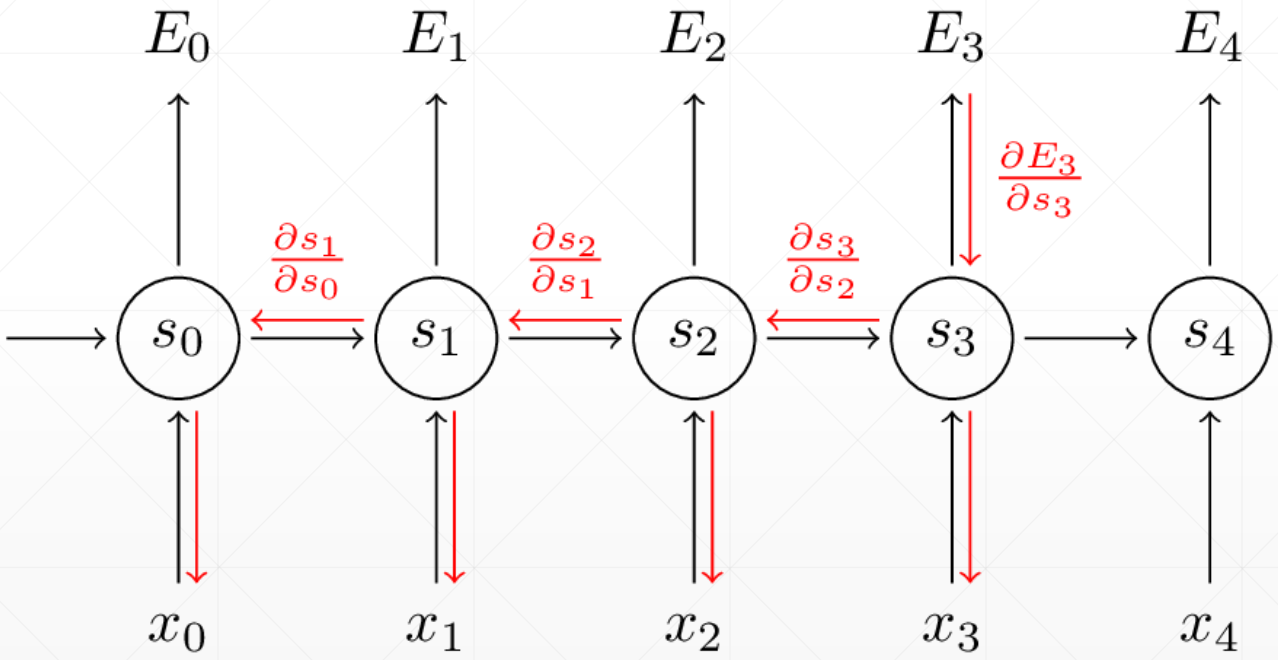
- 두 개의 인풋
  - x_t, h_{t-1}
- 세 개의 웨이트 파라미터
  - W_{hh}, W_{xh}, W_{hy}
- 하나의 아웃풋
  - y_t

RNN

- 그림을 펼쳐서 보면 오른쪽과 같이 각 타임 스텝별로 볼 수도 있다.
- 각 타임 스텝별로 W_{hh}, W_{xh}, W_{hy}는 모두 공유한다!



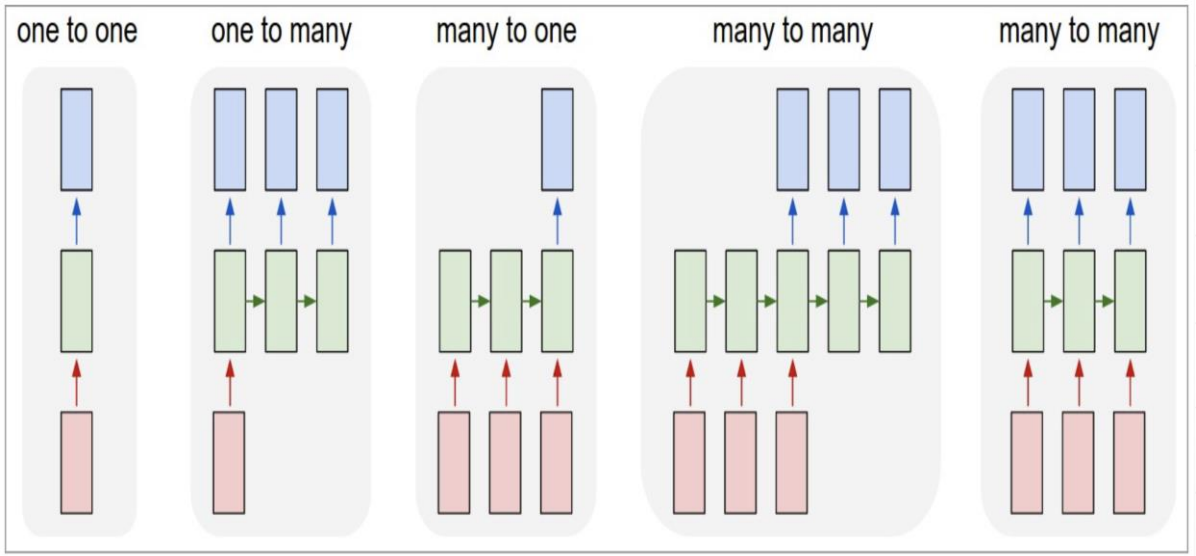
Backpropagation Through Time (BPTT)



- E3에서 시작된 오류가 시간을 거슬러 가며 전파.
- 또한 모든 시간에서의 웨이트는 공유가 되기 때문에 각 시간 별로의 오류를 합하면 최종적인 오류가 나옴.

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

RNN의 종류 & 응용 기술



1. RNN 아님;;

2. 고정크기 입력 & 시퀀스 출력.
   예) 이미지를 입력해서 이미지에 대한 설명을 문장으로 출력하는 이미지 캡션 생성

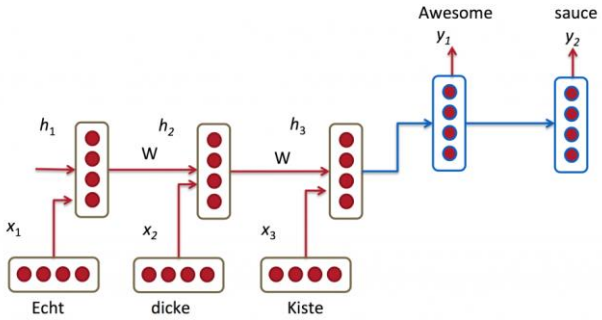
3. 시퀀스 입력 & 고정크기 출력.
   예) 문장을 입력해서 긍정 및 부정 정도를 출력하는 감성 분석기

4. 시퀀스 입력 & 시퀀스 출력.
   예) 영어를 한국으로 번역하는 자동 번역기

5. 동기화된 시퀀스 입력 & 시퀀스 출력.
   예) 문장에서 다음에 나올 단어를 예측하는 언어 모델

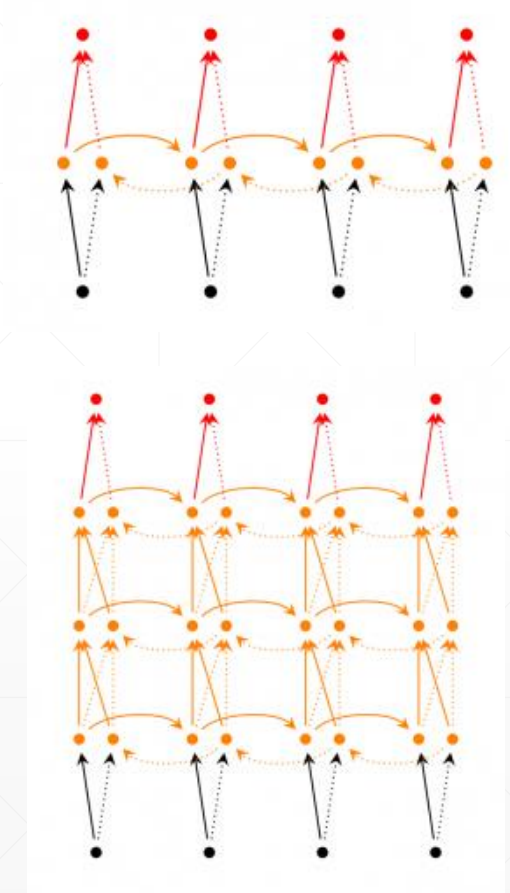
RNN의 종류 & 응용 기술

- NLP
  - Language Model (언어를 이해)
  - 감정 분석
  - 기계 번역
- 이미지 캡셔닝
- 음성 인식

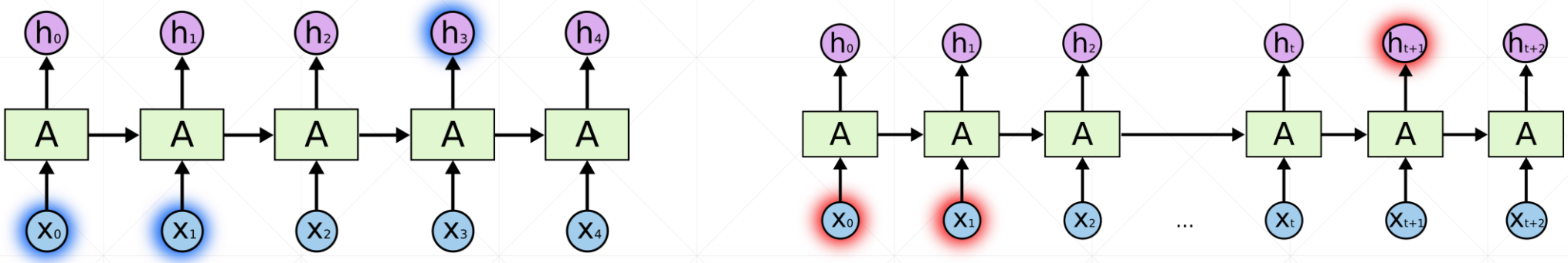


RNN의 종류 & 응용 기술

- Bidirectional RNN
  - 문장으로 치면 양쪽에서 오는 Context를 모두 알 수 있음.
  - 기본적으로 요즘 모델은 Bidirectional한 경우가 대다수
- Deep (Bidirectional) RNN
  - RNN을 여러 층으로 쌓음.
  - 보통 2~3개 층 이상은 잘 쌓지 않음.



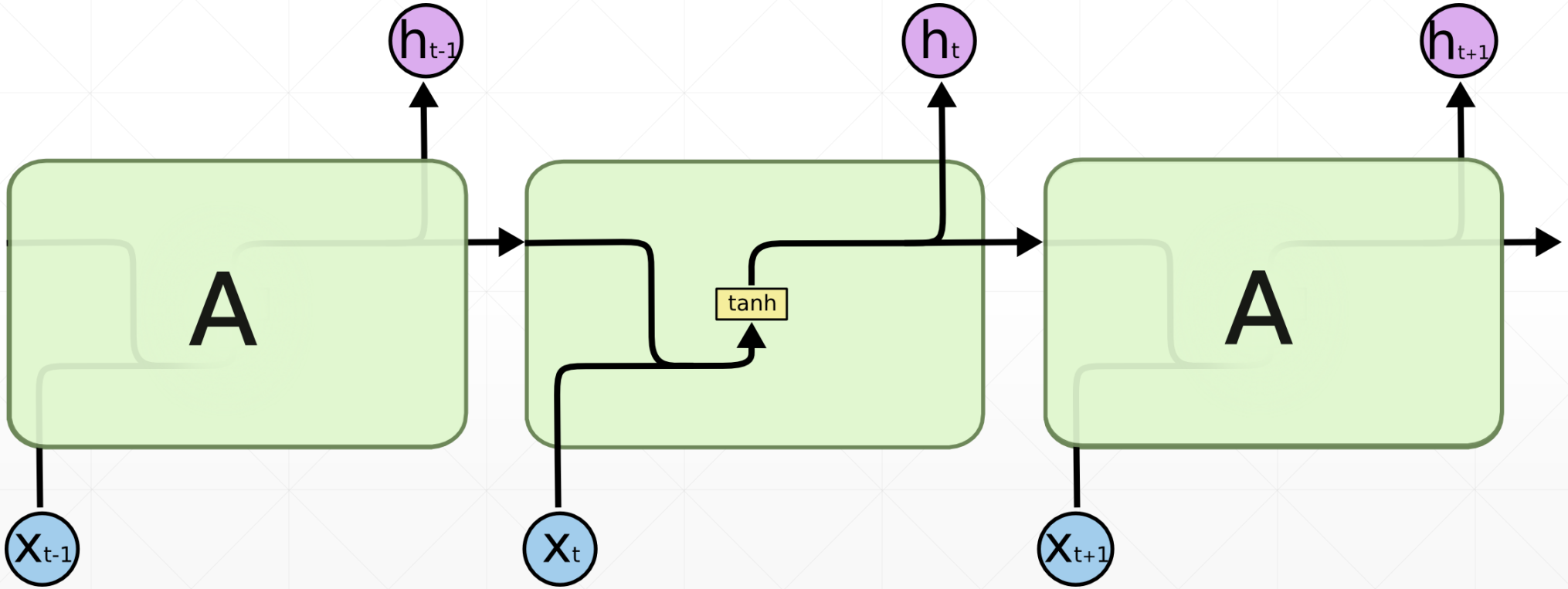
RNN : long-term dependency problem



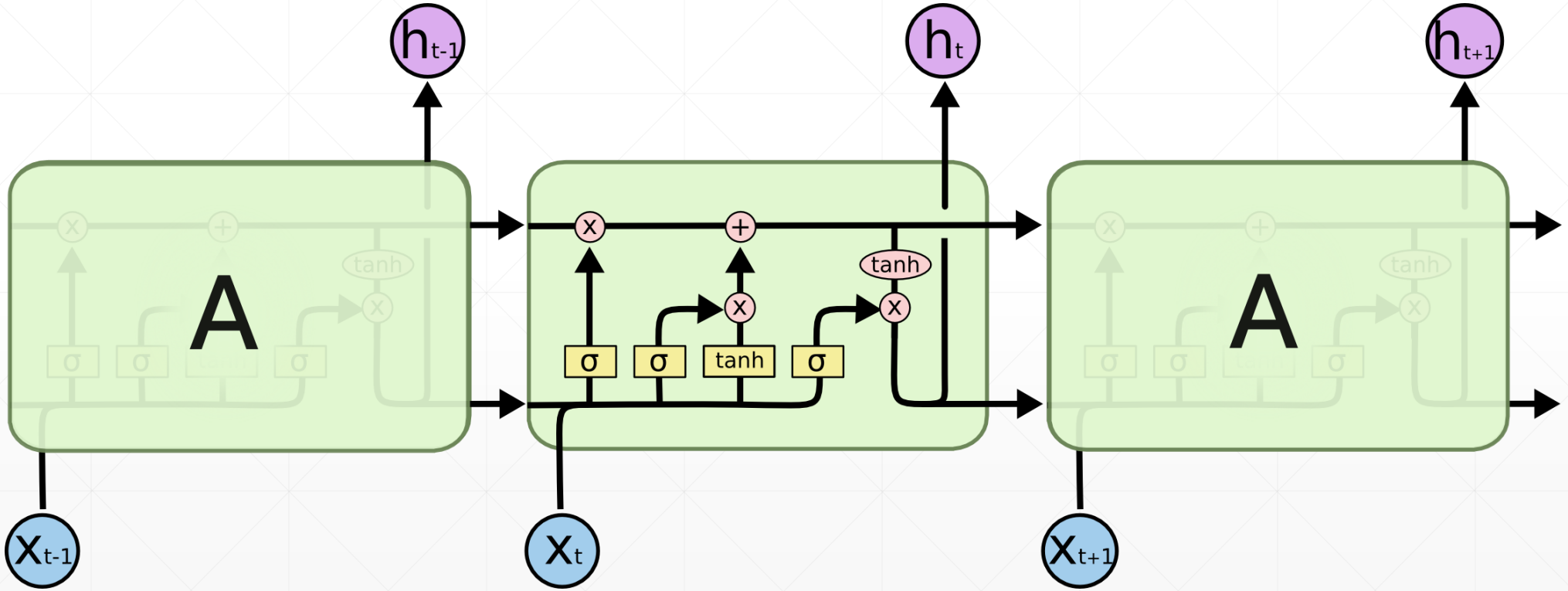
RNN을 펼치게(unfold)되면 매우 깊은 네트워크가 될 것이며, 이러한 네트워크는 **그래디언트 소실 및 폭주**(vanishing & exploding gradient) 문제가 발생할 가능성이 크다.

타임 스텝을 일정 구간(보통 5-steps)으로 나누어 역전파(backprop)를 계산하여, 전체 역전파로 근사시키는 방법인 **Truncated BPTT**를 대안
(하지만 long-term dependency problem를 해결 불가능)

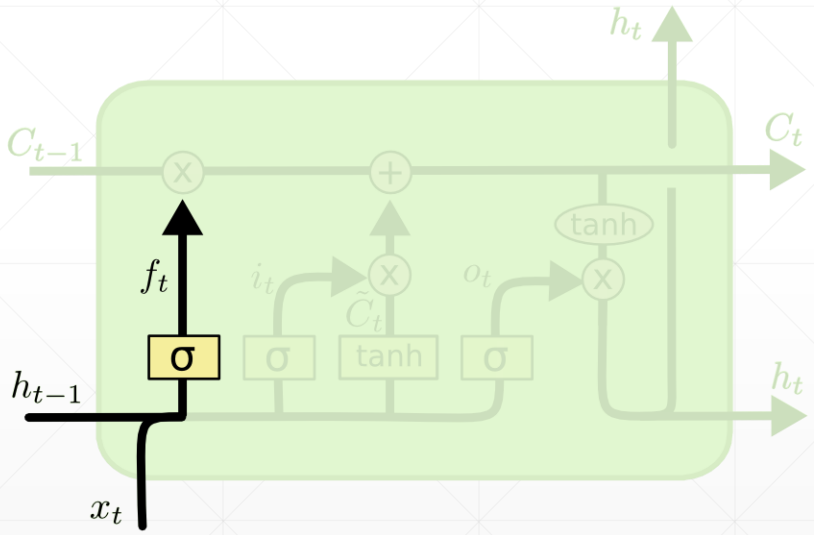
RNN



LSTM!



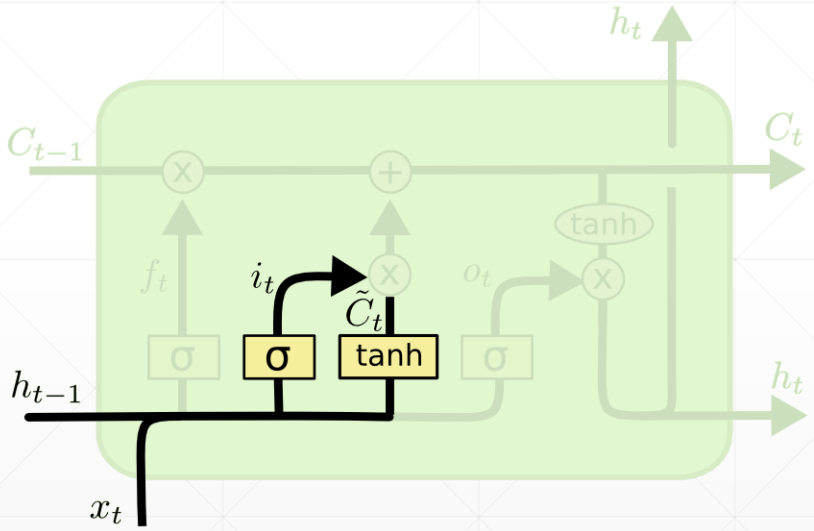
LSTM의 forget gate layer



현재의 계산으로는 단순히 linear한 연산의 결과이며 forget gate의 의미는 이 이후에 생김

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

LSTM의 input gate layer

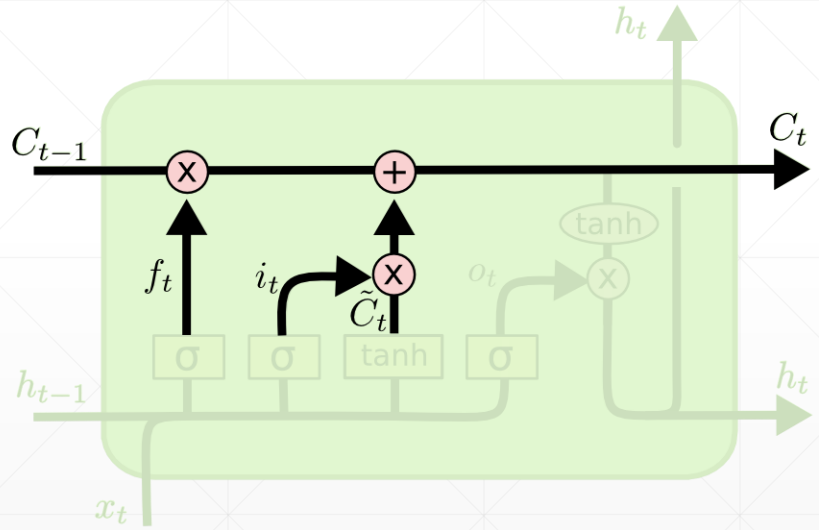


Input gate의 의미 또한 차후이며
$\tilde{C}_t$ 는 현재의 의미를 말함
(RNN 때는 이 벡터를 히든으로 사용함.)

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

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

LSTM의 cell state 업데이트

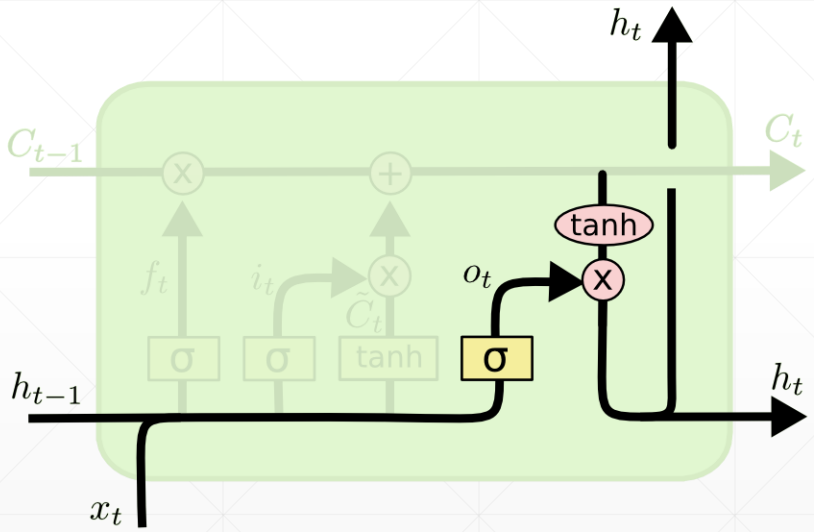


$\tilde{C}_t$ 에 i_t (정보를 얼마나 입력시키는가)를 곱하며 이전 셀인 C_{t-1}에 f_t(정보를 얼마나 잊을까)를 곱하여 현재 셀 state를 만듬.

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

(*은 element-wise 곱셈을 의미한다.)

LSTM의 output gate layer



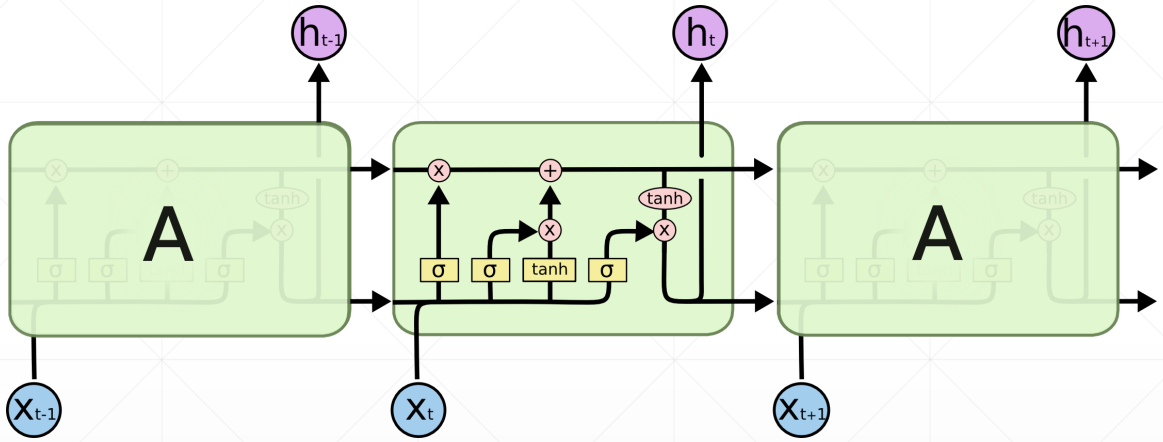
C_t에다가 o_t만큼을 곱하여 최종적인 히든 벡터를 생성

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

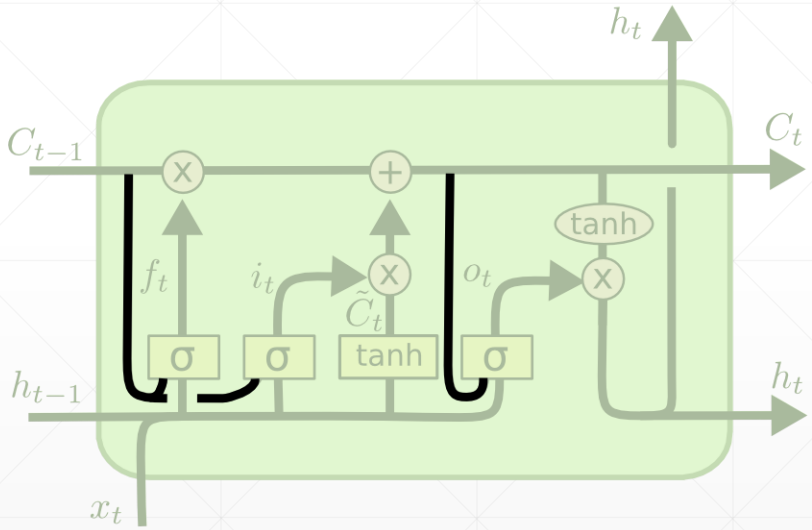
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM!

- LSTM은 어떻게 long-term dependency problem을 해결하는가?
  - Cell state의 유무!
  - 직접적으로 이전 셀과 현재 셀의 값을 더하기 때문에 이전의 셀에 대한 기억을 더 잘 할 수 있다.
- **Vanishing Gradient 또한 해결 가능**



LSTM 변형 : peephole connection



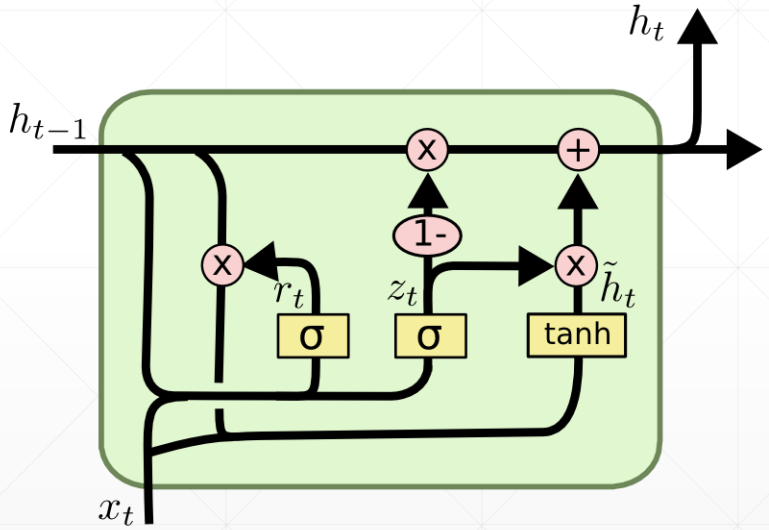
gate controller에 이전 타임스텝의 cell state가 입력으로 추가되며, 좀 더 많은 맥락(context)를 인식

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

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

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

LSTM 변형 : GRU



기존의 forget, input gate를 하나로 해결

웨이트 파라미터가 더 적어진다는 장점

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

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

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

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