Idea Factory Intensive Program #2

딥러닝 홀로서기

#28

이론강의/PyTorch실습/코드리뷰

딥러닝(Deep Learning)에 관심이 있는 학생 발굴을 통한
딥러닝의 이론적 배경 강의 및 오픈소스 딥러닝 라이브러리 PyTorch를 활용한 실습

Acknowledgement

Sung Kim`s 모두를 위한 머신러닝/딥러닝 강의

- https://hunkim.github.io/ml/
- https://www.youtube.com/playlist?list=PLlMkM4tgfjnLSOjrEJN31gZATbcj_MpUm

Andrew Ng`s and other ML tutorials

- https://class.coursera.org/ml-003/lecture
- http://www.holehouse.org/mlclass/ (note)
- Deep Learning Tutorial
- Andrej Karpathy's Youtube channel

WooYeon Kim & SeongOk Ryu`s KAIST CH485 Artificial Intelligence and Chemistry

- https://github.com/SeongokRyu/CH485---Artificial-Intelligence-and-Chemistry

SungJu Hwang`s KAIST CS492 Deep Learning Course Material

Many insightful articles, blog posts and Youtube channels

Facebook community

- Tensorflow KR (https://www.facebook.com/groups/TensorFlowKR/)
- Pytorch KR (https://www.facebook.com/groups/PyTorchKR/)

Medium Channel and Writers

- Toward Data Science (https://towardsdatascience.com/)

Today`s Time Schedule

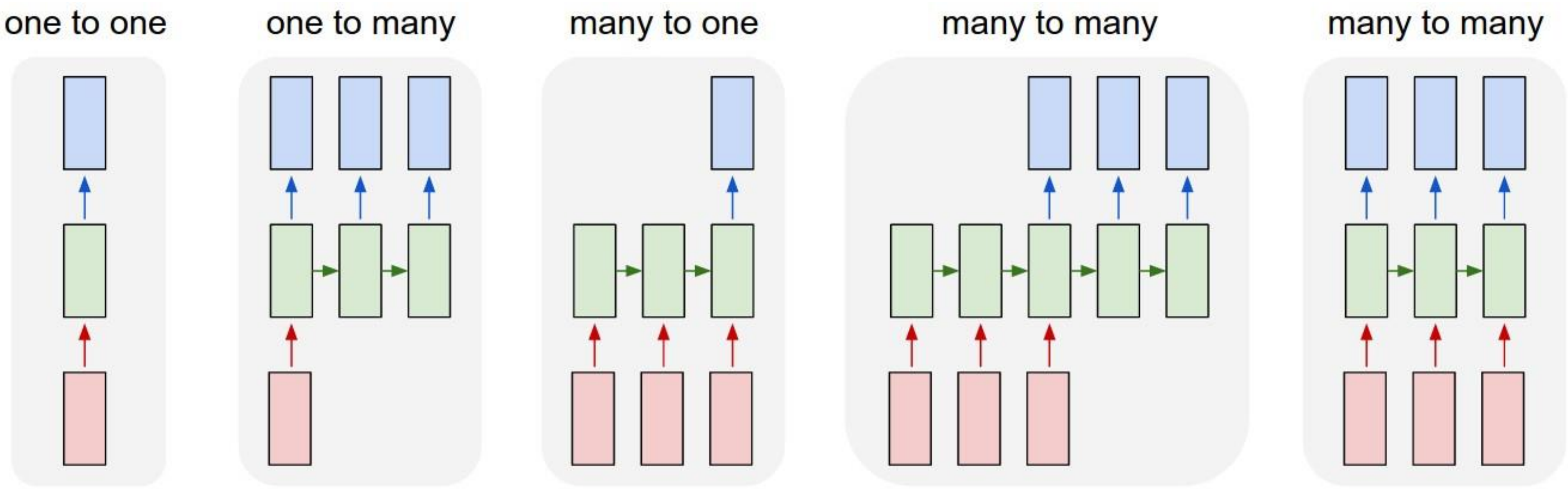
Advanced RNN Architecture (LSTM, GRU) — 30 mins

Predict Stock Price with LSTM — 2 hour

Fix Lab 8 – Learning Trigonometric Function with RNN — 1.5 hour

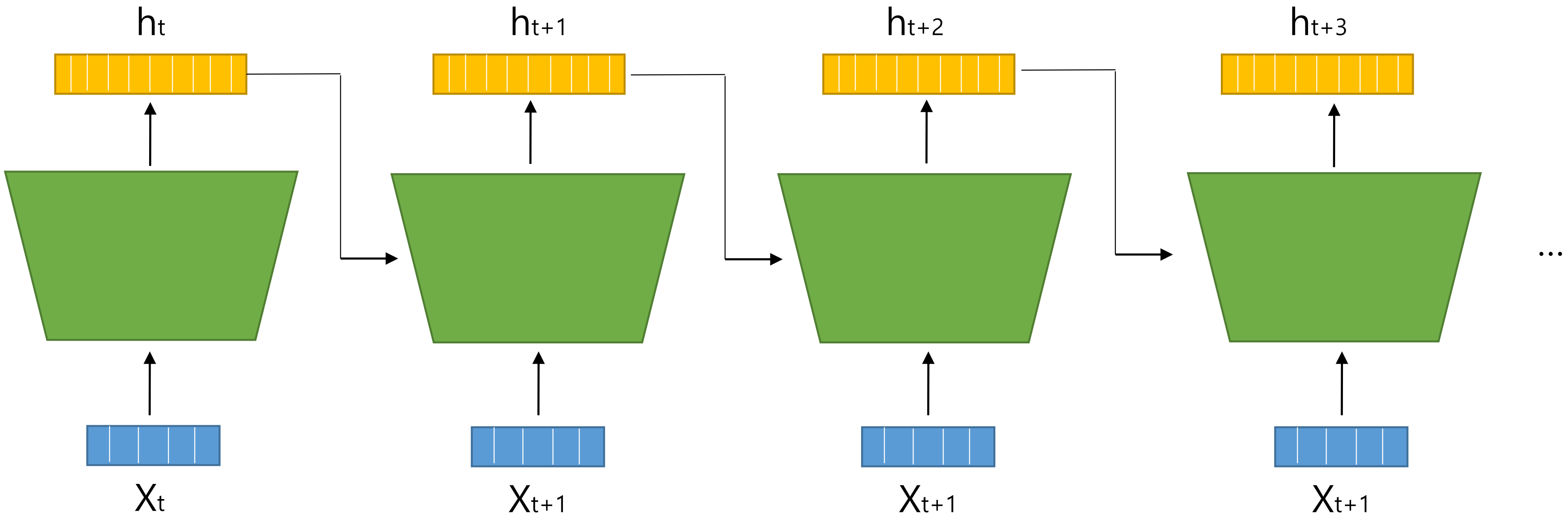
Review From Last Lecture

Review From Last Lecture

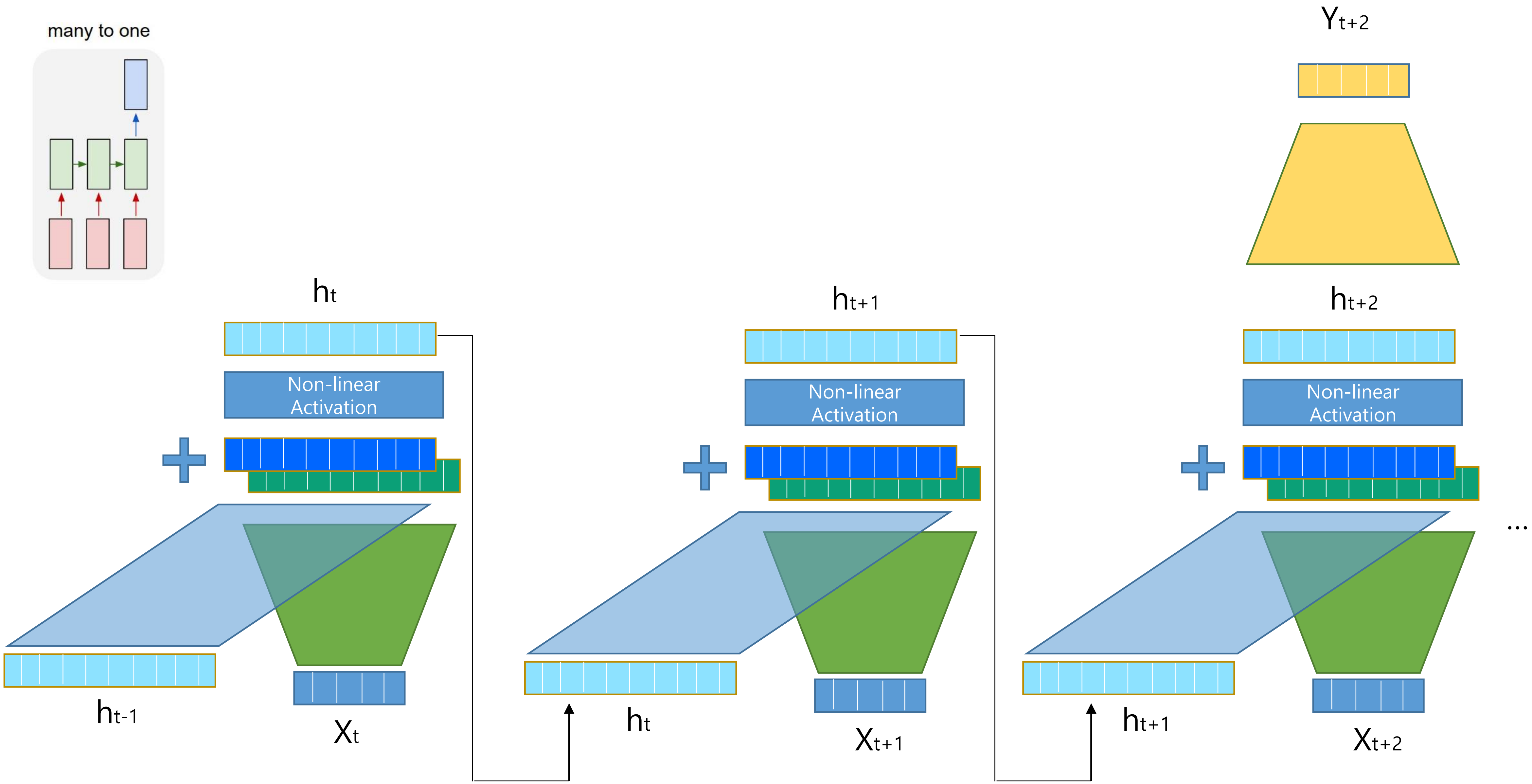


Review From Last Lecture

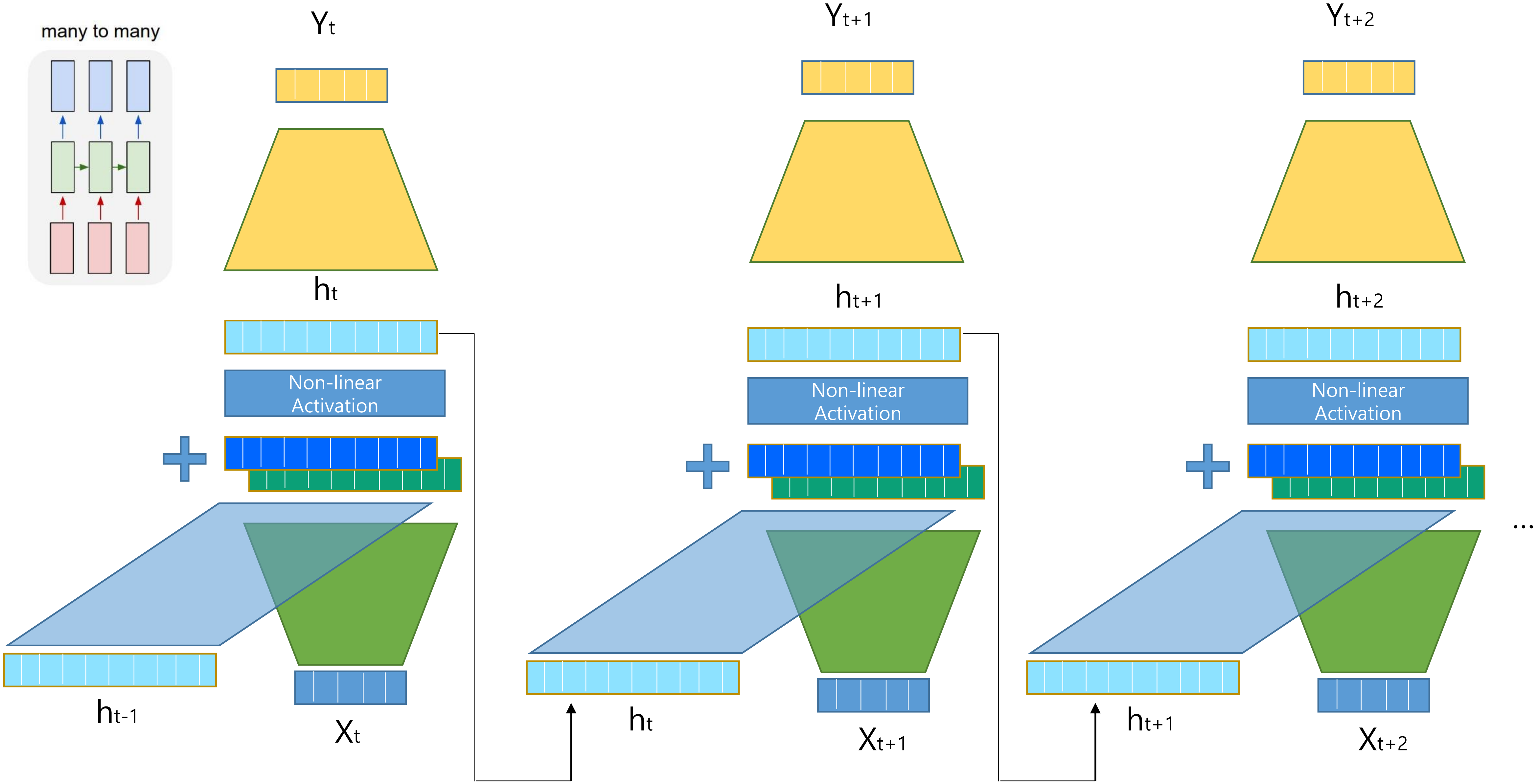
Process both new inputs and model output of previous input!



Review From Last Lecture

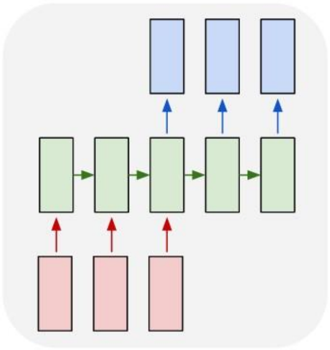


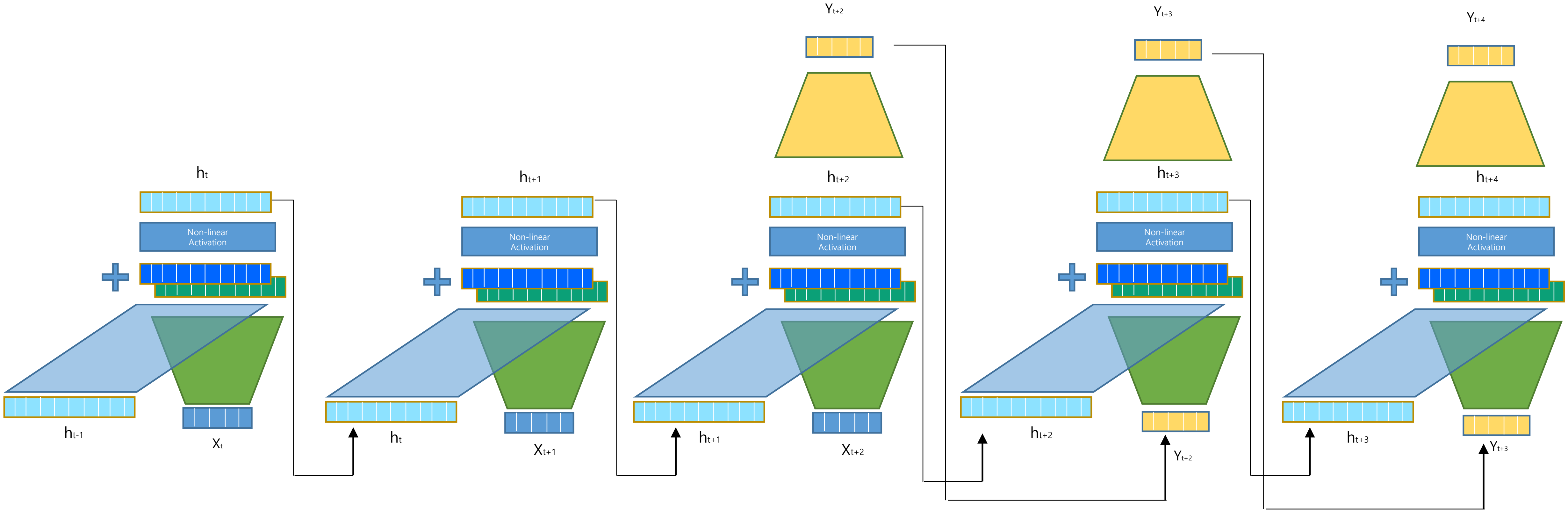
Review From Last Lecture



Review From Last Lecture

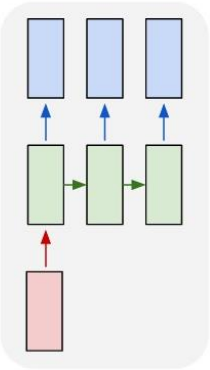
many to many

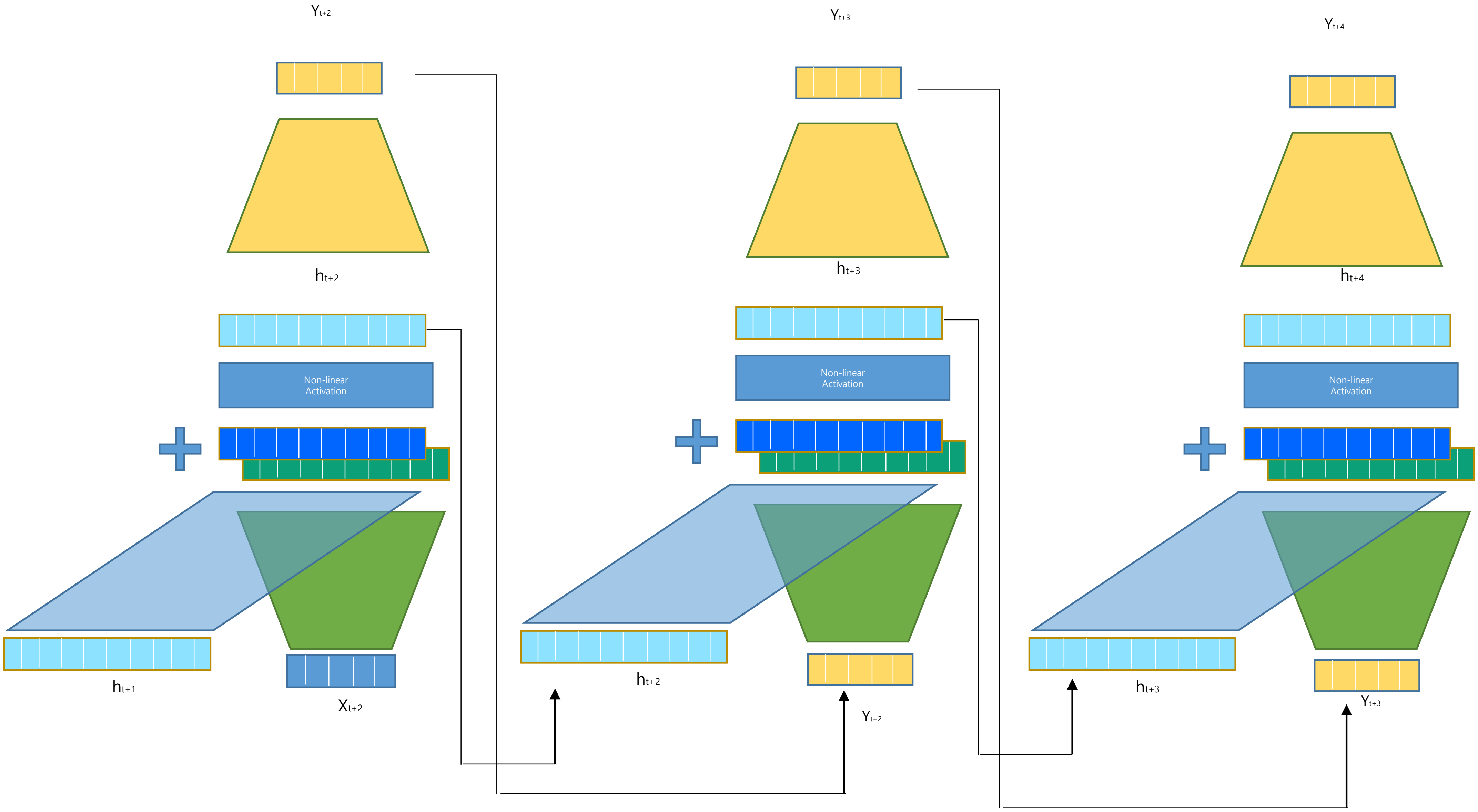




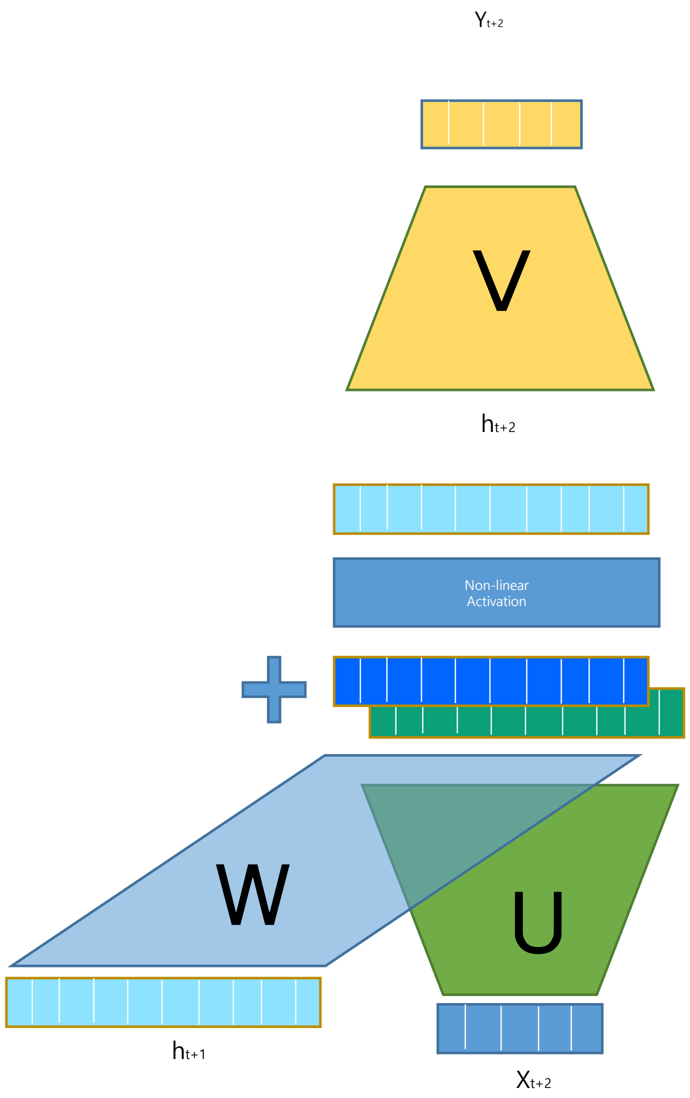
Review From Last Lecture

one to many





Review From Last Lecture



f(x) = tanh(x)

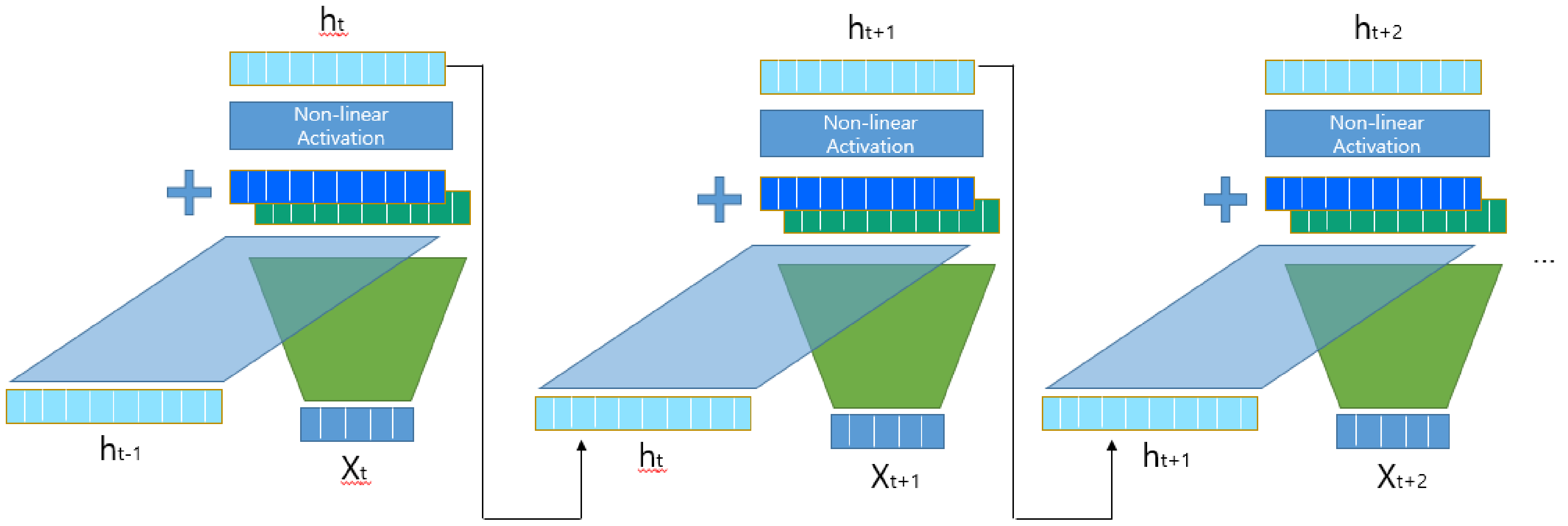
$$h_t = f(Ux_t + Wh_{t-1})$$

$$y_t = f(Vh_t)$$

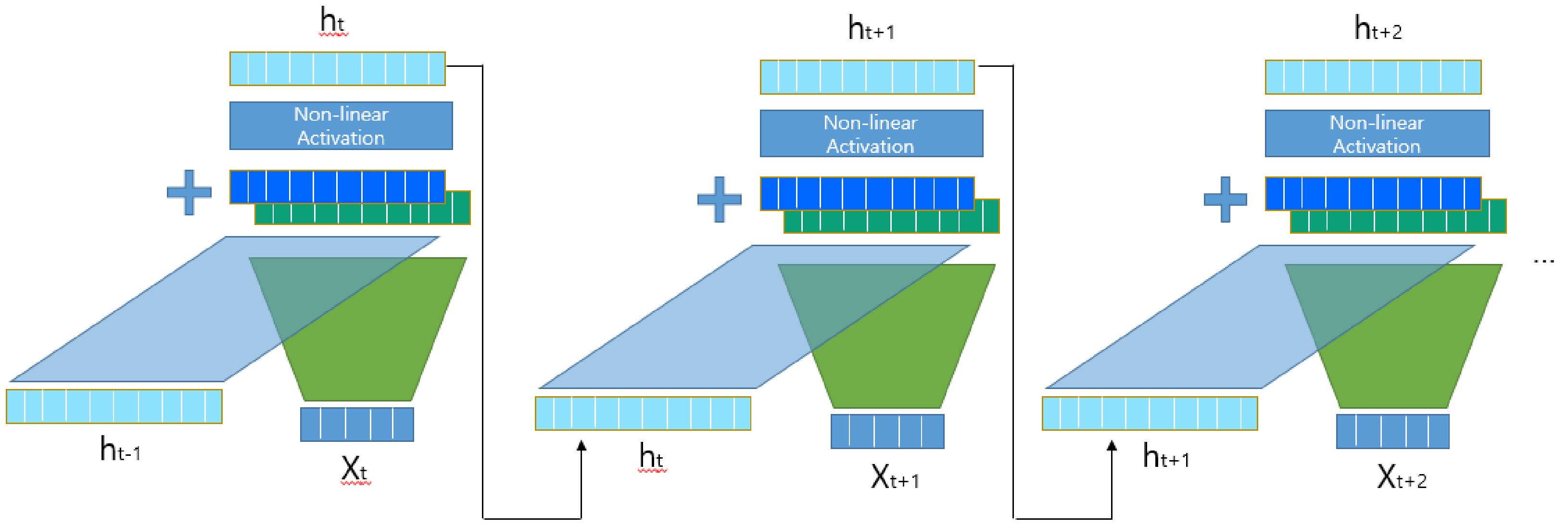
f(x) = x

Vanishing Gradient Problem

Vanishing Gradient Problem

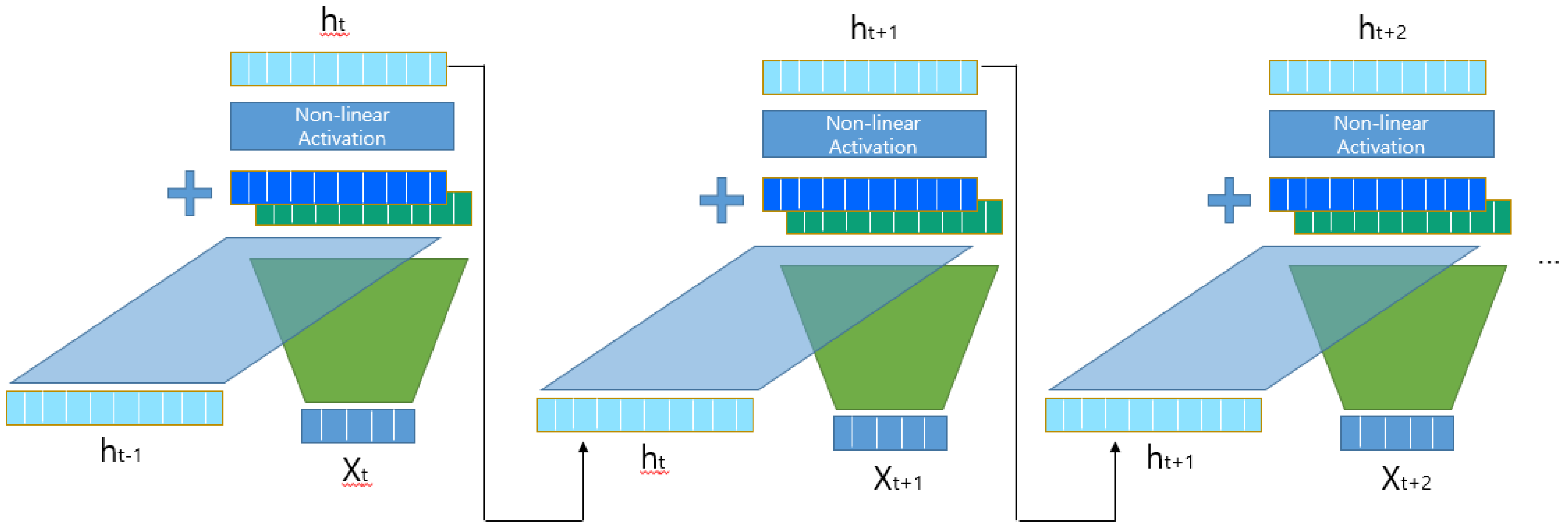


Vanishing Gradient Problem



$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$

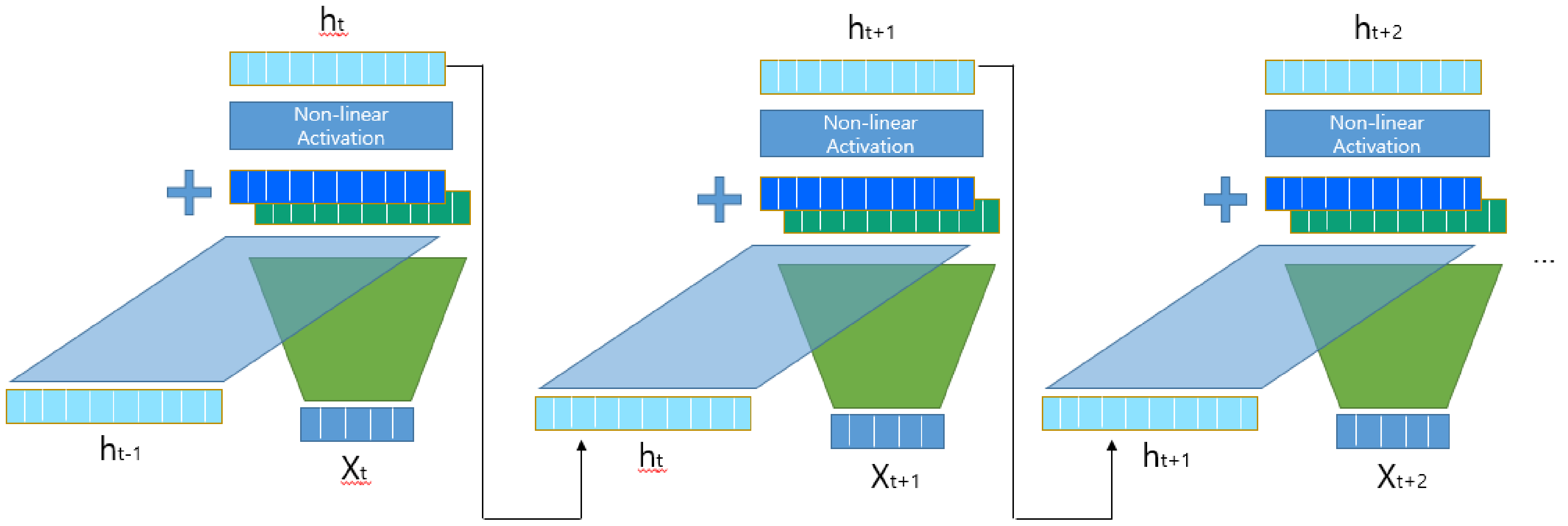
Vanishing Gradient Problem



$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$

$$h_{t-1} = tanh(W[h_{t-2}, x_{t-1}])$$

Vanishing Gradient Problem

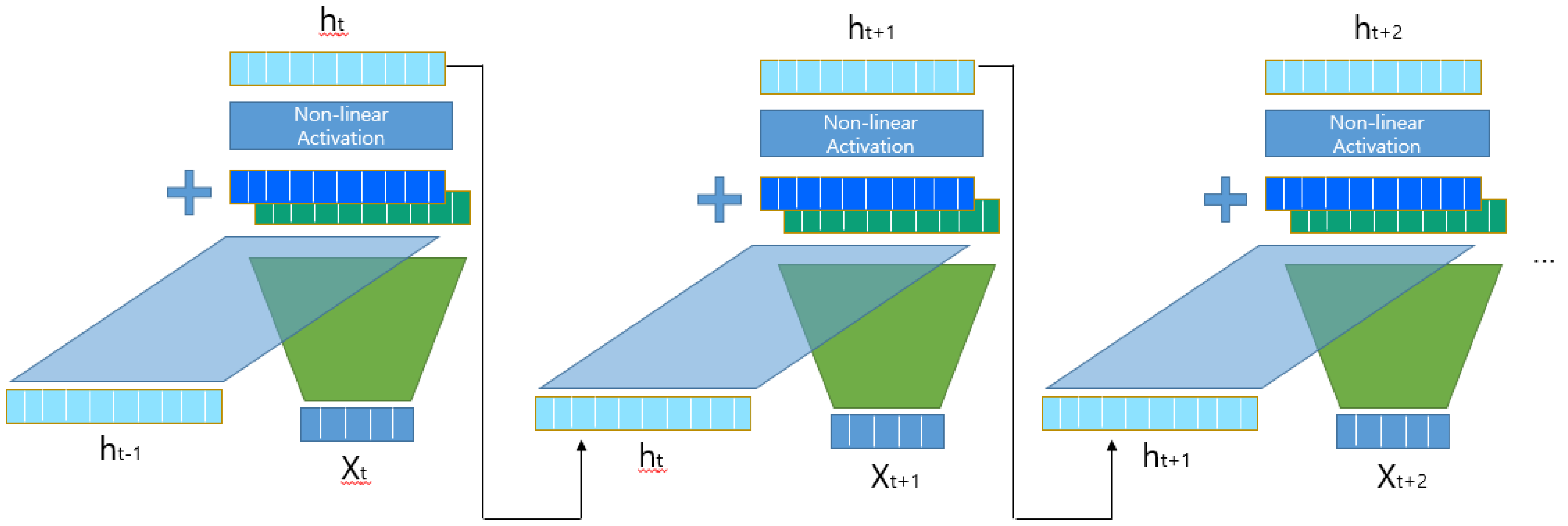


$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$

$$h_{t-1} = tanh(W[h_{t-2}, x_{t-1}])$$

$$h_t = tanh(W[h_{t-1}, x_t])$$

Vanishing Gradient Problem



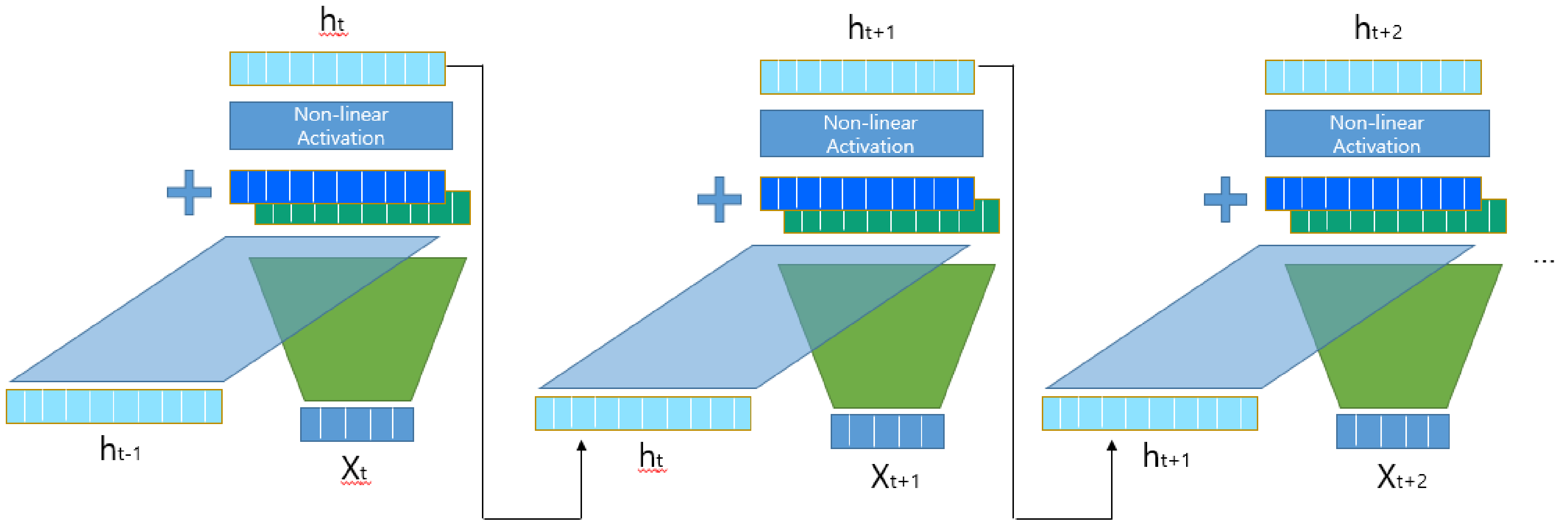
$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$

$$h_{t-1} = tanh(W[h_{t-2}, x_{t-1}])$$

$$h_t = tanh(W[h_{t-1}, x_t])$$

$$h_t = tanh(W[tanh(..tanh(..h_{t-3})), x_t])$$

Vanishing Gradient Problem



$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$

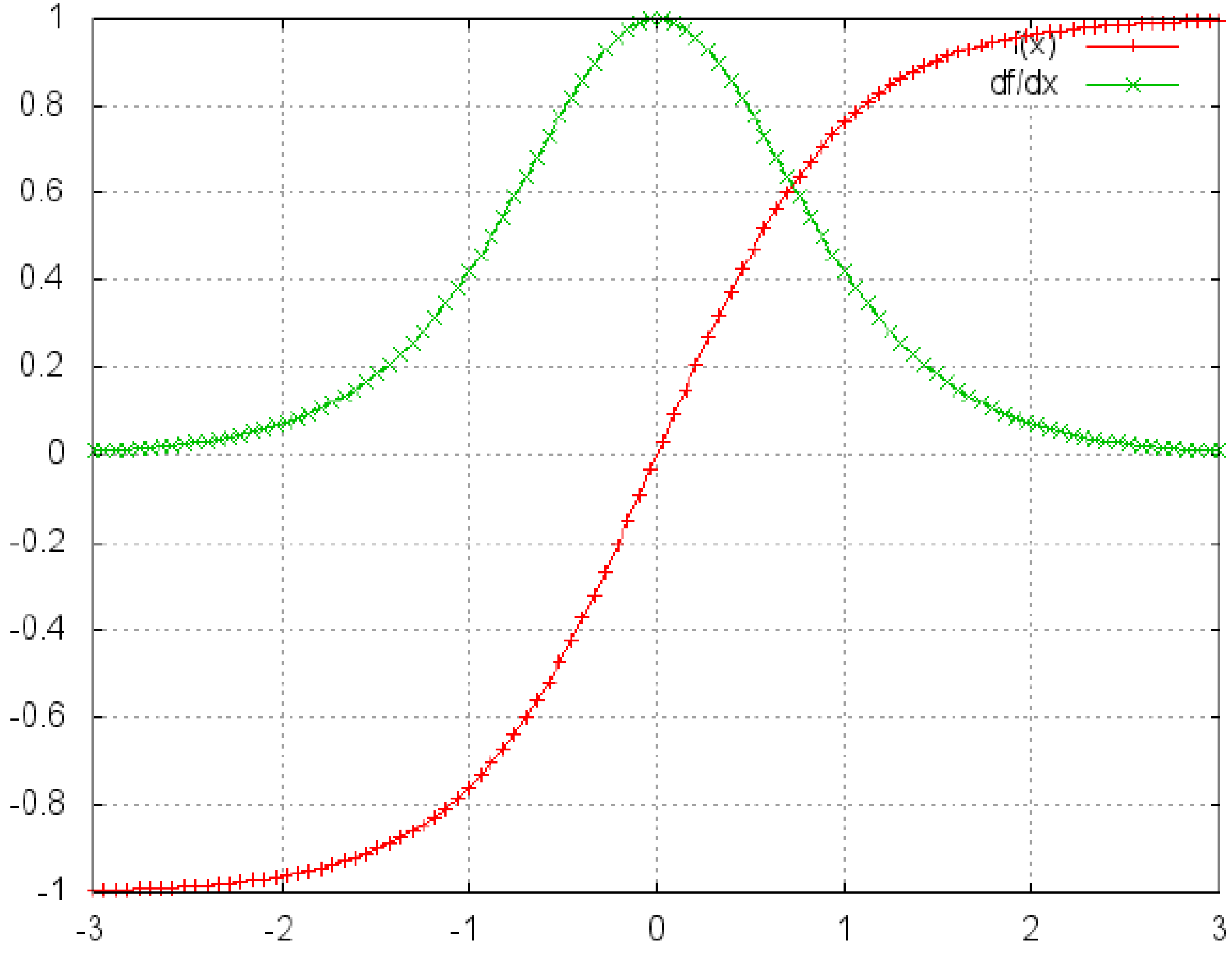
$$h_{t-1} = tanh(W[h_{t-2}, x_{t-1}])$$

$$h_t = tanh(W[h_{t-1}, x_t])$$

$$h_t = tanh(W[tanh(..tanh(..h_{t-3})), x_t])$$

So many tanh(x)!

Vanishing Gradient Problem



Graph of $tanh(x)$ (red) $\frac{d}{dx}tanh(x)$ (green)

Vanishing Gradient Problem

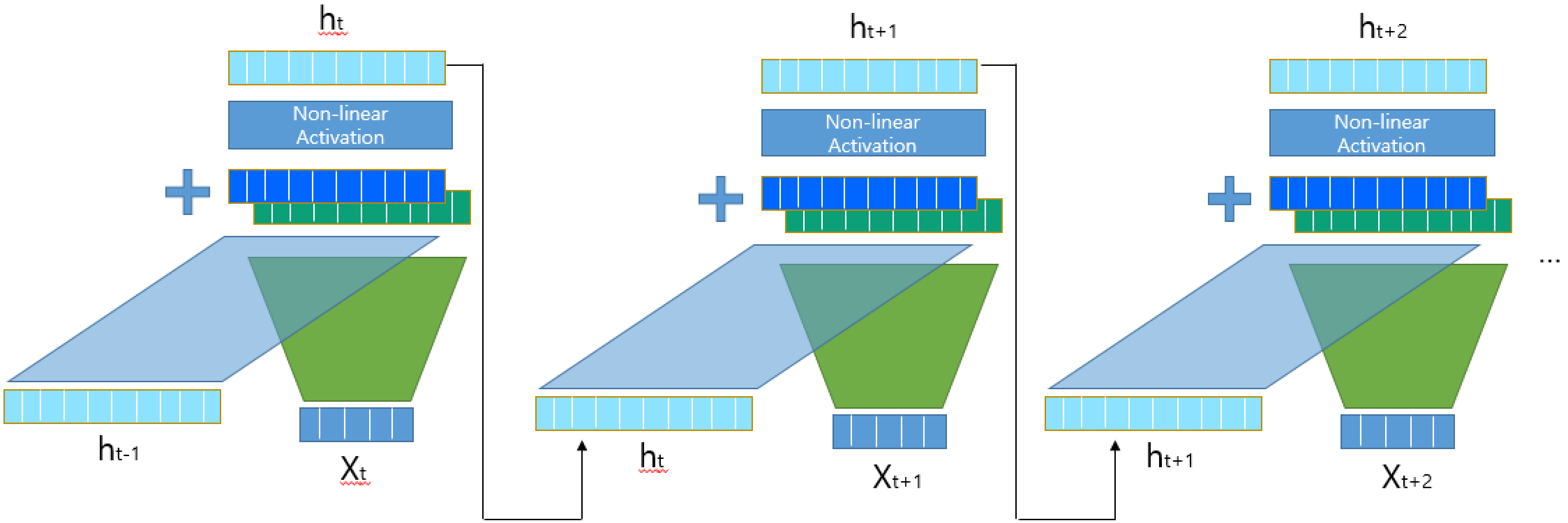
With long sequence, gradient could be vanished

Back-propagation could not be done properly

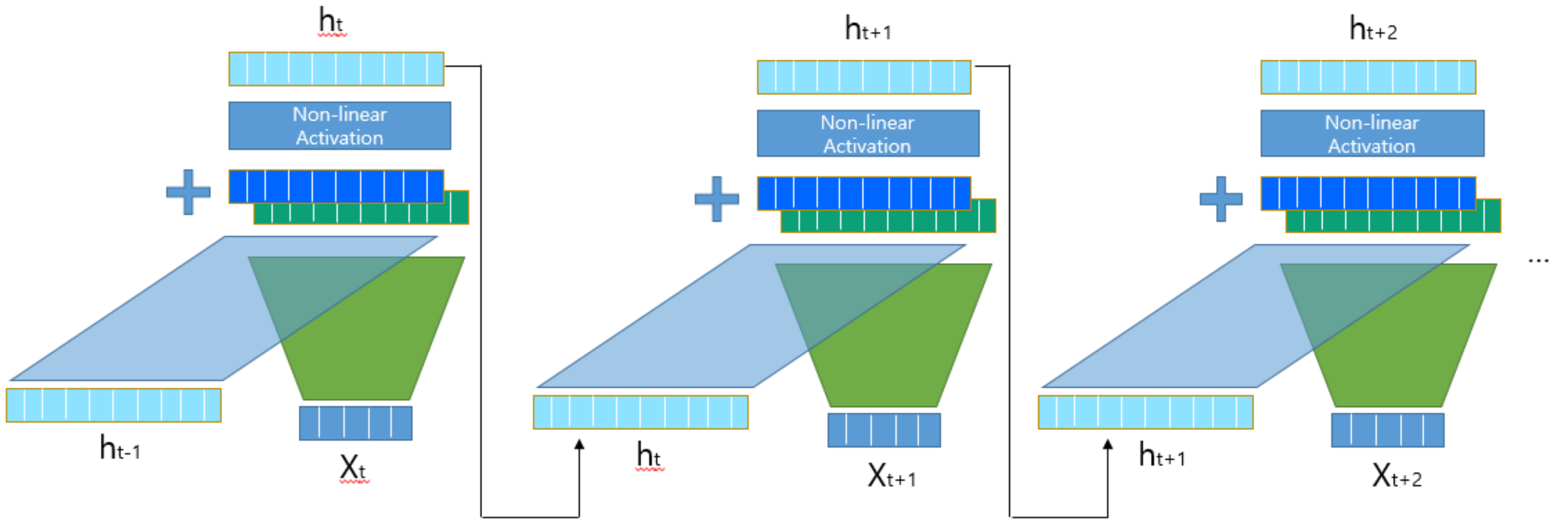
Vanilla RNN is weak to learn long sequence

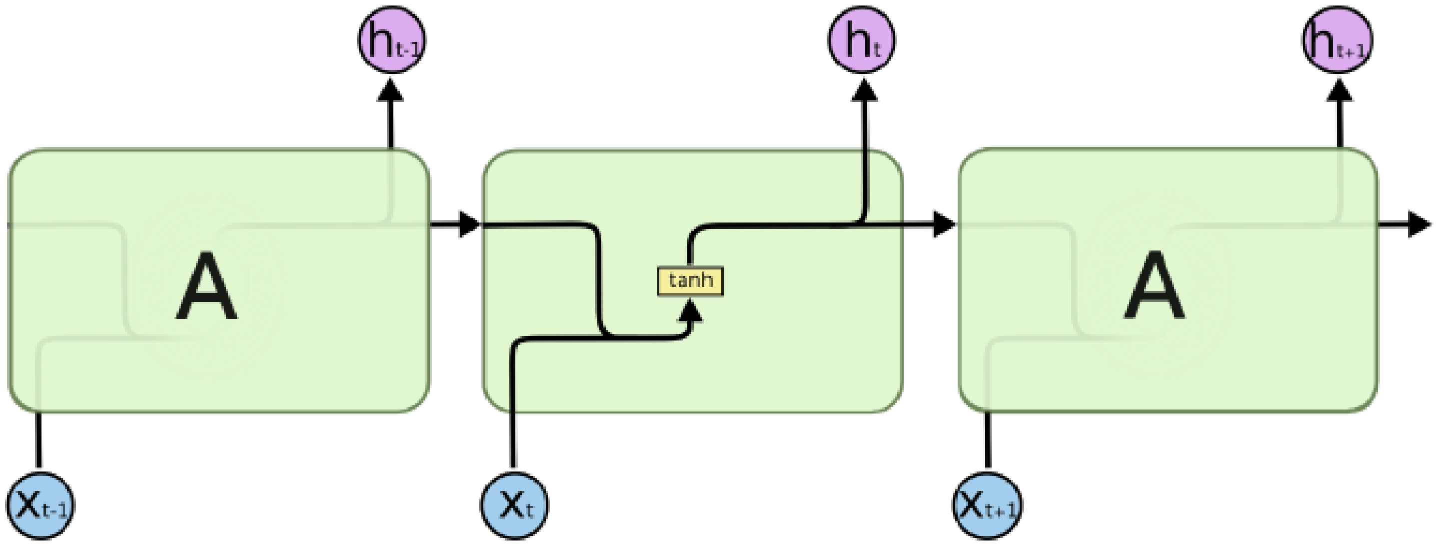
Long Short Term Memory Network

Long Short Term Memory Network



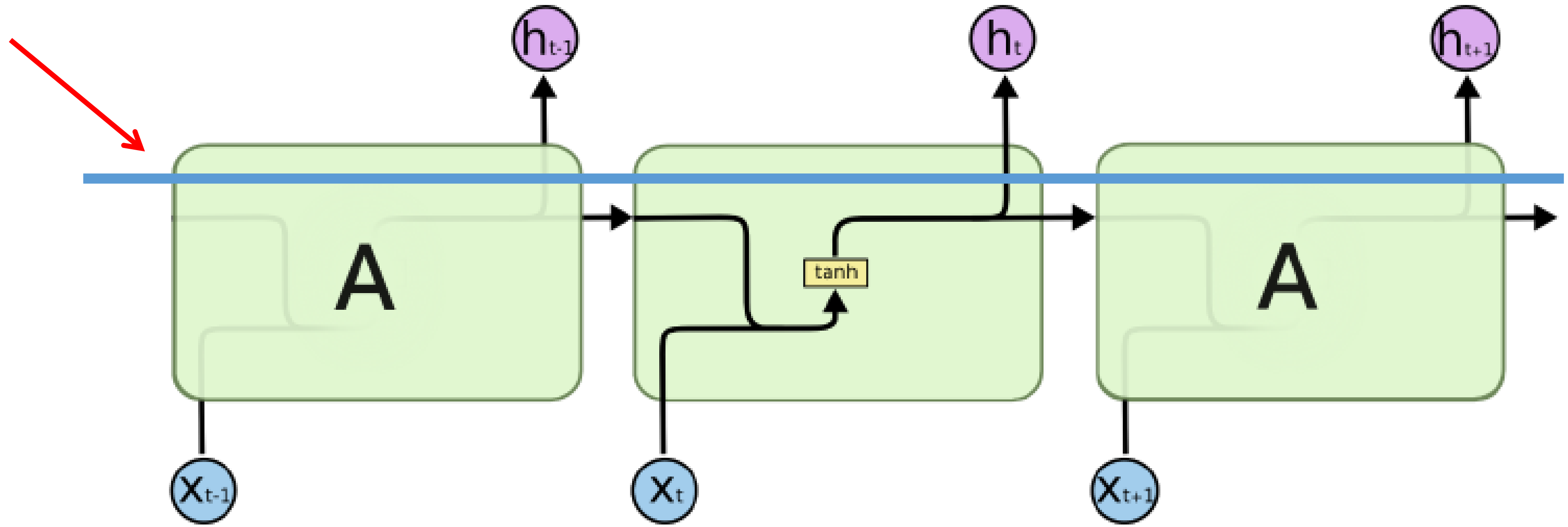
Long Short Term Memory Network





Long Short Term Memory Network

Add Information Flow?



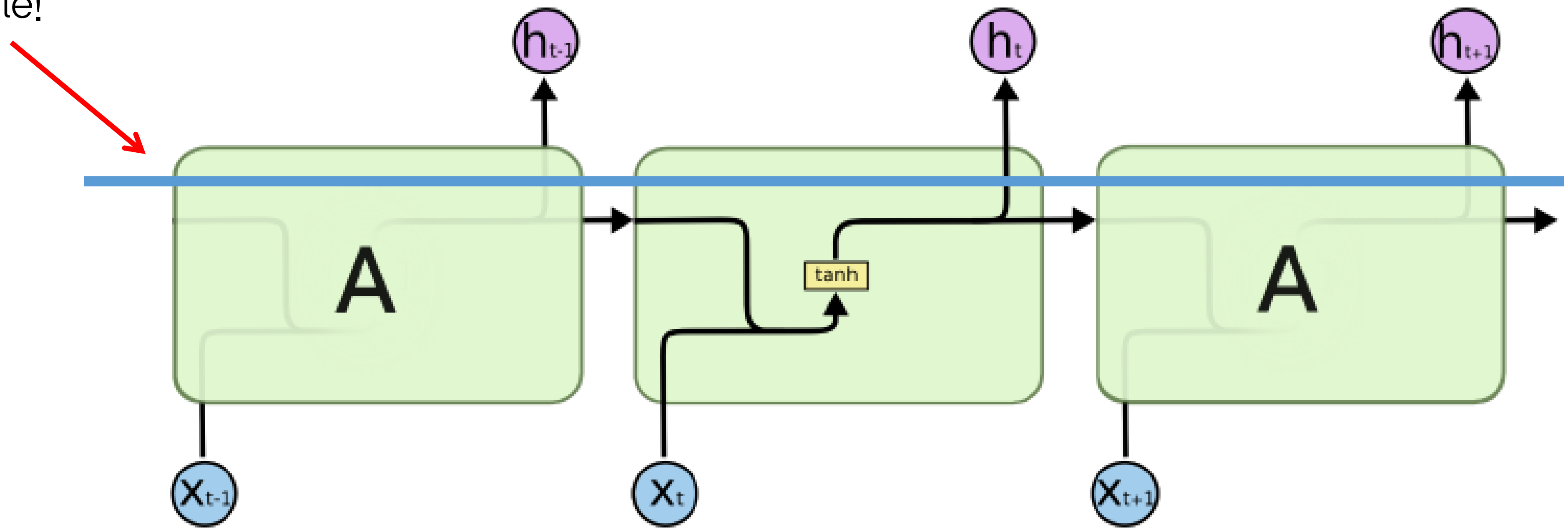
Standard RNN

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

Long Short Term Memory Network

Add Information Flow?

Cell State!



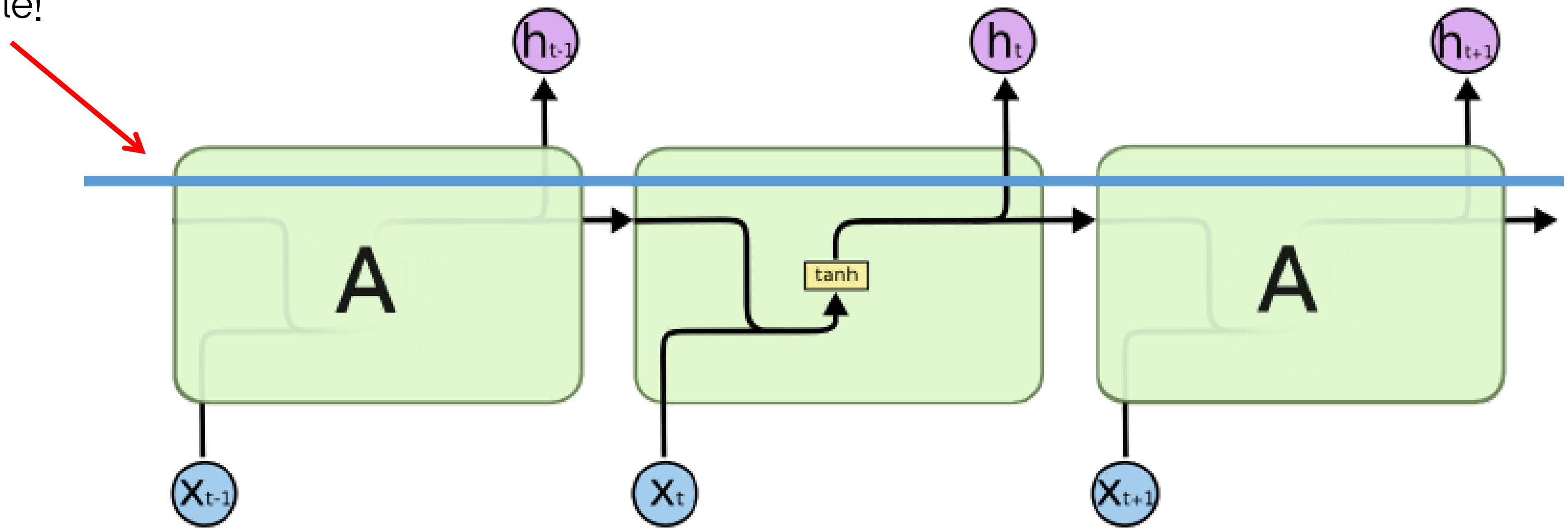
Standard RNN

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

Long Short Term Memory Network

Add Information Flow?

Cell State!



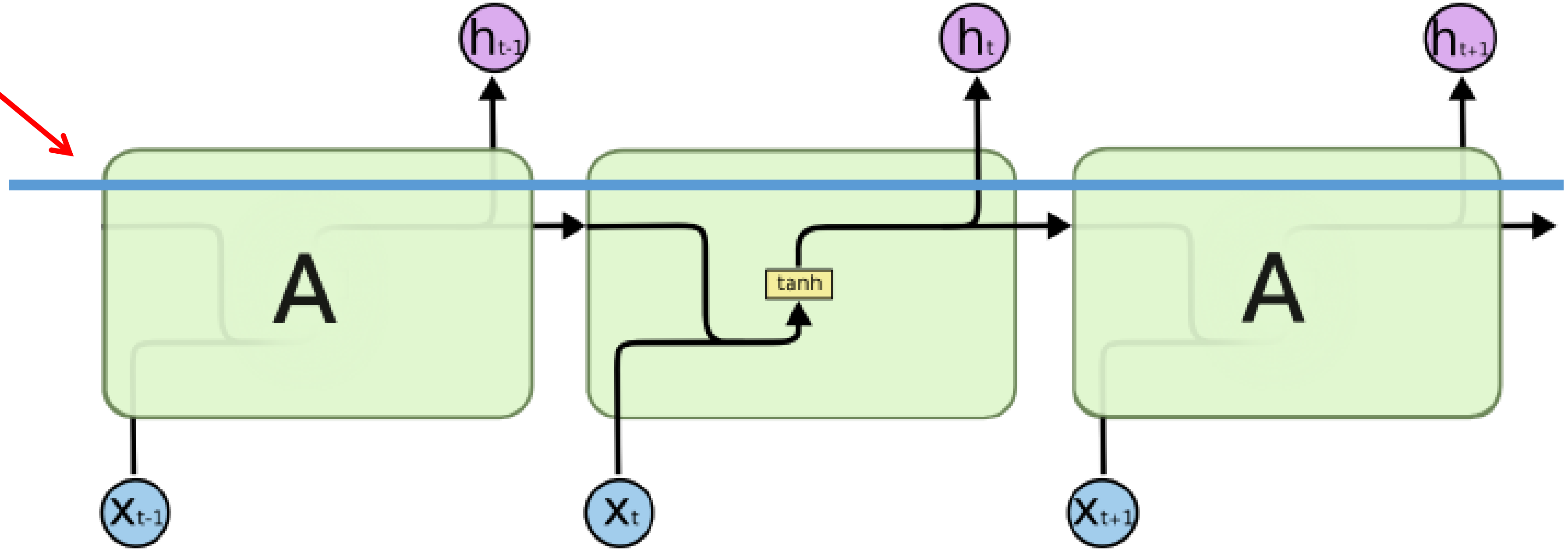
남길 건 남기고, 잊어버릴 건 잊어버리고, 새로 추가할 건 추가해서
Cell State에 중요한 정보만 계속 흘러가도록!

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

Long Short Term Memory Network

Add Information Flow?

Cell State!



남길 건 남기고, 잊어버릴 건 잊어버리고, 새로 추가할 건 추가해서
Cell State에 중요한 정보만 계속 흘러가도록!

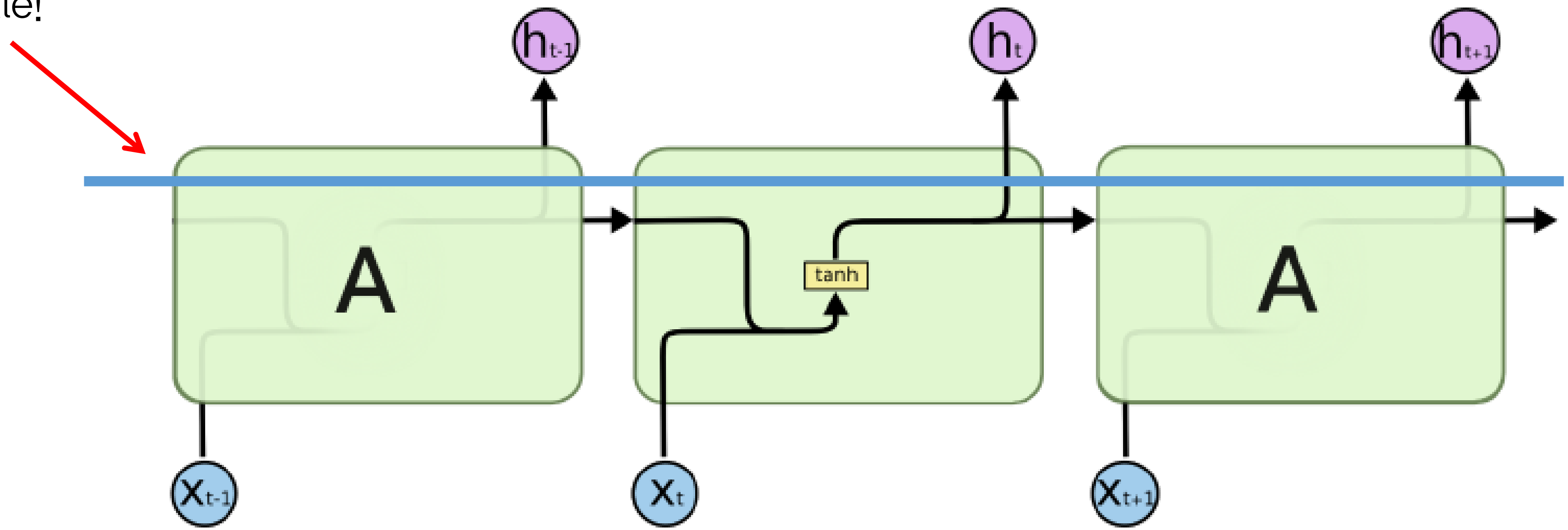
Hidden State는 Cell State를 적당히 가공해서 내보내자!

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

Long Short Term Memory Network

Add Information Flow?

Cell State!



남길 건 남기고, 잊어버릴 건 잊어버리고, 새로 추가할 건 추가해서
Cell State에 중요한 정보만 계속 흘러가도록!

Hidden State는 Cell State를 적당히 가공해서 내보내자!

어떻게? → Using Gate!

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

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

0.1	0.3	−0.1	0.2	0.5	0.9	−1.9	2.0	−1.1	0.3	C_{t-1}

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1	G
0.1	0.3	−0.1	0.2	0.5	0.9	−1.9	2.0	−1.1	0.3	C_{t-1}

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

X

0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1	G
0.1	0.3	−0.1	0.2	0.5	0.9	−1.9	2.0	−1.1	0.3	C_{t-1}

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

	0.01	0.03	−0.03	0.198	0.49	0.18	−0.19	1.8	−0.11	0.03	G C_{t-1}
X	0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1	G
	0.1	0.3	−0.1	0.2	0.5	0.9	−1.9	2.0	−1.1	0.3	C_{t-1}

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

	0.01	0.03	−0.03	0.198	0.49	0.18	−0.19	1.8	−0.11	0.03	$G\ C_{t-1}$
X	0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1	G
	0.1	0.3	−0.1	0.2	0.5	0.9	−1.9	2.0	−1.1	0.3	C_{t-1}

How to judge value of each gate coefficient?

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

X

0.01	0.03	−0.03	0.198	0.49	0.18	−0.19	1.8	−0.11	0.03	G C_{t-1}
0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1	G
0.1	0.3	−0.1	0.2	0.5	0.9	−1.9	2.0	−1.1	0.3	C_{t-1}

How to judge value of each gate coefficient?

→ Using small non–linear layer!

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0∼1

$$g_t = \sigma(W_g \cdot v_{input})$$

Input Vector

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

Linear Transformation

$$g_t = \sigma(W_g \cdot v_{input})$$

Input Vector

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

Linear Transformation

$$g_t = \sigma(W_g \cdot v_{input})$$

Input Vector

Mapping transformation output into 0~1 range

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

Linear Transformation

Vector with 0~1 elements

$$g_t = \sigma(W_g \cdot v_{input})$$

Input Vector

Mapping transformation output into 0~1 range

Long Short Term Memory Network

Gate – element wise coefficient multiplication

Control whether pass or block the information of each dimension with coefficient 0~1

Linear Transformation

Vector with 0~1 elements

$$g_t = \sigma(W_g \cdot v_{input})$$

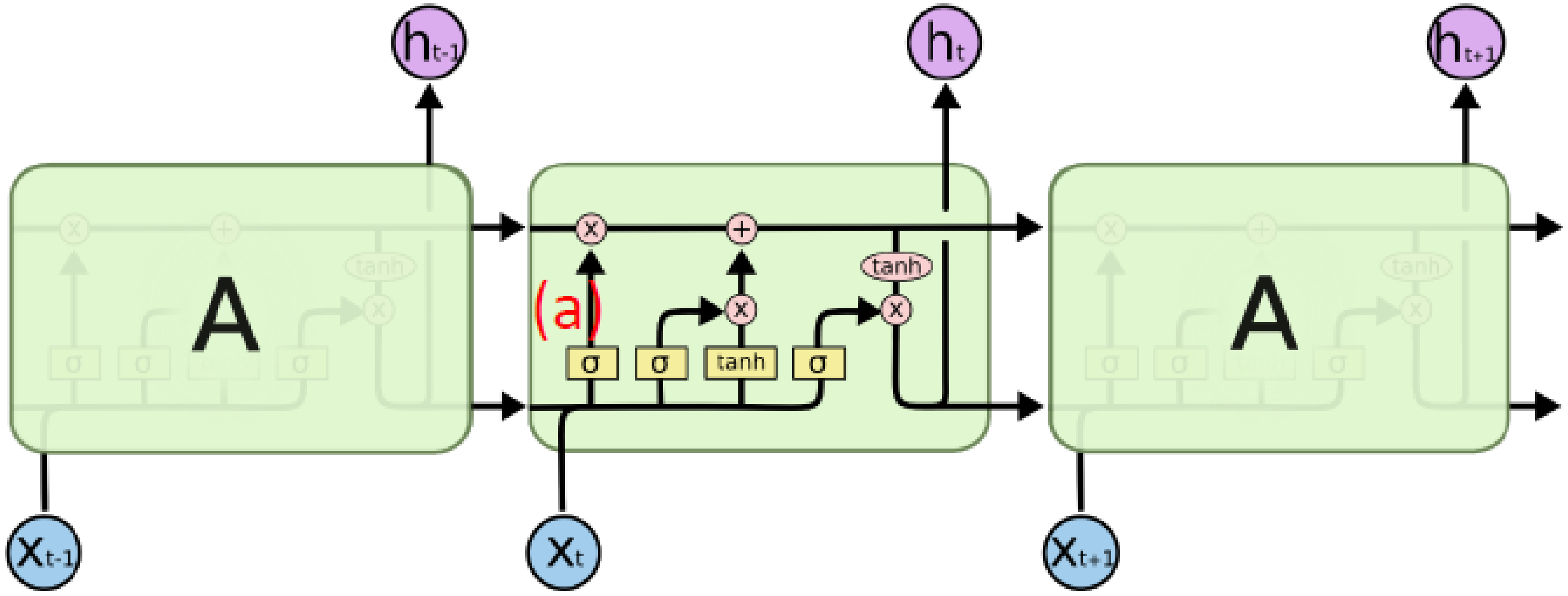
Input Vector

Mapping transformation output into 0~1 range

$$C_t' = g_t \cdot C_t$$

Gating Information from C_t

Long Short Term Memory Network

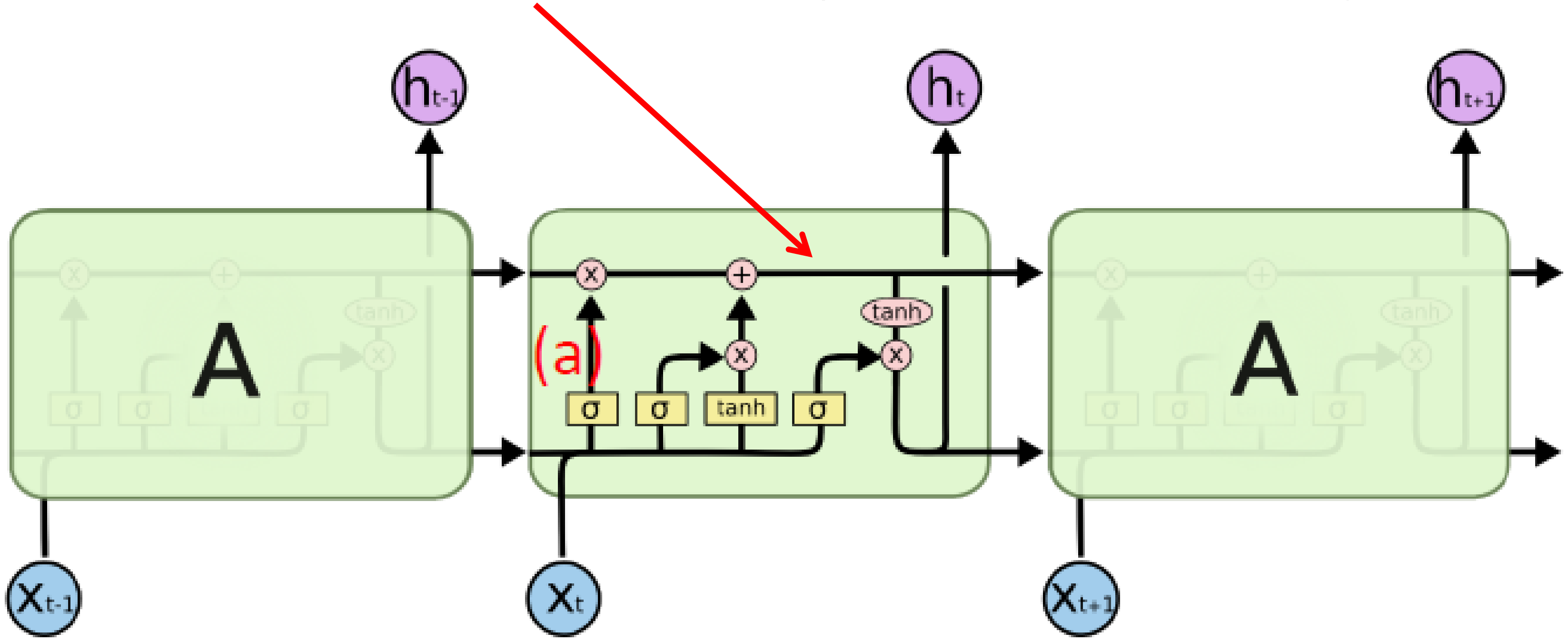


LSTM

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

Long Short Term Memory Network

Cell State: Flow of precious information through time

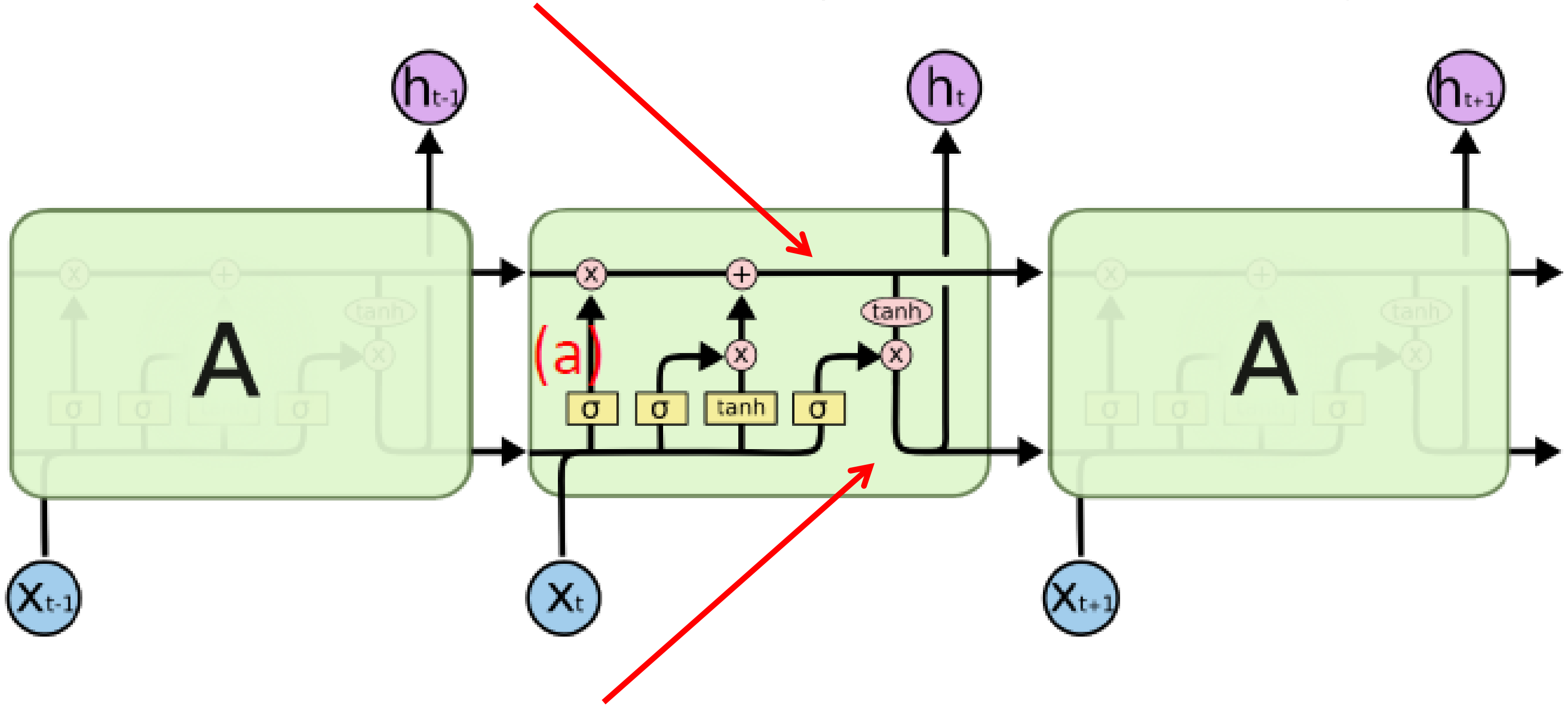


LSTM

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

Long Short Term Memory Network

Cell State: Flow of precious information through time



Hidden State: Output of each time step

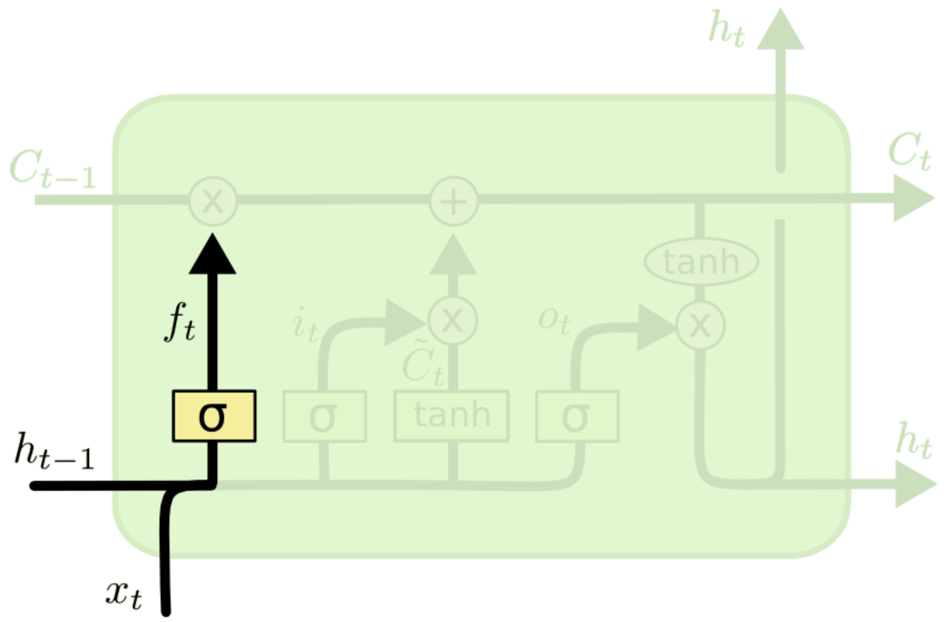
LSTM

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

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 C_t를 만든다.
4. C_t를 적절히 가공해 해당 t에서의 h_t를 만든다.
5. C_t와 h_t를 다음 스텝 t+1로 전달한다.

Long Short Term Memory Network



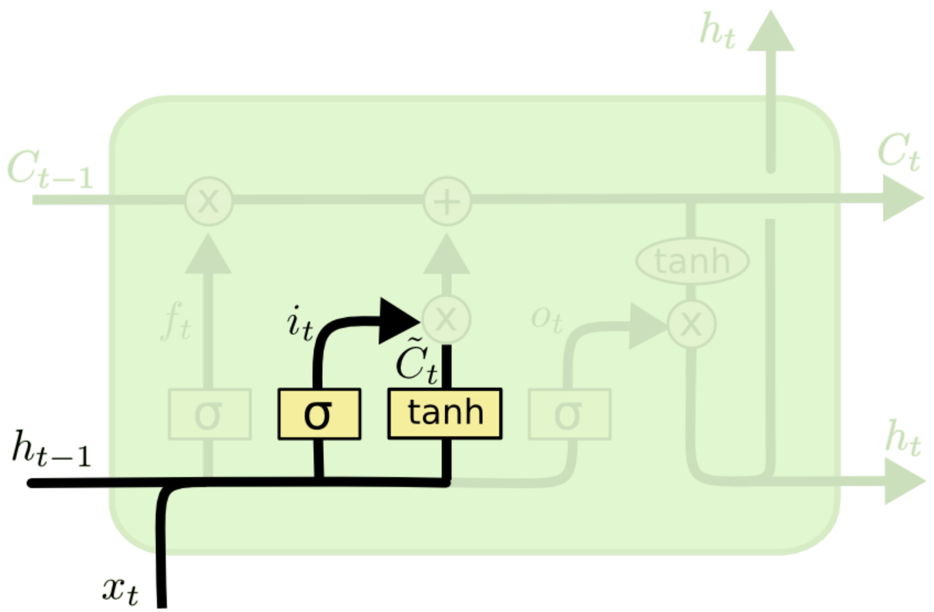
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

1. C_{t-1} 에서 불필요한 정보를 지운다.

   f는 forget gate를 의미

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

Long Short Term Memory Network



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

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

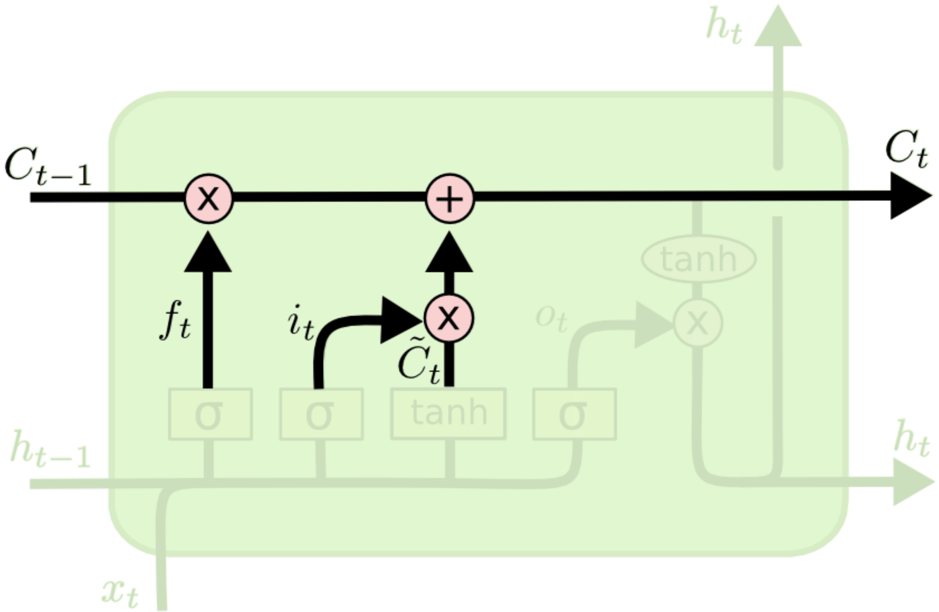
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.

i는 input gate를 의미

임시 C_t를 계산

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

Long Short Term Memory Network



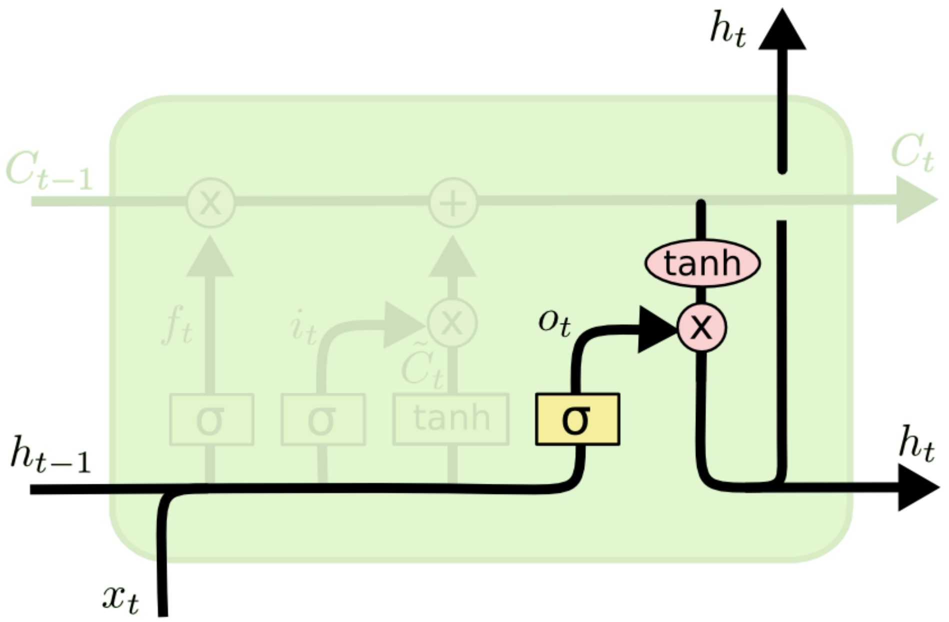
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

3. 위 과정을 통해 C_t를 만든다.

f_t를 이용해서 C_{t-1}의 일부 정보를 날리고 임시 C_t 정보를 추가한다

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

Long Short Term Memory Network



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

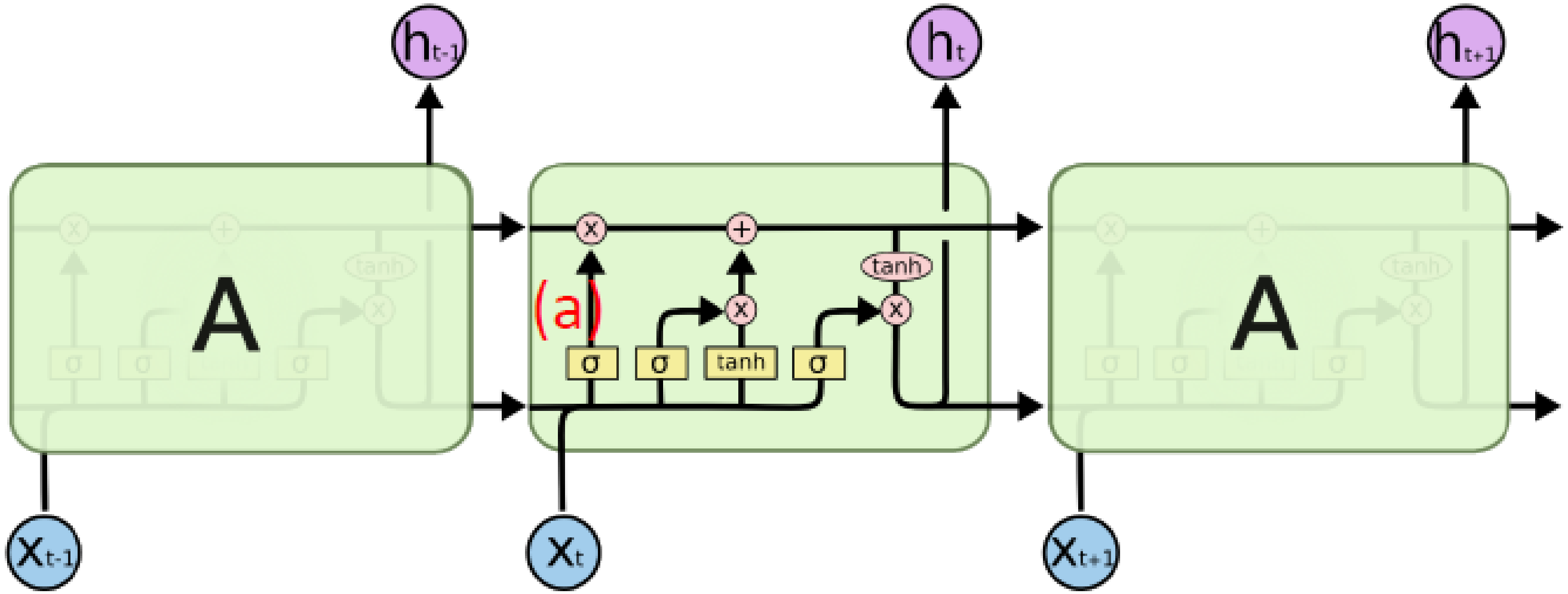
$$h_t = o_t * \tanh\left(C_t\right)$$

4. Ct를 적절히 가공해 해당 t에서의 ht를 만든다.

Ct를 가공할 output gate ot를 바탕으로 ht를 계산

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

Long Short Term Memory Network



5. C_t와 h_t를 다음 스텝 t+1로 전달한다.

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

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.

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

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.

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

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

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

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 C_t를 만든다.

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

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

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

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

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 C_t를 만든다.
4. C_t를 적절히 가공해 해당 t에서의 h_t를 만든다.

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

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

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

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

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

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

Long Short Term Memory Network

1. Ct–1 에서 불필요한 정보를 지운다.
2. Ct–1에 새로운 인풋 xt와 ht–1를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 Ct를 만든다.
4. Ct를 적절히 가공해 해당 t에서의 ht를 만든다.
5. Ct와 ht를 다음 스텝 t+1로 전달한다.

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

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

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

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

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

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

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 C_t를 만든다.
4. C_t를 적절히 가공해 해당 t에서의 h_t를 만든다.
5. C_t와 h_t를 다음 스텝 t+1로 전달한다.

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

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

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

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

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

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

뭔가 너무 중복되는 느낌..?

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 C_t를 만든다.
4. C_t를 적절히 가공해 해당 t에서의 h_t를 만든다.
5. C_t와 h_t를 다음 스텝 t+1로 전달한다.

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

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

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

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

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

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

뭔가 너무 중복되는 느낌..?

→ 더 간단하게 forget gate, input gate, output gate를 해결 할 수는 없을까?

Long Short Term Memory Network

1. C_{t-1} 에서 불필요한 정보를 지운다.
2. C_{t-1}에 새로운 인풋 x_t와 h_{t-1}를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해 C_t를 만든다.
4. C_t를 적절히 가공해 해당 t에서의 h_t를 만든다.
5. C_t와 h_t를 다음 스텝 t+1로 전달한다.

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

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

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

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

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

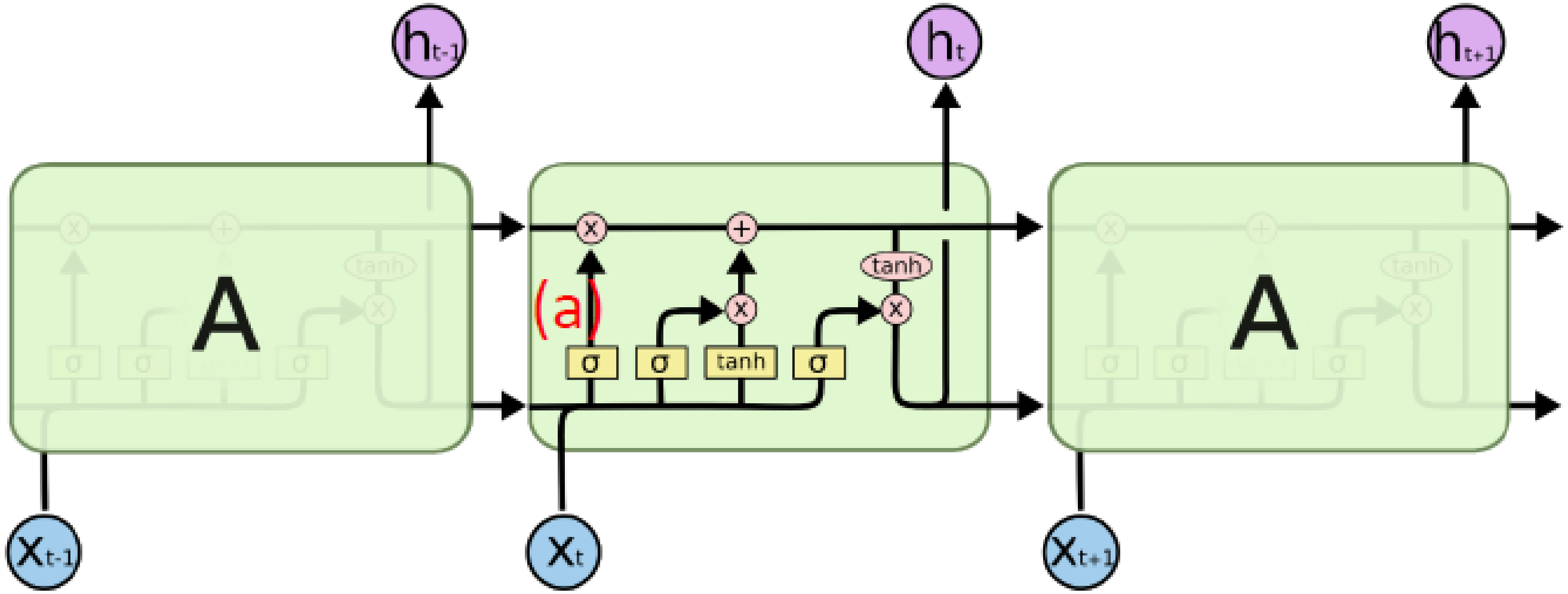
$$h_t = o_t * \tanh\left(C_t\right)$$

뭔가 너무 중복되는 느낌..?

→ 더 간단하게 forget gate, input gate, output gate를 해결 할 수는 없을까?

So, How LSTM could overcome vanishing gradient problem?

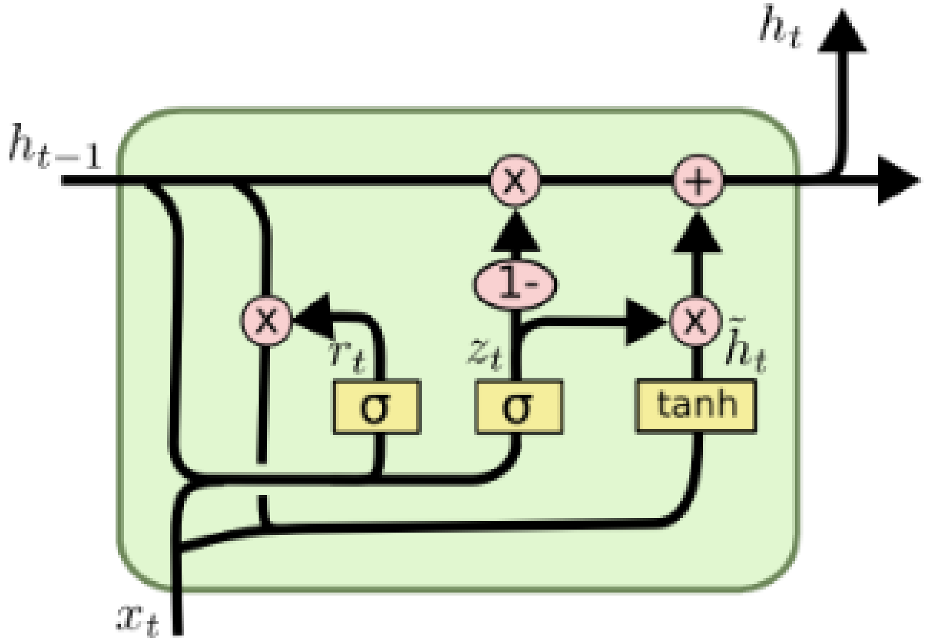
So, How LSTM could overcome vanishing gradient problem?



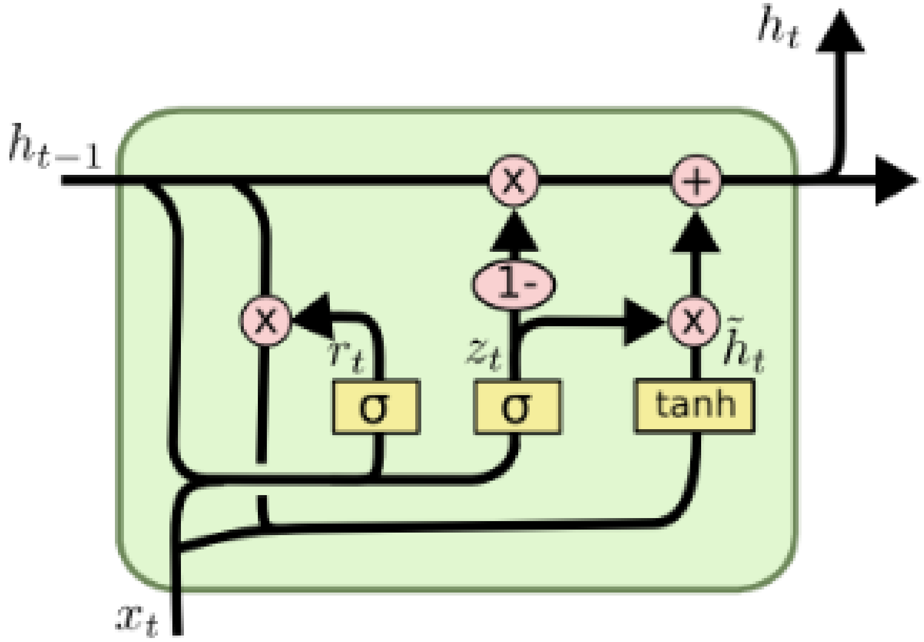
t 시점에서 cell state에 기록된 정보가 만약 지워지지 않고 t+n에서 활용 되었을 때 중간에 non-linear activation function을 거치치 않고 t+n 시점까지 흘러오기 때문에 Vanishing Gradient Problem을 상당 부분 해결 할 수 있다.

Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU)



Gated Recurrent Unit (GRU)

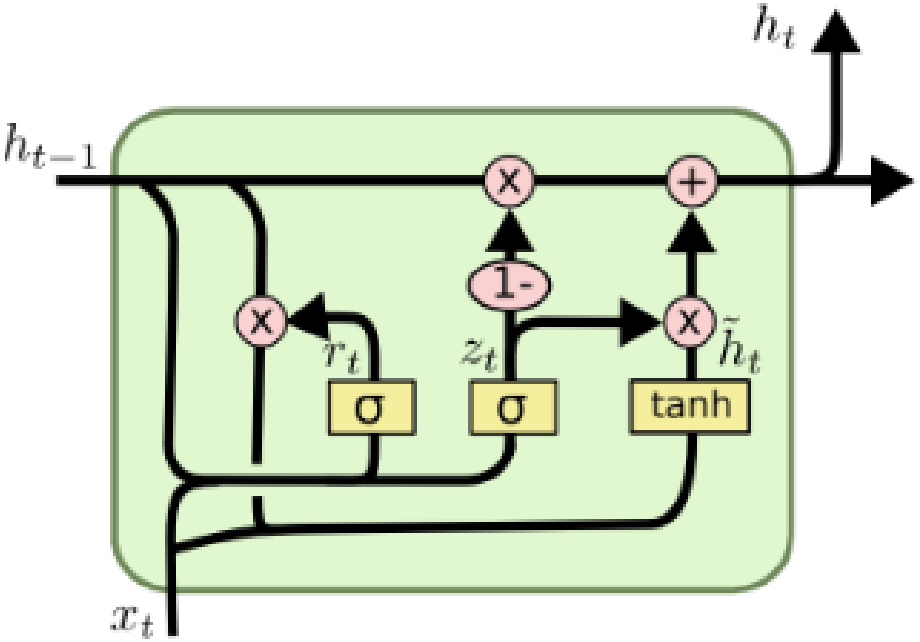


$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

1. Reset gate를 계산해서 임시 h_t를 만든다.

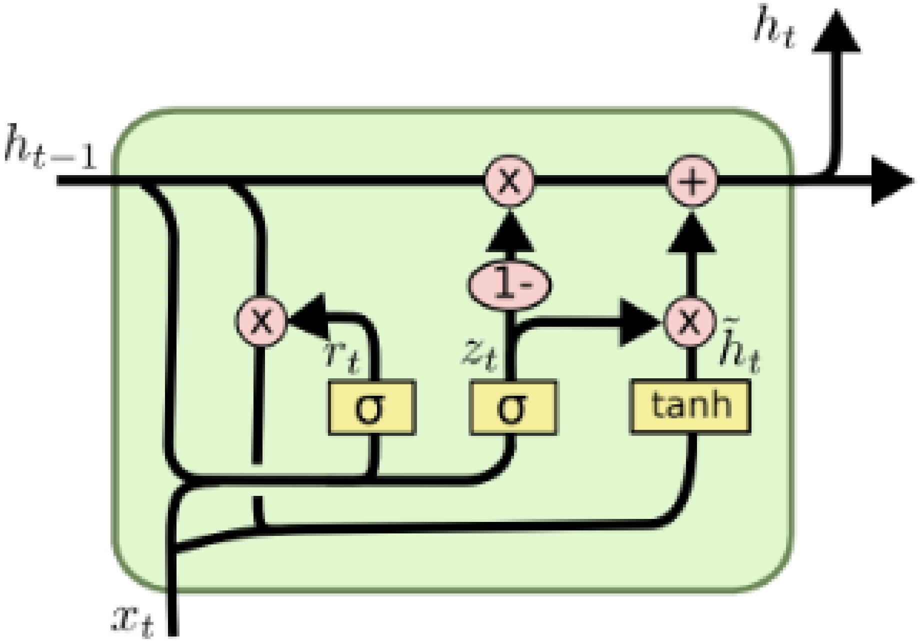
Gated Recurrent Unit (GRU)



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

2. Update gate를 통해 h_{t-1}과 h_t간의 비중을 결정한다.

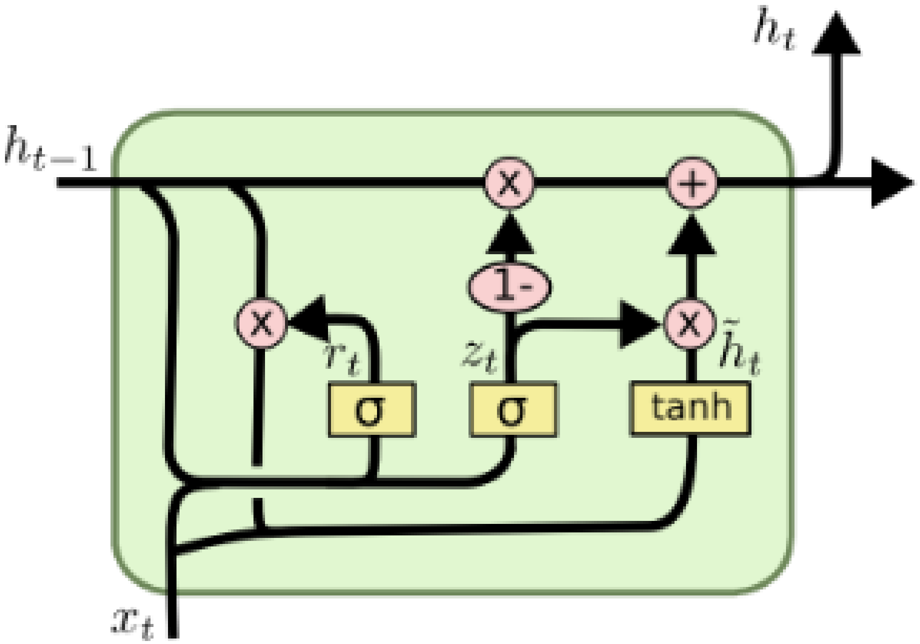
Gated Recurrent Unit (GRU)



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

3. z_t를 이용해 최종 h_t를 계산한다.

Gated Recurrent Unit (GRU)



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

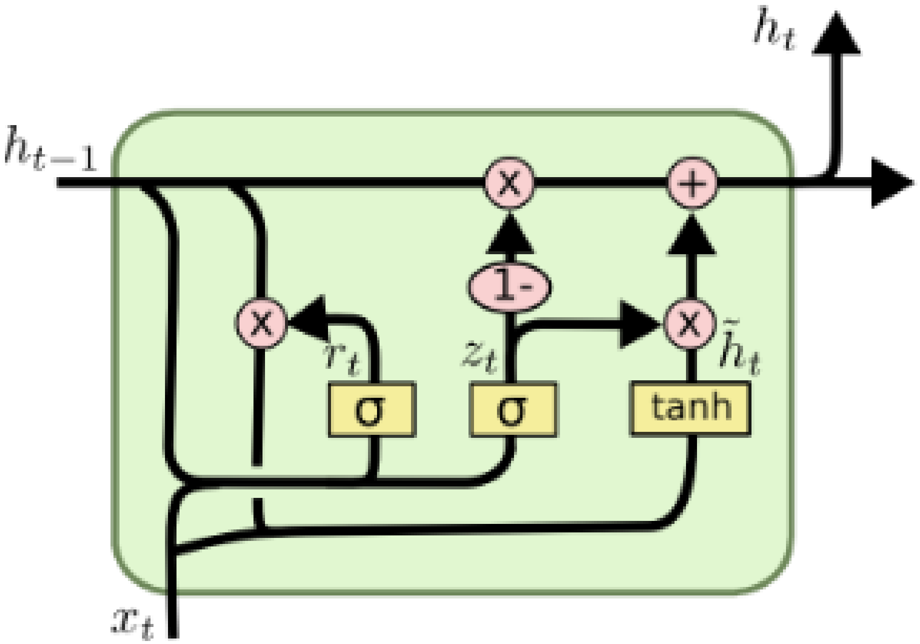
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

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

1. Reset gate를 계산해서 임시 ht를 만든다.
2. Update gate를 통해 ht−1과 ht간의 비중을 결정한다.
3. Zt를 이용해 최종 ht를 계산한다.

Gated Recurrent Unit (GRU)



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

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

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

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

1. Reset gate를 계산해서 임시 ht를 만든다.
2. Update gate를 통해 ht−1과 ht간의 비중을 결정한다.
3. zt를 이용해 최종 ht를 계산한다.

Empirical Exploration of RNN Architectures

Arch.	N	N-dropout	P
Tanh	3.612	3.267	6.809
LSTM	3.492	3.403	6.866
LSTM-f	3.732	3.420	6.813
LSTM-i	3.426	**3.252**	6.856
LSTM-o	3.406	**3.253**	6.870
LSTM-b	3.419	3.345	6.820
GRU	3.410	3.427	6.876
MUT1	**3.254**	3.376	**6.792**
MUT2	3.372	3.429	6.852
MUT3	3.337	3.505	6.840

Table 2. Negative Log Likelihood on the music datasets. N stands for Nottingham, N-dropout stands for Nottingham with nonzero dropout, and P stands for Piano-Midi.

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

Table 3. Perplexities on the PTB. The prefix (e.g., 5M) denotes the number of parameters in the model. The suffix "v" denotes validation negative log likelihood, the suffix"tst" refers to the test set. The perplexity for select architectures is reported in parentheses. We used dropout only on models that have 10M or 20M parameters, since the 5M models did not benefit from dropout at all, and most dropout-free models achieved a test perplexity of 108, and never greater than 120. In particular, the perplexity of the best models without dropout is below 110, which outperforms the results of Mikolov et al. (2014).

Most RNN variants are almost the same

[Greff15] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steuuebrink, J. Schmidhuber, LSTM: A Search Space Odyssey, arXiv:1503.04069
[Jozefowicz15] R. Jozefowicz, W. Zaremba, I. Sutskever, An Empirical Exploration of Recurrent Network Architectures, ICML 2015

Summary

- Vanila RNN has vanishing gradient problem

- LSTM architecture is useful to overcome vanishing gradient problem
  due to Cell State which enables information flow through time
  without passing non-linear activation function

- GRU architecture could also overcome same problem but more efficiently