Lecture 14: Recurrent Neural Networks II: LSTM

Xuming He
SIST, ShanghaiTech
Fall, 2019

Previously on RNNs

- RNN
  - RNNs allow a lot of flexibility in architecture design
  - BP through time is used to compute the gradient descent update
- Problems
  - The updates are mathematically correct, but gradient descent fails because the gradients explode or vanish
  - This limits the scope of the dependencies over time

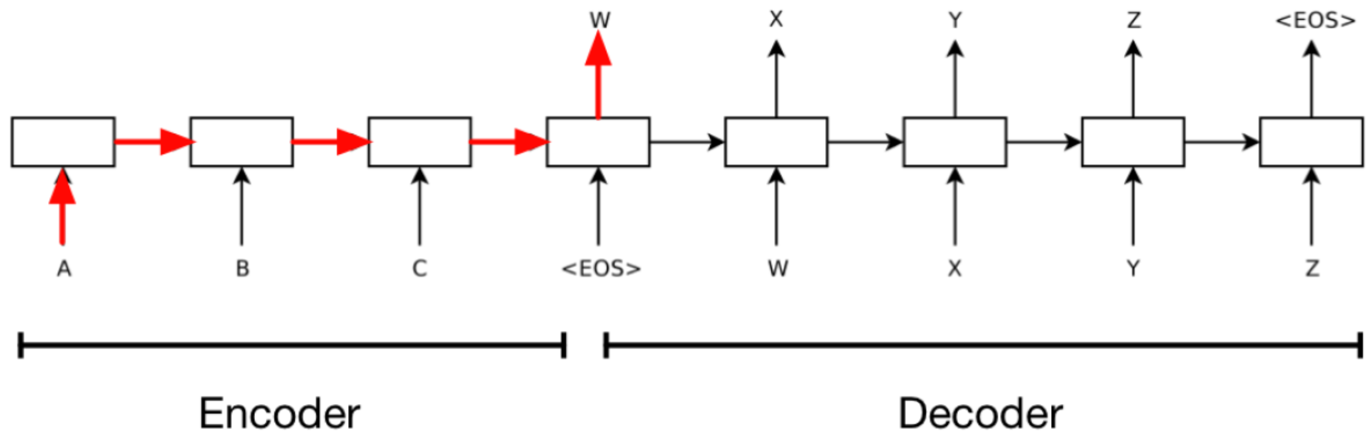
Outline

- Recurrent Neural Networks
  - Gradient problems in training RNNs
  - Stabilizing RNN training
- Long-Term Short Term Memory (LSTM)
  - LSTM/GRU unit
  - RNNs with LSTM

Acknowledgement: Feifei Li et al's cs231n notes

Why gradients explode or vanish

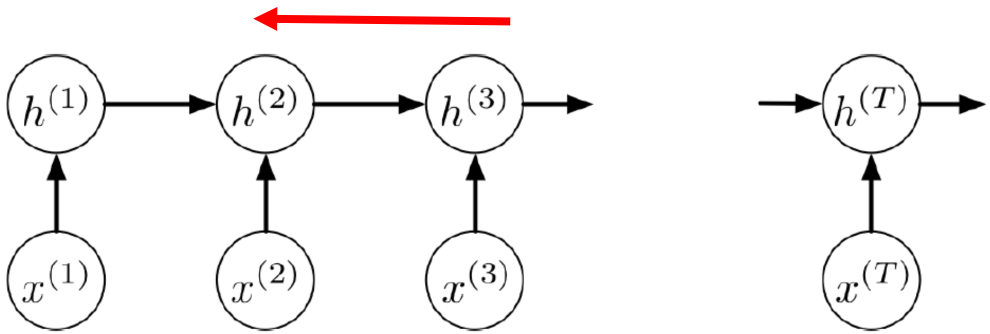
- Motivating example: machine translation



- The derivatives need to travel over this entire pathway
  - A typical sentence length is about 20 words

Why gradients explode or vanish

- Motivating example: machine translation
  - Consider a univariate version of the encoder network



$$z^{(t+1)} = wh^{(t)} + vx^{(t+1)}$$
$$h^{(t)} = \phi(z^{(t)})$$

Backprop updates:

$$\overline{h^{(t)}} = \overline{z^{(t+1)}}\, w$$
$$\overline{z^{(t)}} = \overline{h^{(t)}}\, \phi'(z^{(t)})$$

Applying this recursively:

$$\overline{h^{(1)}} = \underbrace{w^{T-1}\phi'(z^{(2)})\cdots\phi'(z^{(T)})}_{\text{the Jacobian } \partial h^{(T)}/\partial h^{(1)}}\overline{h^{(T)}}$$

With linear activations:

$$\partial h^{(T)}/\partial h^{(1)} = w^{T-1}$$

Exploding:

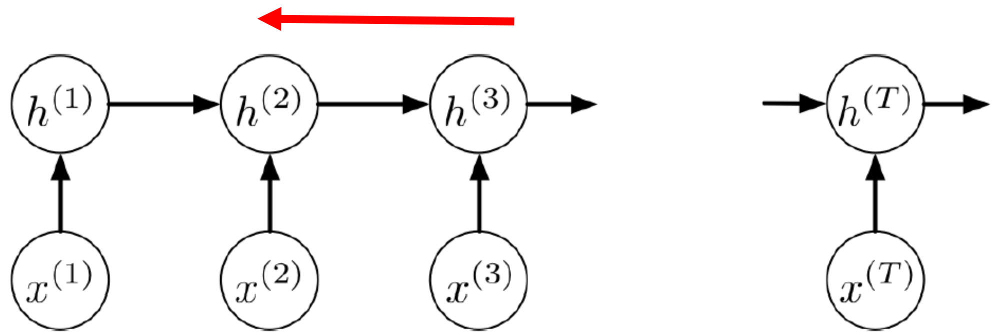
$$w = 1.1,\ T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 117.4$$

Vanishing:

$$w = 0.9,\ T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 0.00515$$

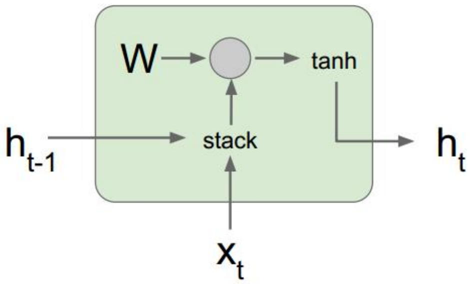
Why gradients explode or vanish

- Motivating example: machine translation
  - Consider a univariate version of the encoder network



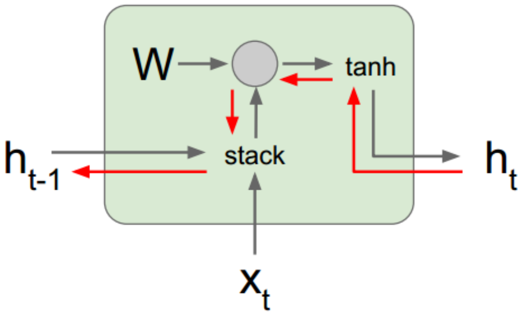
$$z^{(t+1)} = wh^{(t)} + vx^{(t+1)}$$
$$h^{(t)} = \phi(z^{(t)})$$

- General example on the multivariate case



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

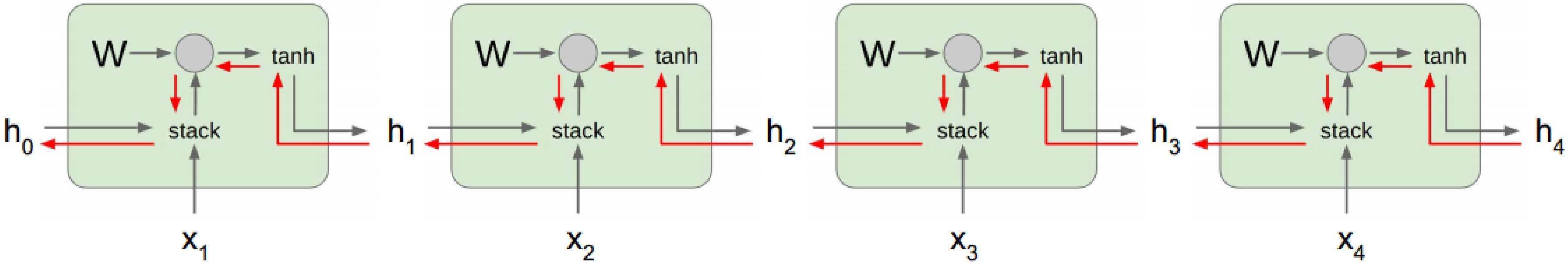
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



Why gradients explore or vanish

- In the multivariate case, the Jacobians multiply:

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest Eigen value > 1:
Exploding gradients

Largest Eigen value < 1:
Vanishing gradients

Why gradients explore or vanish

- In the multivariate case, the Jacobians multiply:

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$

- Contrast this with the forward pass
  - The forward pass has nonlinear activation functions which squash the activations, preventing them from blowing up.
  - The backward pass is linear, so it's hard to keep things stable. There's a thin line between exploding and vanishing.

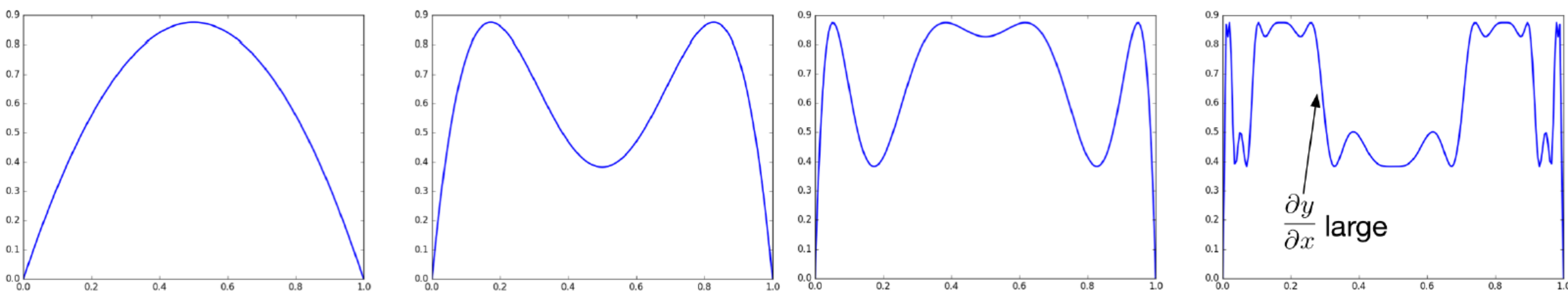
A dynamic system perspective

- RNN can be viewed as an iterative process
  - Each hidden layer computes some function of the previous hiddens and the current input:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

  - Iterated functions are complicated, e.g.:

$$f(x) = 3.5\,x\,(1-x)$$

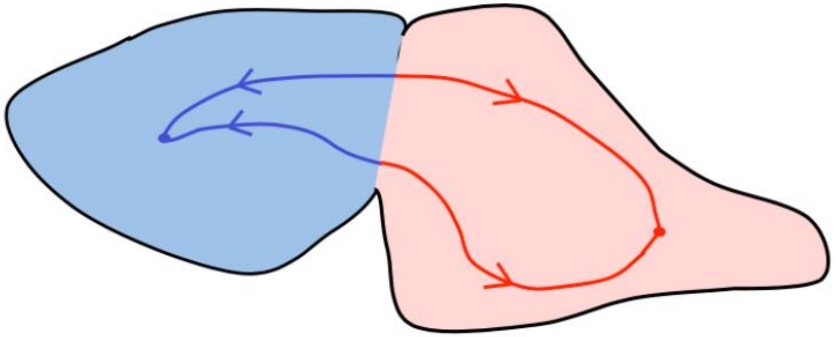


$y = f(x)$ $\quad$ $y = f(f(x))$ $\quad$ $y = f(f(f(x)))$ $\quad$ $y = \underbrace{f \circ \cdots \circ f}_{6 \text{ times}}(x)$

A dynamic system perspective

- RNN can be viewed as an iterative process
  - As a dynamical system, it has various attractors:

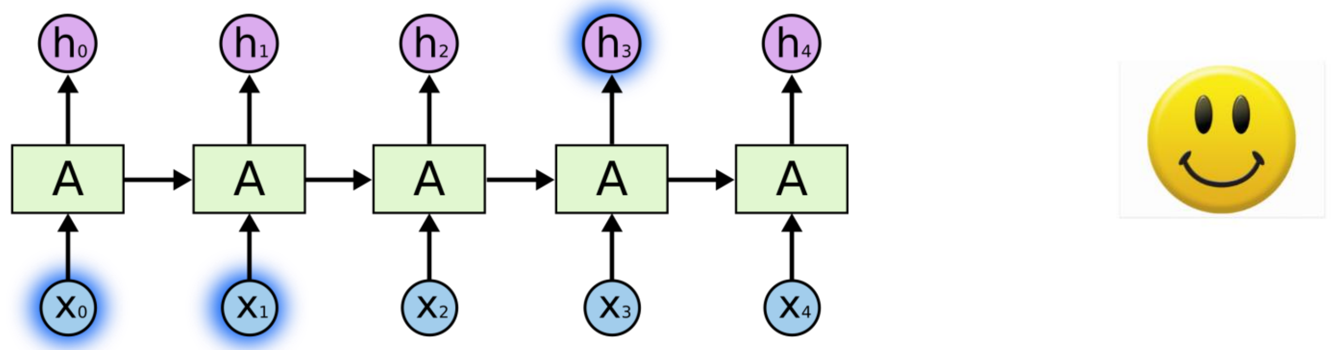
$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

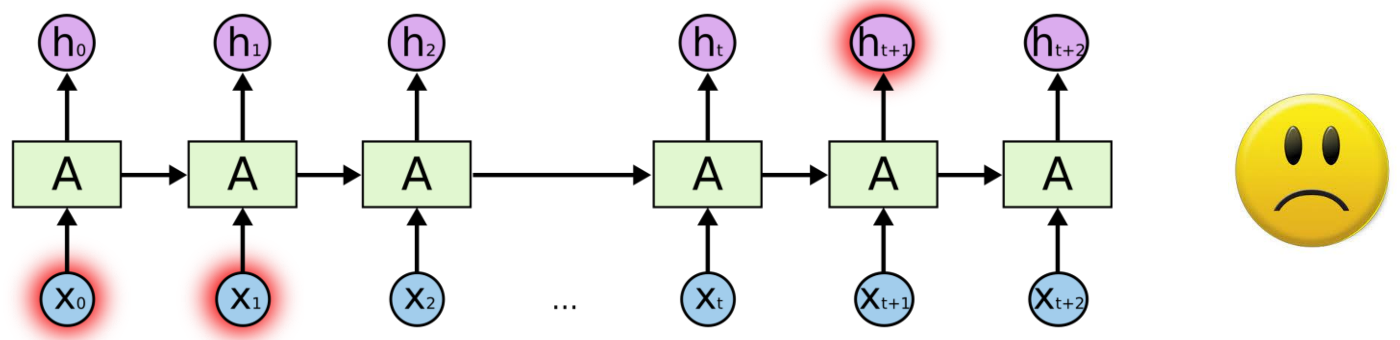


  - Within one of the colored regions, the gradients vanish because even if you move a little, you still wind up at the same attractor.
  - If you're on the boundary, the gradient blows up because moving slightly moves you from one attractor to the other.

Vanilla RNN

- Difficulty in modeling long-term dependency





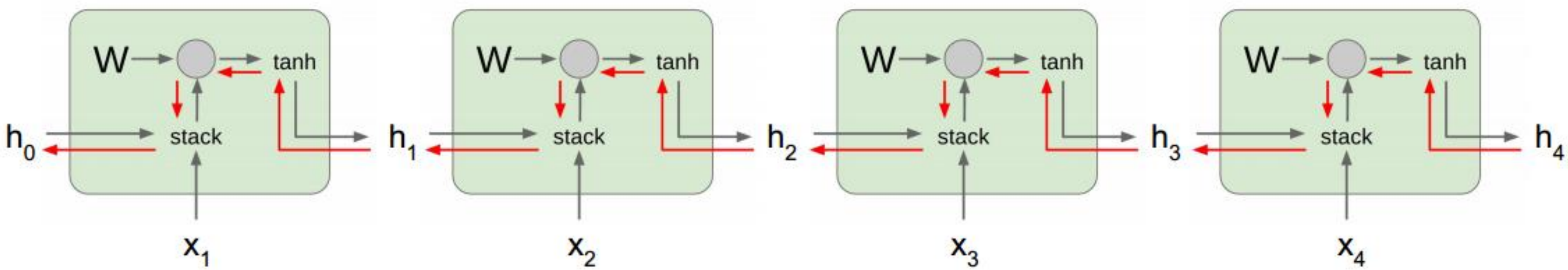
Outline

- Recurrent Neural Networks
  - Gradient problems in training RNNs
  - Stabilizing RNN training
- Long-Term Short Term Memory (LSTM)
  - LSTM/GRU unit
  - RNNs with LSTM

Acknowledgement: Feifei Li et al's cs231n notes

Stabilizing RNN training

- Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1: **Vanishing gradients**

Stabilizing RNN training

- Gradient clipping

  Clip the gradient $\mathbf{g}$ so that it has a norm of at most η:

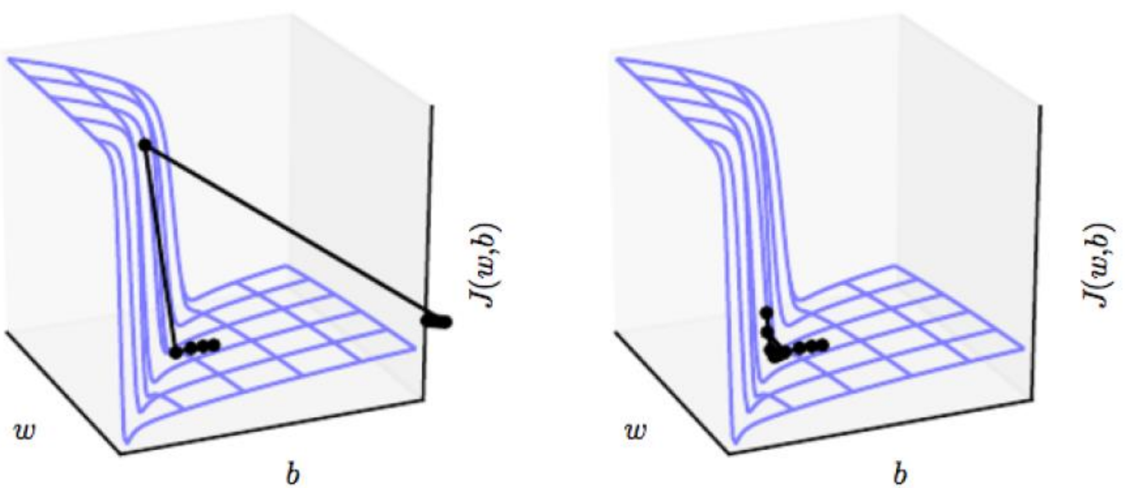
  if $\|\mathbf{g}\| > \eta$:

  $$\mathbf{g} \leftarrow \frac{\eta \mathbf{g}}{\|\mathbf{g}\|}$$

- The gradients are biased, but at least they don't blow up

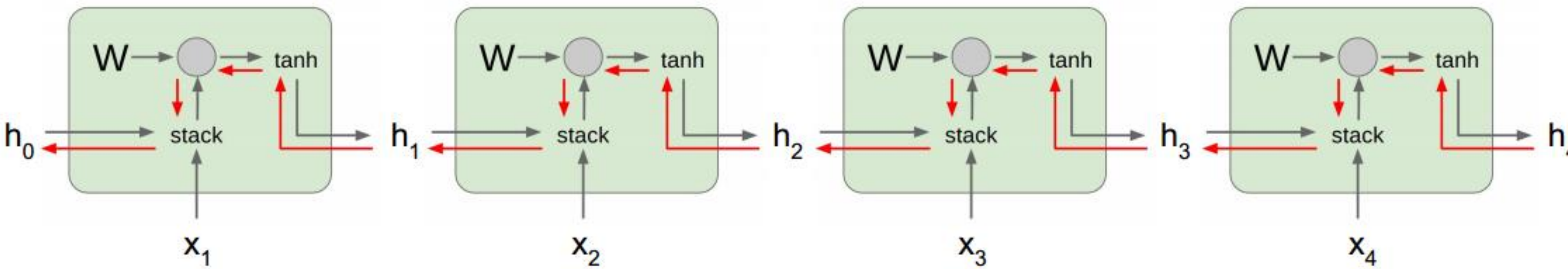
Without clipping

With clipping



Stabilizing RNN training

- Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest Eigen value > 1:
Exploding gradients

Largest Eigen value < 1:
Vanishing gradients

→ Change RNN architecture

Stabilizing RNN training

- Architecture re-design:
  - The hidden units are a kind of memory. Therefore, their default behavior should be to keep their previous value.
- If the function is close to the identity, the gradient computations are stable
  - The Jacobians are close to the identity matrix and so they can be multiplied together safely.
- Example: Identity RNN
  - Use the ReLU activation function
  - Initialize all the weight matrices to the identity matrix
  - It was able to learn to classify MNIST digits, input as sequence one pixel at a time!

Le et al., 2015. A simple way to initialize recurrent networks of rectified linear units.

Outline

- Recurrent Neural Networks
  - Gradient problems in training RNNs
  - Stabilizing RNN training
- Long-Term Short Term Memory (LSTM)
  - LSTM/GRU unit
  - RNNs with LSTM

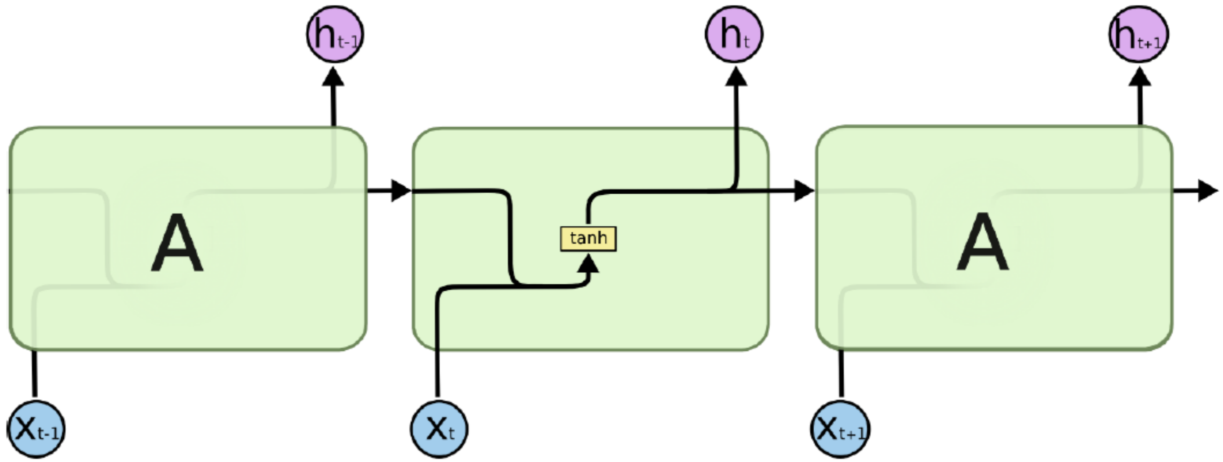
Acknowledgement: Feifei Li et al's cs231n notes

Long-term Short Term Memory

- Replacing a vanilla RNN neuron by the LSTM unit
- Why it is called LSTM
  - A network's activations are its short-term memory and its weights are its long-term memory
  - The LSTM architecture wants the short-term memory to last for a long time period
- Key idea
  - Composed of memory cells which have controllers that decide when to store or forget information

Standard RNN

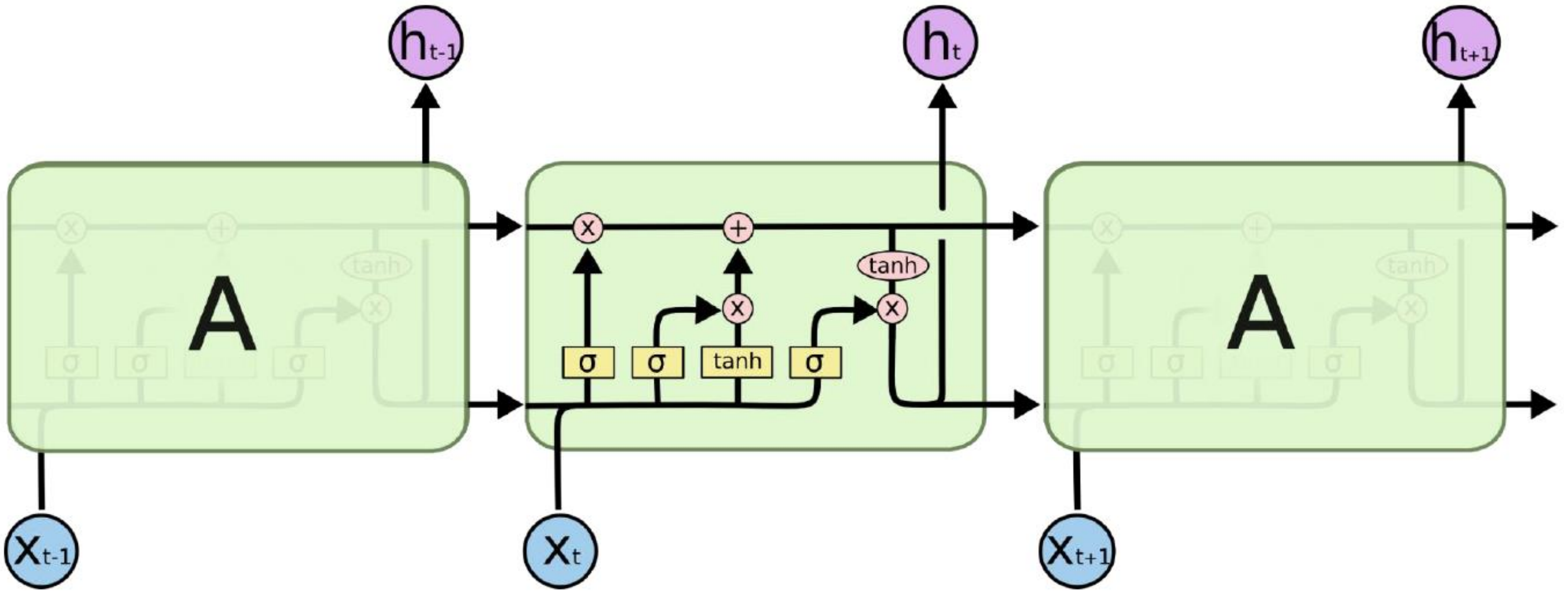
- Recall



- Each recurrent neuron receives past outputs and current input
- Pass through a tanh function

Long Short Term Memory(LSTM)

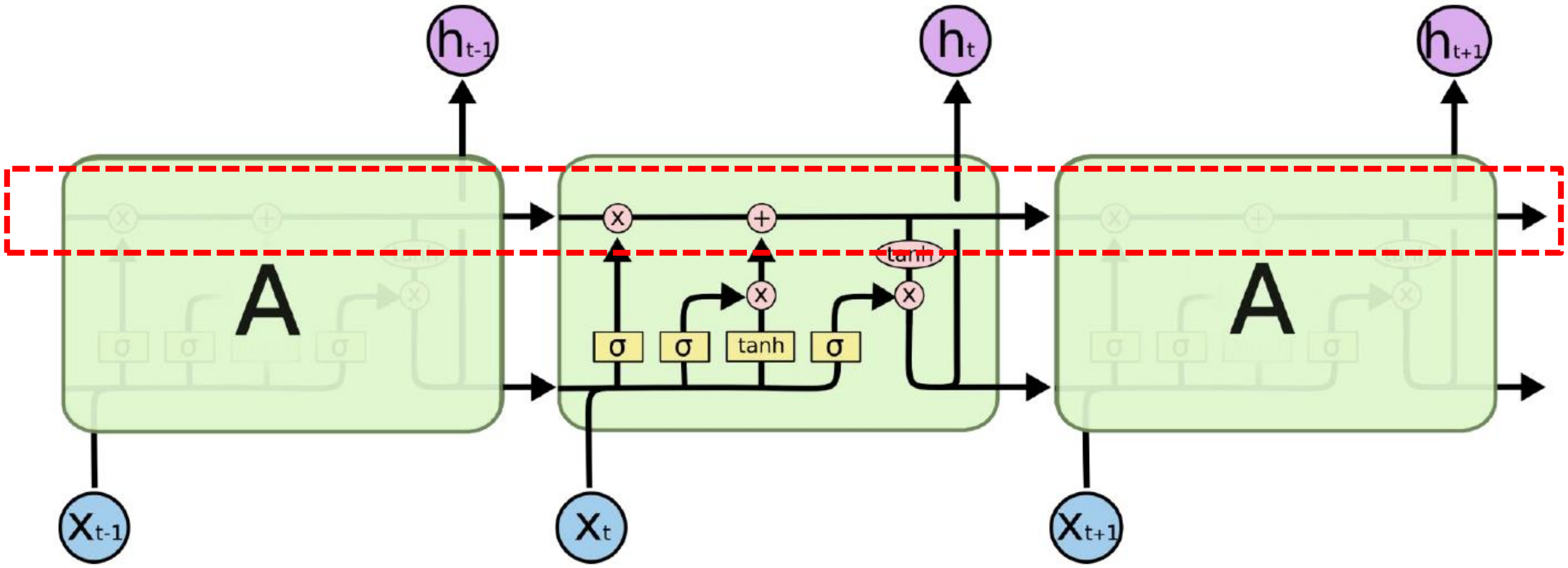
- LSTM uses multiplicative gates that decide if something is important or not



Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation

Long Short Term Memory(LSTM)

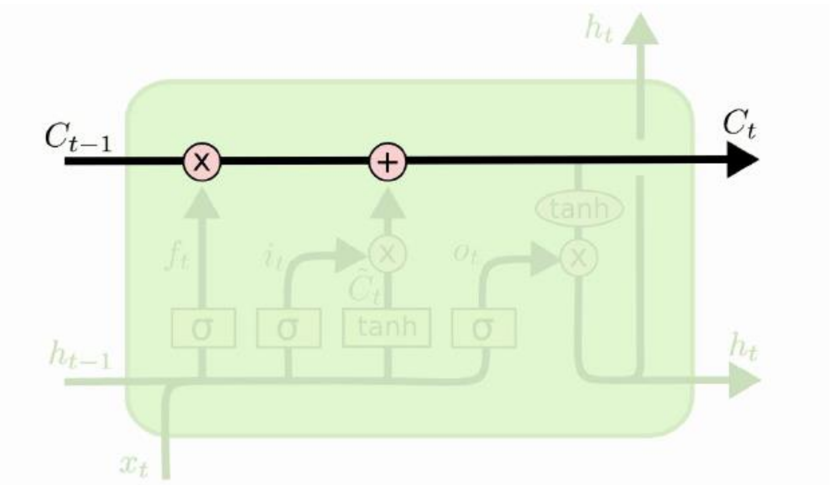
- Key component: a remembered cell state



Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation

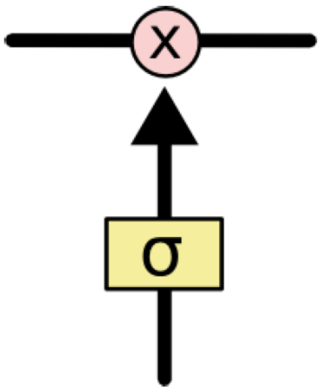
LSTM: cell state

- A linear history
  - Carries information through
  - Only affected by a gate and addition of current information, which is also gated



LSTM: gates

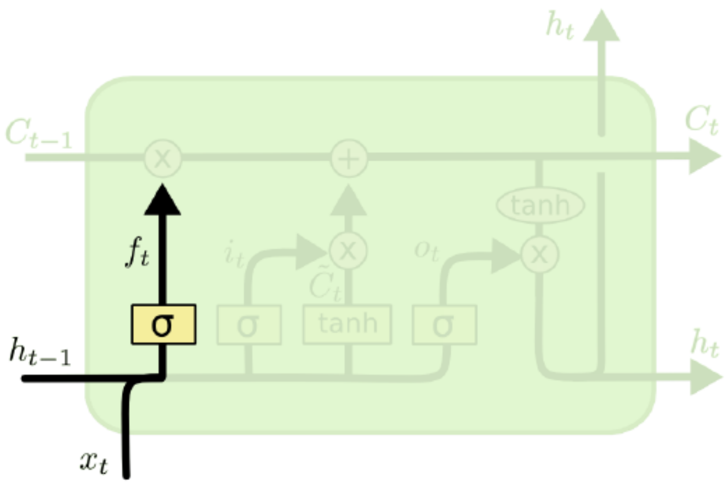
- Gates are simple sigmoid units with output range in (0,1)
- Controls how much of the information will be let through



- Three gates
  - Forget gate
  - Input gate
  - Output gate

LSTM: forget gate

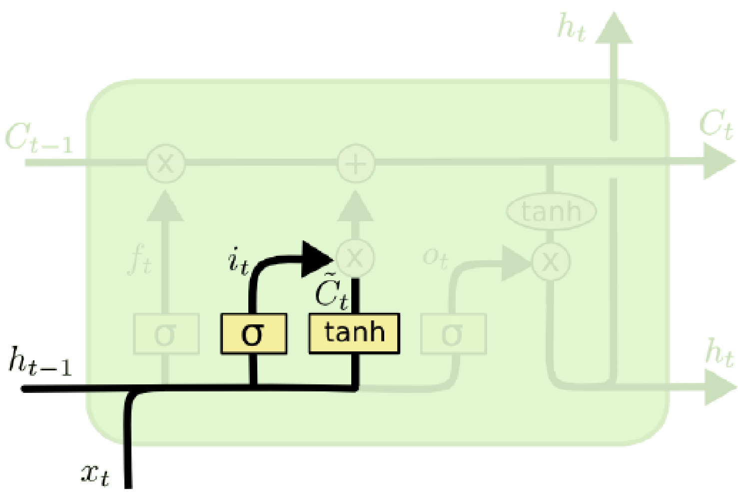
- The first gate determines whether to carry over the history or to forget it
  - Soft decision: how much of the history C_{t-1} to carry over
  - Determined by the current input x_t and the previous state h_{t-1}
  - $\langle h_{t-1}, C_{t-1} \rangle$ can be viewed as partial key-value pairs



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

LSTM: input gate

- The second gate has two parts
  - A gate that decides if it is worth remembering
  - A nonlinear transformation that extracts new and interesting information from the input
  - Both use the current input and the previous state

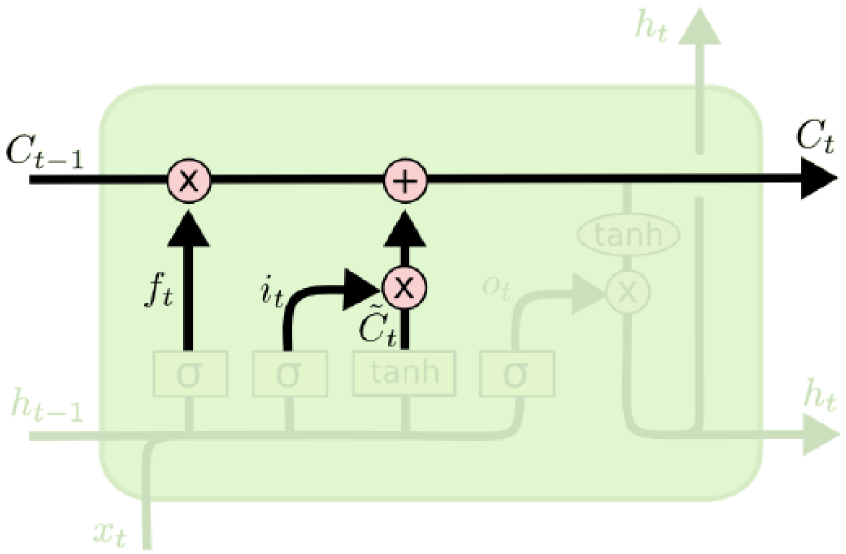


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTM: Memory cell update

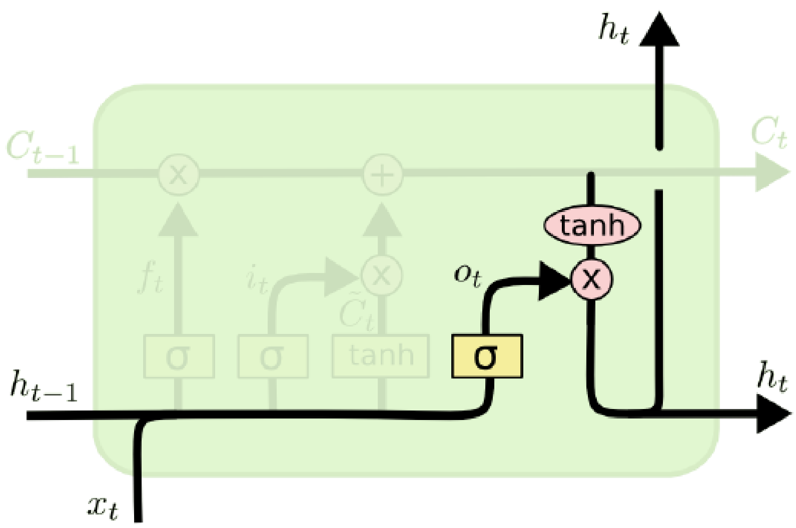
- The output of the second part is added into the current memory cell
  - Can be viewed as value update in a key-value pair
  - The input and state jointly decide how much of history info is kept and how much of embedded input info is added
  - A dynamic mixture of experts at each time step



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

LSTM: Output gate

- The third gate is the output gate
  - To decide if the memory cell contents are worth reporting at this time using the current input and previous state
- The output of the cell or the state
  - A nonlinear transform of the cell values
  - Compress it with tanh to make it in (-1,1)
  - Note the separation of key-value representation

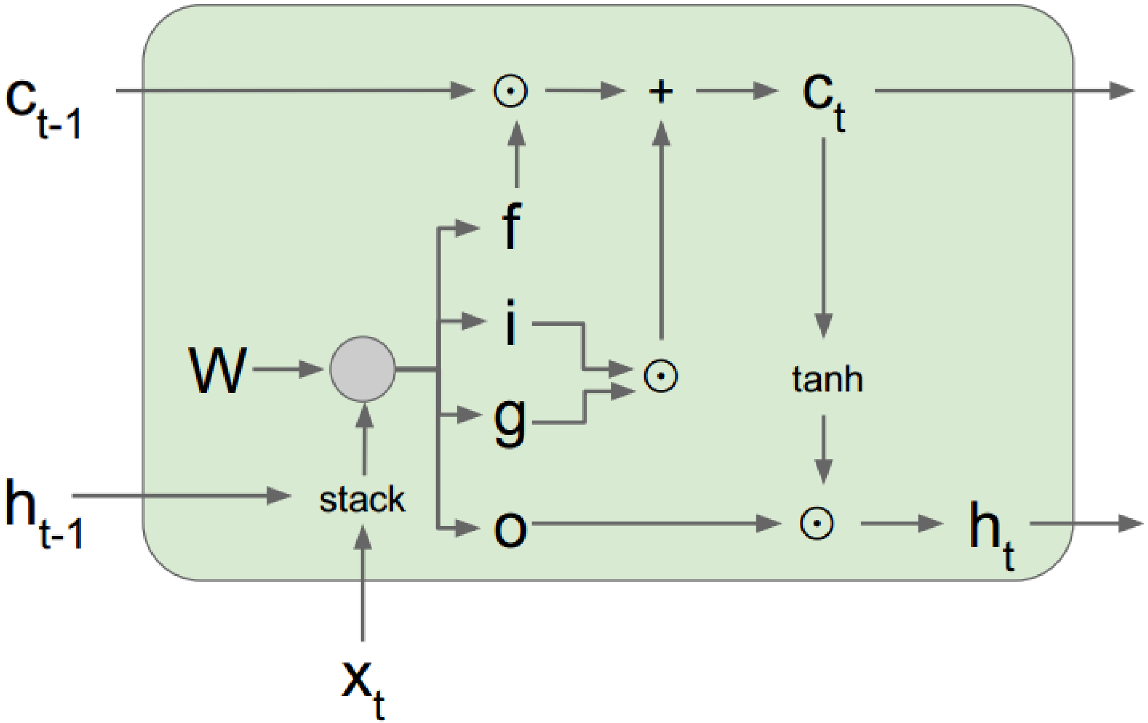


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Long Short Term Memory(LSTM)

[Hochreiter et al., 1997]



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

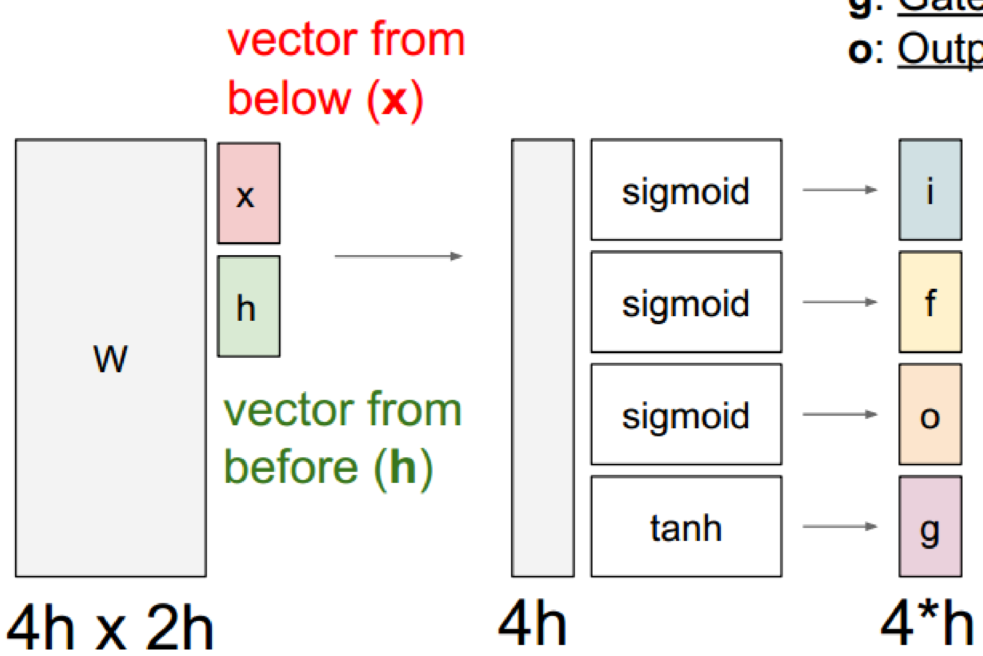
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory(LSTM)

[Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



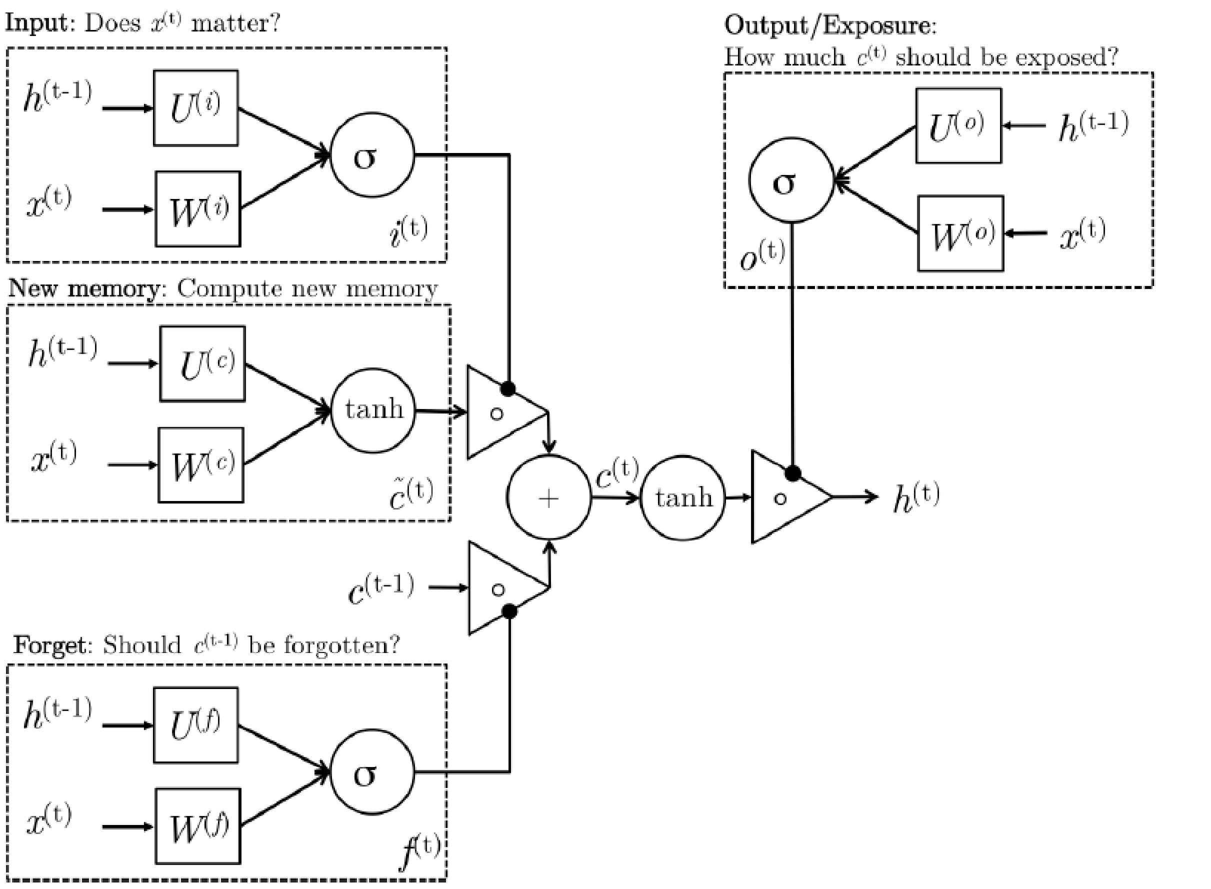
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

LSTM: as feedforward layer

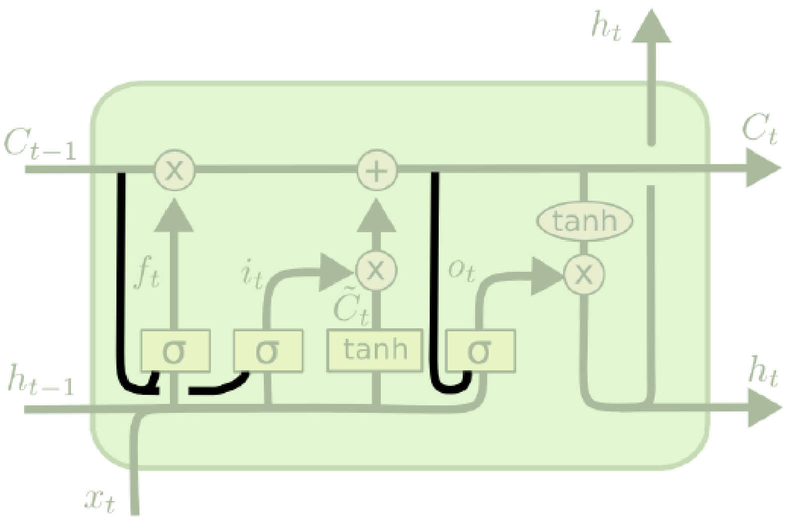
- As a gated feedforward network



Richard Socher's CS224D notes

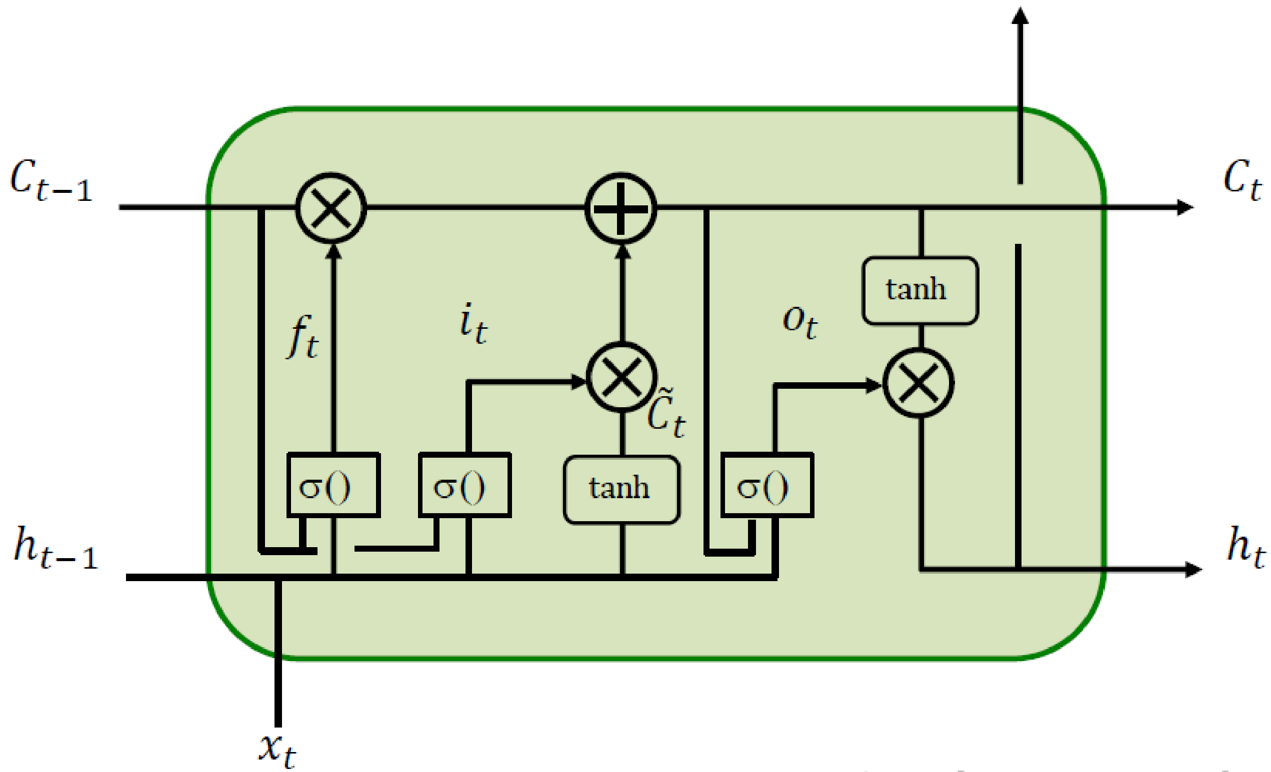
LSTM: the "peephole" connection

- All three gates can also use the memory cell info
  - Complementary to the state and input
  - Rich history information



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Computation: forward in full model



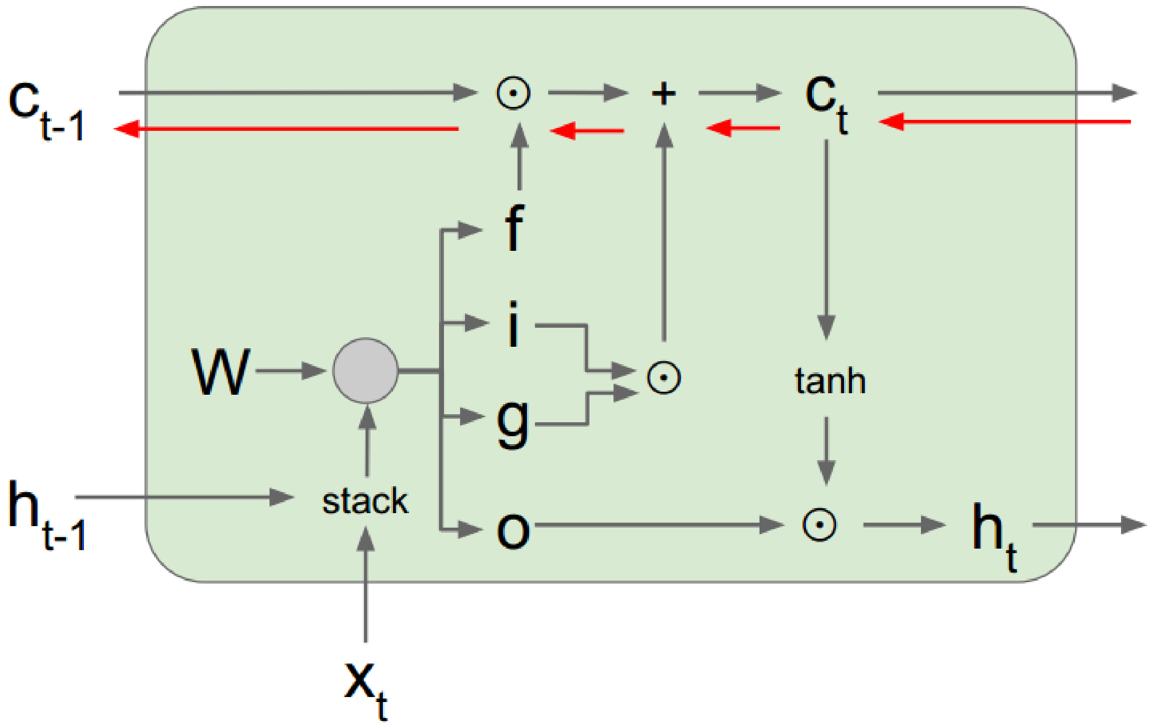
- Forward rules:

Gates
$$f_t = \sigma\left(W_f \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\mathbf{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Variables
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM: Backpropagation

[Hochreiter et al., 1997]



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

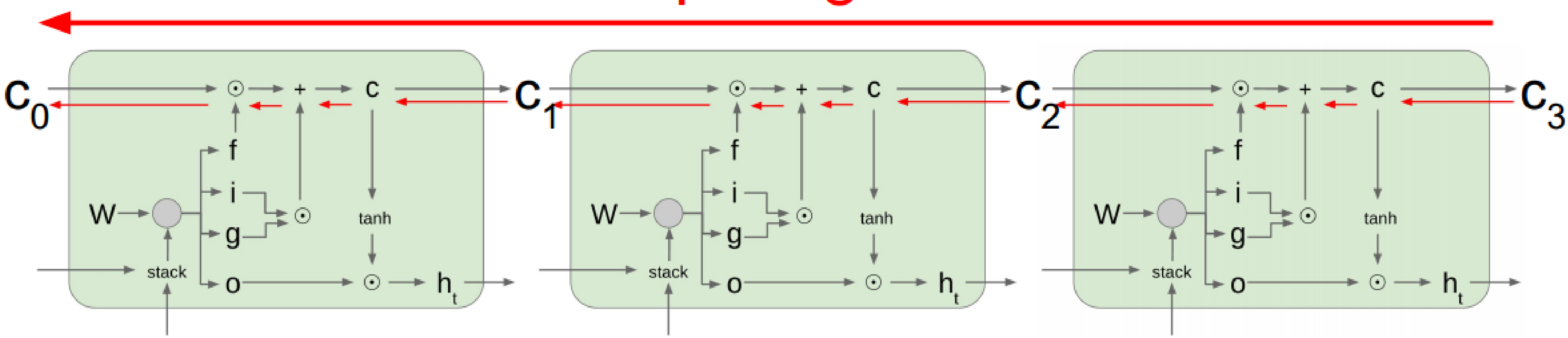
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

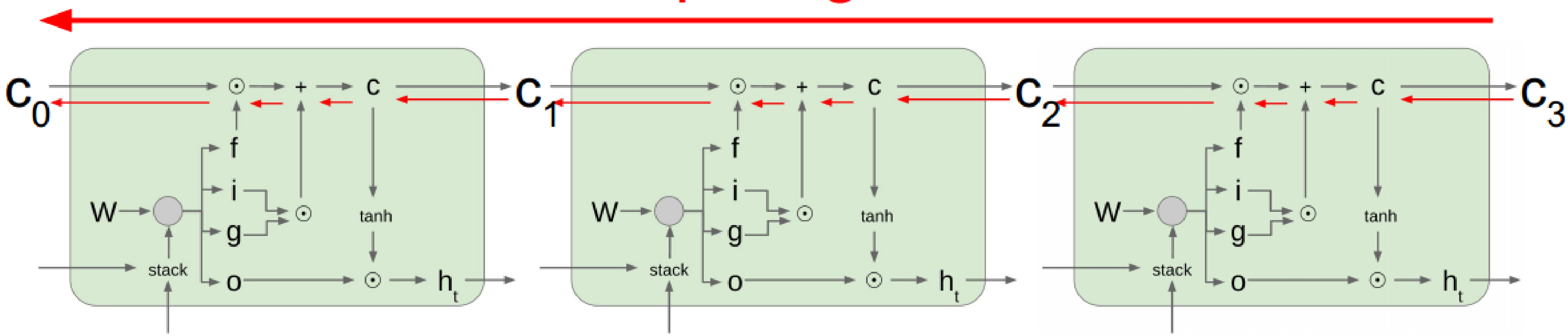
LSTM: Backpropagation

Uninterrupted gradient flow!

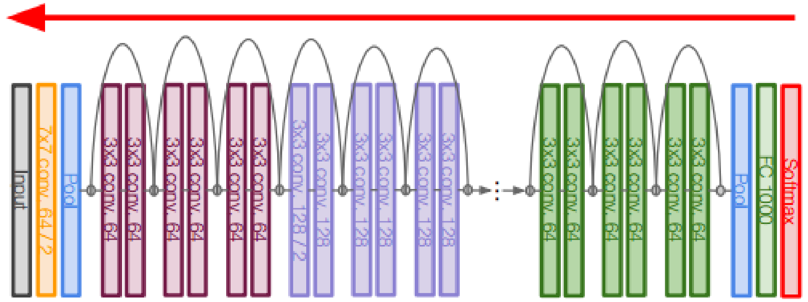


LSTM: Backpropagation

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

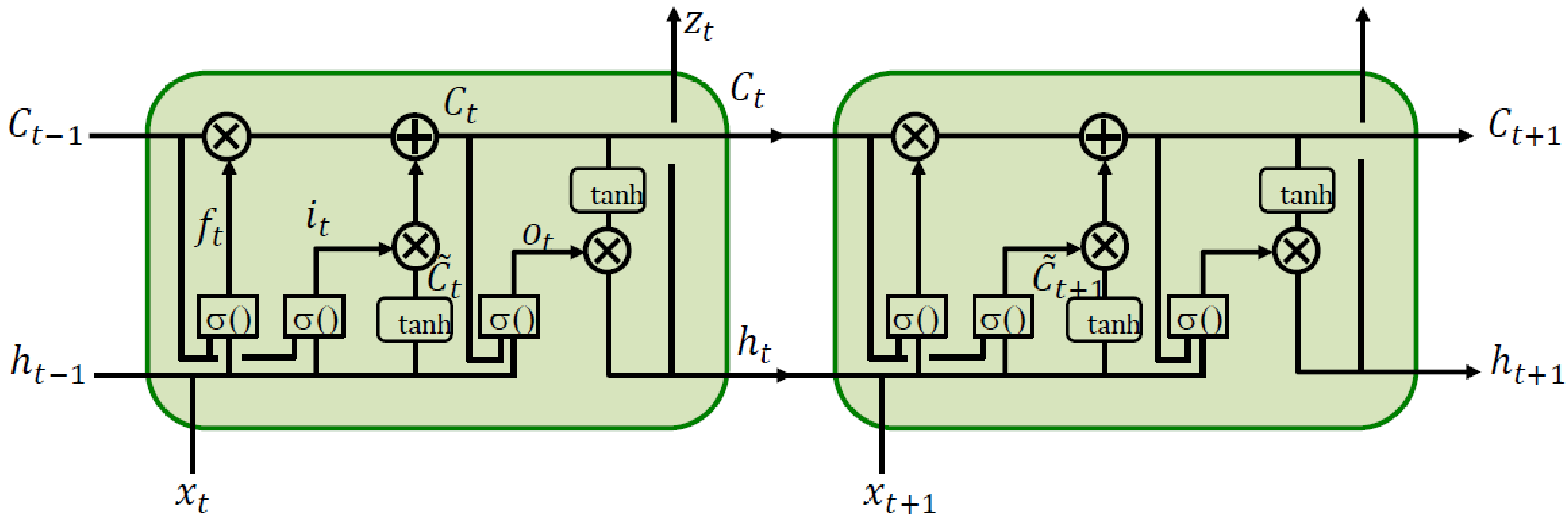
$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

LSTM: Backpropagation

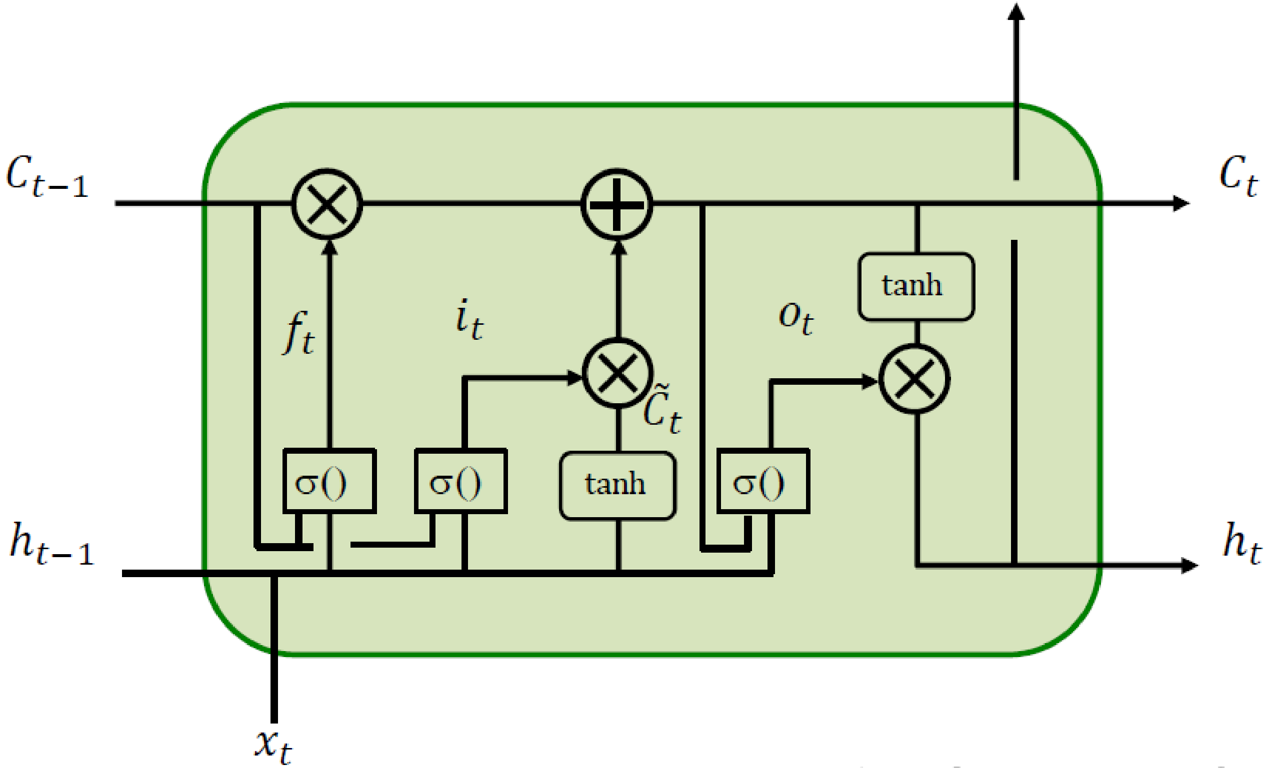
- Full model version



$$\nabla_{C_t} L =$$

$$\nabla_{h_t} L =$$

Computation: forward in full model



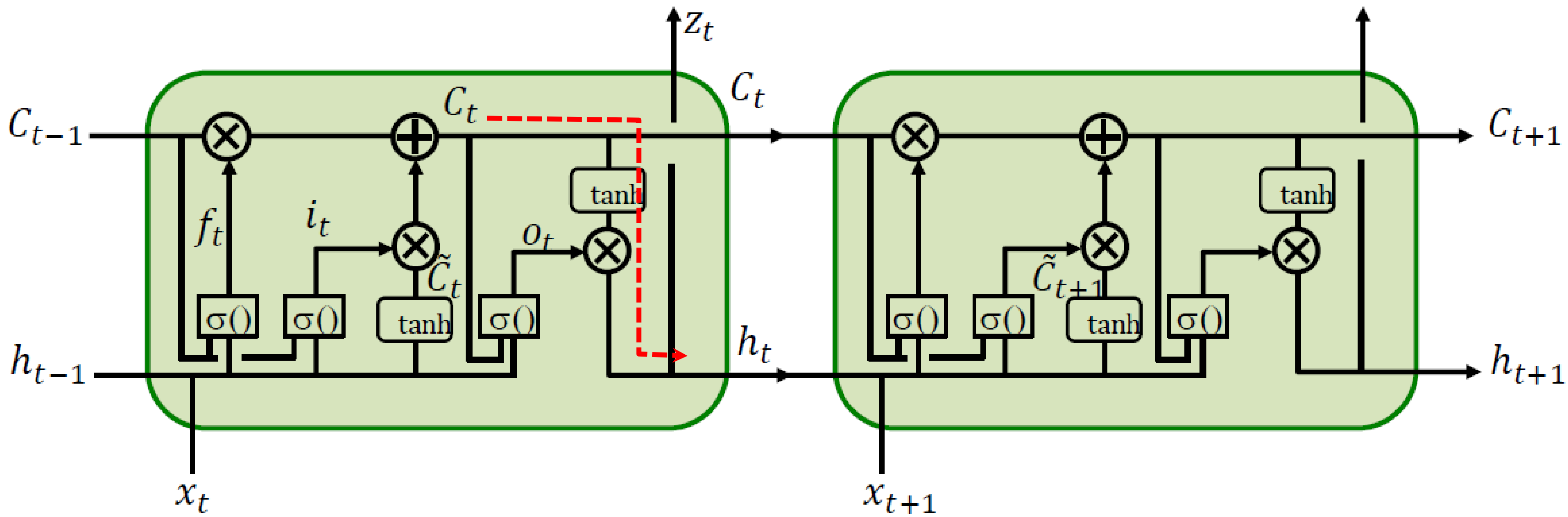
- Forward rules:

Gates
$$f_t = \sigma\left(W_f \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\mathbf{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

Variables
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM: Backpropagation

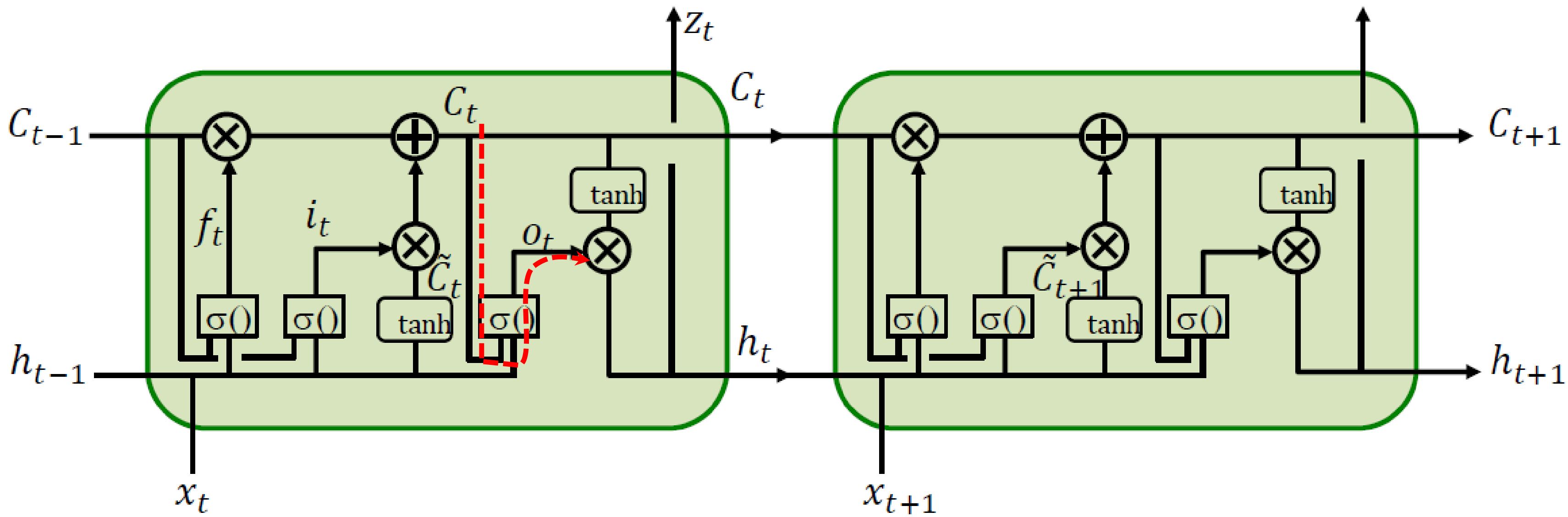
- Full model version



$$\nabla_{C_t} L = \nabla_{h_t} L \circ o_t \circ \tanh'(\cdot) W_{Ch}$$

LSTM: Backpropagation

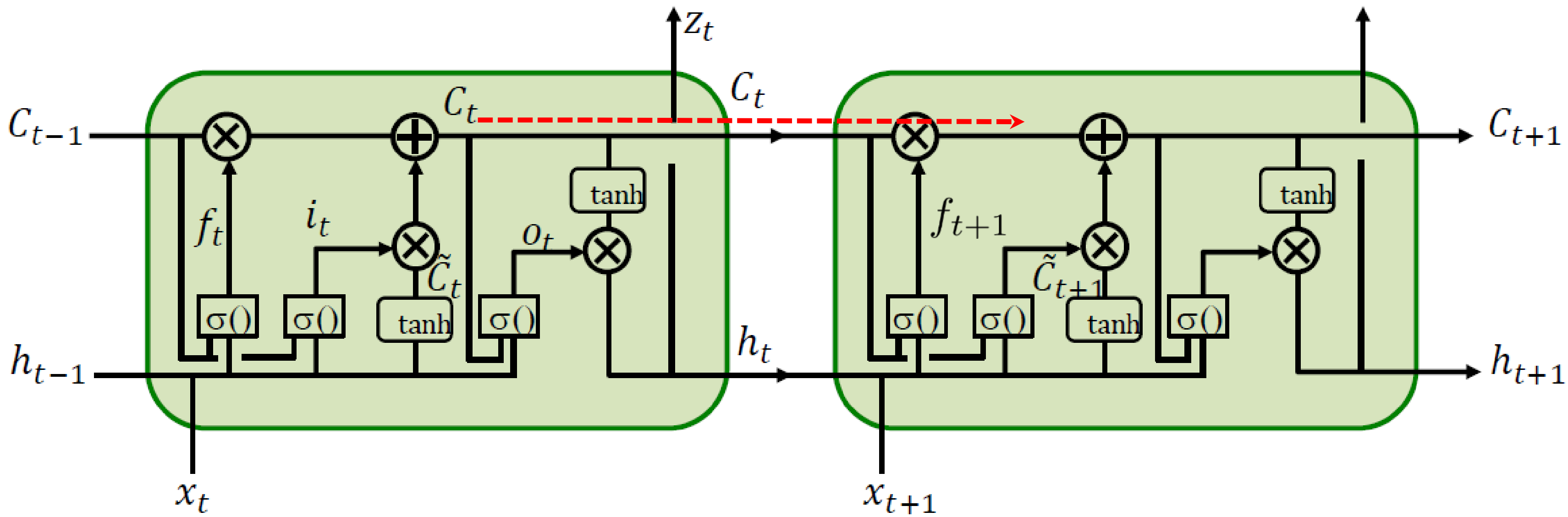
- Full model version



$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

LSTM: Backpropagation

- Full model version

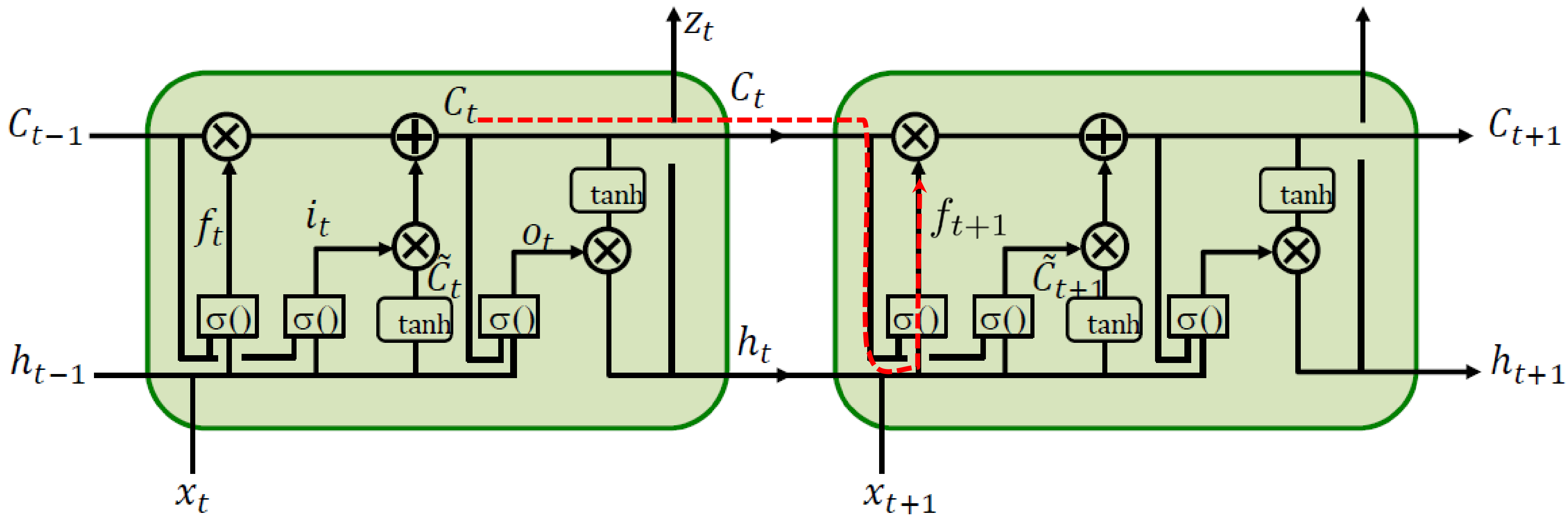


$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

$$+ \nabla_{h_t} C_{t+1} \circ f_{t+1}$$

LSTM: Backpropagation

- Full model version

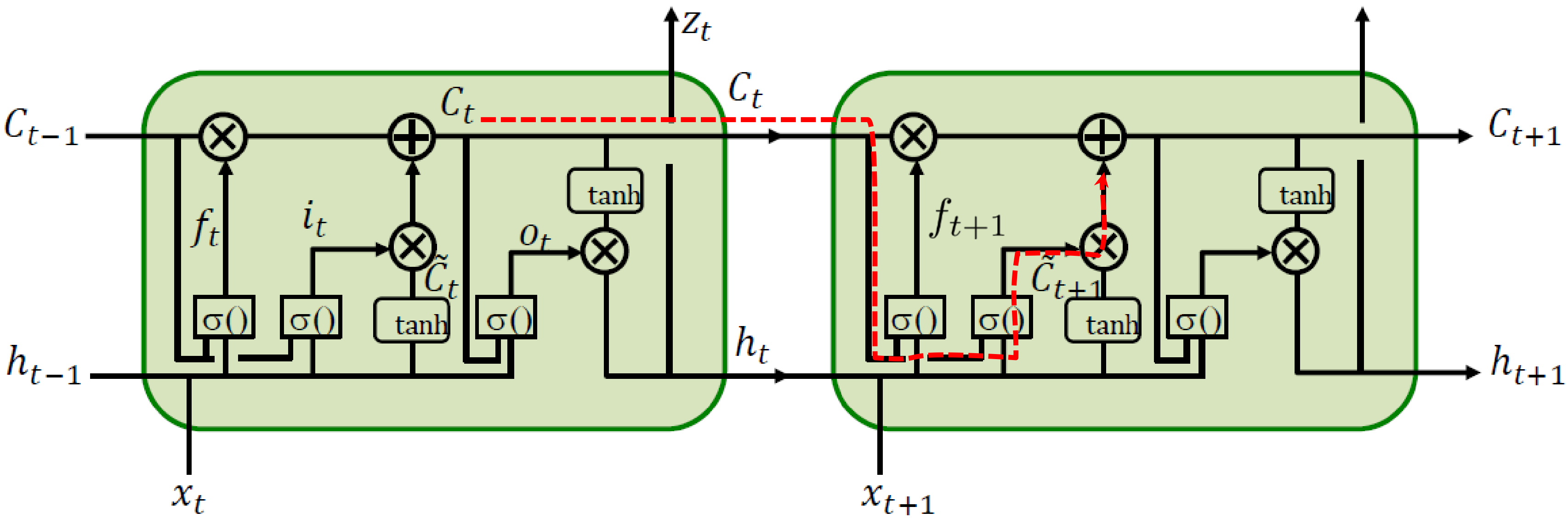


$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

$$+\nabla_{h_t} C_{t+1} \circ (f_{t+1} + C_t \circ \sigma'(\cdot) W_{Cf})$$

LSTM: Backpropagation

- Full model version

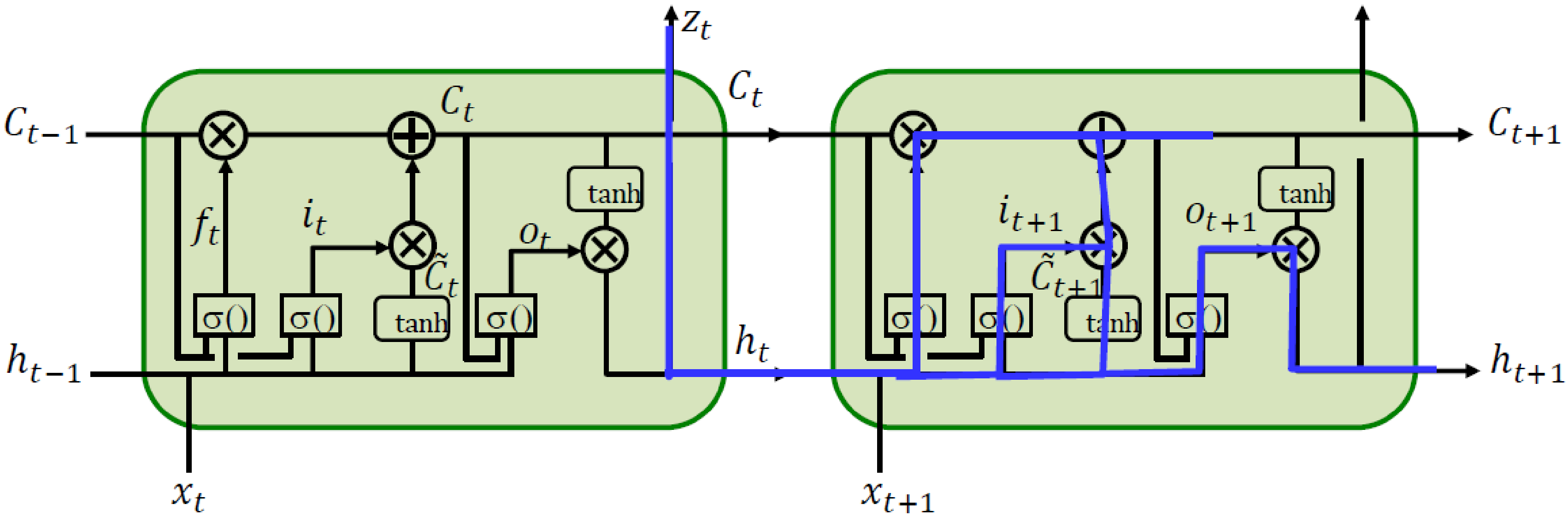


$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

$$+\nabla_{h_t} C_{t+1} \circ (f_{t+1} + C_t \circ \sigma'(\cdot) W_{Cf} + \tilde{C}_{t+1} \circ \sigma'(\cdot) W_{Ci})$$

LSTM: Backpropagation

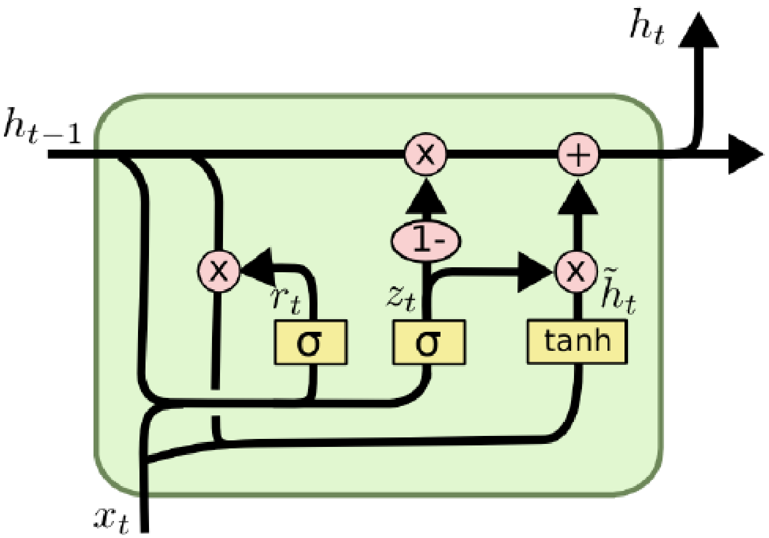
- Full model version



$$\nabla_{h_t} L = \nabla_{z_t} L \nabla_{h_t} z_t + \nabla_{h_t} C_{t+1} \circ (C_t \circ \sigma'(\cdot) W_{hf} + \tilde{C}_{t+1} \circ \sigma'(\cdot) W_{hi})$$
$$+ \nabla_{C_{t+1}} L \circ o_{t+1} \circ \tanh'(\cdot) W_{hi} + \nabla_{h_{t+1}} L \circ \tanh(\cdot) \circ \sigma'(\cdot) W_{ho}$$

Gated Recurrent Units

- Simplified LSTM
  - Can we merge some operations?



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

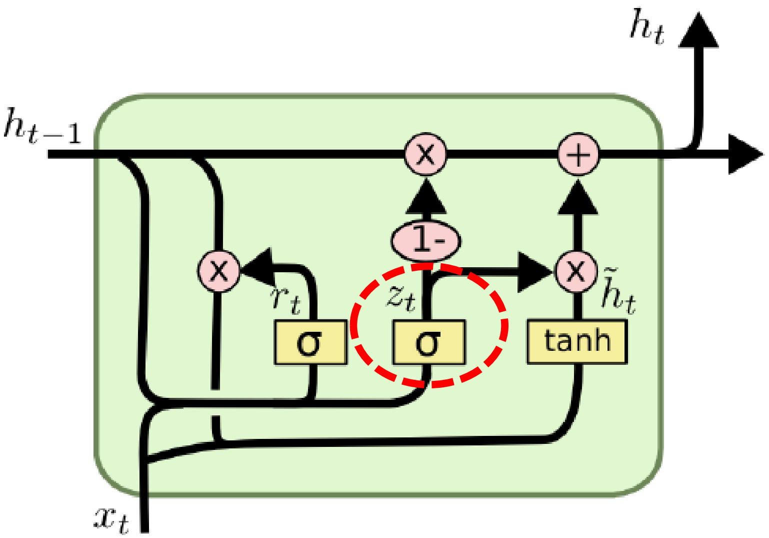
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

Gated Recurrent Units

- Simplified LSTM
  - Combine the forget and input gates



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

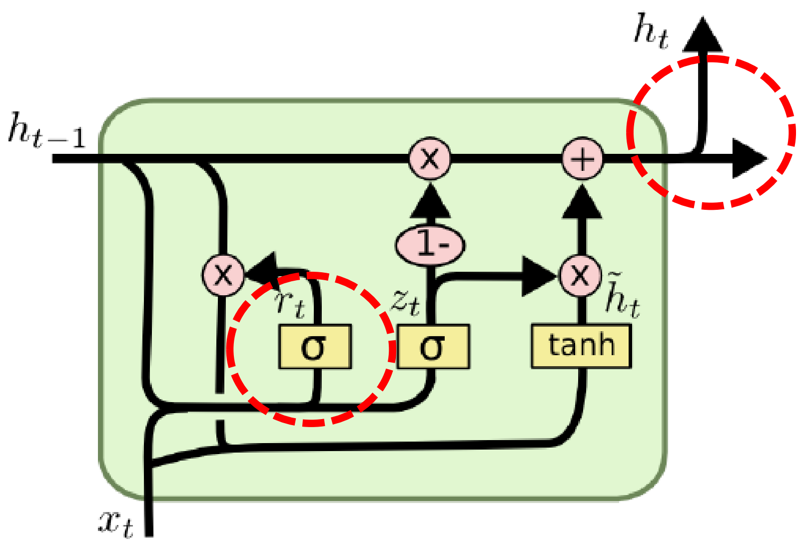
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

Gated Recurrent Units

- Simplified LSTM
  - Don't bother to separately maintain compressed and regular memories
  - Compress it before using it to decide on the usefulness of the current input



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

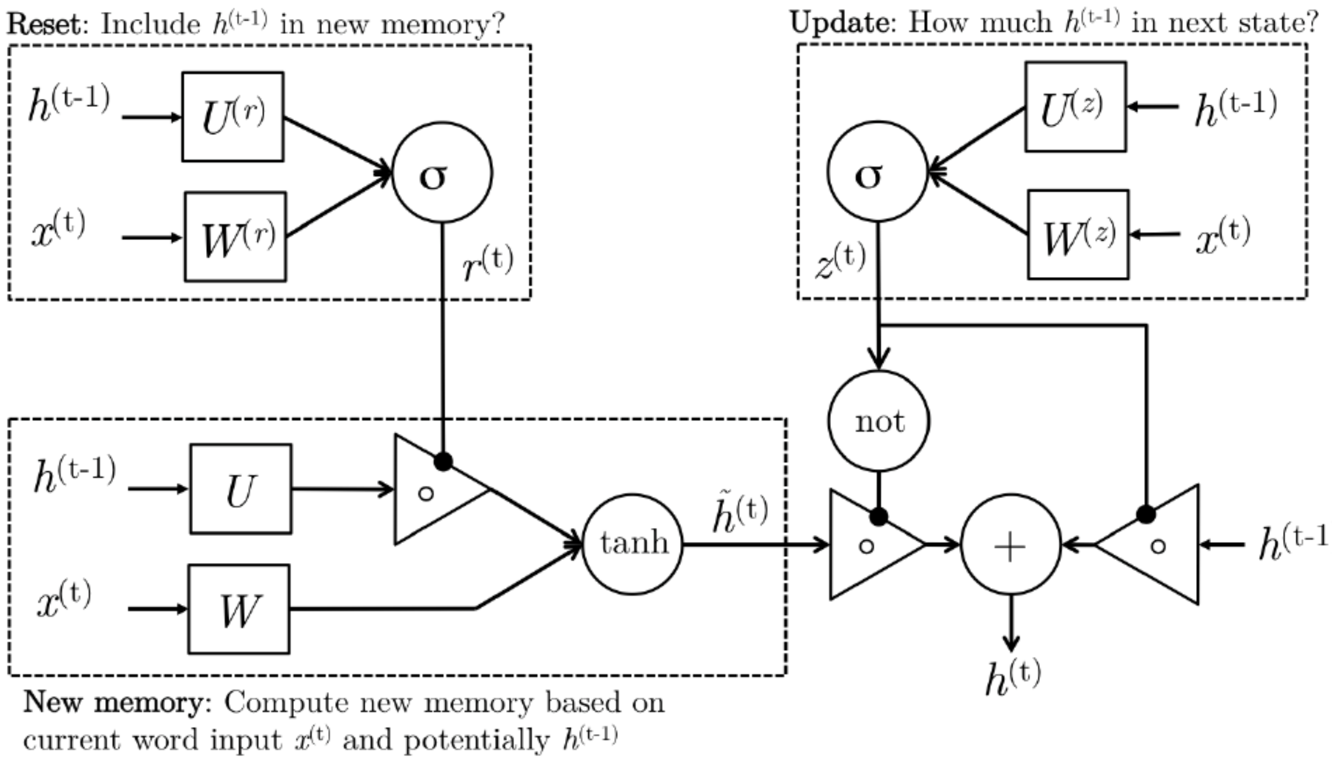
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

GRU: As a feedforward layer

- As a gated feedforward network



$$z^{(t)} = \sigma(W^{(z)}x^{(t)} + U^{(z)}h^{(t-1)}) \quad \text{(Update gate)}$$
$$r^{(t)} = \sigma(W^{(r)}x^{(t)} + U^{(r)}h^{(t-1)}) \quad \text{(Reset gate)}$$
$$\tilde{h}^{(t)} = \tanh(r^{(t)} \circ Uh^{(t-1)} + Wx^{(t)}) \quad \text{(New memory)}$$
$$h^{(t)} = (1 - z^{(t)}) \circ \tilde{h}^{(t)} + z^{(t)} \circ h^{(t-1)} \quad \text{(Hidden state)}$$

Richard Socher’s CS224D notes

Other RNN Variants

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{xz}x_t + b_z) \\ r &= \mathrm{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\ r &= \mathrm{sigm}(x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{xz}x_t + W_{hz}\tanh(h_t) + b_z) \\ r &= \mathrm{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Multi-Layer RNNs

Multilayer RNNs

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

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

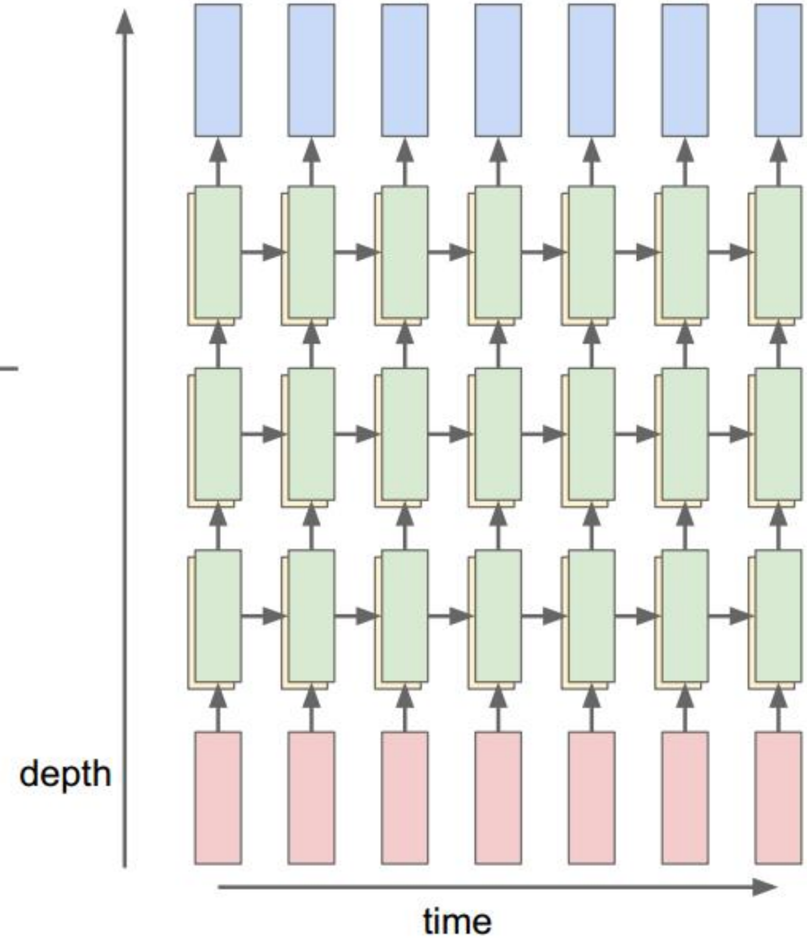
LSTM:

$W^l \; [4n \times 2n]$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

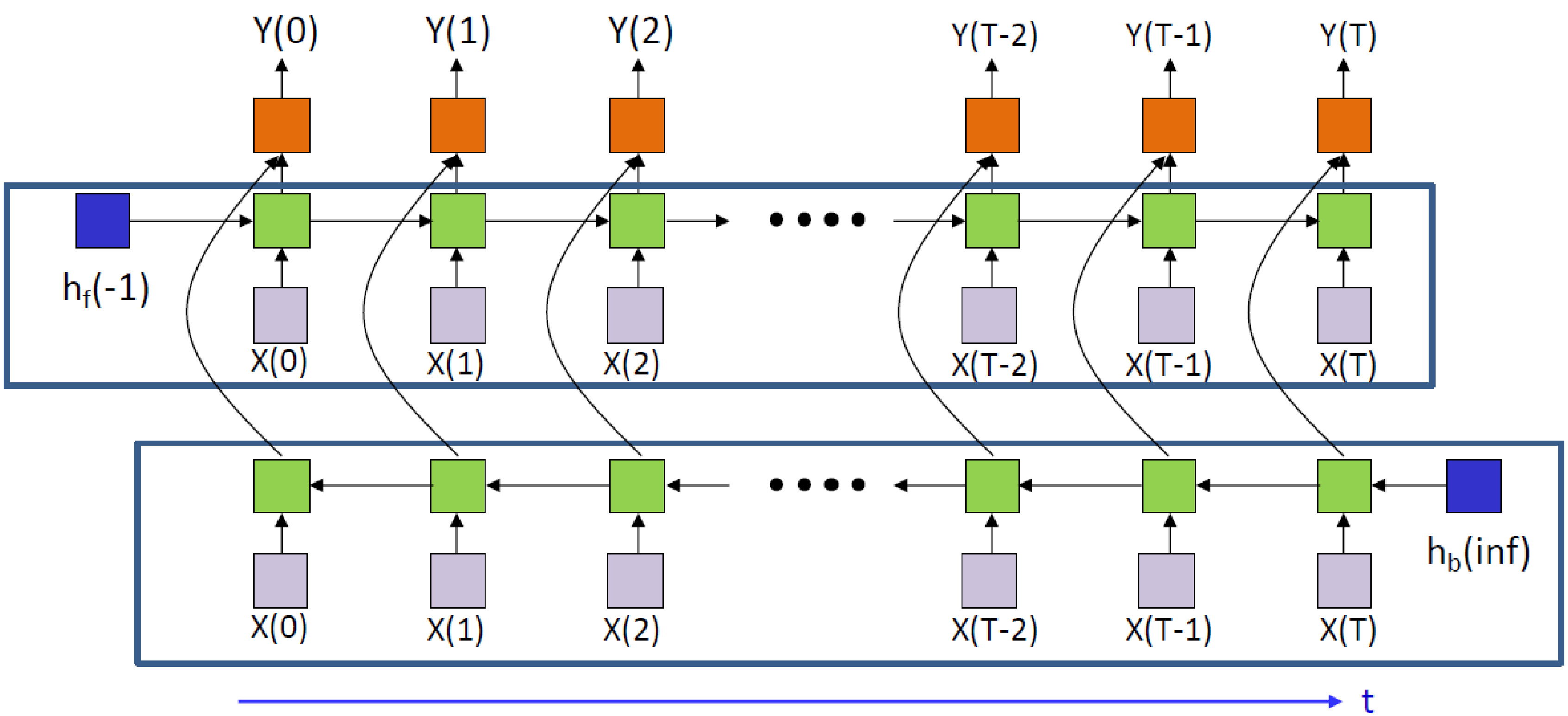
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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



Bidirectional LSTM

- Two opposite directions
  - Noncausal but complementary global context
  - Can have multiple layers of LSTM units in either direction



Summary

- RNN
  - Training vanilla RNNs has gradient explosion/vanishing problem
  - Two strategies
    - Gradient clipping
    - Change model structure
  - LSTM structure and learning
  - LSTM-based RNN networks
- Next time:
  - Examples of RNNs in Vision and NLP applications
  - Attention models