Recurrent Neural Networks

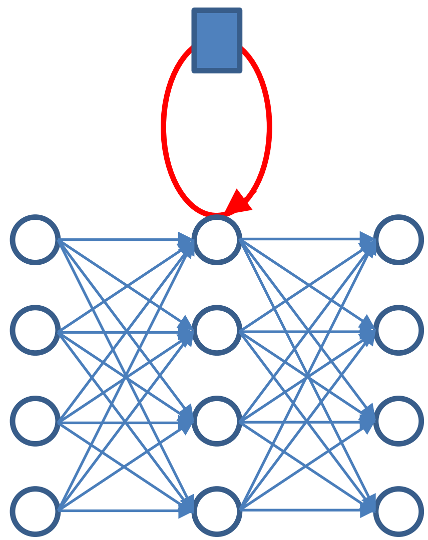
Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

2019

HJYU@UOS.AC.KR

Sequence Modeling

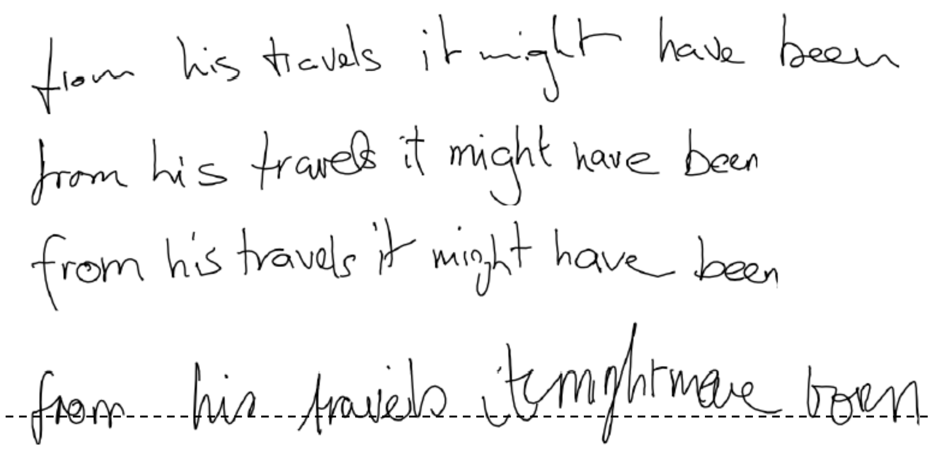
- Recurrent neural networks or RNNs (Rumelhart et al., 1986a)
- Sequential data $x^{(1)}, \ldots, x^{(\tau)}$ 처리 (다양한 길이)
  - Speech, text, video, handwriting, …

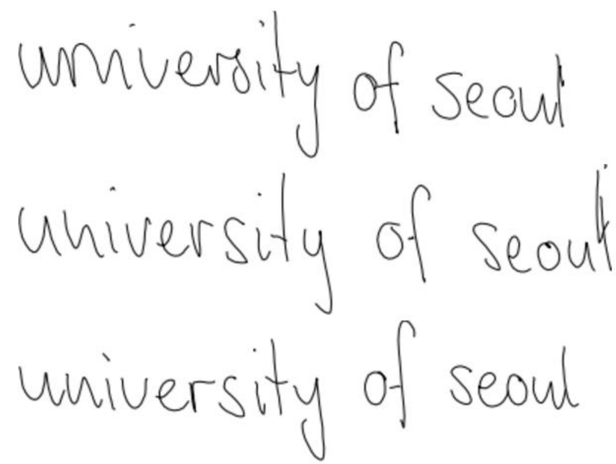


RNN applications

- Unconstrained handwriting recognition (graves *et al.*, 2009),
- Speech recognition (graves *et al.*, 2013; graves and jaitly, 2014),
- Handwriting generation (graves, 2013),
- Machine translation (sutskever *et al.*, 2014),
- Image captioning (kiros *et al.*, 2014b; vinyals *et al.*, 2014b; xu *et al.*, 2015)

Generating hand-writing using RNN





Graves, Alex. "Generating sequences with recurrent neural networks." *arXiv preprint arXiv:1308.0850* (2013).

RNN (Recurrent Neural Network)

- Feedforward : $\boldsymbol{h}(t) = f\,(\,\boldsymbol{x}(t)\,;\,\boldsymbol{\theta}\,)$

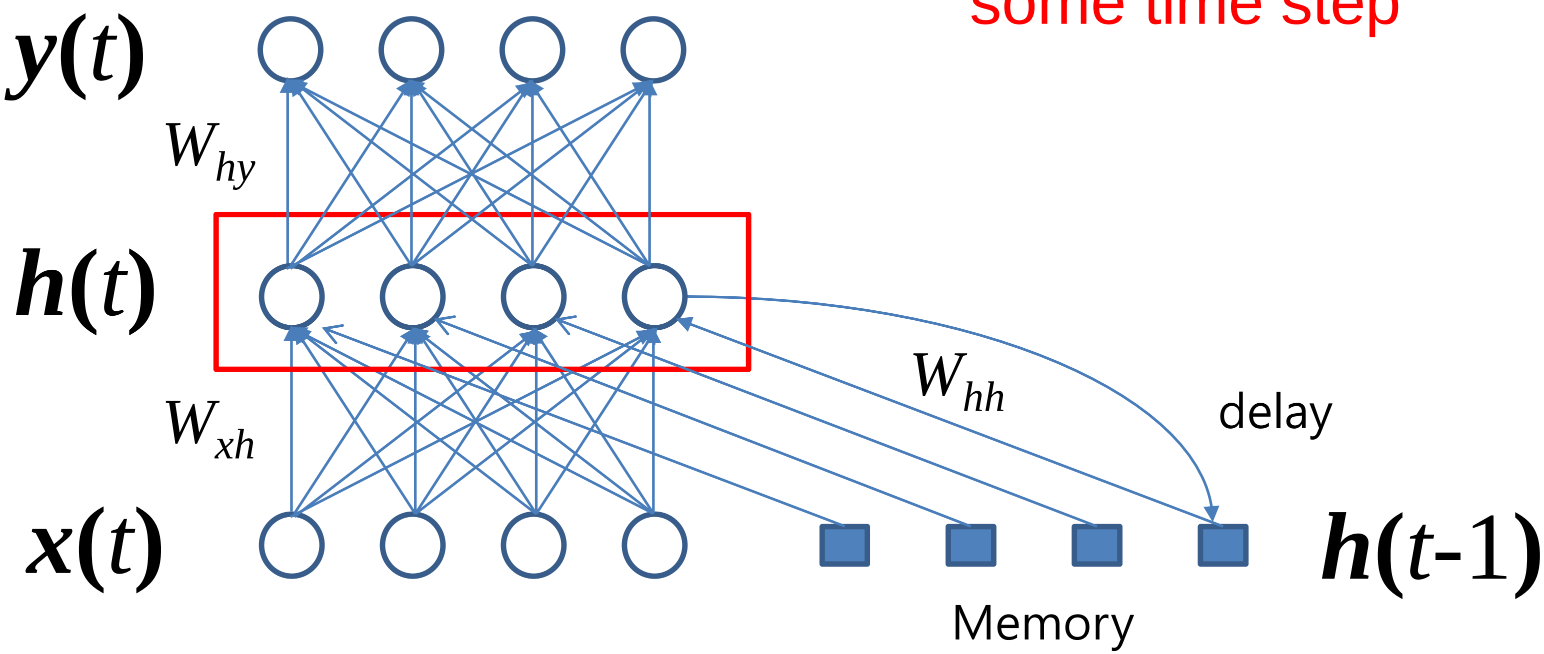
parameters W

- RNN : $$\boldsymbol{h}(t) = f\,(\,\boldsymbol{h}(t-1)\,,\,\boldsymbol{x}(t)\,;\,\boldsymbol{\theta}\,)$$

new state

old state

input vector at some time step



Simple RNN

- Feedforward : $\boldsymbol{h}(t) = f\,(\,\boldsymbol{x}(t)\,;\,\boldsymbol{\theta}\,)$
- RNN : $\boldsymbol{h}(t) = f\,(\,\boldsymbol{h}(t-1)\,,\,\boldsymbol{x}(t)\,;\,\boldsymbol{\theta}\,)$

$$\boldsymbol{h}(t) = \tanh\,(\,\mathrm{W_{hh}}\,\boldsymbol{h}(t-1) + \mathrm{W_{xh}}\,\boldsymbol{x}(t)\,)$$

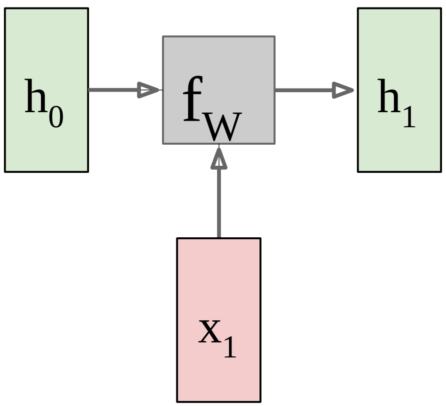
$$\boldsymbol{y}(t) = \mathrm{W_{hy}}\,\boldsymbol{h}(t)$$

$$\begin{aligned}\boldsymbol{h}(t) &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix}\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)\end{aligned}$$

- Also called as “Vanilla RNN”
  or “Elman RNN” after Prof. Jeffrey Elman

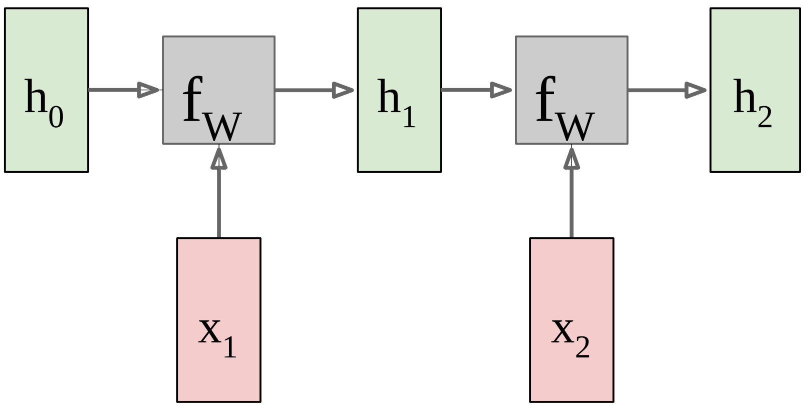
RNN: Computational Graph

h_t : time t 에서 hidden node 들의 값
x_t : time t 에서 input x 값



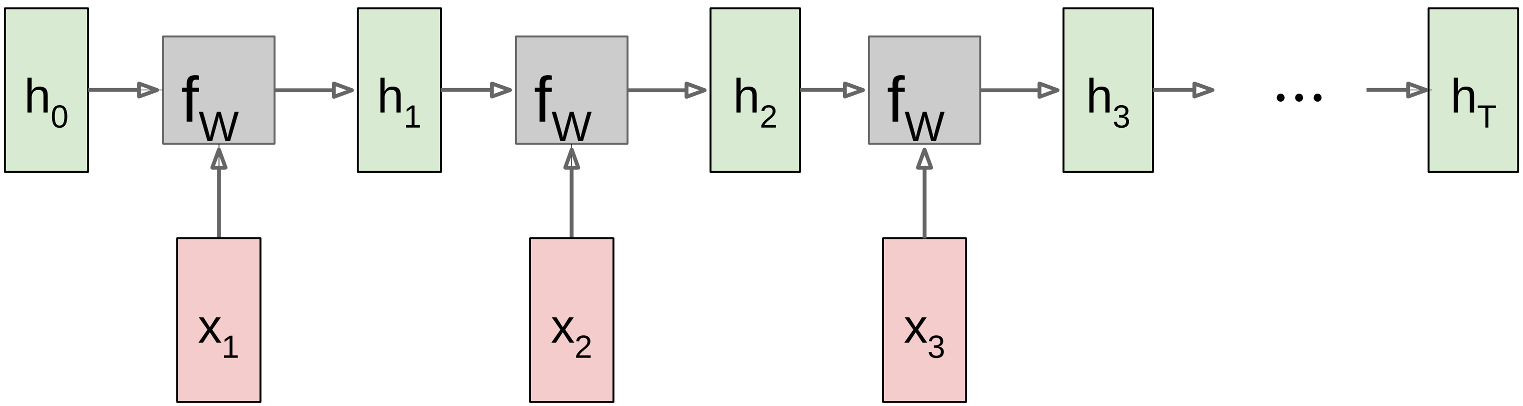
http://cs231n.stanford.edu/

RNN: Computational Graph



http://cs231n.stanford.edu/

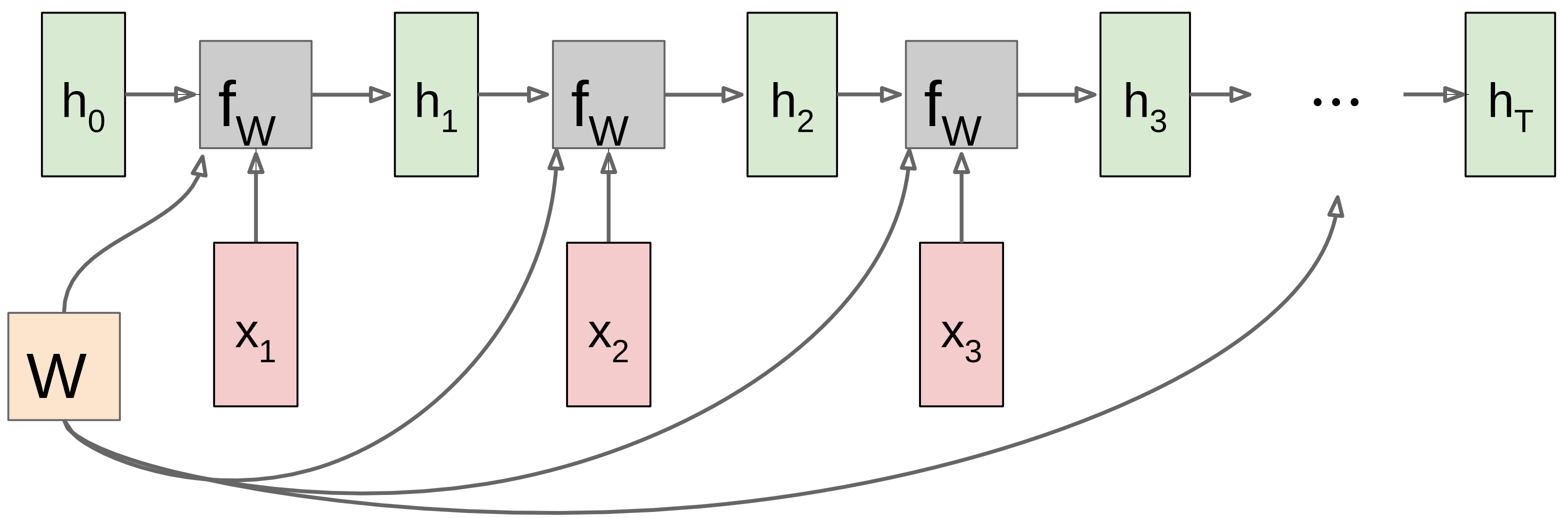
RNN: Computational Graph



http://cs231n.stanford.edu/

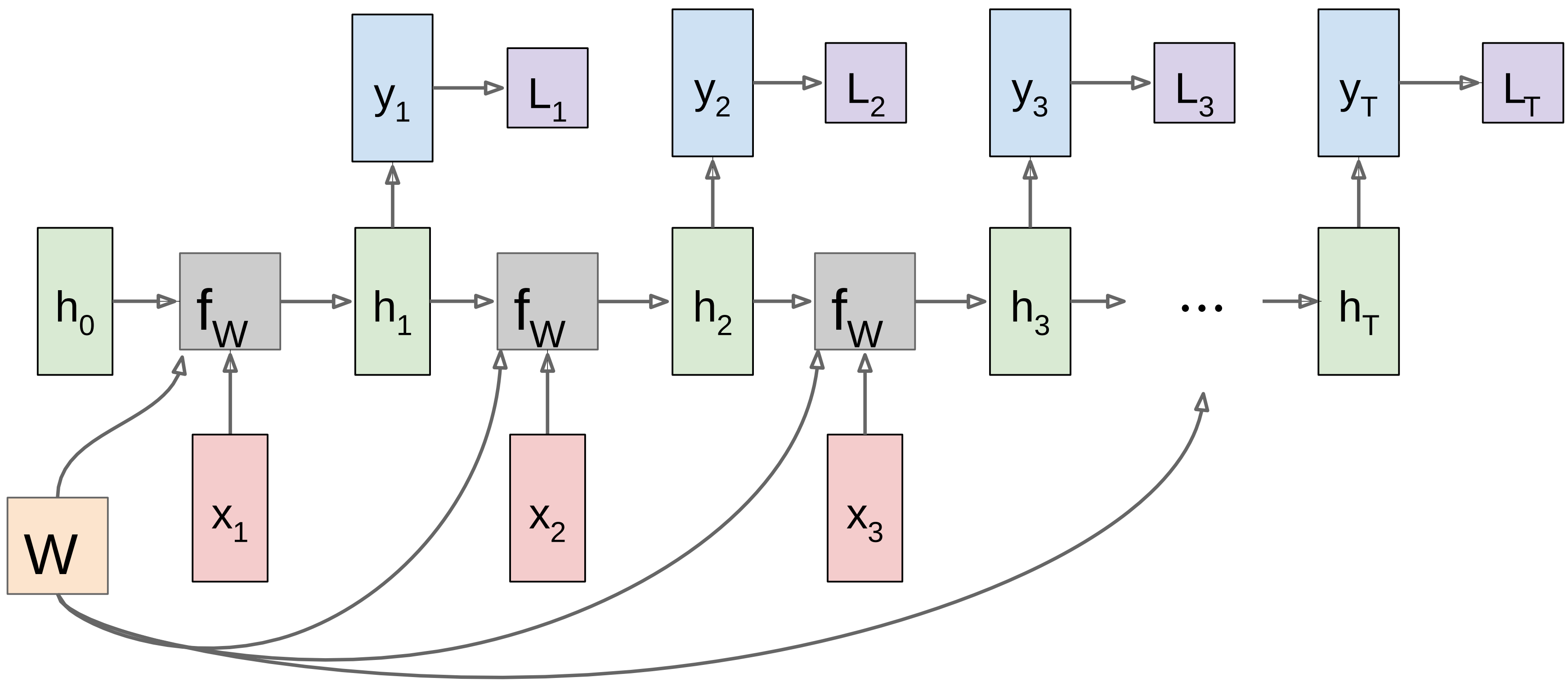
RNN: Computational Graph

Re-use the same weight matrix at every time-step



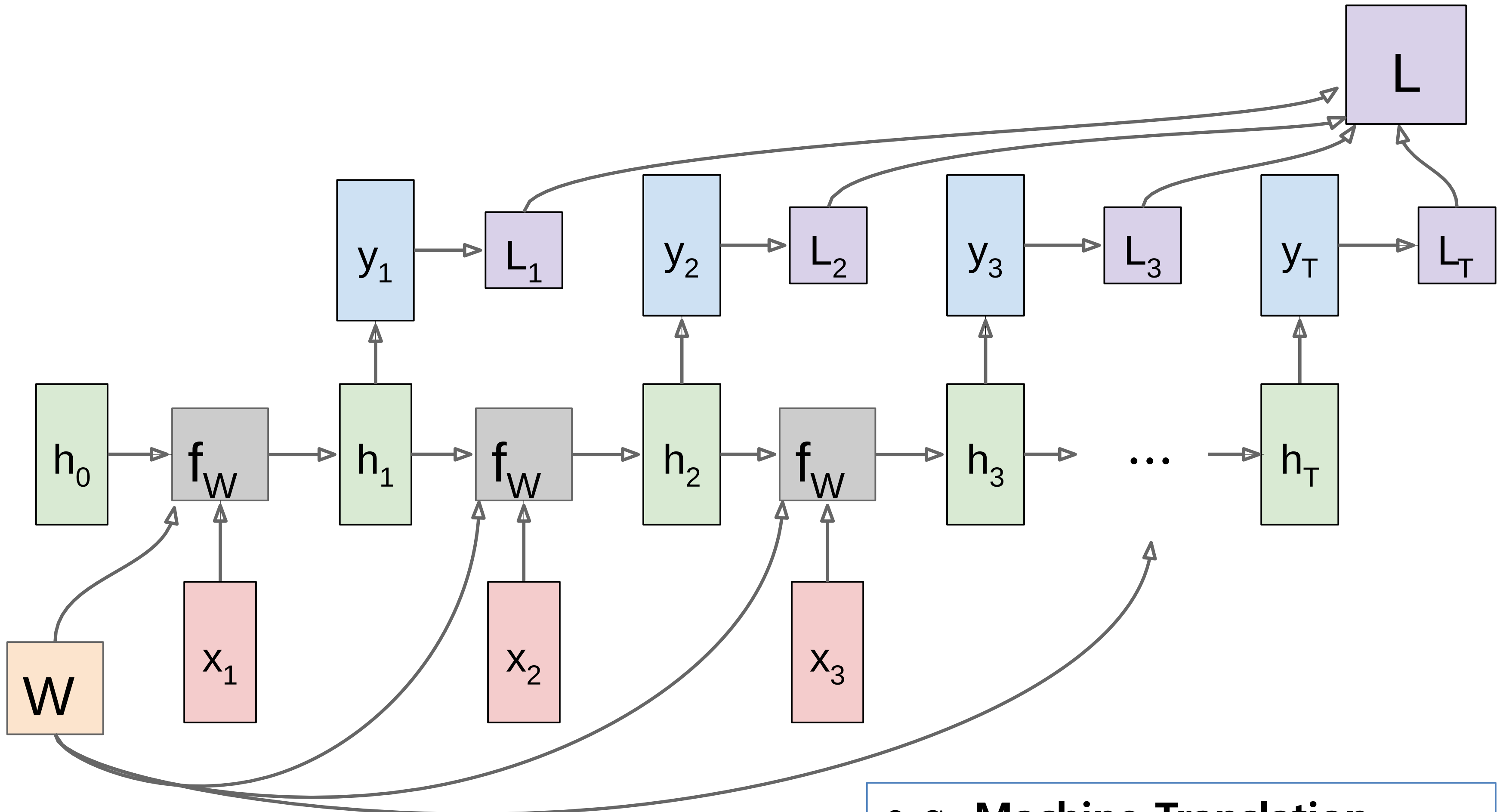
http://cs231n.stanford.edu/

RNN: Computational Graph: Many to Many



http://cs231n.stanford.edu/

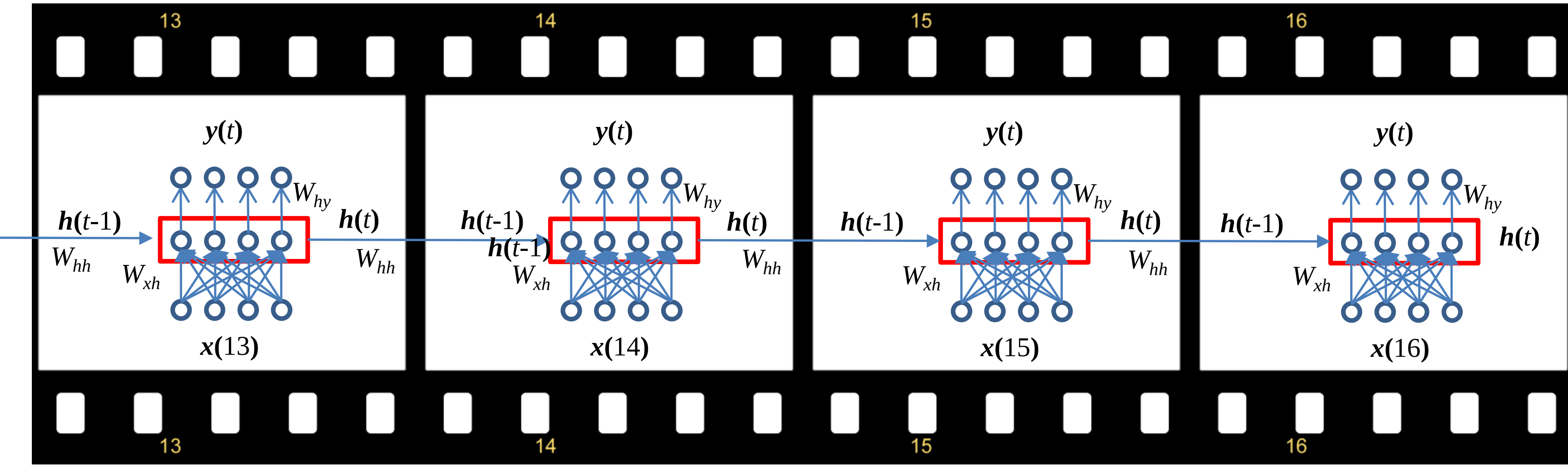
RNN: Computational Graph: Many to Many



e.g. **Machine Translation**
seq of words → seq of words

http://cs231n.stanford.edu/

Unfolding



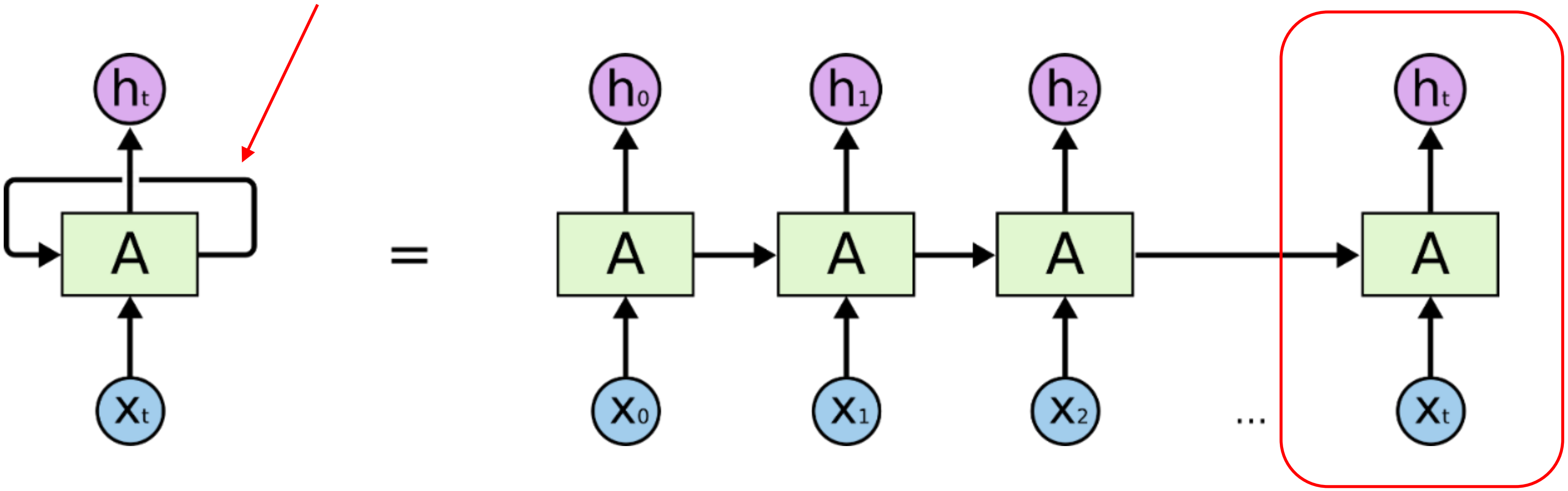
t = 13 t = 14 t = 15 t = 16

- 매 시간 별 상태를 보여주는 그림

RNN의 기본적인 구조

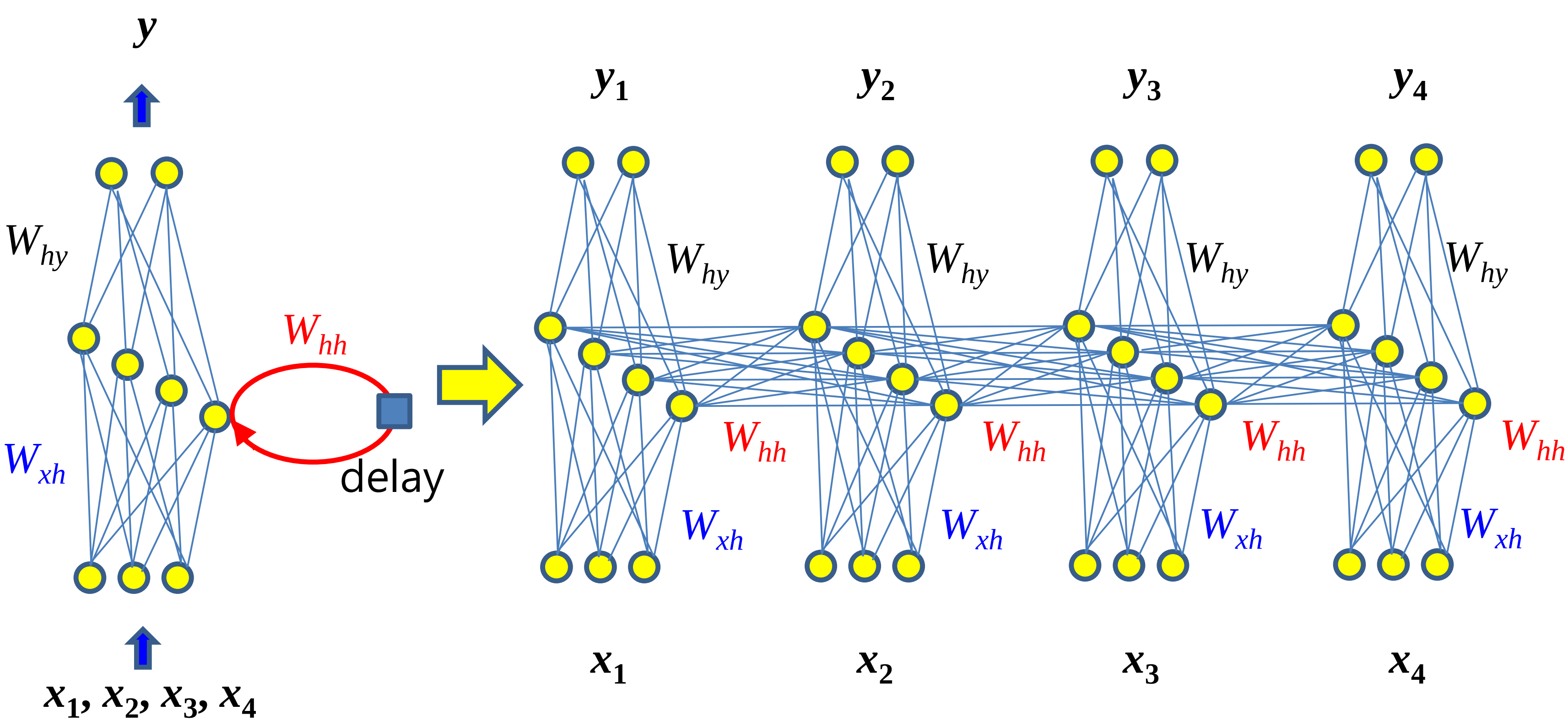
loop가 존재하여,
이전 단계의 정보가
현재 단계에 반영되도록 함

unfolding



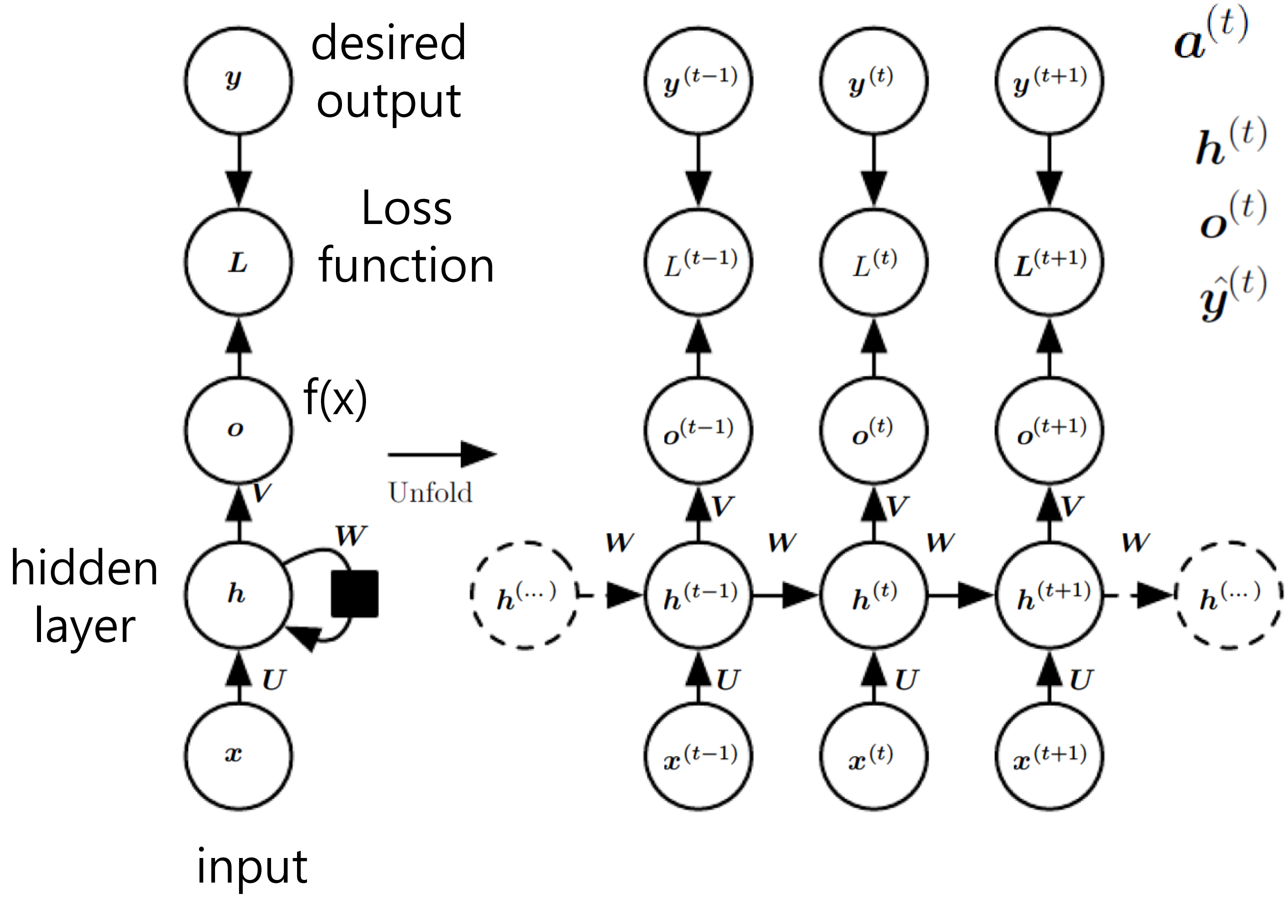
x_t: 시간 t 에서의 입력
h_t: 시간 t 에서의 출력

RNN의 unfolding



RNN의 기본적인 구조 (Many to Many)

- Recurrent connections between hidden units

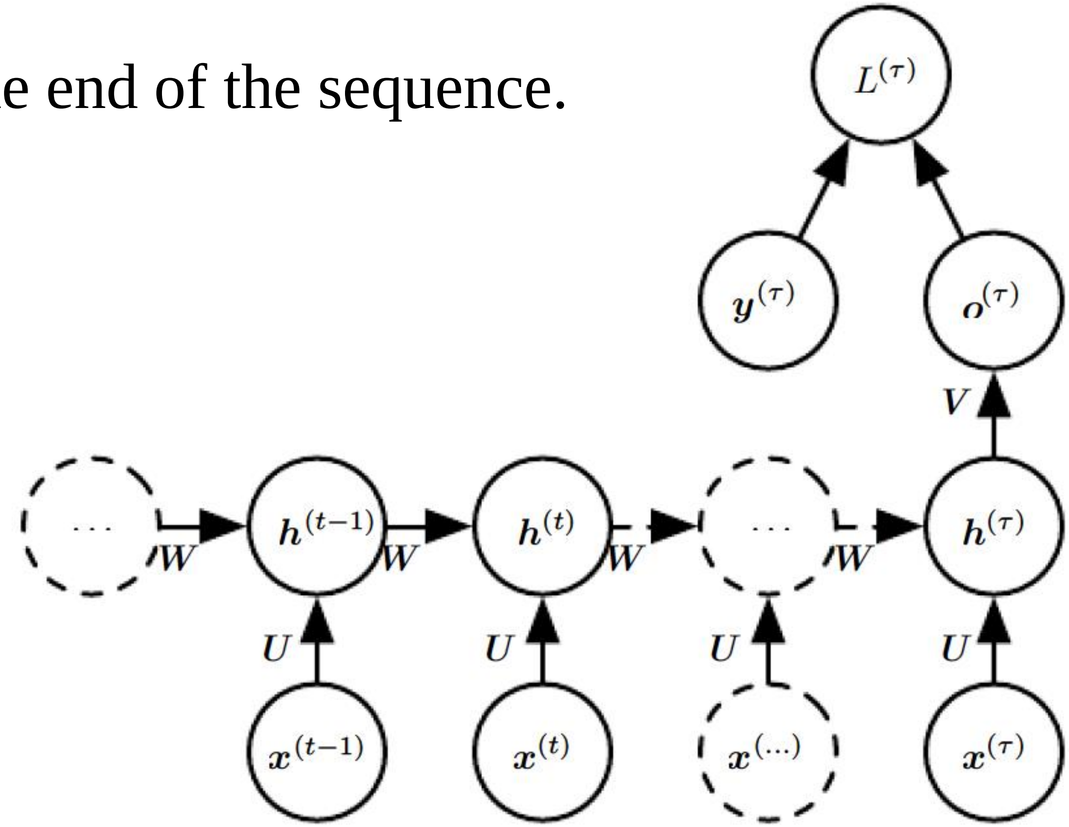


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

http://www.deeplearningbook.org

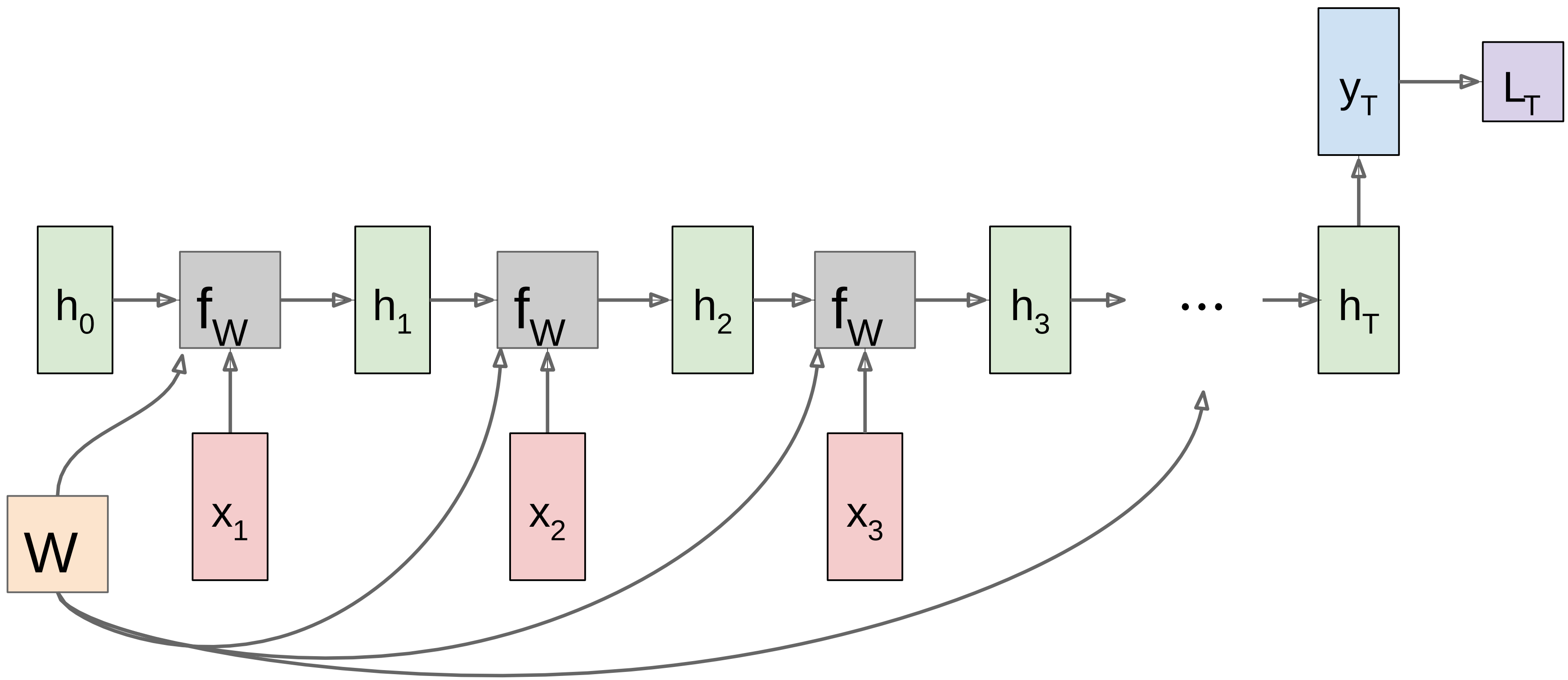
RNN 구조 (Many to One)

- A single output at the end of the sequence.



http://www.deeplearningbook.org

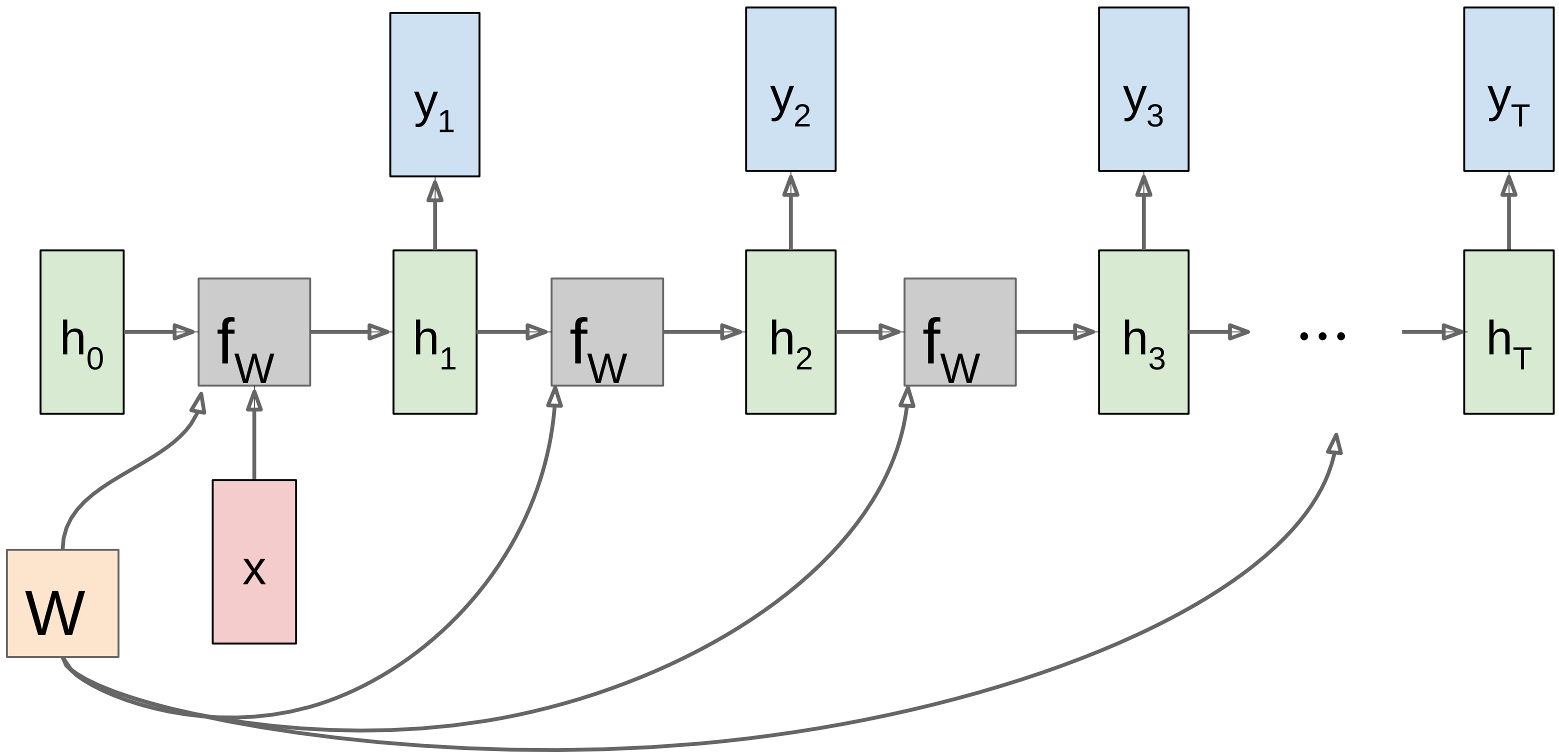
RNN: Computational Graph: Many to One



e.g. **speech recognition**
Speech waveform → word

http://cs231n.stanford.edu/

RNN: Computational Graph: One to Many



e.g. **Image Captioning**
image → sequence of words

http://cs231n.stanford.edu/

영상 주석 생성

- http://deeplearning.cs.toronto.edu/i2t

a man wearing a blue shirt with his arms on the grass.
a man holding a frisbee bat in front of a green field.
a man throwing a frisbee in a green field.
a boy playing ball with a disc in a field.
a young man playing in the grass with a green ball.

a red car on the side of the road in the small race.
a truck driving uphill on the side of the road.
a person driving a truck on the road.
a small car driving down a dirt and water.
a truck in a field of car is pulled up to the back.

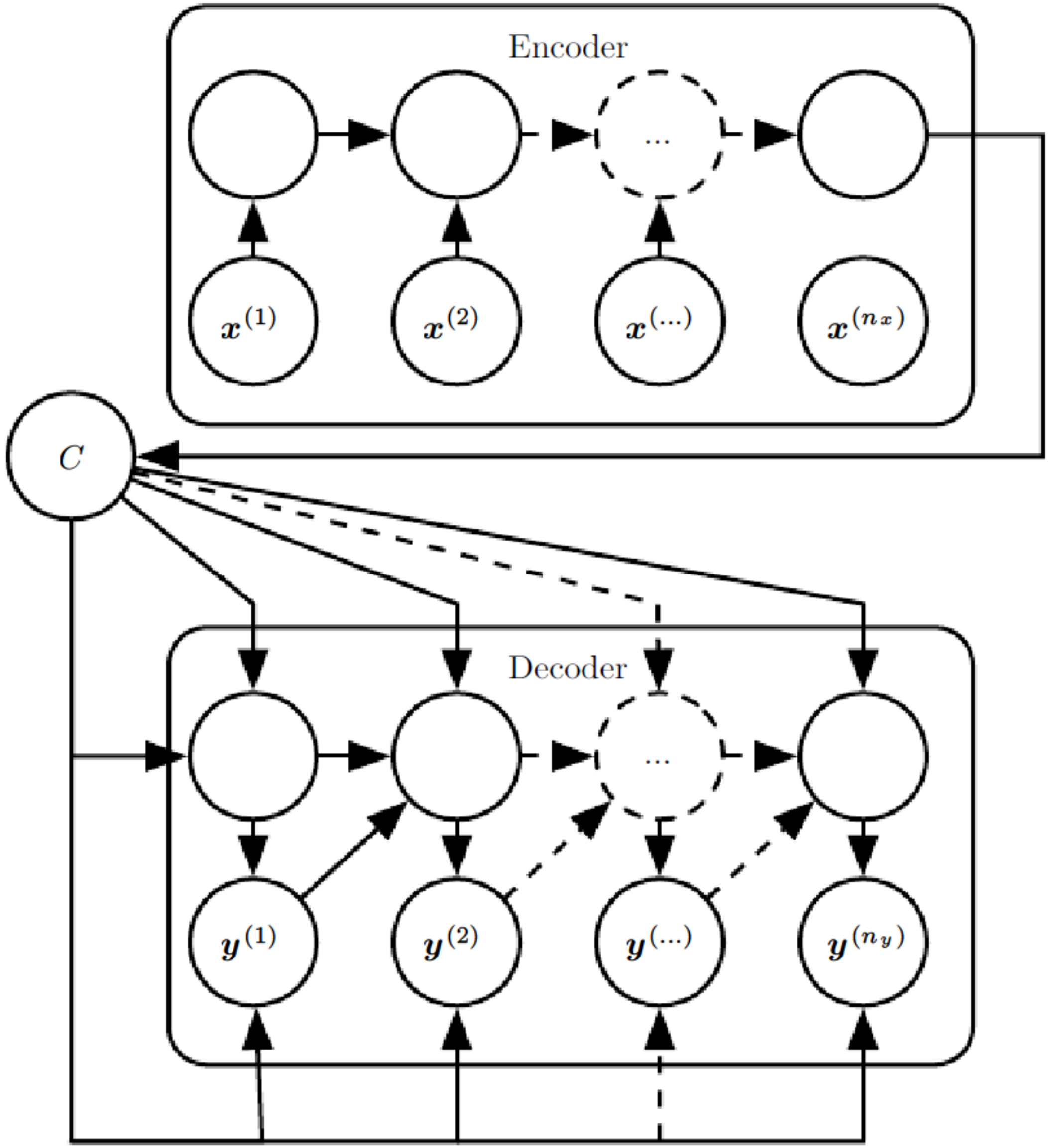
a group of birds standing next to each other.
a group of ducks that are standing in a row.
a group of ducks that are standing on each other.
a group of sheep next to each other on sand.
a group of small birds is standing in the grass.

a kite flying over the ocean on a sunny day.
a person flying over the ocean on a sunny day.
a person flying over the ocean on a cloudy day.
a kite on the beach on the water in the sky.
a large flying over the water and rocks.

서울시립대학교
UNIVERSITY OF SEOUL

Encoder-Decoder Sequence-to-Sequence Architectures

context



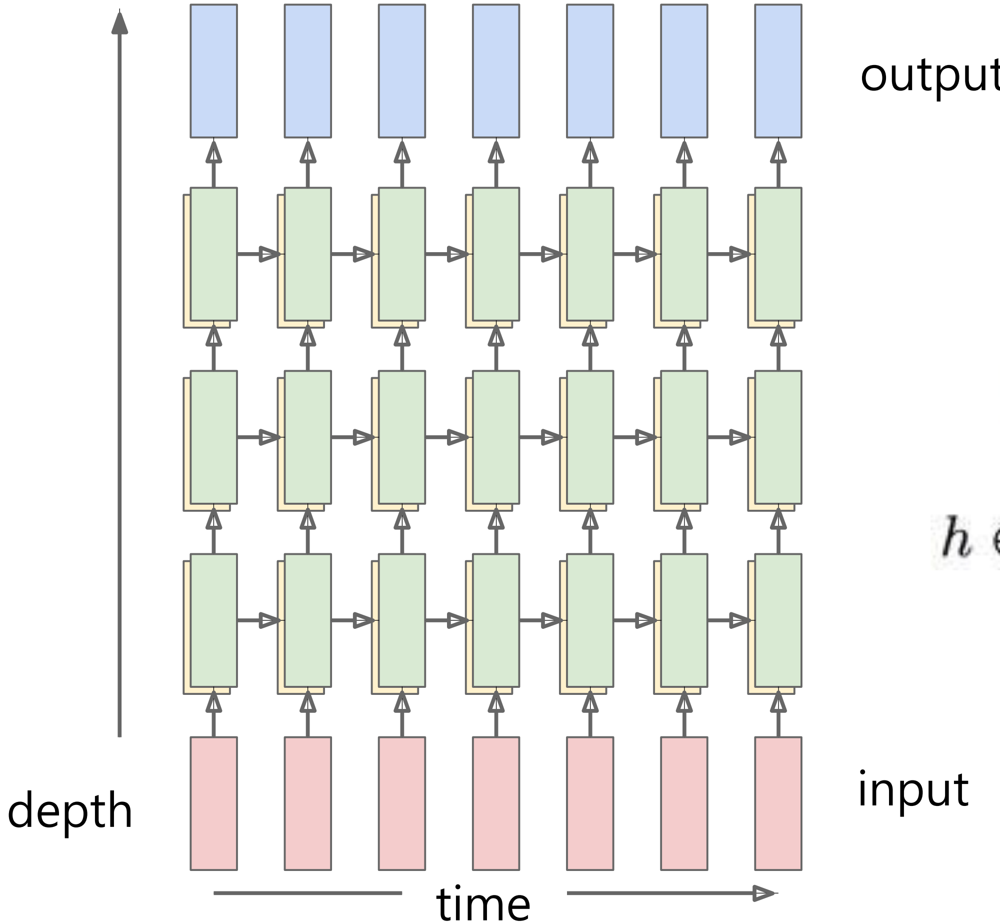
예) 입력 $\boldsymbol{x}$: 한국어

예) 출력 $\boldsymbol{y}$: 영어

RNN의 학습 : *BPTT*

- Back-propagation algorithm 그대로 적용
- *Back-propagation through time*
  - The back-propagation algorithm applied to the unrolled graph
- Forward pass 과정의 States 는 backward pass 계산을 위해서 저장됨
- Memory cost is $O(\tau)$.

Multilayer RNNs



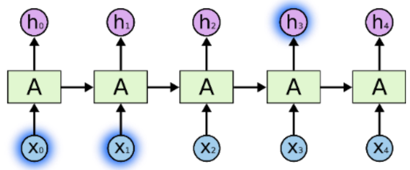
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad W^l \; [n \times 2n]$

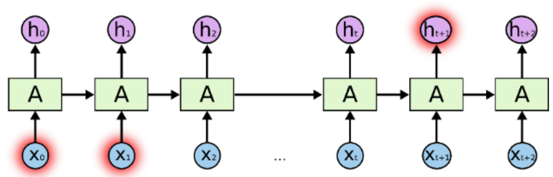
http://cs231n.stanford.edu/

The Challenge of Long-Term Dependencies

- 예: RNN을 이용해 문장의 마지막 단어를 추측
  - "The <u>clouds</u> are in the (   )"
    - 추측 대상인 '*sky*'와 추측에 필요한 'clouds' 간의 간격이 작음
    - RNN으로 추측 가능



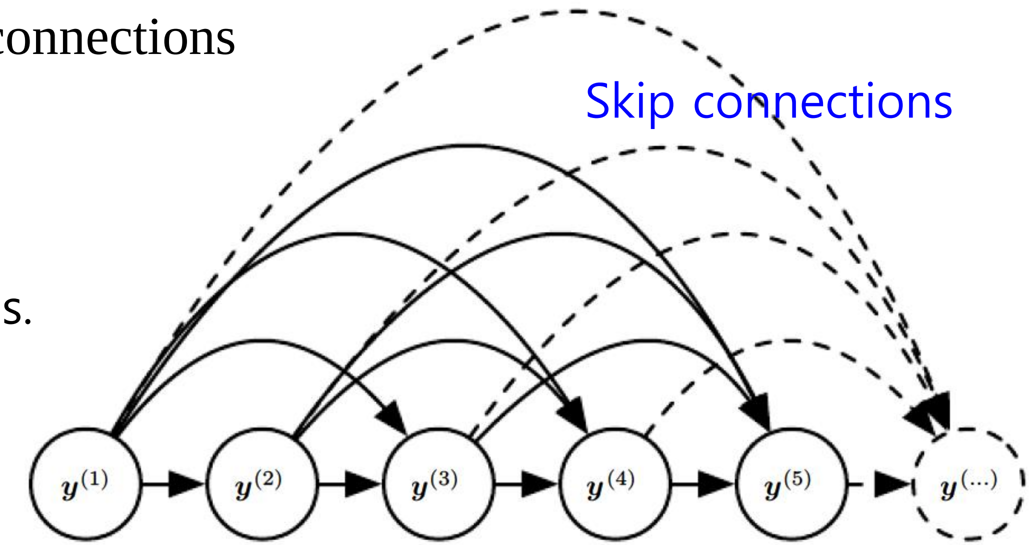
  - "I grew up in <u>France</u> …(중략)… I <u>speak</u> fluent (   )"
    - 'speak' 을 통해 마지막 단어는 언어 이름일 것임을 추측 가능
    - 어떤 언어인지까지 추측하고 싶다면?
      - 문장 맨 앞부분에 'France' 가 나왔다는 정보가 추가로 필요
      - RNN으로는 추측이 어려움
        - » 'France'와 '*French*' 간 간격이 크기 때문 (long-term dependency)



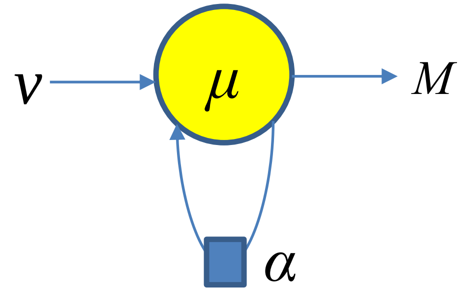
Long-term dependencies 처리 방법

- Multiple time scales 을 처리하는 model
  - Fine-grained time scales - small details
  - Coarse time scales – 먼 과거에서 현재로 정보 전달
- 해결 방법 들
  - Skip connections
  - Leaky units :
    Hidden units with linear self-connections
  - Removing Connections:

    removing length-one connections and replacing them with longer connections.



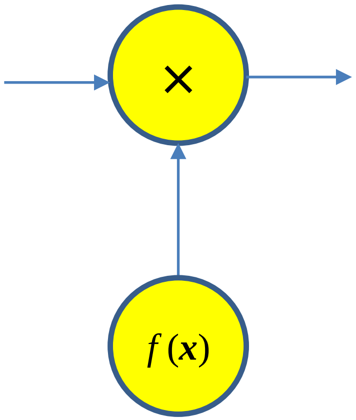
Leaky Units



- Integrate signals with different time constants
- $M^{(t)} \leftarrow \alpha\mu^{(t)} + (1-\alpha)v^{(t)}$
- $M^{(t)}$: running average of some value $v^{(t)}$
- Linear self-connection from $\mu(t-1)$ to $\mu(t)$.
- α parameter : when $\alpha = 1$: 오랫동안 기억

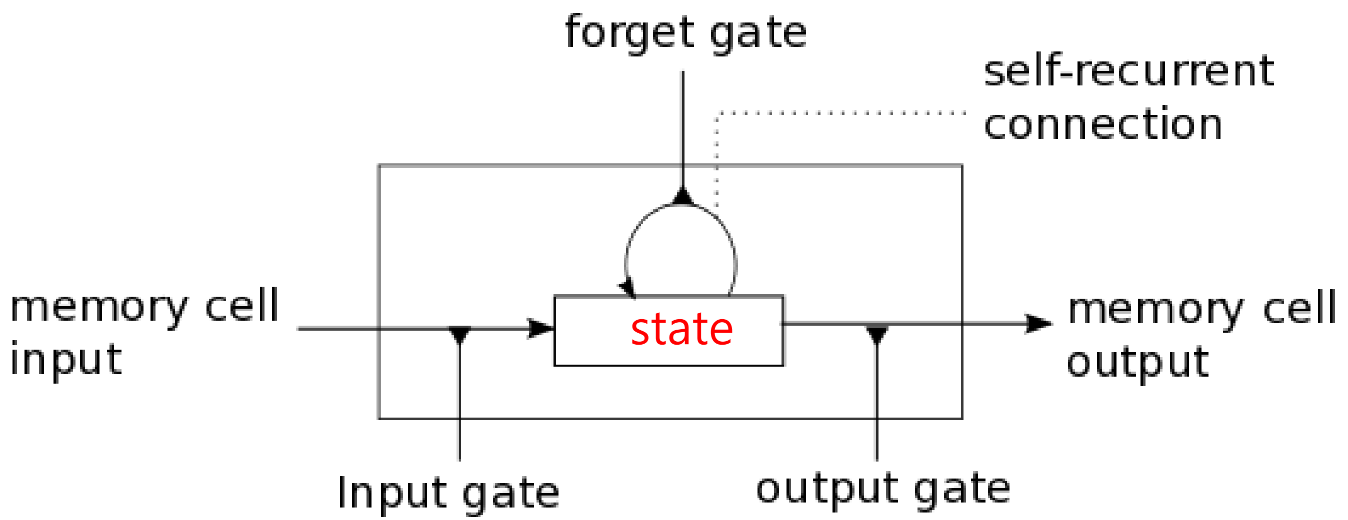
  $\alpha = 0$: 빨리 잊어버림
- 얼마나 오래 전 정보까지 기억할지를 실수 α 로 결정
- α 는 상수이거나 또는 학습되거나

Gated RNNs



- Gating units : controls the flow of information
- Gated RNNs : leaky unit의 일반화.
  Connection weights 가 시간에 따라 변할 수 있음.
- Leaky units : 정보를 오랫동안 축적
- 정보가 사용되고 나면 잊어버릴 (forget) 필요가 있음.
- 필요한 구간의 정보를 처리하고 나면 α 를 0으로 setting 하도록 학습
- Gated : self-loop의 weight를 다른 hidden unit 이 제어하도록하여 입력에 따라 시간 scale 을 dynamically변경
- Forget gate
  - 이전 단계의 정보 중, 현재 단계부터는 더 이상 필요하지 않은 정보를 제거

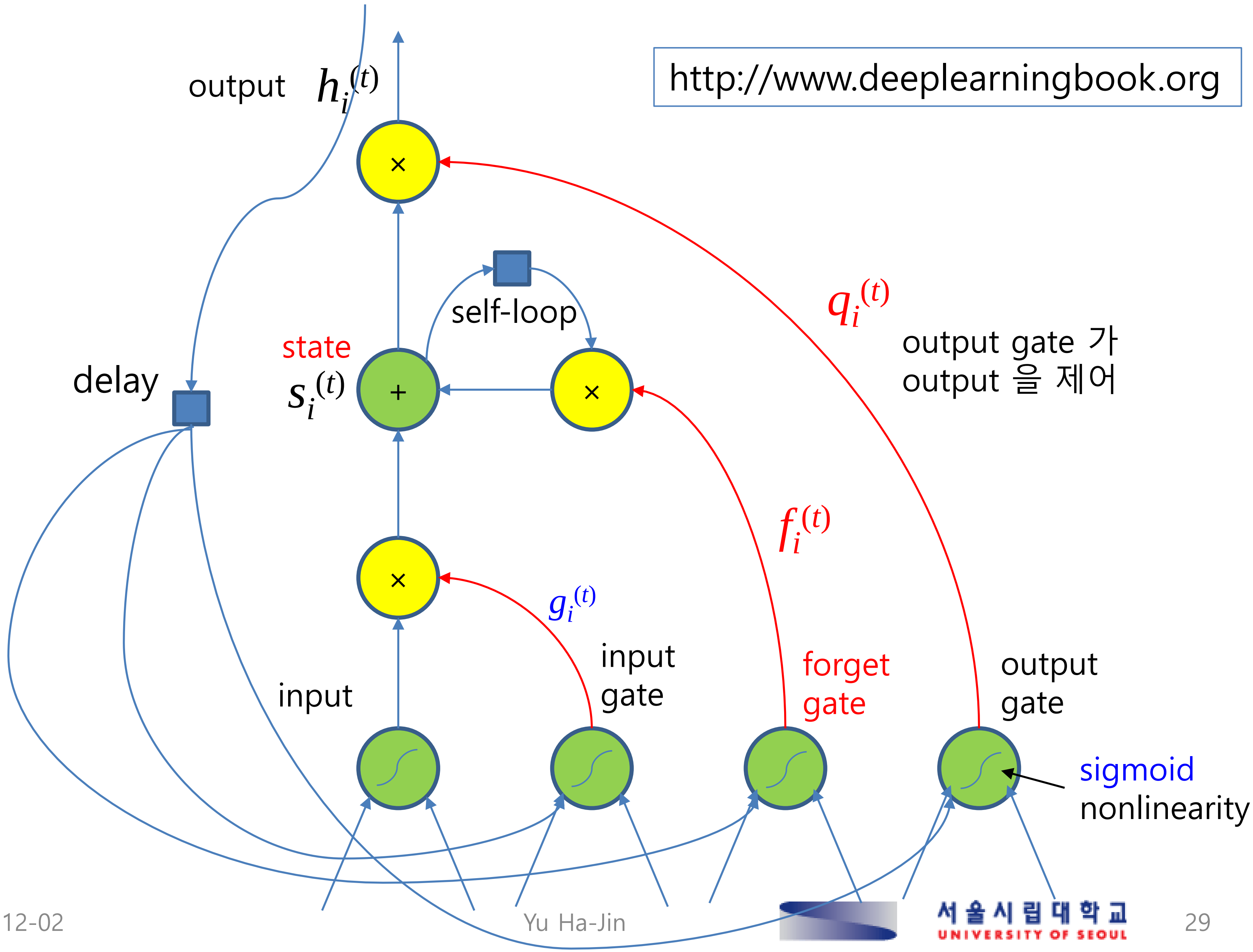
Long Short Term Memory (LSTM)



LSTM memory cell

http://deeplearning.net/tutorial/lstm.html

LSTM recurrent network "cell."



Forget gate unit

- The Self-loop weight is controlled by a *forget gate* unit $f_i^{(t)}$ (for time step t and cell i)
- A sigmoid unit to obtain a gating value between 0 and 1

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

- $x(t)$ = current input vector
- $h(t)$ = the current hidden layer vector, containing the outputs of all the LSTM cells
- $\boldsymbol{b}^f$ = biases
- $\boldsymbol{U}^f$ = input weights
- $\boldsymbol{W}^f$ = recurrent weights for the forget gates.

LSTM cell internal state $s_i^{(t)}$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \tanh\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

- $f_i^{(t)}$:conditional self-loop weight
- b :biases,
- U :input weights
- W :recurrent weights into the LSTM cell.

The *external input gate* unit $g_i^{(t)}$

- A Sigmoid unit to obtain a gating value between 0 and 1

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right)$$

Output gate

- The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the *output gate* $q_i^{(t)}$
- also uses a sigmoid unit for gating:

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

- parameters

  b^o:biases,

  U^o:input weights,

  W^o :recurrent weights

Long Short Term Memory (LSTM)

- 모든 weights를 ***W*** 하나로 표현하면

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

stack

LSTM

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

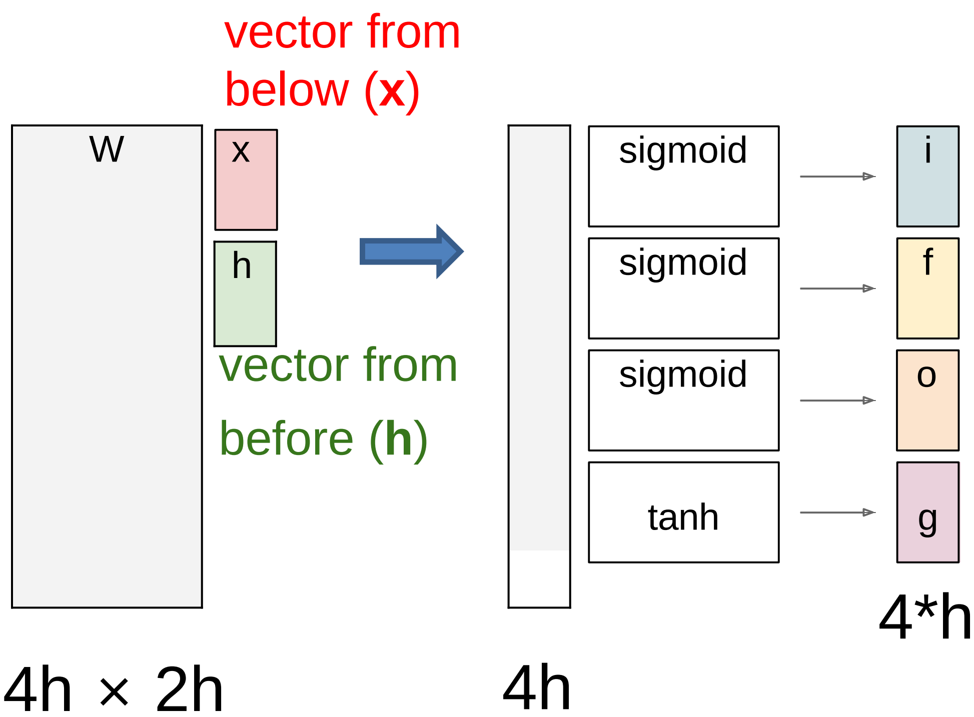
state

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation, 1997
http://cs231n.stanford.edu/

Long Short Term Memory (LSTM)



i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell

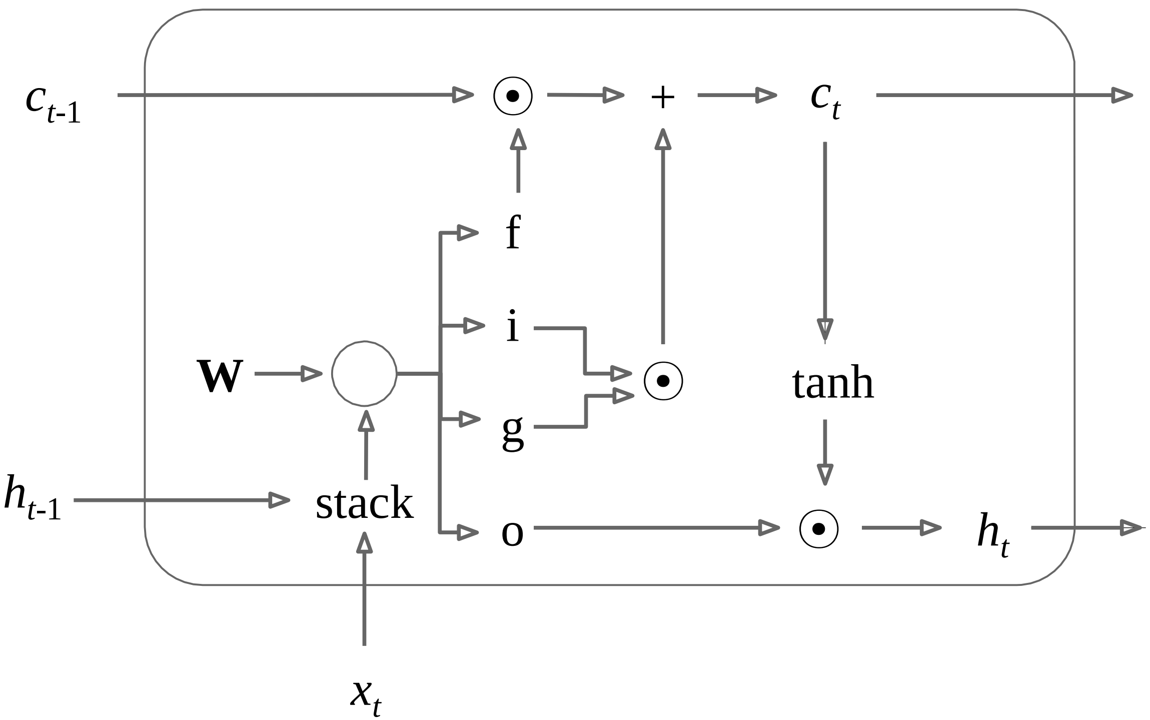
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

http://cs231n.stanford.edu/

LSTM



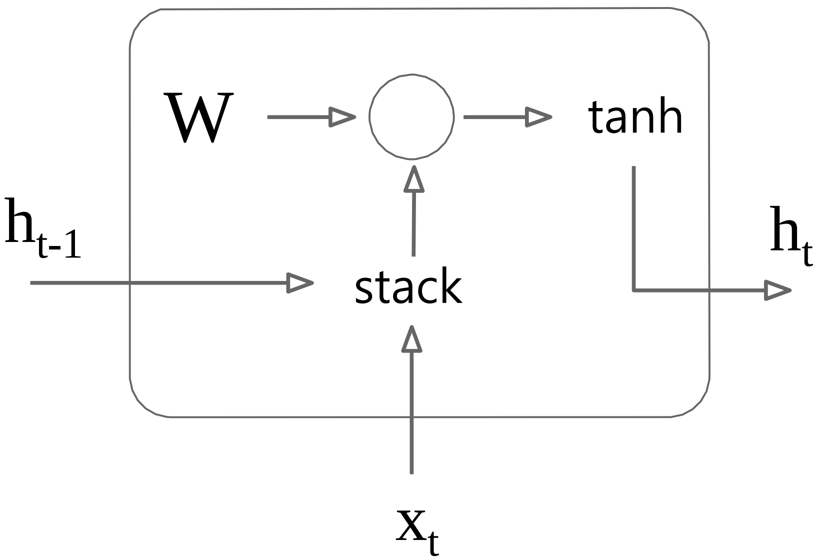
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

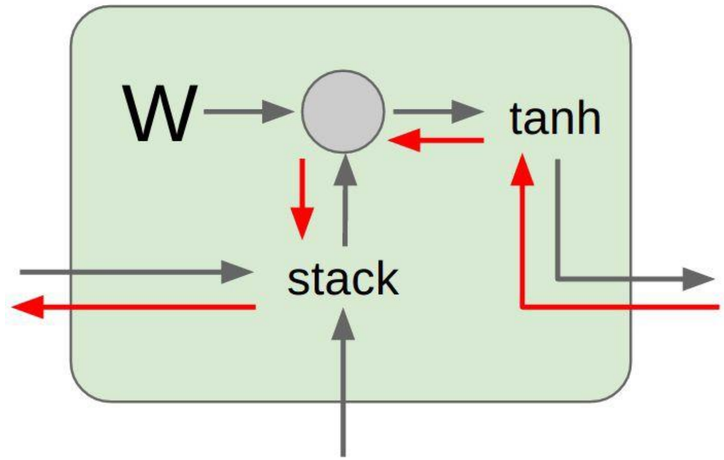
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

http://cs231n.stanford.edu/

Vanilla RNN Gradient Flow





$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

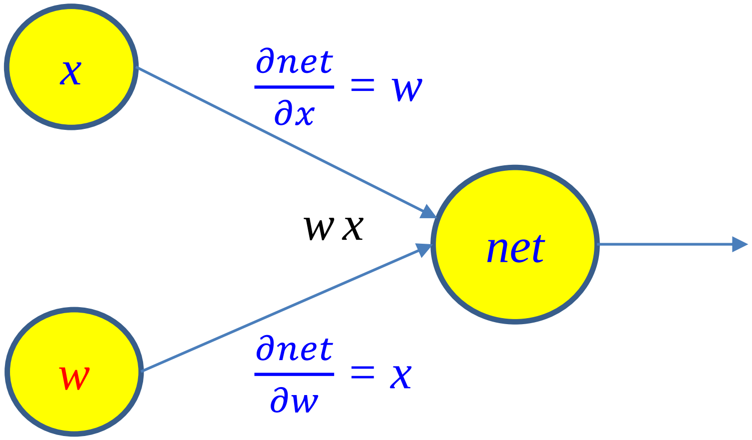
Backpropagation 할 때 W 가 반복해서 곱해짐 →

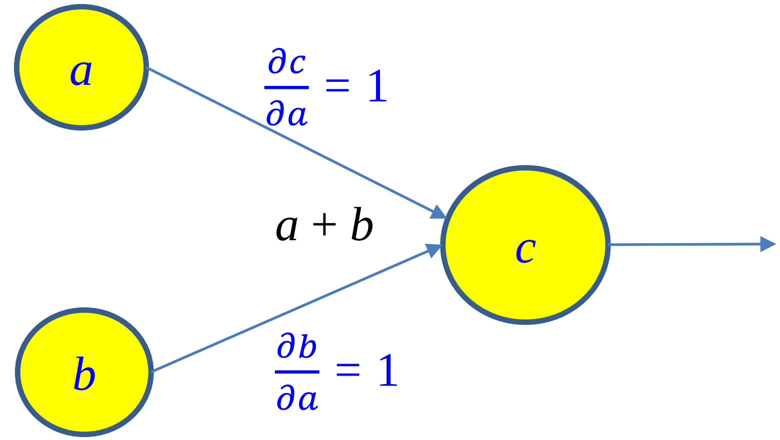
Exploding gradients

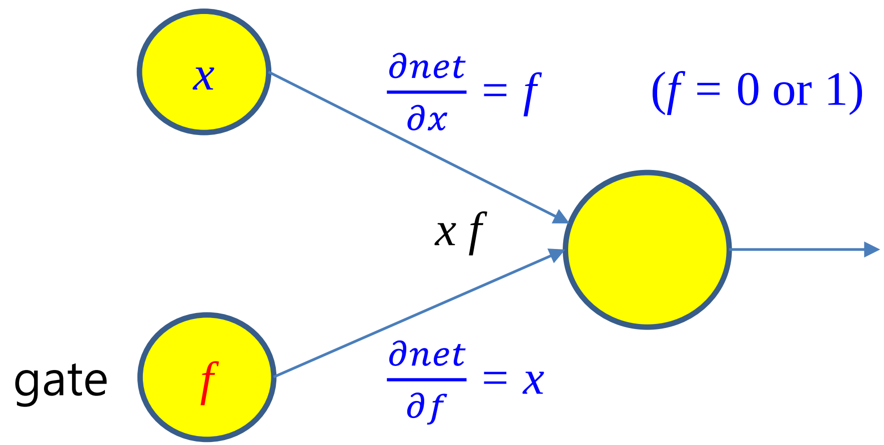
Vanishing gradients

http://cs231n.stanford.edu/
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Recall: gradient calculation



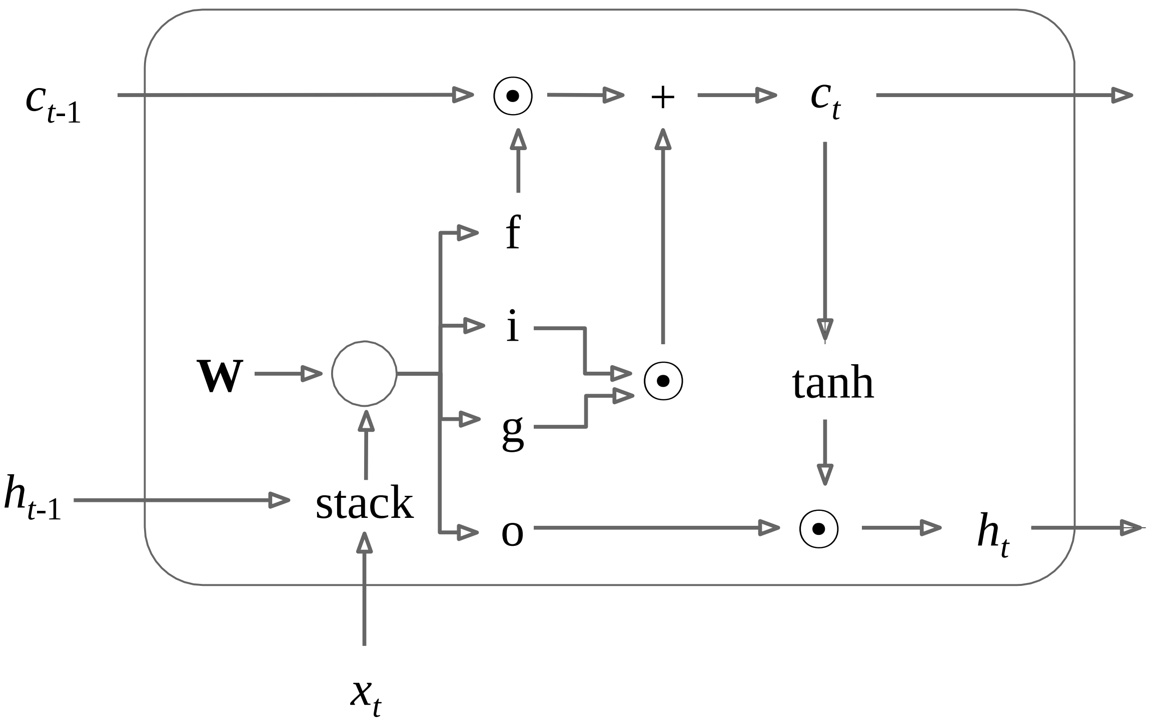




- 곱셈보다 덧셈이 유리함.

LSTM: Gradient Flow

Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by **W**



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

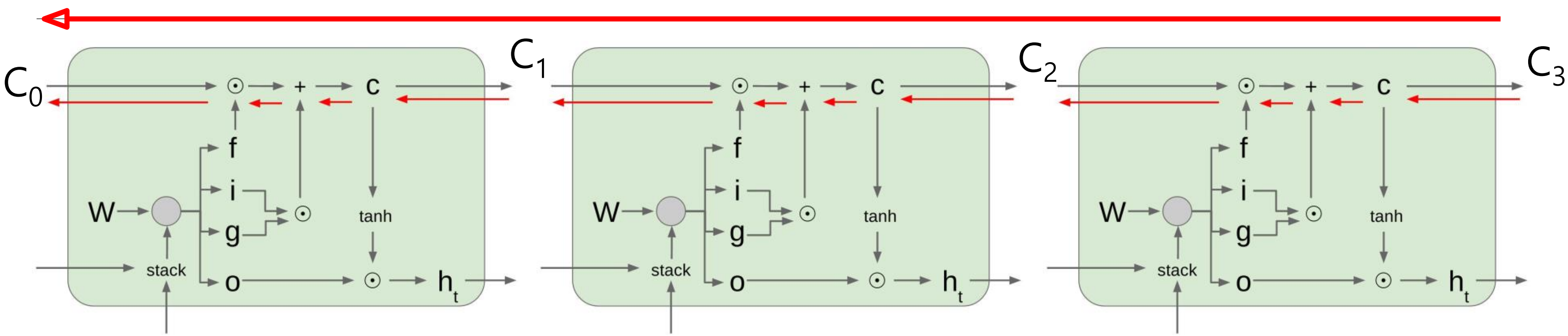
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

http://cs231n.stanford.edu/

LSTM: Gradient Flow

gradient 계산할 때 W가 곱해지지 않음



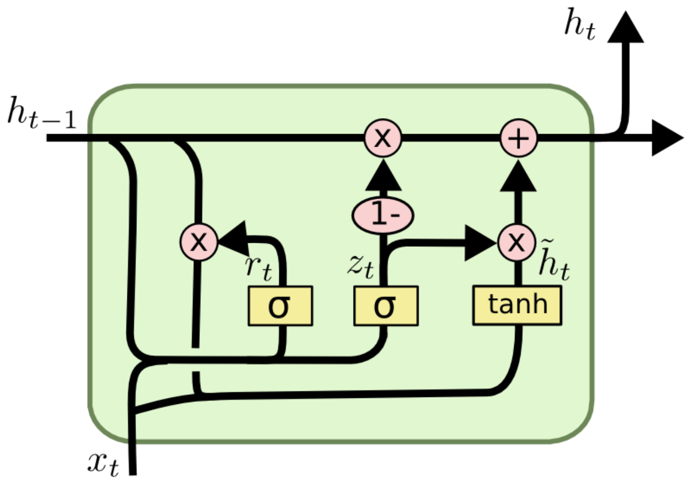
f 값은 일정하지 않음.

→ Vanishing gradients 문제 해결

http://cs231n.stanford.edu/

Gated Recurrent Unit (GRU)

- Two gates: reset gate r and update gate z
- No Output gate



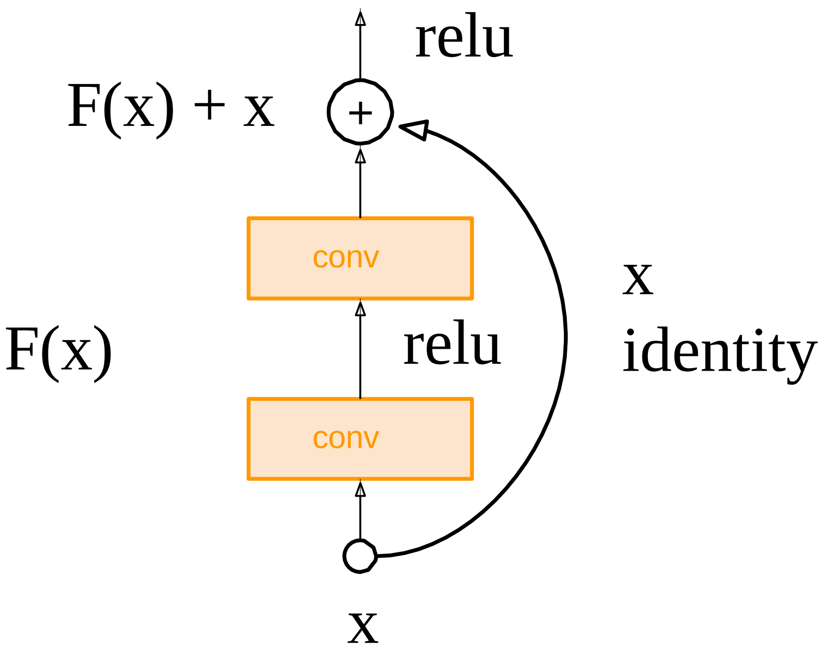
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

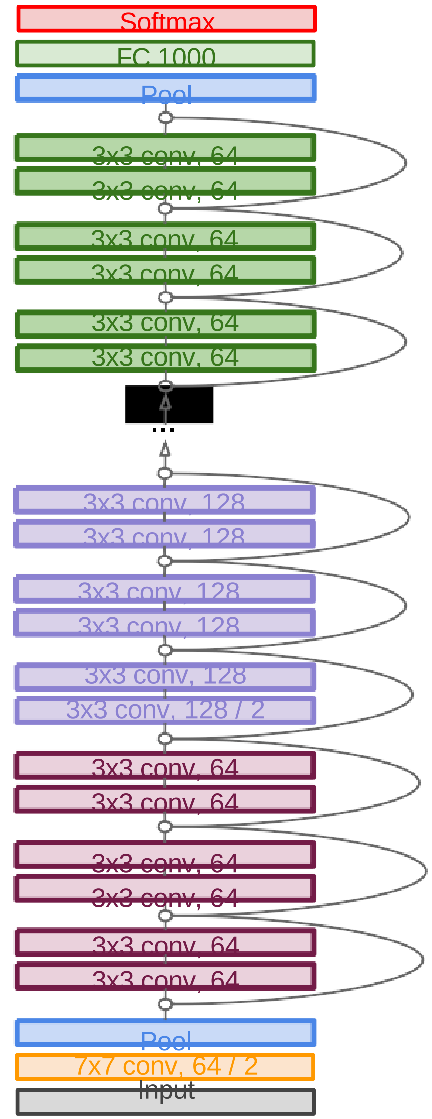
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

Residual Net과 비슷한 원리





- Gradient Flow 를 위한 통로를 만들어 깊은 층 구현 가능