Recurrent Neural Networks

And Long Short Term Memory

"어제의 생각이 오늘의 당신을 만들고 오늘의 생각이 내일의 당신을 만든다."
블레즈 파스칼

Outline

- Introduction
- Motivation
- RNN architecture
- RNN problems
- LSTM
- How LSTM solves the problem
- Paper experiments
- Conclusions

Introduction

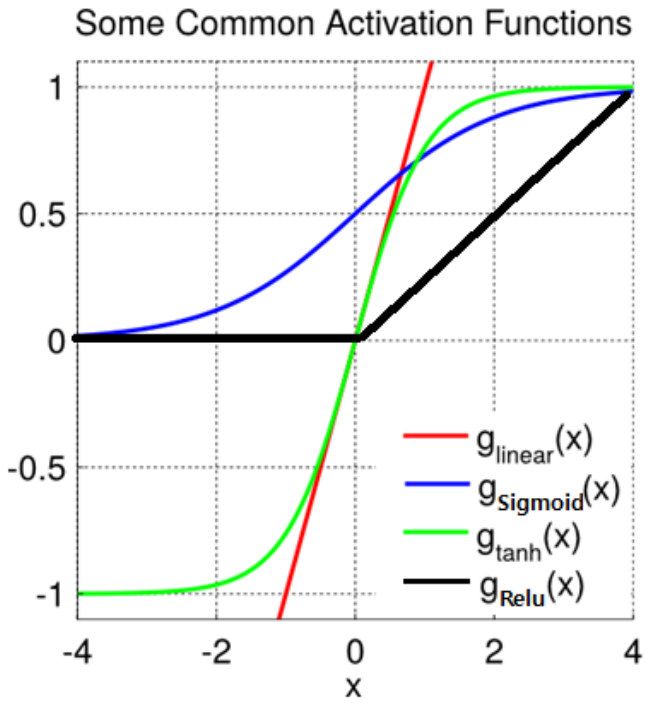
- RNN were introduced in the late 80's.
- Hochreiter discovers the 'vanishing gradients' problem in 1991.
- Long Short Term Memory published in 1997.
- LSTM a recurrent network to overcome the problem.

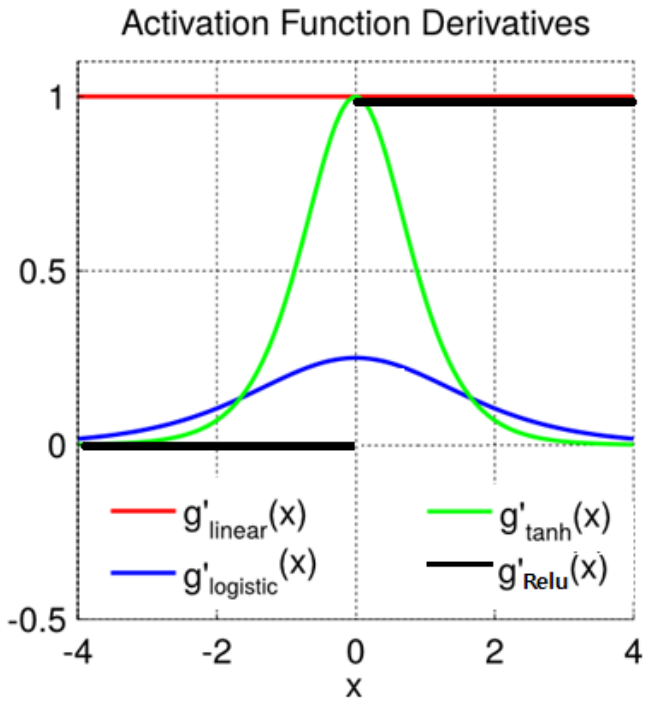
Motivation

- Feed forward networks accept a fixed-sized vector as input and produce a fixed-sized vector as output
- fixed amount of computational steps
- recurrent nets allow us to operate over *sequences* of vectors

Activation Functions

인공신경망에서 활성, 비활성을 결정하는데 사용되는 함수





- Activation Functions과 미분

sigmoid 함수 : $\sigma(x) = 1 / (1+ e^{-x})$ → $\sigma'(x) = \sigma(x)(1-\sigma(x))$

하이퍼볼릭 탄젠트 : $\tanh(x)$ → $\tanh'(x) = 2\sigma(2x) - 1 = (e^{x} - e^{-x}) / (e^{x} + e^{-x}) = 1 - \tanh^2(x)$

Rectified Linear Unit (ReLU) : $f(x) = \max(0, x)$ → $f'(x) = 0$ if $x \le 0$, $f'(x) = 1$ if $x > 0$

Motivation

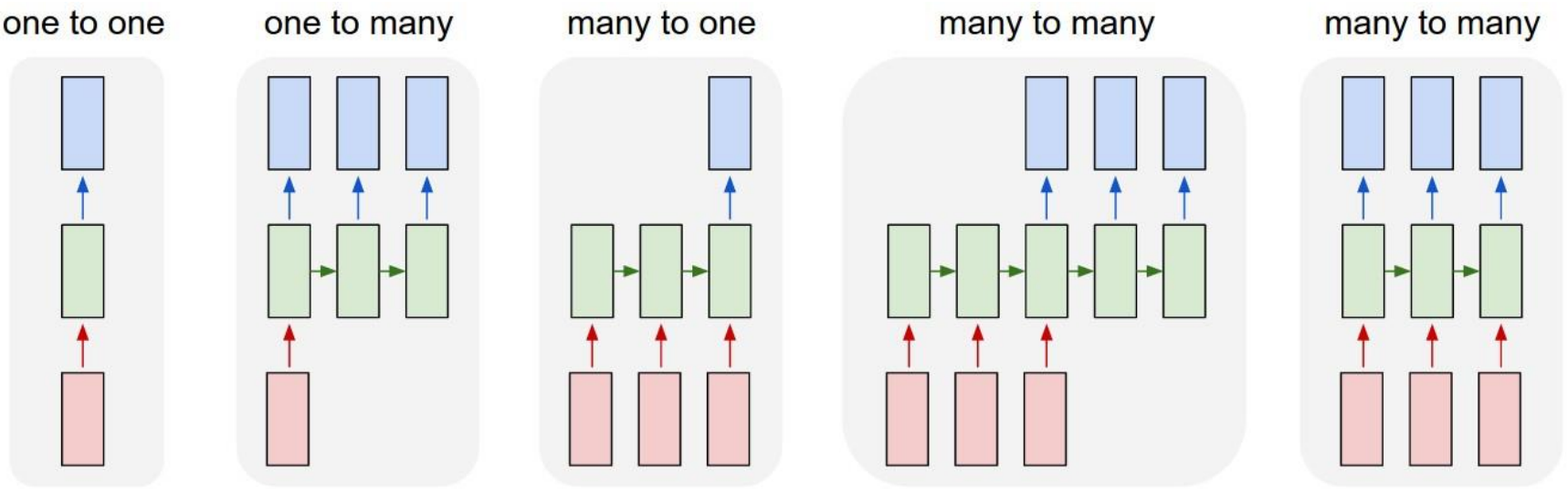


Image classification

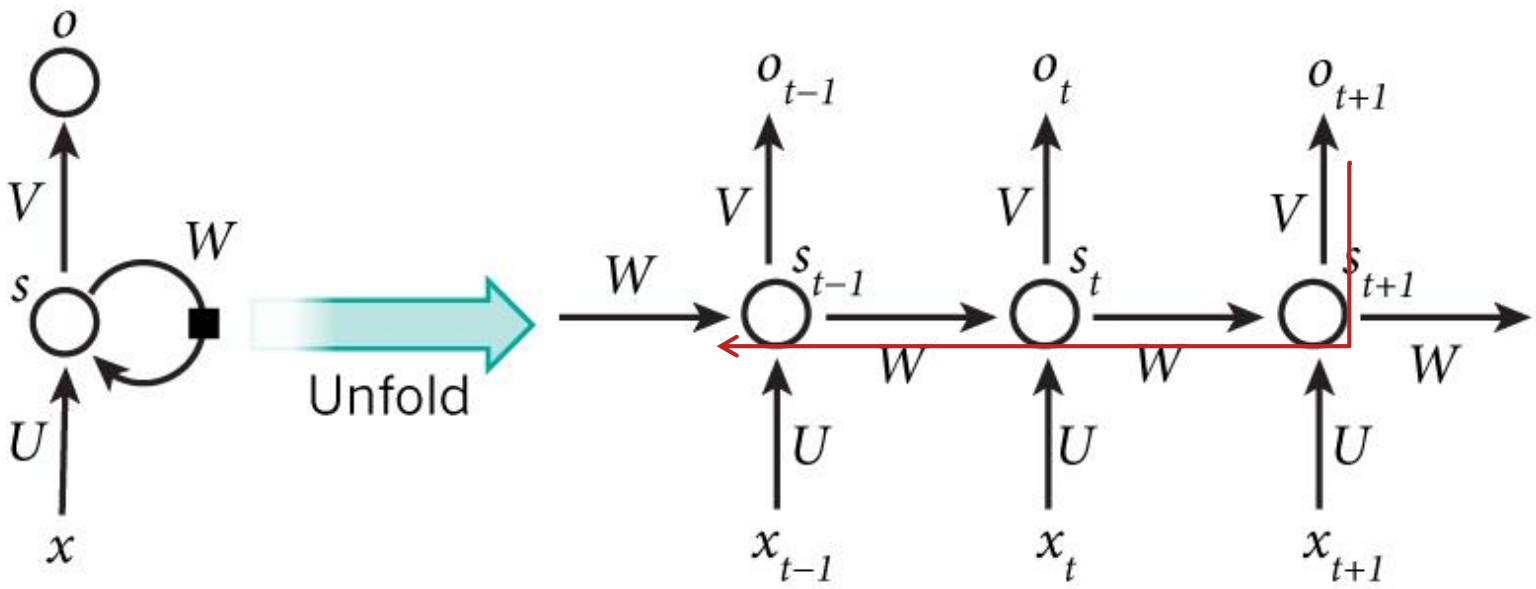
Image captioning

Sentiment analysis

Machine translation

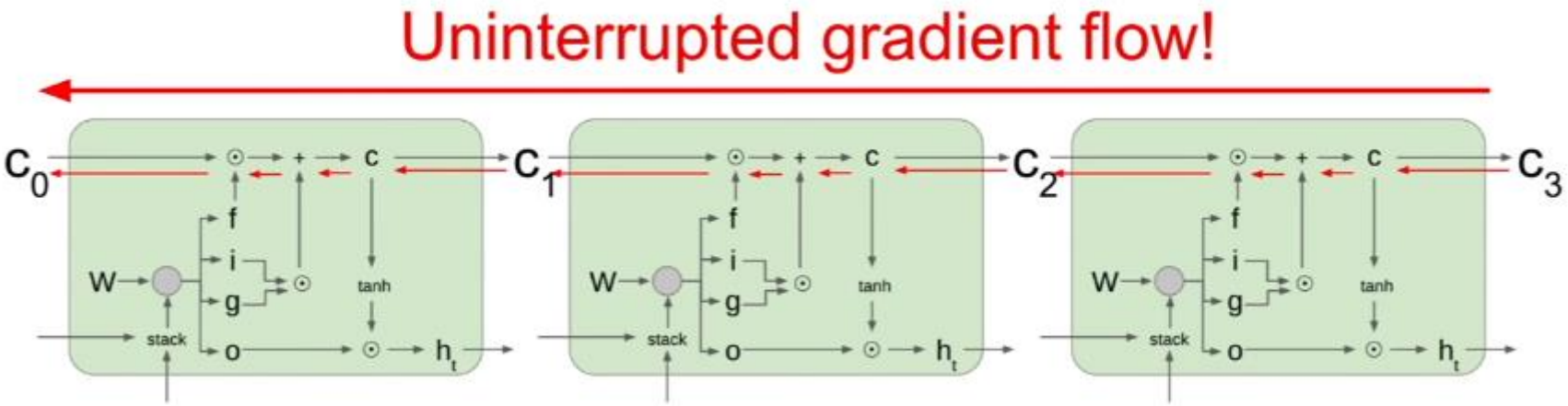
Synced sequence (video classification 여러 개의 이미지 프레임에 대해 여러 개의 설명이나 번역 형태로 결과를 반환)

RNN Architecture



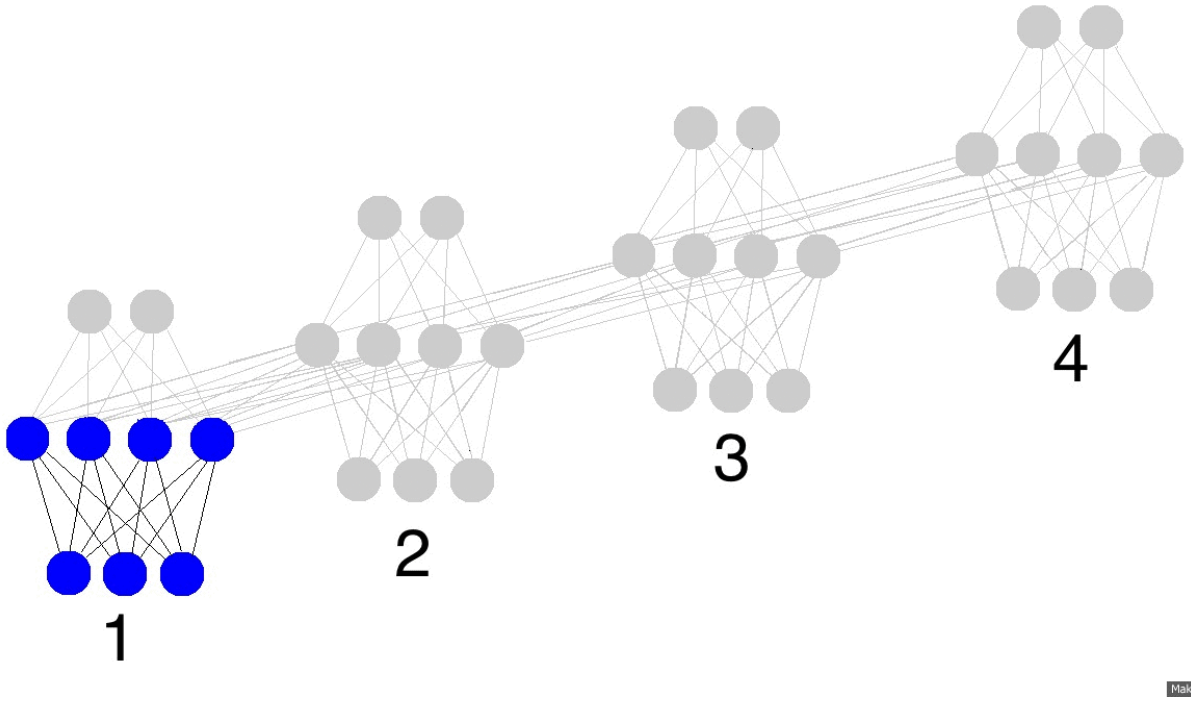
- The recurrent network can be converted into a feed forward network by **unfolding over time**
- long-term 정보 손실 문제 - LSTM(Long Short Term Memory)으로 해결

RNN Architecture



Gradient flows smoothly during Backprop

RNN Architecture



RNN Architecture

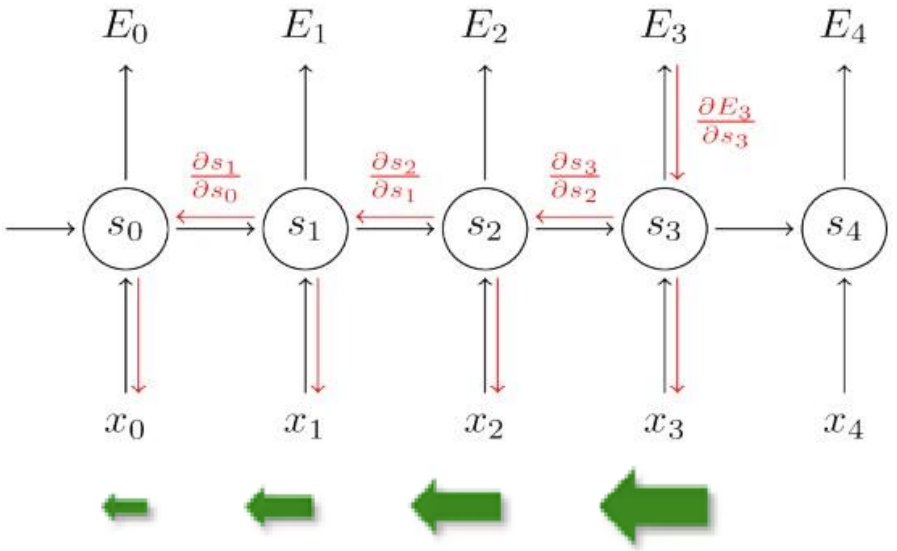
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

But $s_3 = \tanh(Ux_t + Ws_2)$

S_3 depends on s_2, which depends on W and s_1, and so on.

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



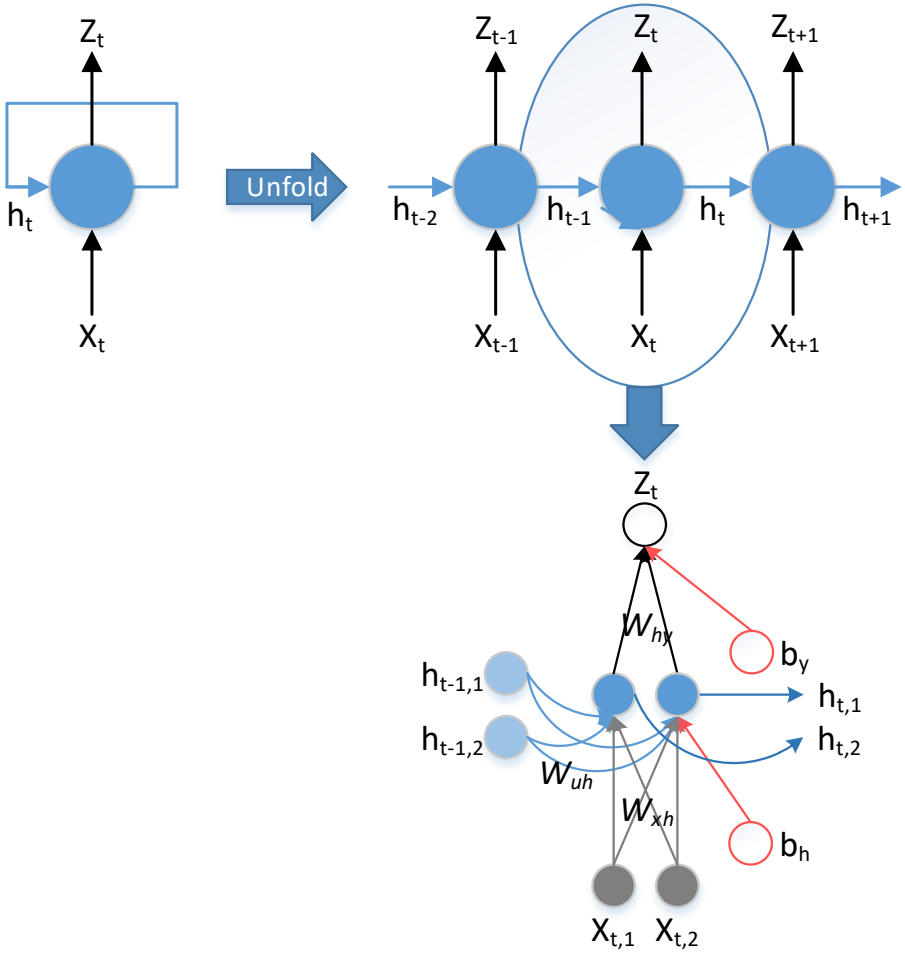
Outline - LSTM

- Introduction
- Motivation
- RNN architecture
- RNN problems
- Long Short Term Memory
- How LSTM solves the problem
- Paper experiments
- Conclusions

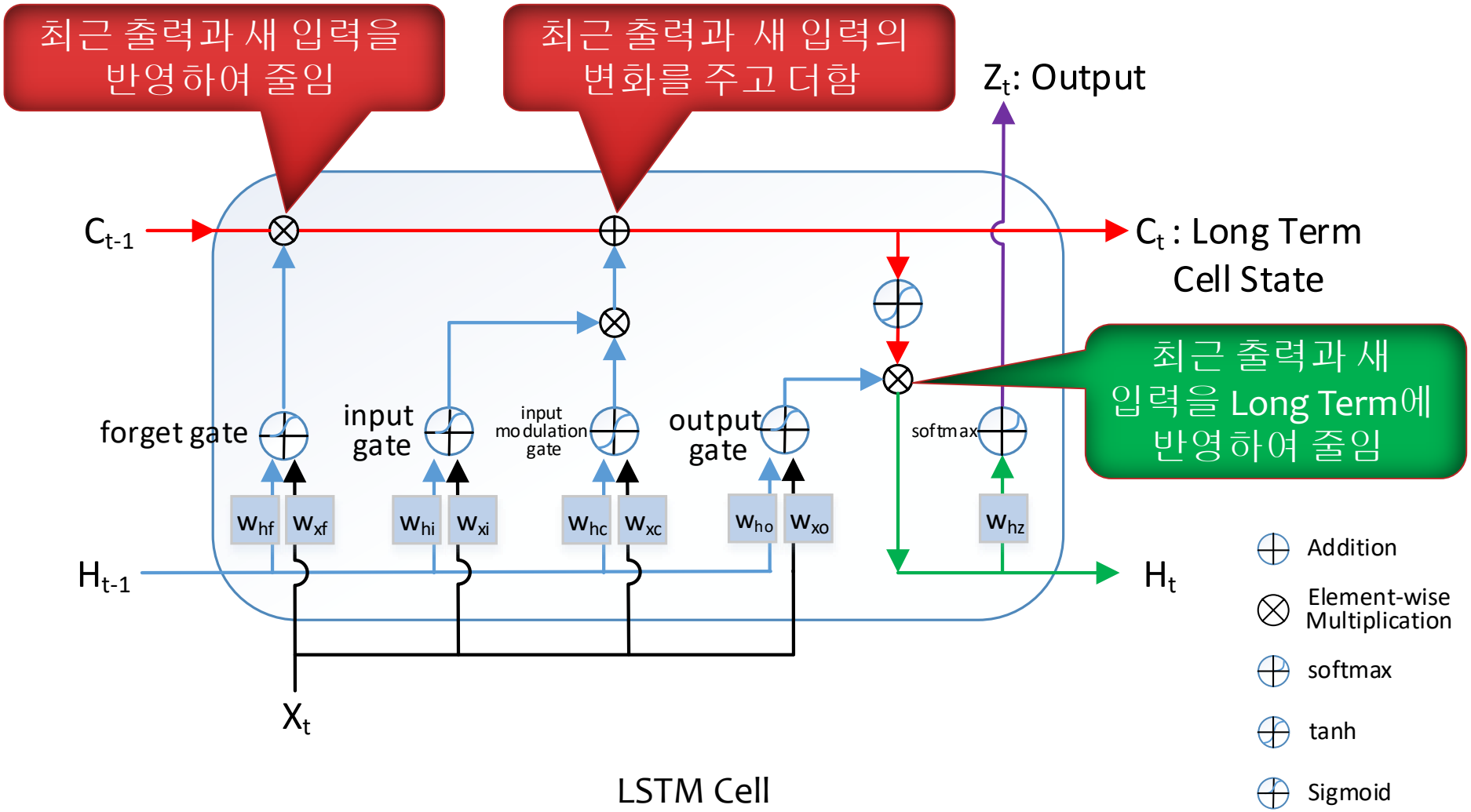
LSTM - introduction

- LSTM was invented to solve the vanishing gradients problem.
- LSTM maintain a more constant error flow in the ba ckpropogation process.
- LSTM can learn over more than 1000 time steps , an d thus can handle large sequences that are linked re motely.

RNN Architecture

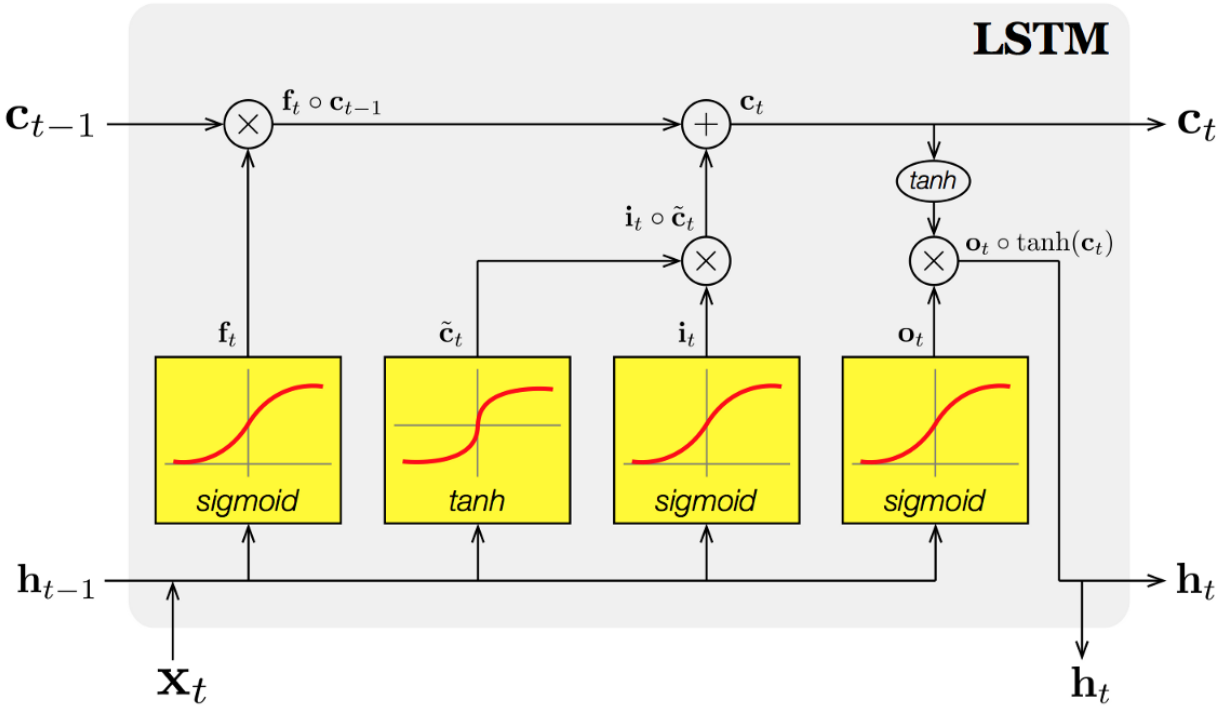


LSTM Architecture - overview



LSTM Cell

LSTM Architecture - overview



Gating variables

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

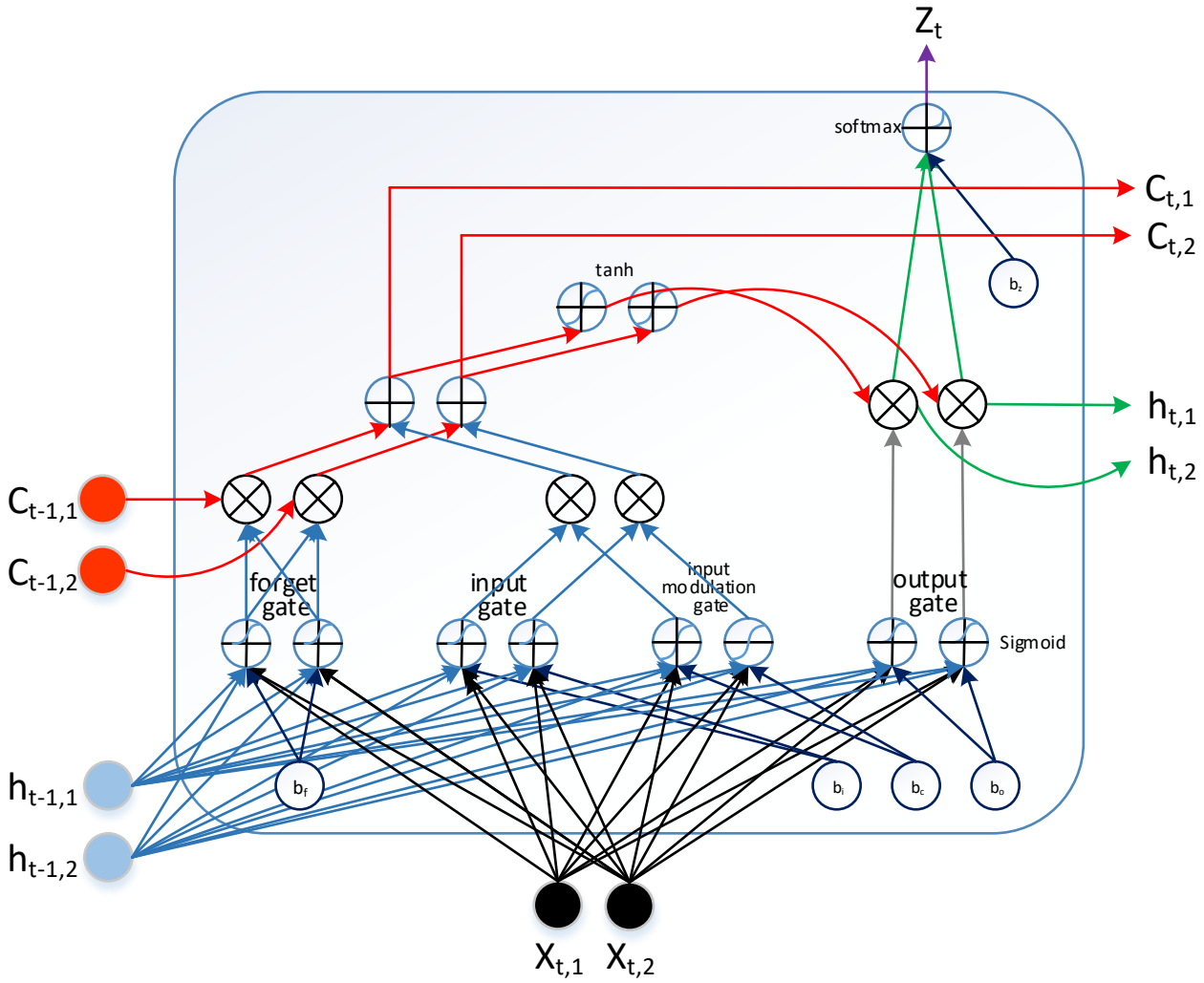
Candidate (memory) cell state

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$

Cell & Hidden state

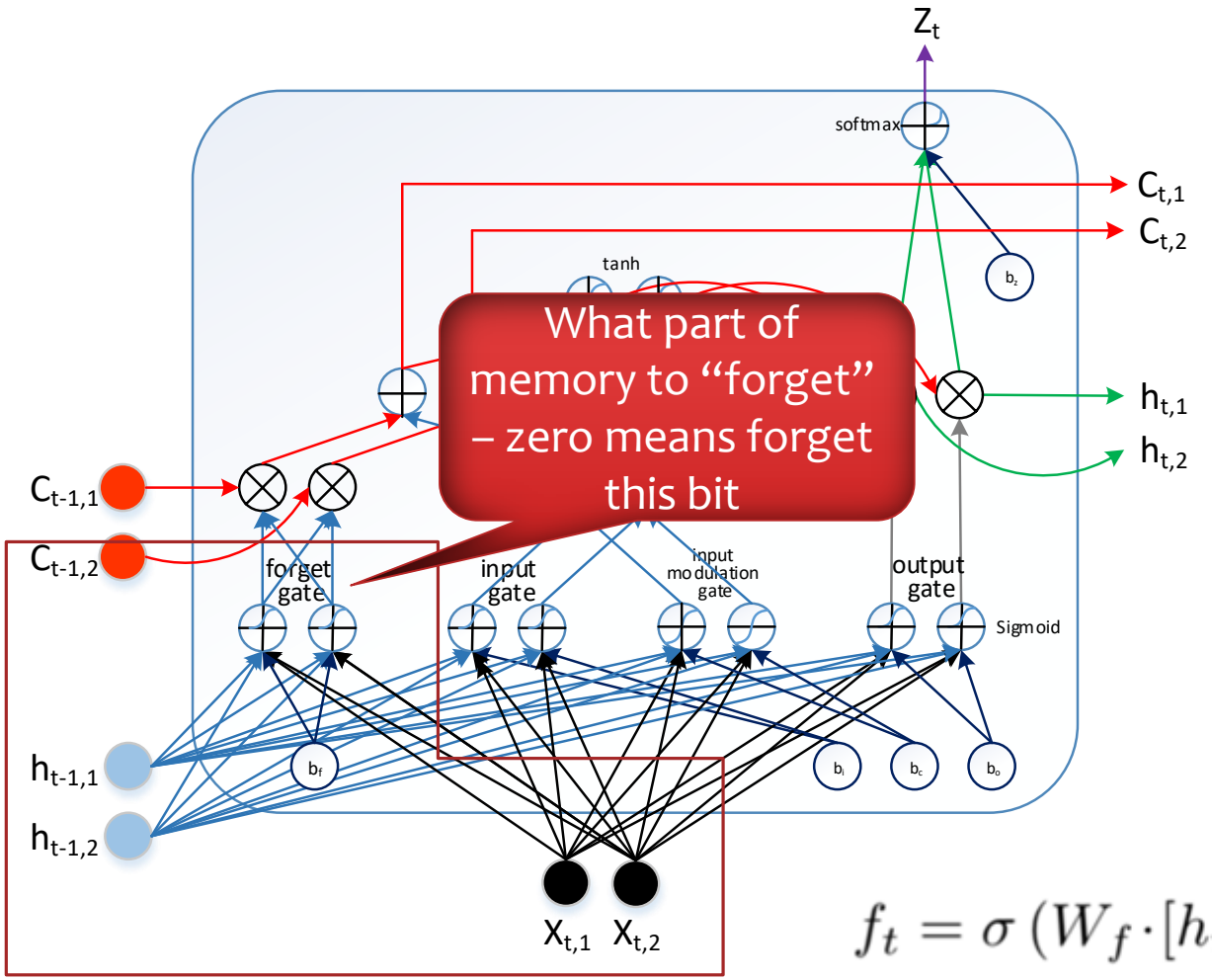
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

LSTM Architecture - overview



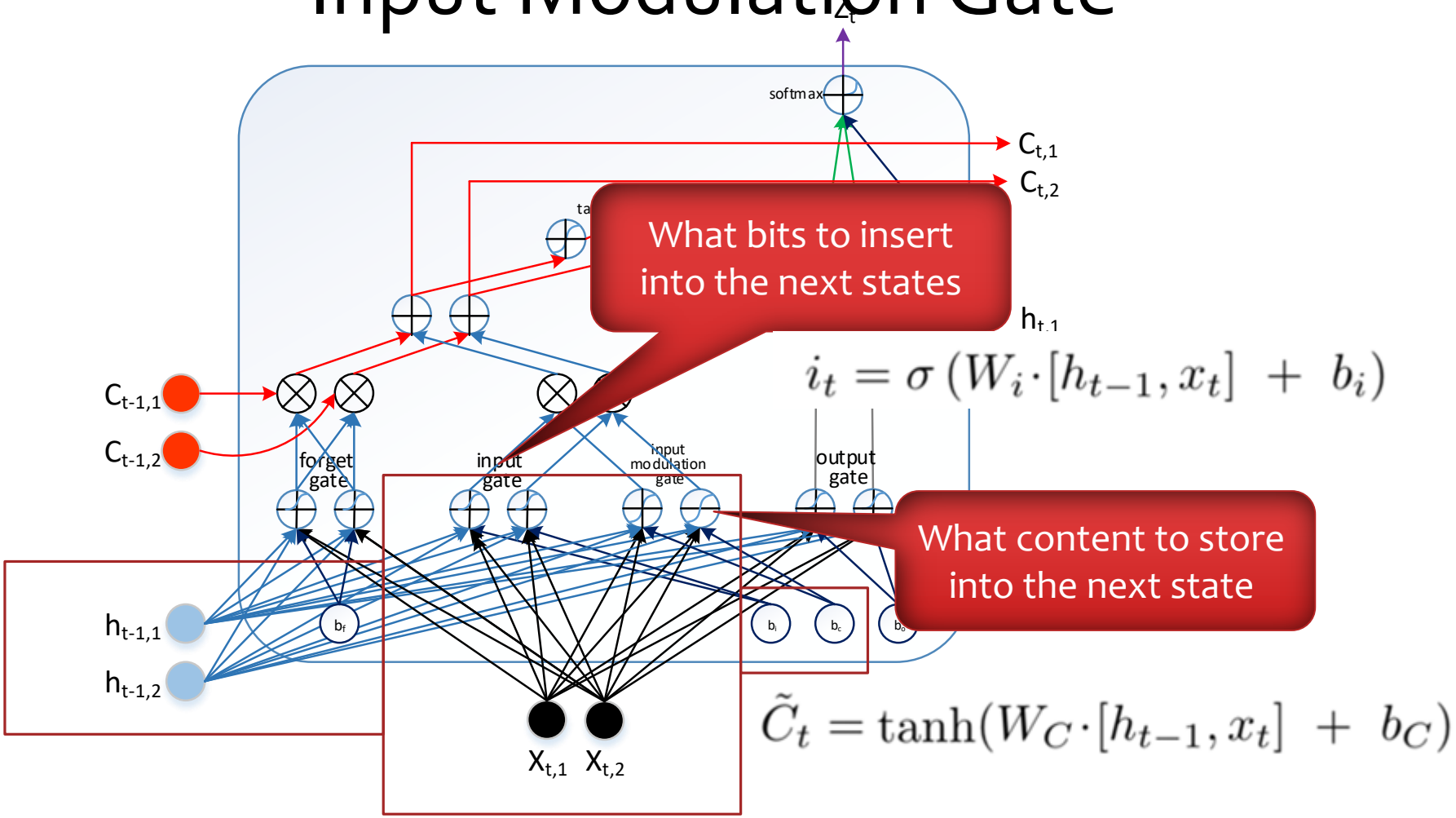
LSTM Cell

LSTM – Forget Gate



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

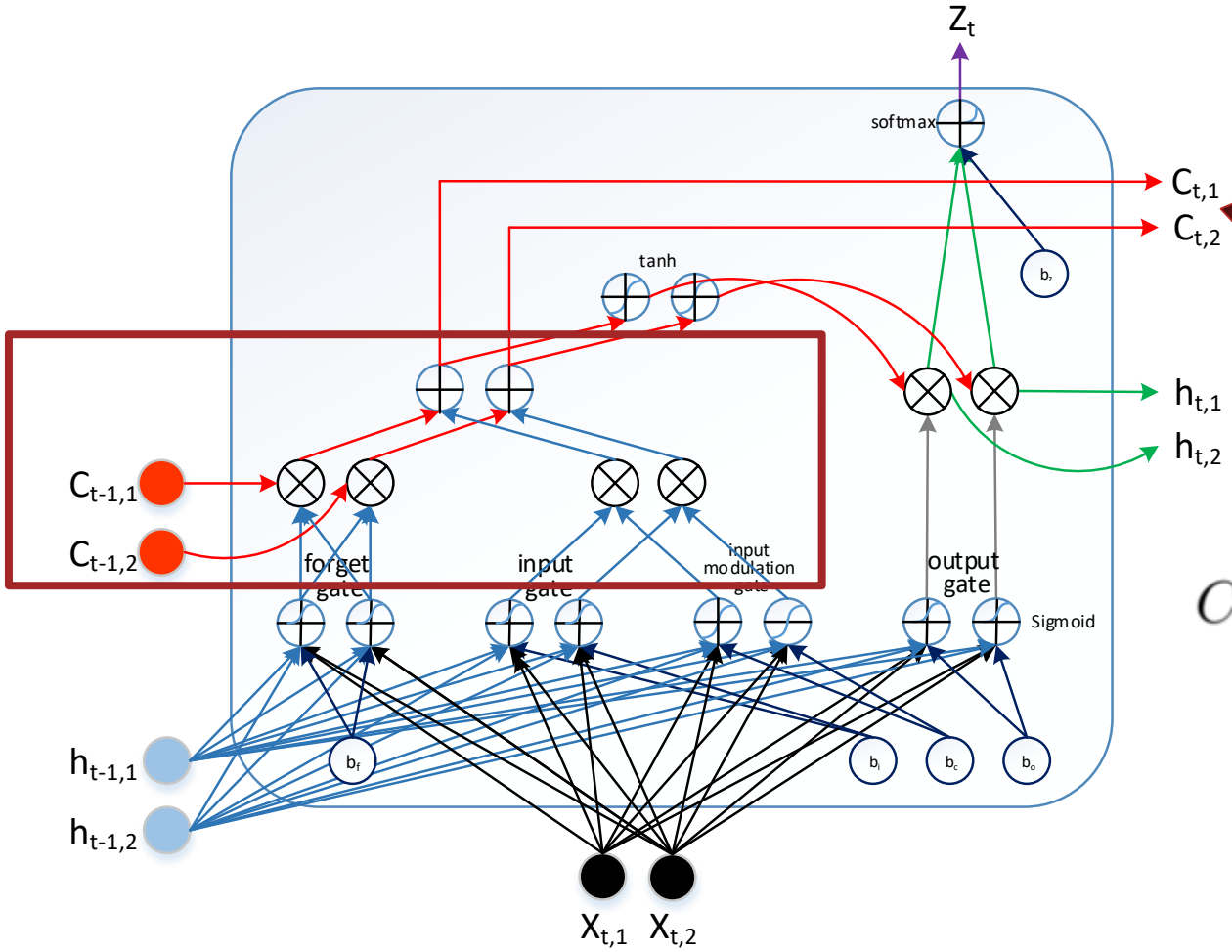
LSTM – Input Gate and Input Modulation Gate



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

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

LSTM – Long Term memory cell content



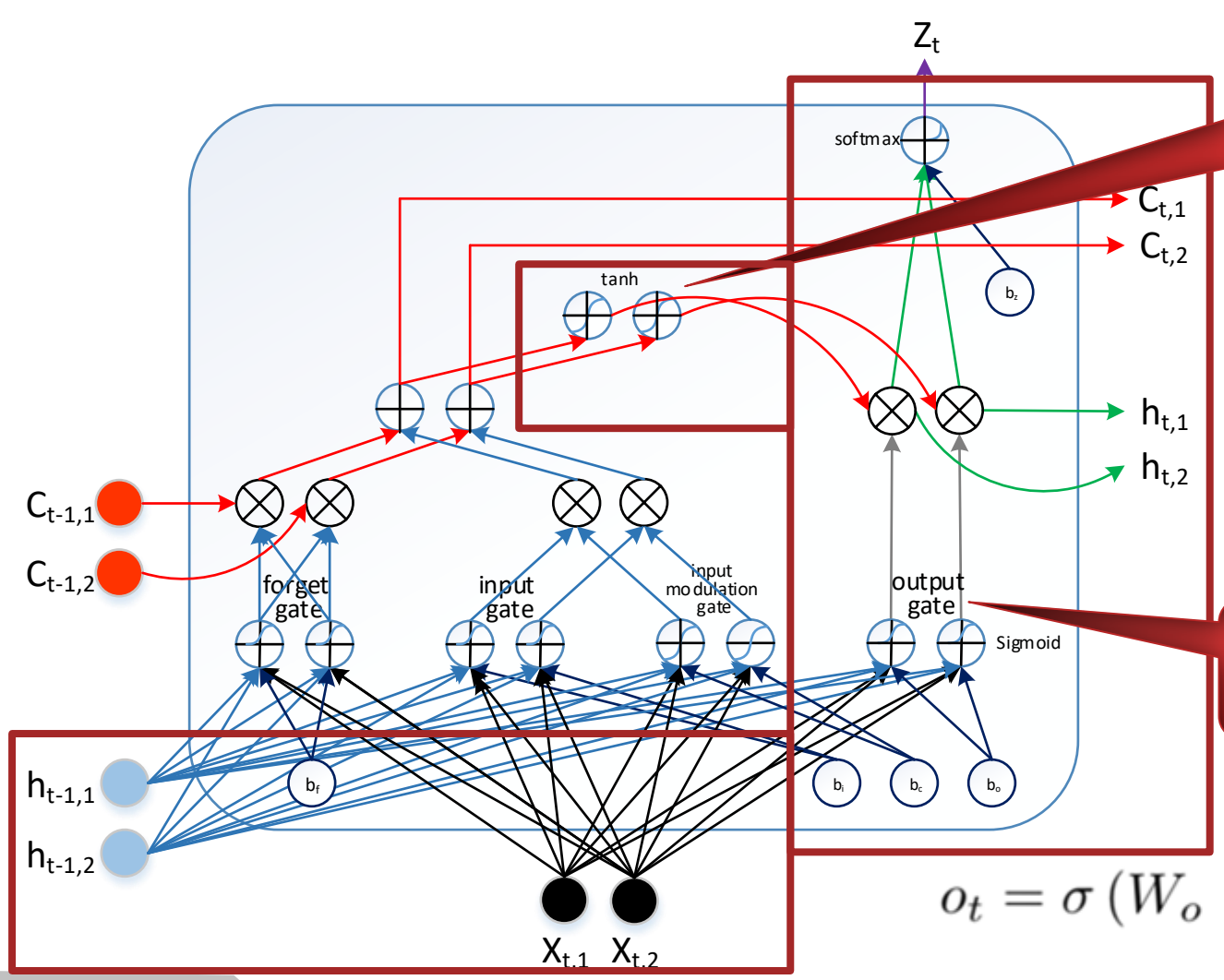
Next memory cell content – mixture of not-forgotten part of previous cell and insertion

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

⊕ 상승 효과

⊗ 억제 효과

LSTM – Output to Next Cell



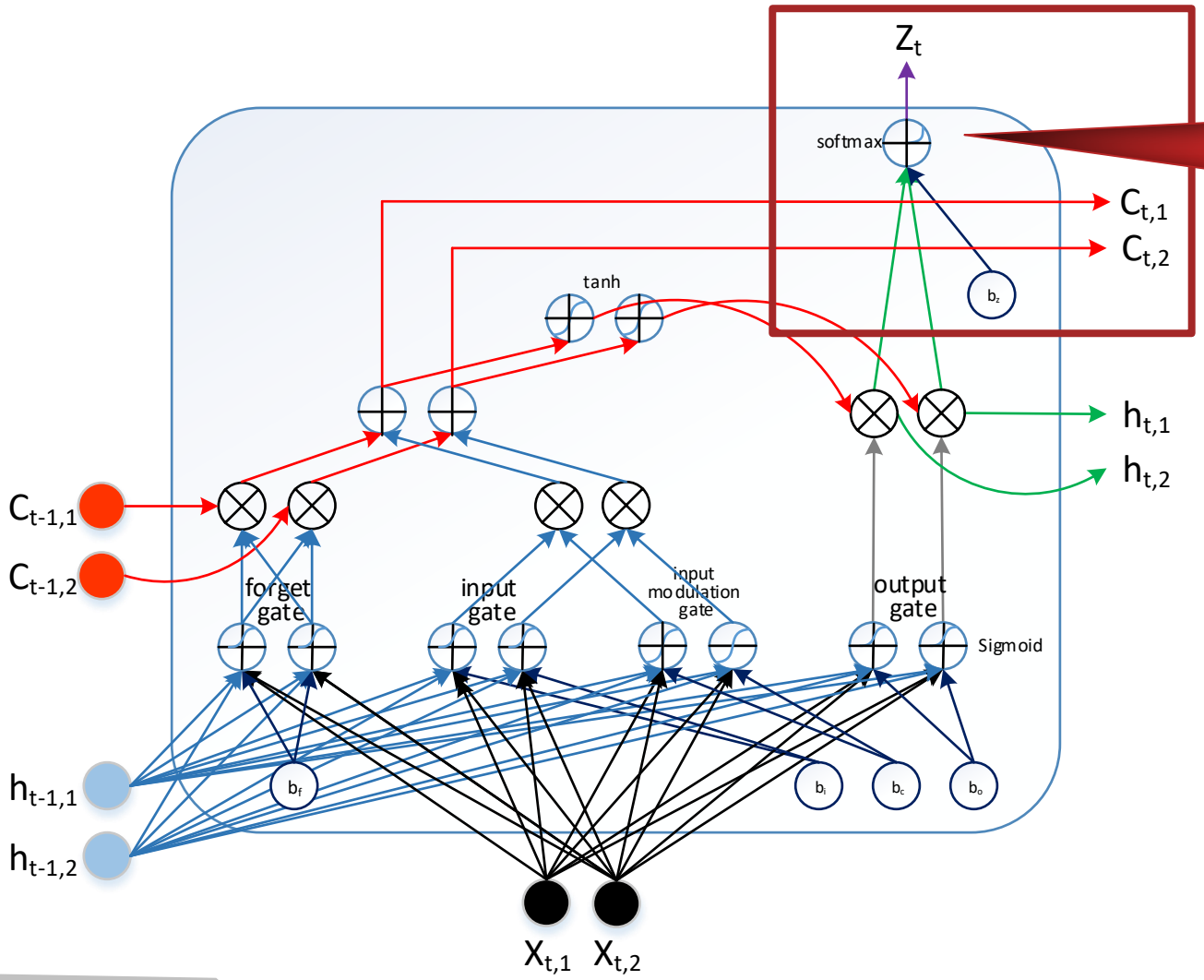
tanh maps bits to [-1,+1] range

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

What part of cell to output

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

LSTM – Softmax Output

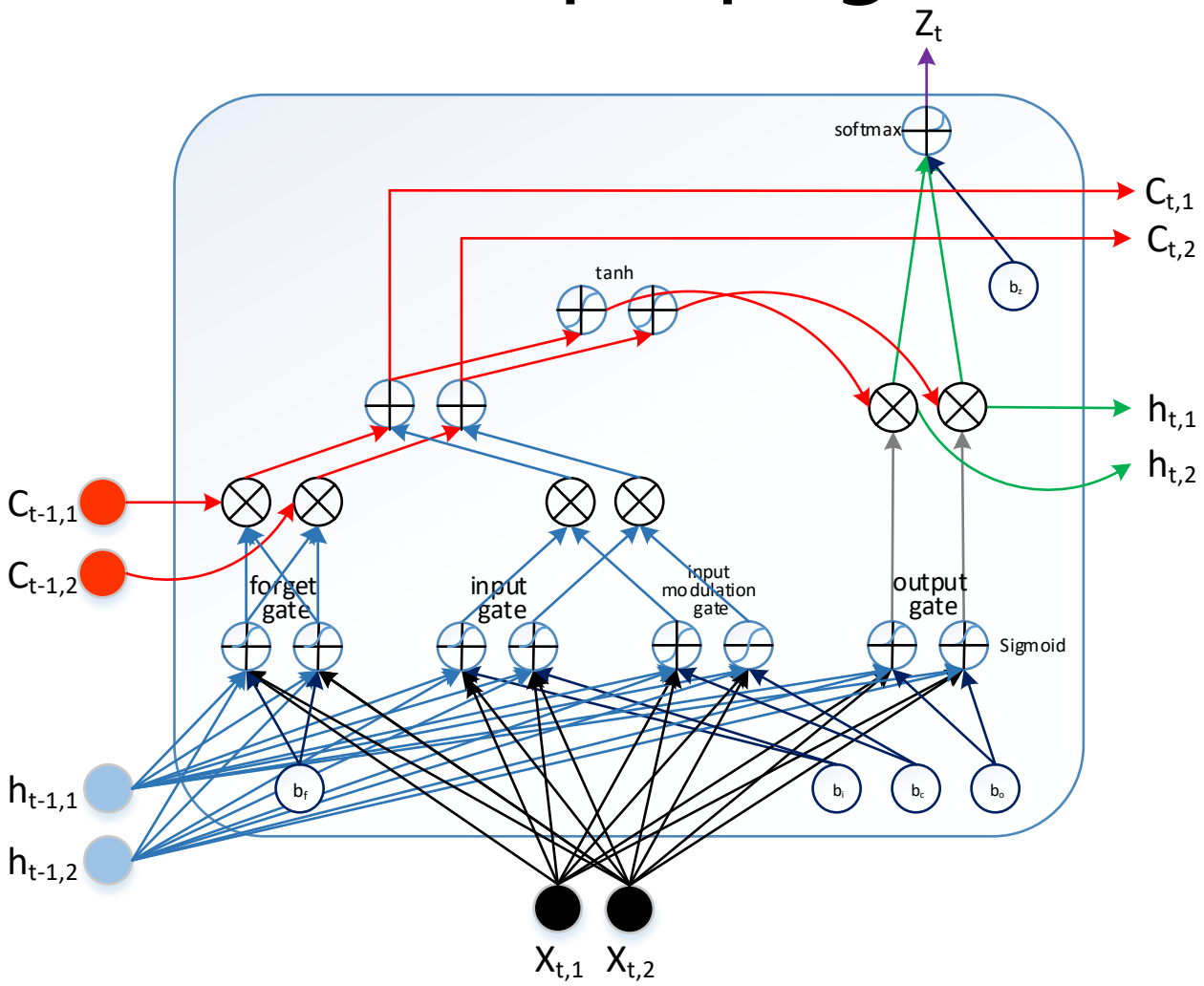


Softmax를 구하여
목표값과의 오차 계산

$$Z_t = s(w_{hz} + b_z)$$

$$s(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

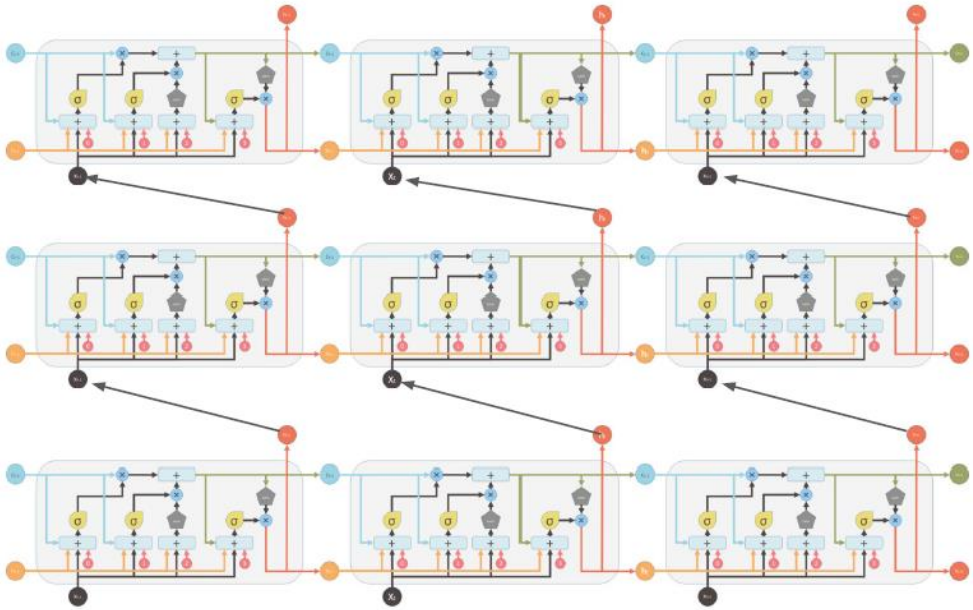
LSTM Backpropogation



LSTM Cell

Deep LSTM

- Deep LSTMs can be created by stacking multiple LSTM layers vertically, with the output sequence of one layer forming the input sequence of the next (in addition to recurrent connections within the same layer)
- Increases the number of parameters - but given sufficient data, performs significantly better than single-layer LSTMs (Graves et al. 2013)
- Dropout usually applied only to non-recurrent edges, including between layers



LSTM Cell

Advantages of LSTM

- For long time lag problems, LSTM can handle noise and continuous values.
- No parameter fine tuning.
- Memory for long time periods