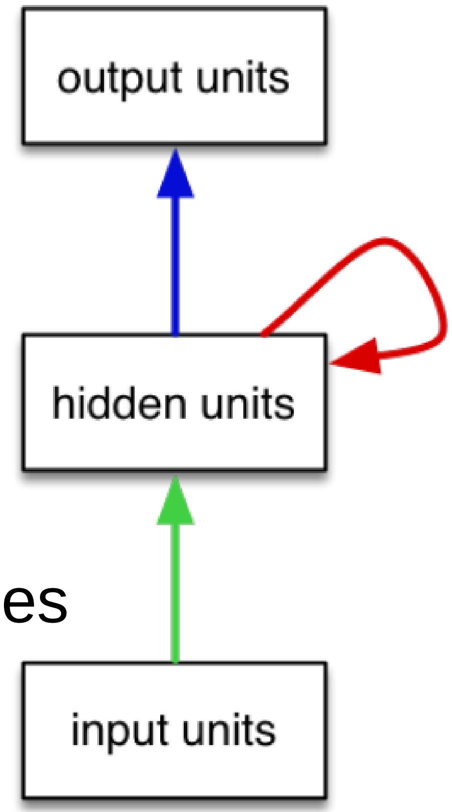
Lecture 6: Recurrent Neural Networks II: LSTM, GRU

Lan Xu
SIST, ShanghaiTech
Fall, 2024

Design Criteria of RNN

- We need to model sequences:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequences

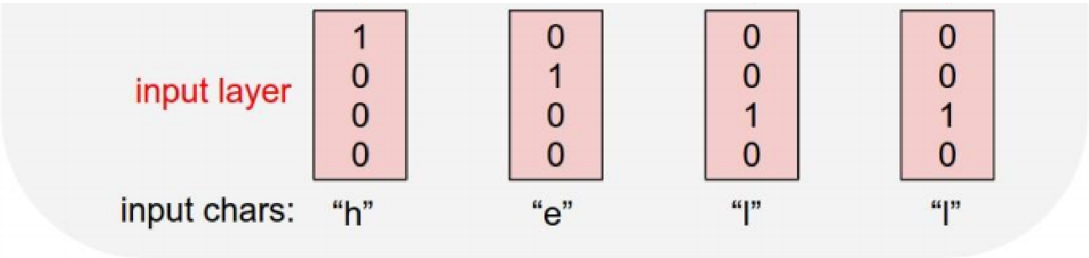


RNN for language modeling

Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



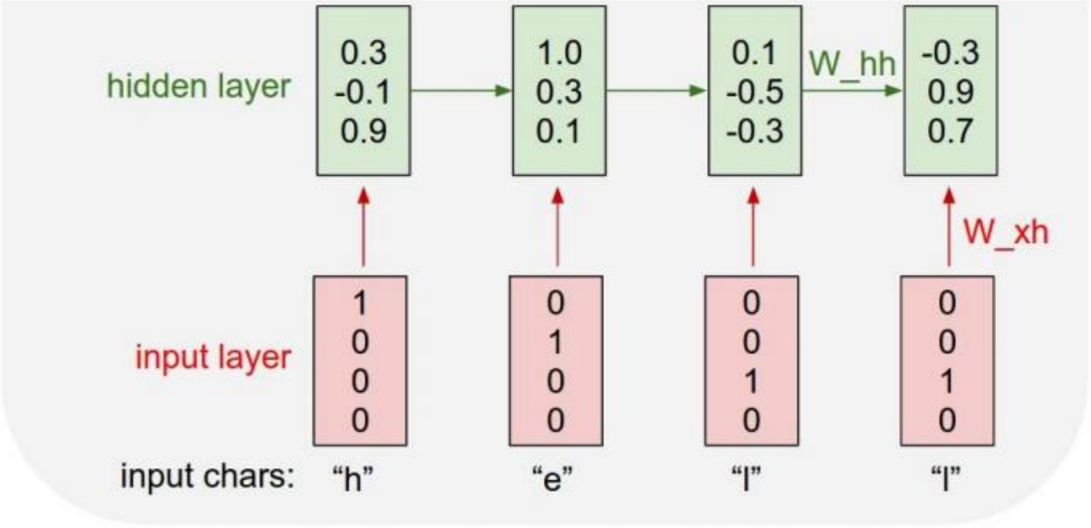
RNN for language modeling

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

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

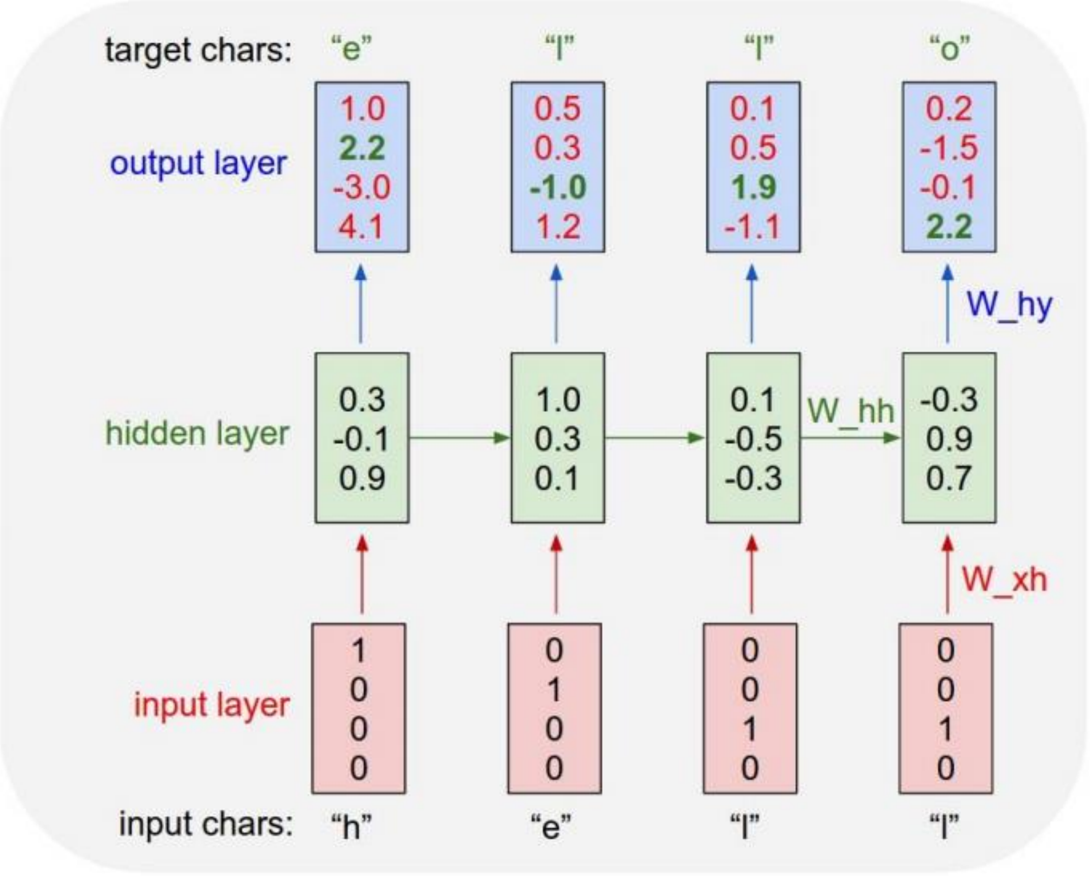


RNN for language modeling

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

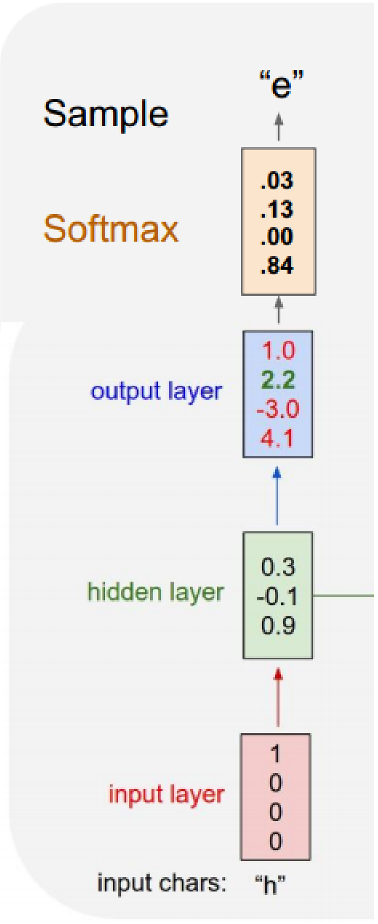


RNN for language modeling

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

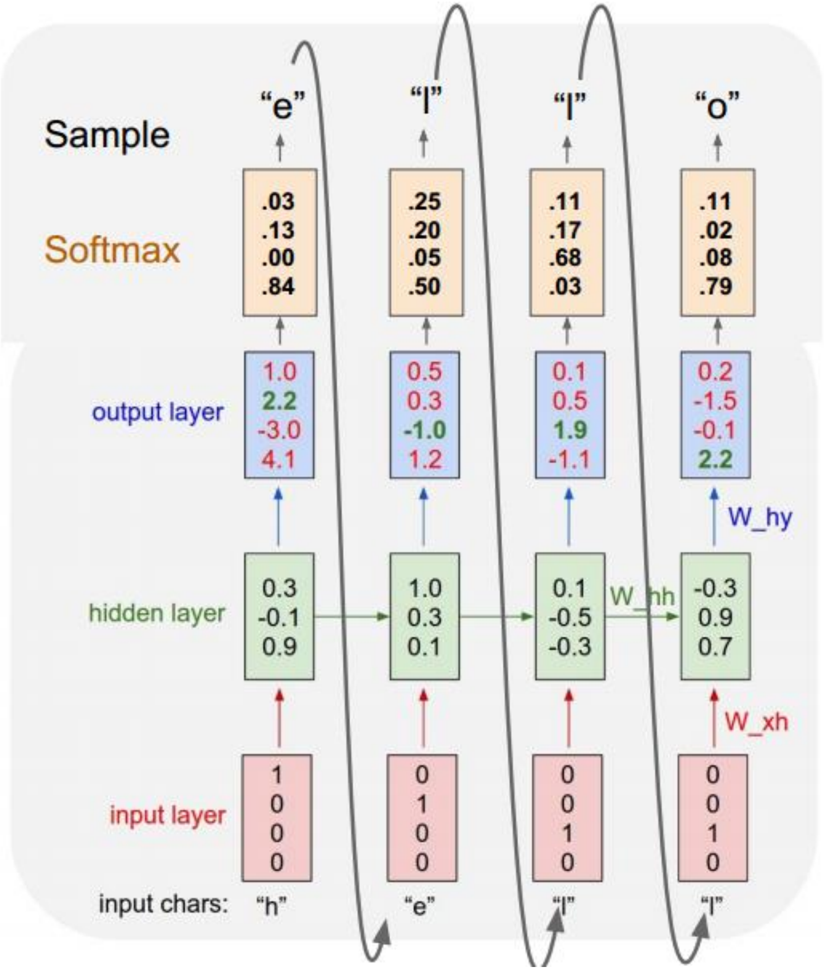


RNN for language modeling

Example:
Character-level Language Model Sampling

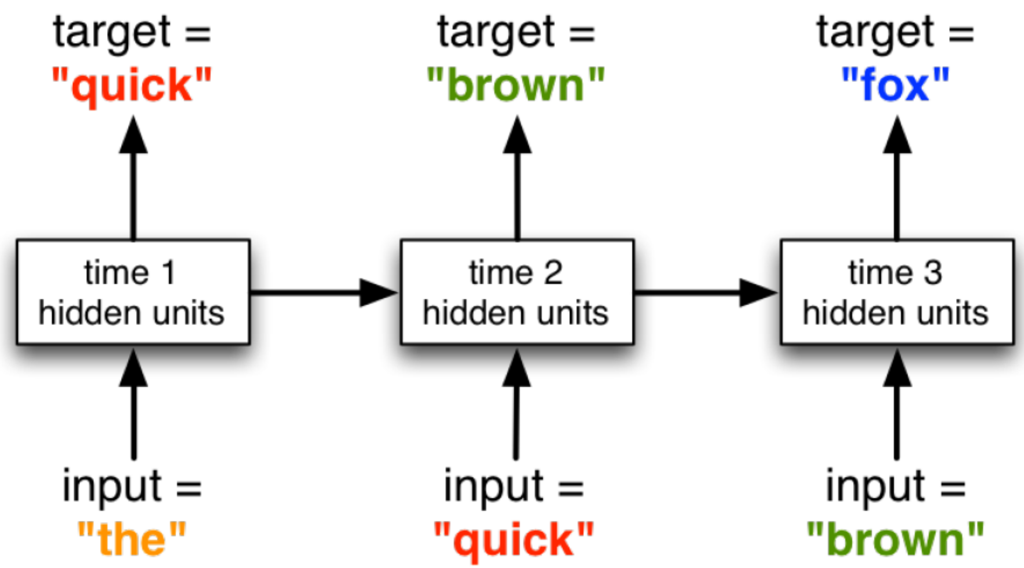
Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



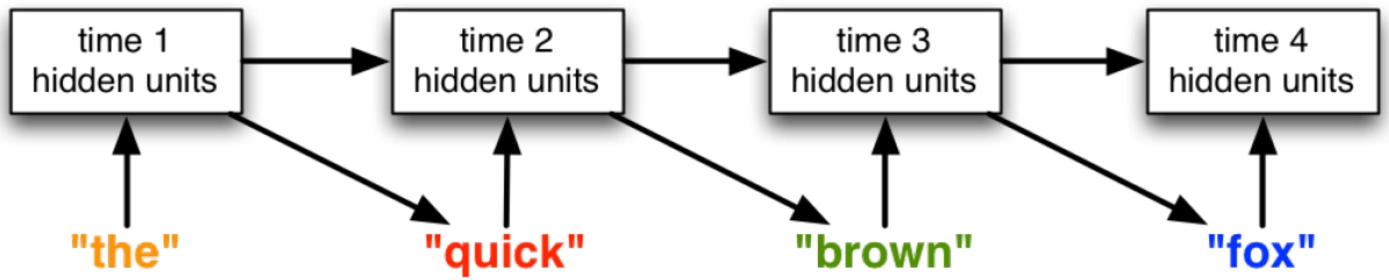
RNN for language modeling

- Modeling at word level
  - Each word is represented as an indicator vector
  - The model predicts a distribution over words



RNN for language modeling

- Generating from a RNN language model
  - The outputs are fed back to the network



- Training time: the inputs are the token from the training set (teacher forcing).

RNN for language modeling

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

RNN for language modeling

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

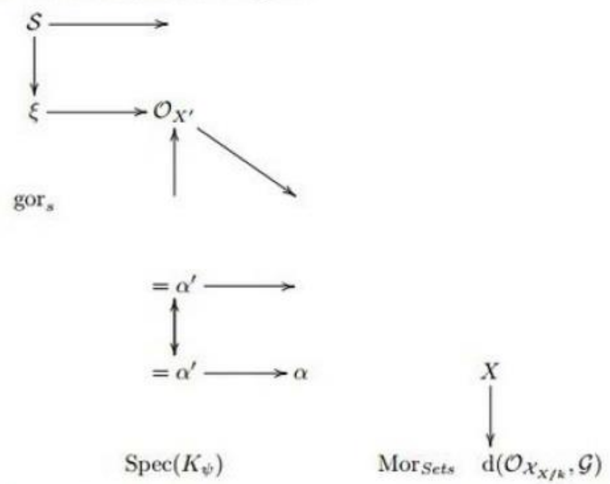
(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

- Generated math from algebraic geometry textbook

RNN for language modeling

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

$\square$

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. $\square$

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. $\square$

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

- Generated math from algebraic geometry textbook

RNN for language modeling

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

- Generated C code

RNN for language modeling

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

- Generated C code

RNN for language modeling

- Some remaining challenges
  - Vocabularies can be very large once you include people, places, etc. It's computationally difficult to predict distributions over millions of words.
  - How do we deal with words we haven't seen before?
  - In some languages (e.g. Chinese), it's hard to determine what should be considered a word.

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

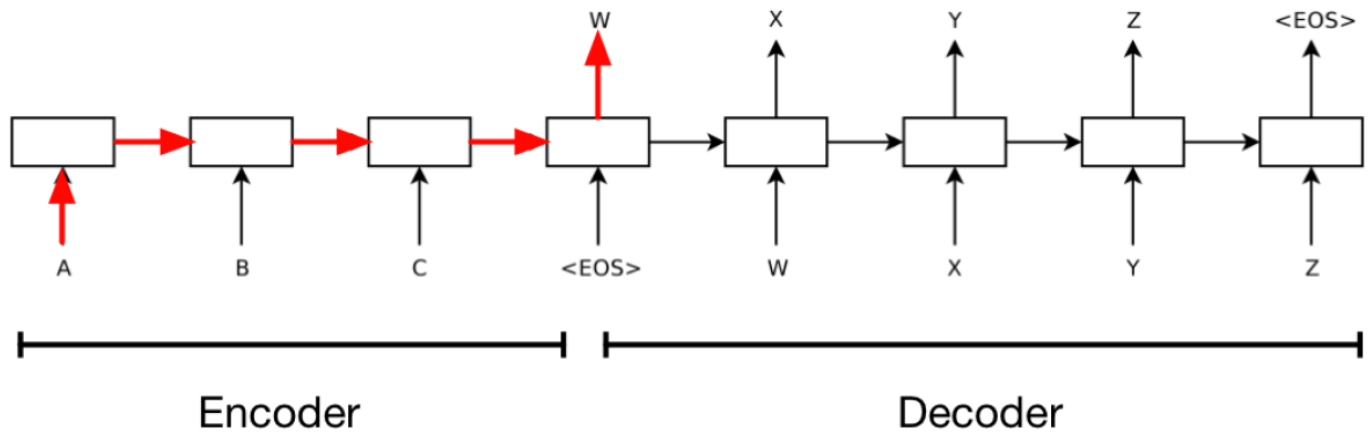
Outline

- 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

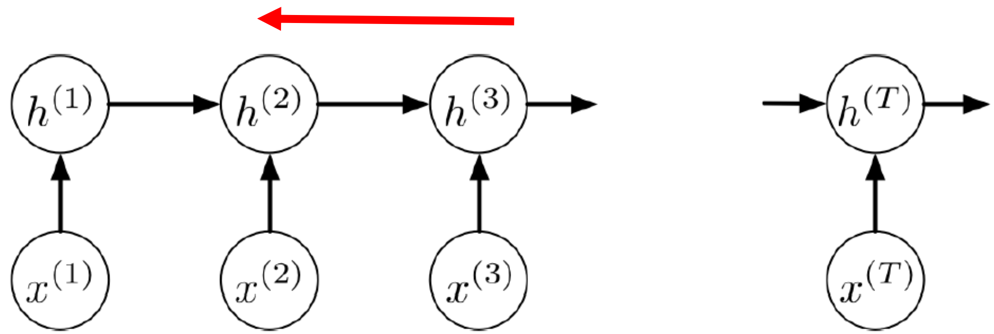
- Motivating example: machine translation



- 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



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

Lecture_notes_5

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:

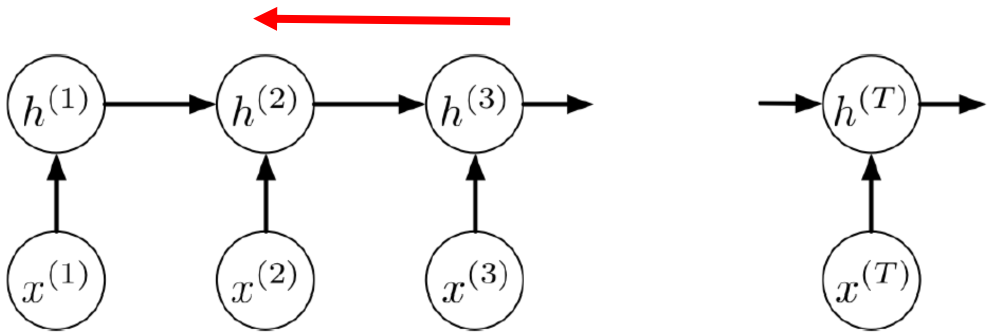
$$w = 1.1, T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 117.4$$

Vanishing:

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

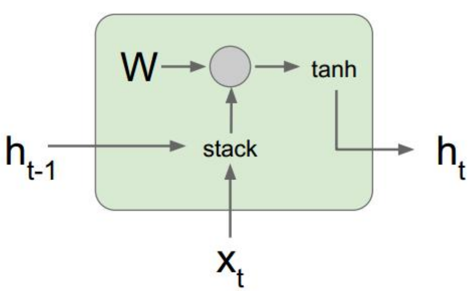
Why gradients explode or vanish

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



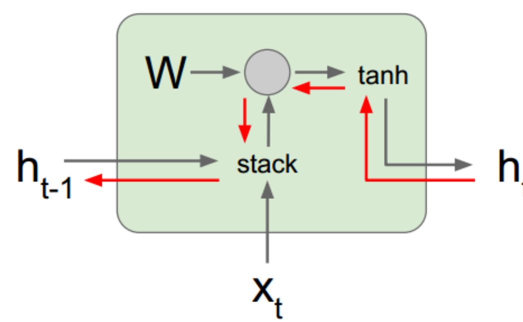
$$z^{(t+1)} = wh^{(t)} + vx^{(t+1)}$$
$$h^{(t)} = \phi(z^{(t)})$$

  - General example on the multivariate case



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

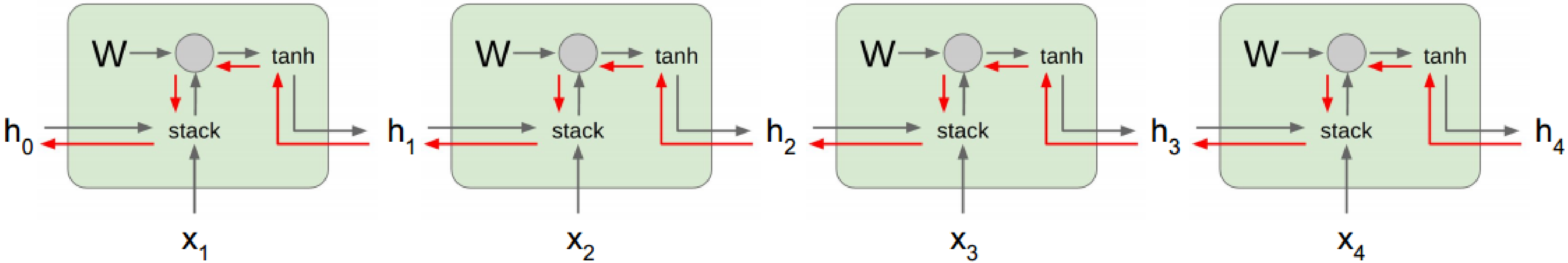
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



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.

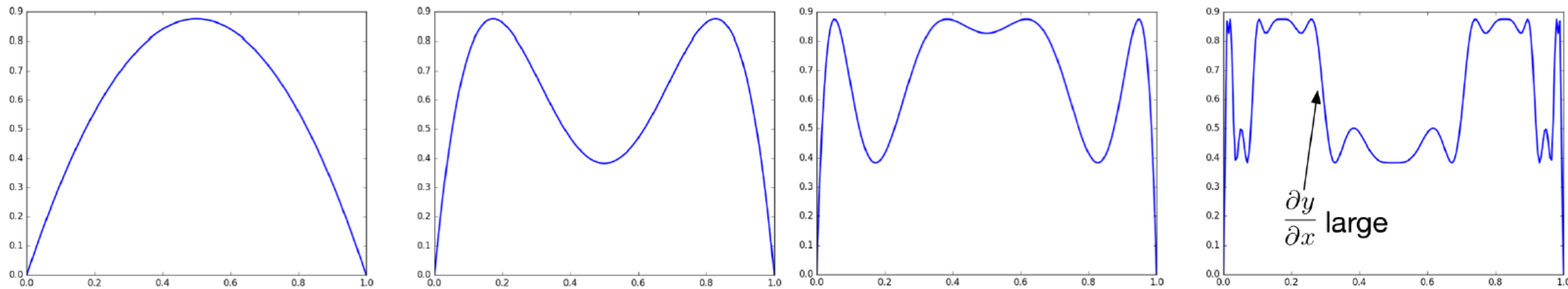
A dynamic system perspective

- 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.:

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

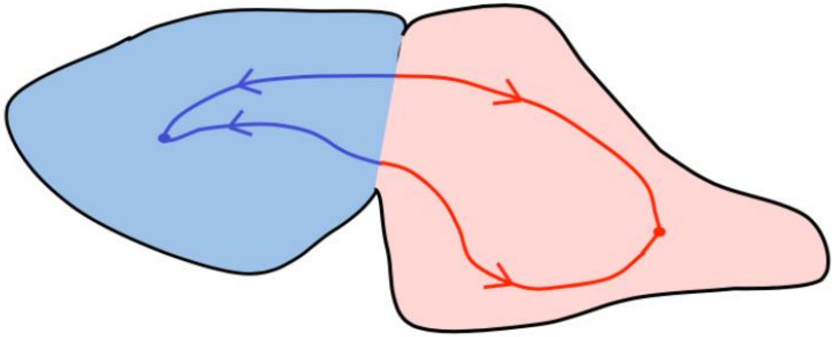


$y = f(x)$ $\quad$ $y = f(f(x))$ $\quad$ $y = f(f(f(x)))$ $\quad$ $y = \underbrace{f \circ \cdots \circ f}_{6\ \text{times}}(x)$

A dynamic system perspective

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

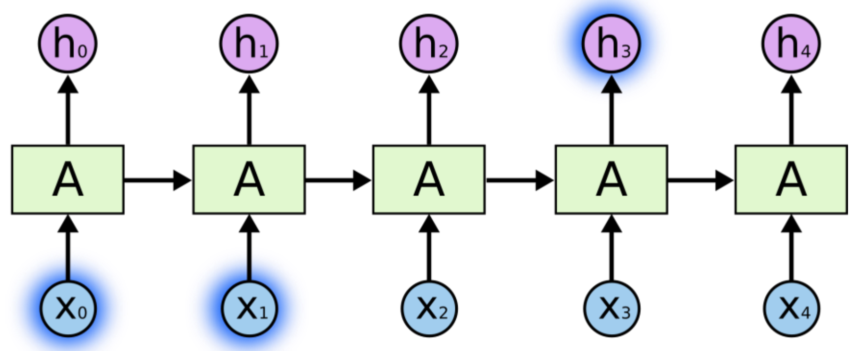
$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

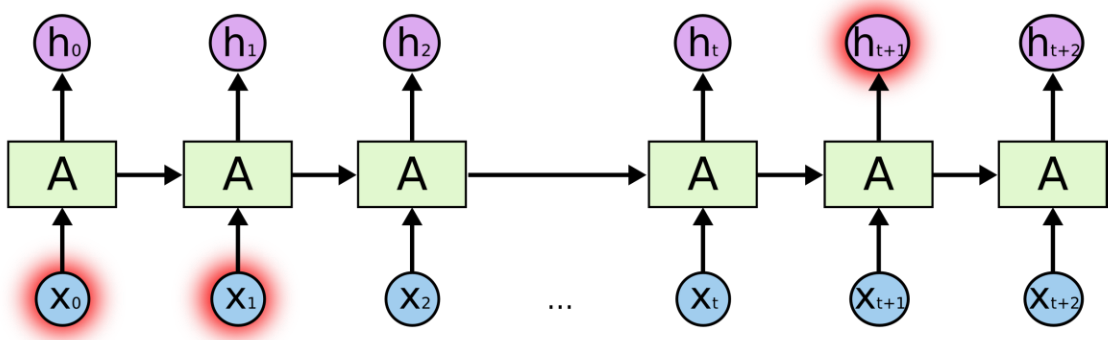


  - Within one of the colored regions, the gradients vanish because even if you move a little, you still wind up at the same attractor.
  - If you're on the boundary, the gradient blows up because moving slightly moves you from one attractor to the other.

Vanilla RNN

- Difficulty in modeling long-term dependency





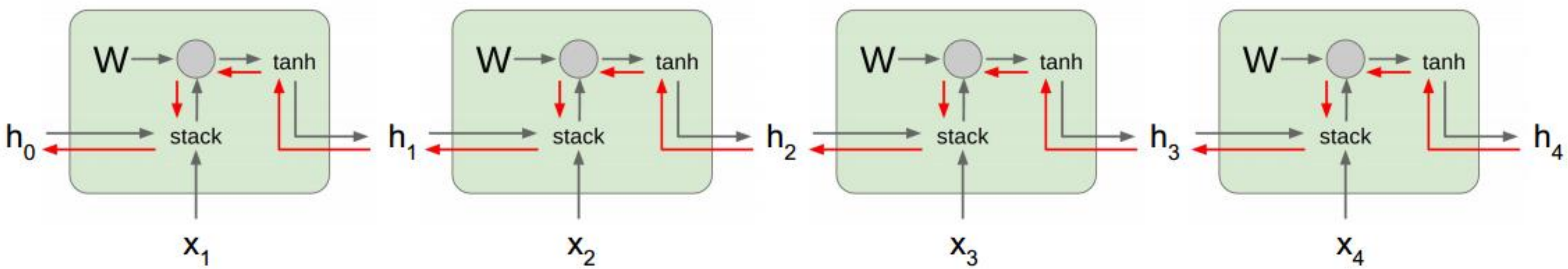
Outline

- 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

Largest singular value < 1:
Vanishing 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)
```

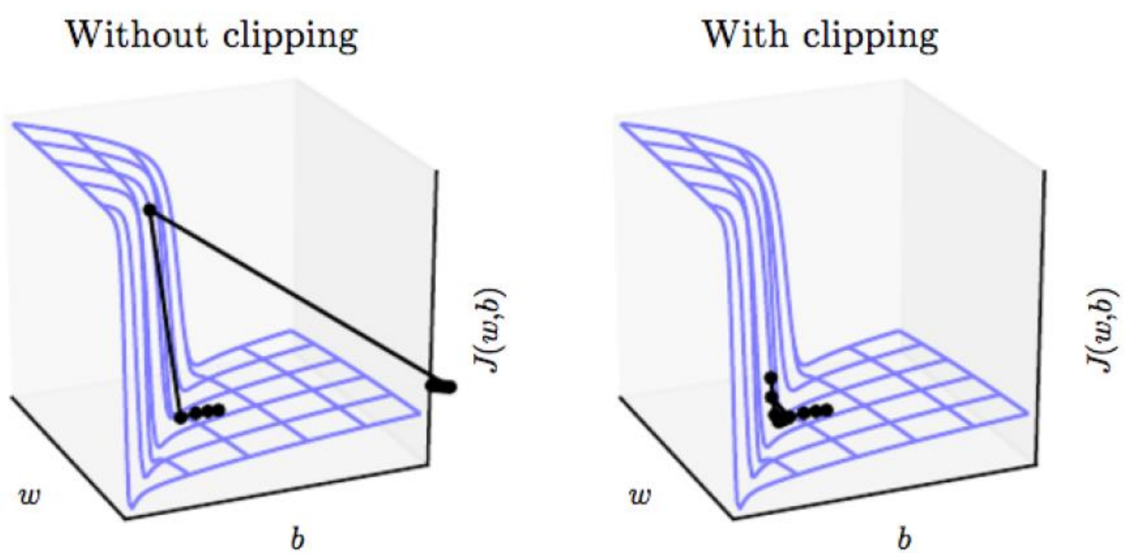
Stabilizing RNN training

- Gradient clipping

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

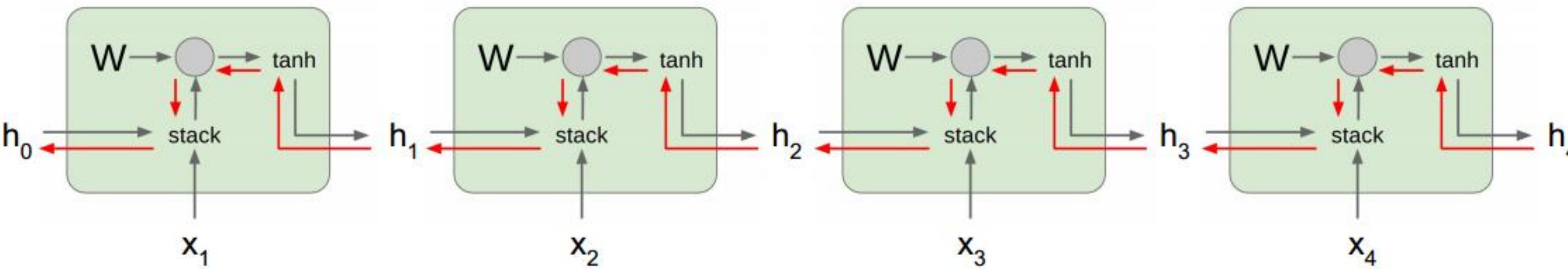
  if $\|\mathbf{g}\| > \eta$:
  $$\mathbf{g} \leftarrow \frac{\eta \mathbf{g}}{\|\mathbf{g}\|}$$

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



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

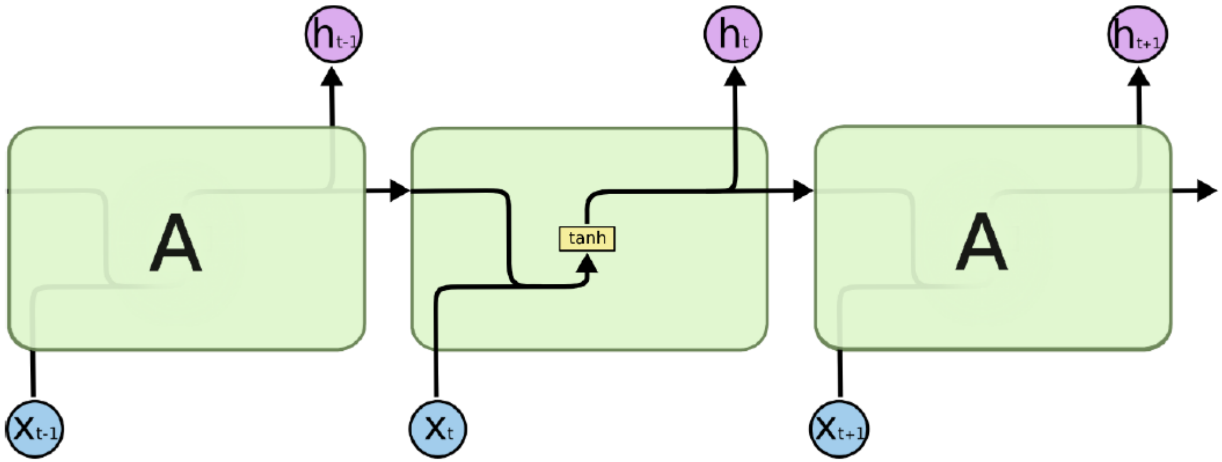
Acknowledgement: Feifei Li et al's cs231n notes

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

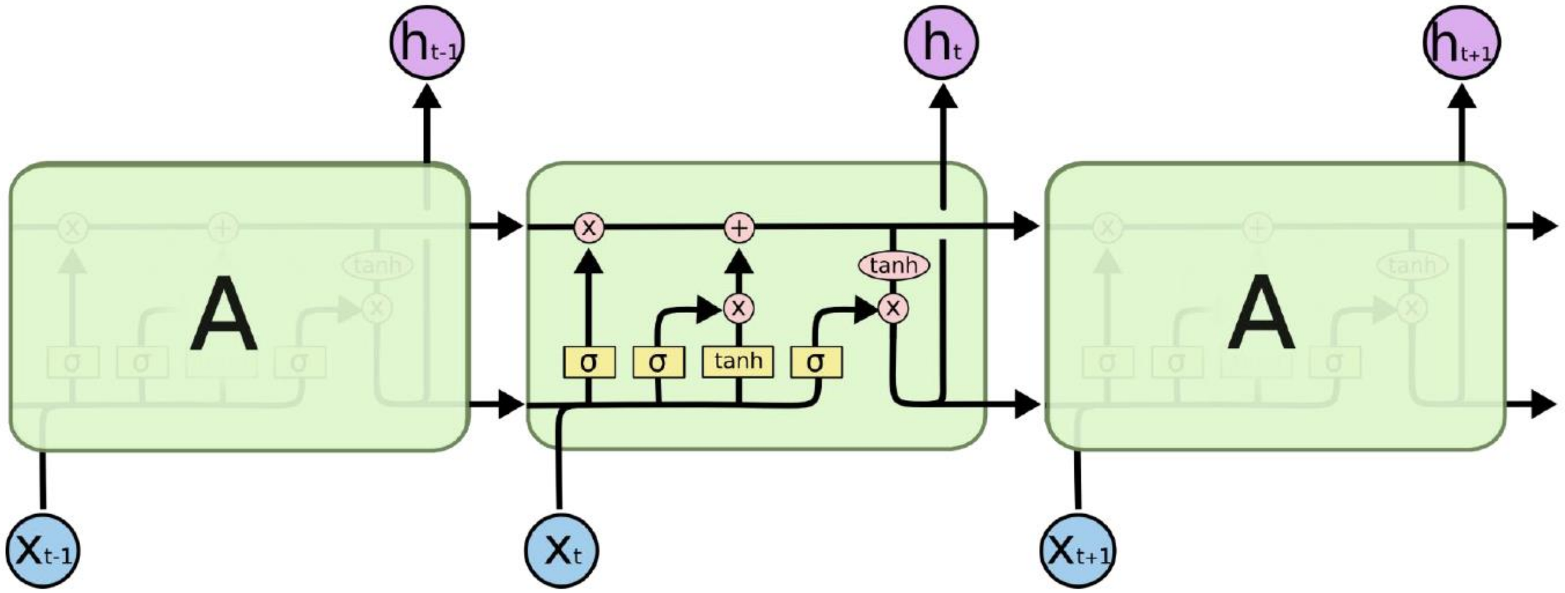
- Recall



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

Long Short Term Memory(LSTM)

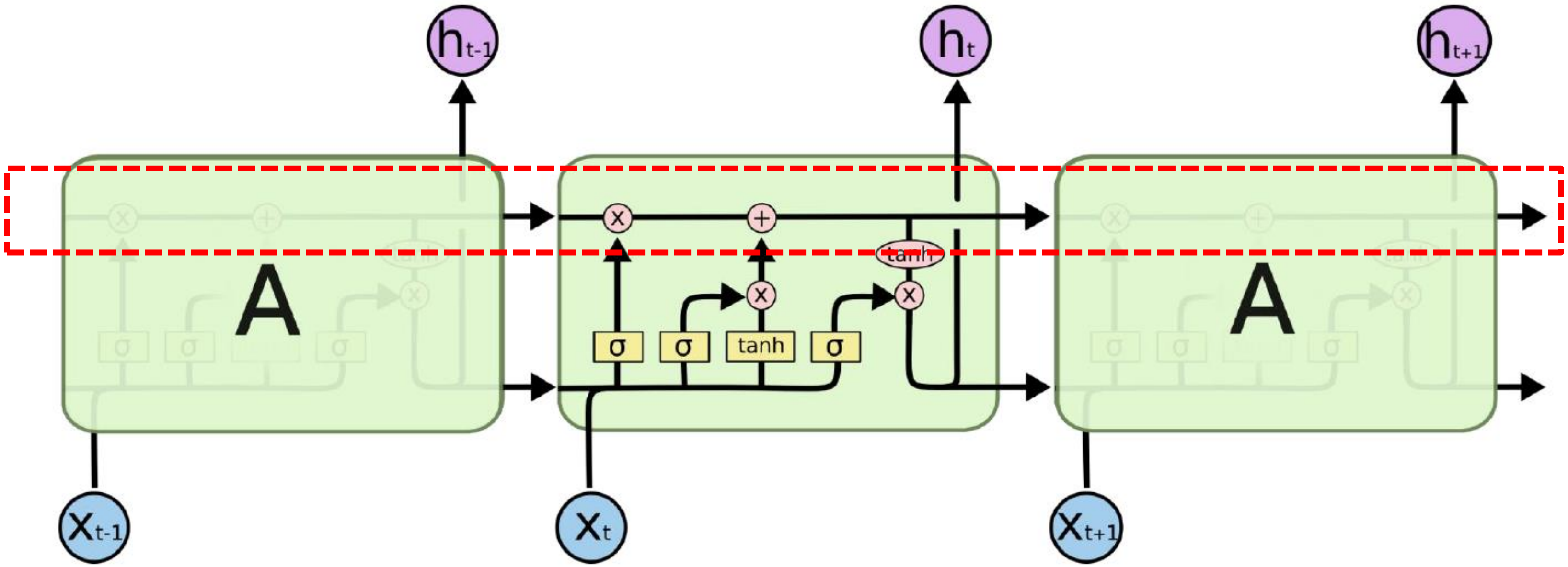
- LSTM uses multiplicative gates that decide if something is important or not



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

Long Short Term Memory(LSTM)

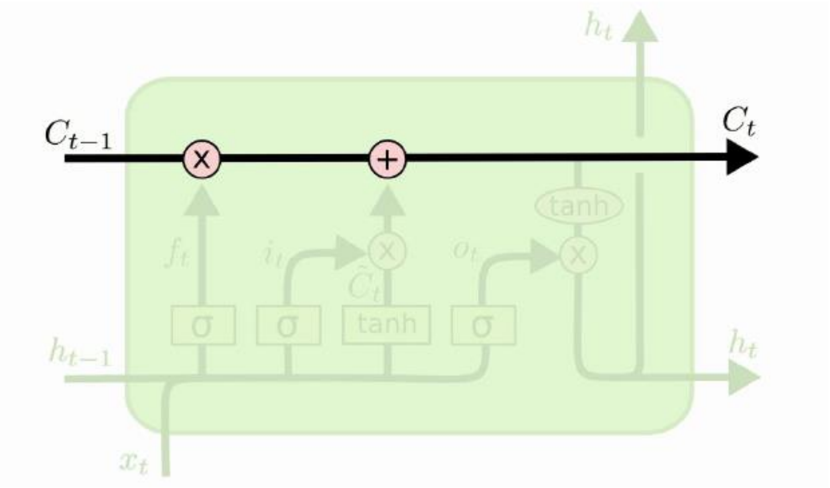
- Key component: a remembered cell state



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

LSTM: cell state

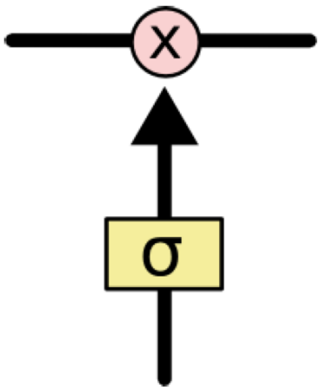
- A linear history
  - Carries information through
  - Only affected by a gate and addition of current information, which is also gated



LSTM: gates

Gates are simple sigmoid units with output range in (0,1)

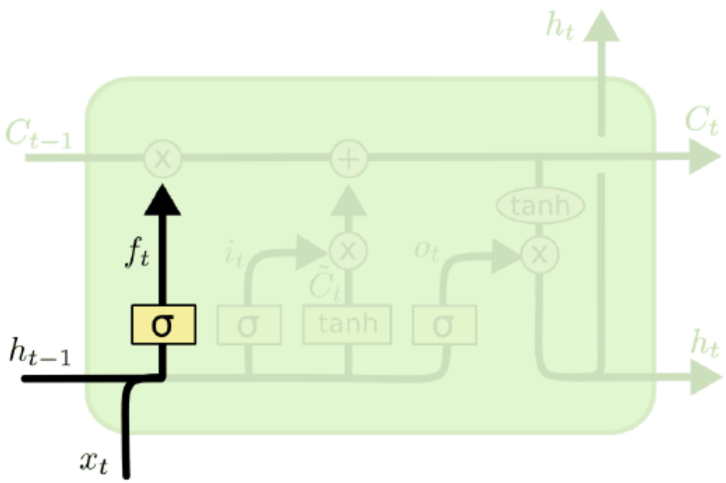
- Controls how much of the information will be let through



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

LSTM: forget gate

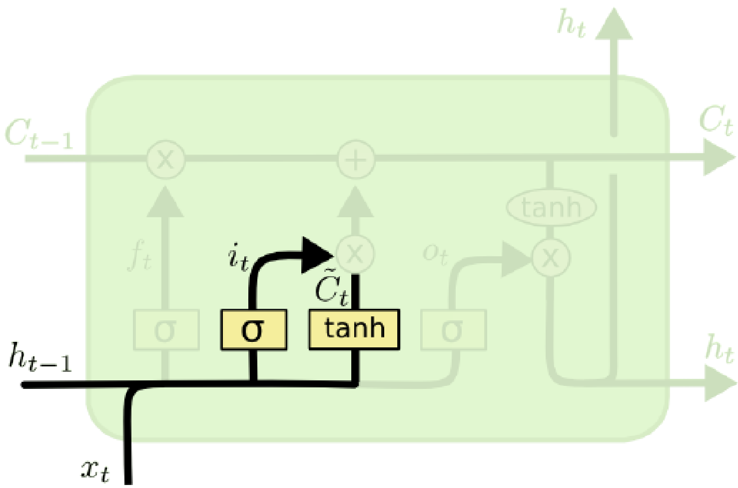
- The first gate determines whether to carry over the history or to forget it
  - Soft decision: how much of the history C_{t-1} to carry over
  - Determined by the current input x_t and the previous state h_{t-1}
  - $\langle h_{t-1}, C_{t-1} \rangle$ can be viewed as partial key-value pairs



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

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

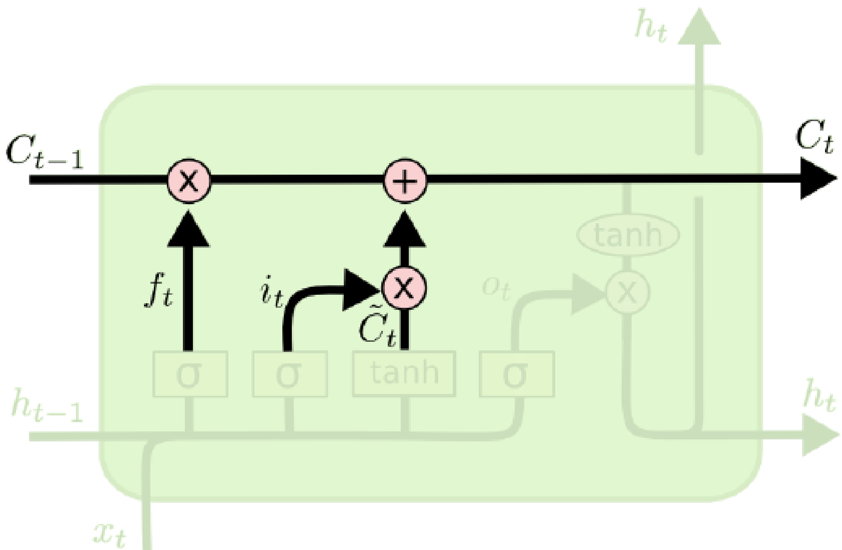


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTM: Memory cell update

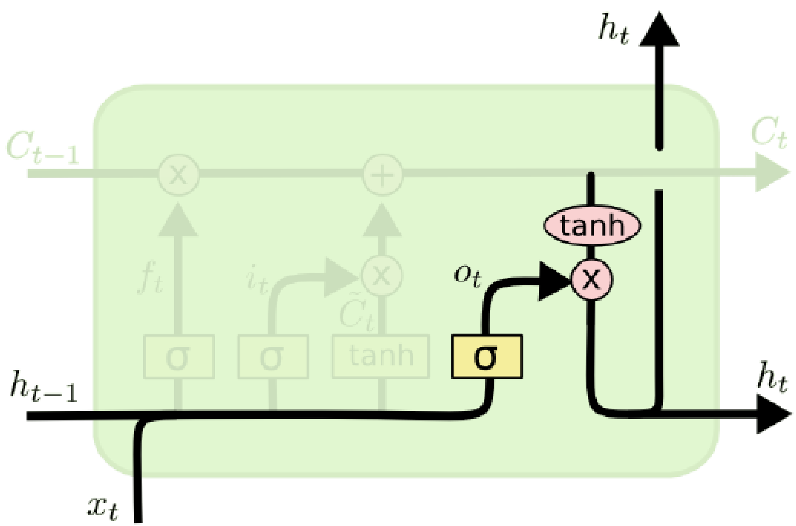
- The output of the second part is added into the current memory cell
  - Can be viewed as value update in a key-value pair
  - The input and state jointly decide how much of history info is kept and how much of embedded input info is added
  - A dynamic mixture of experts at each time step



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

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

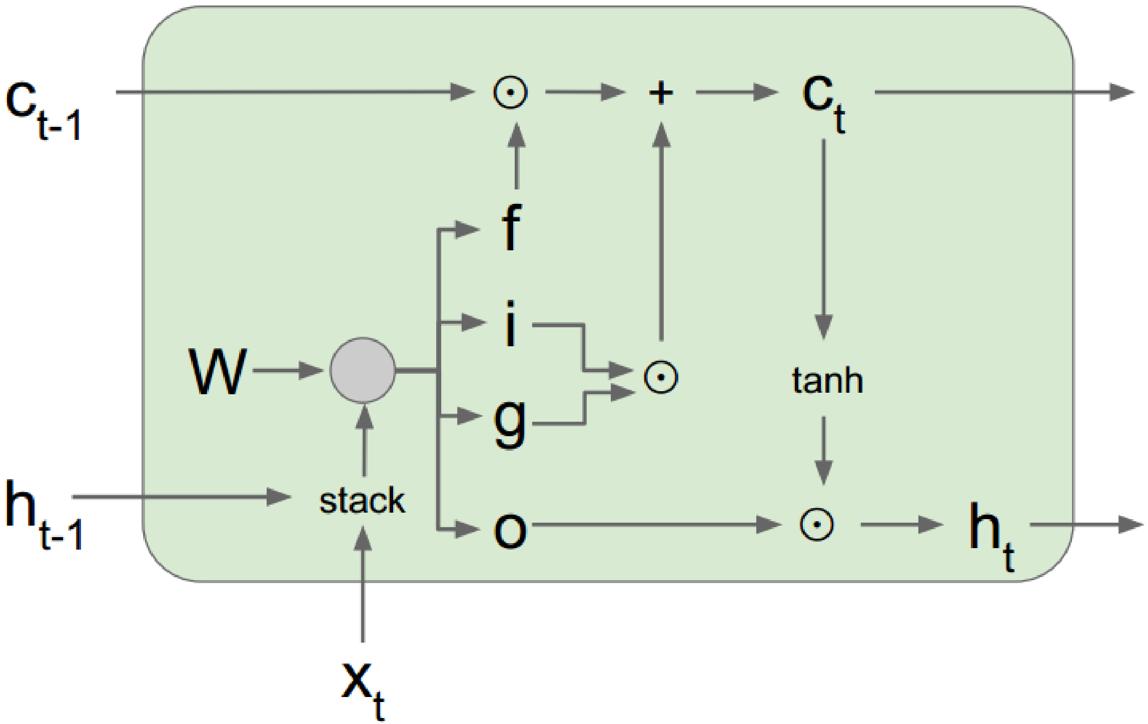


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Long Short Term Memory(LSTM)

[Hochreiter et al., 1997]



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

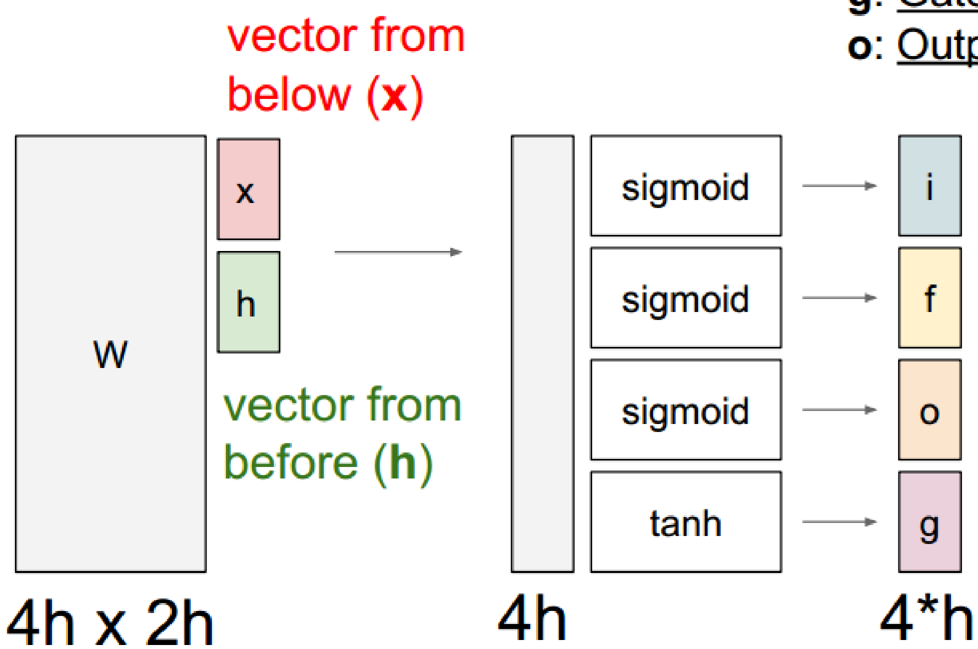
$$c_t = f \odot c_{t-1} + i \odot g$$

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

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



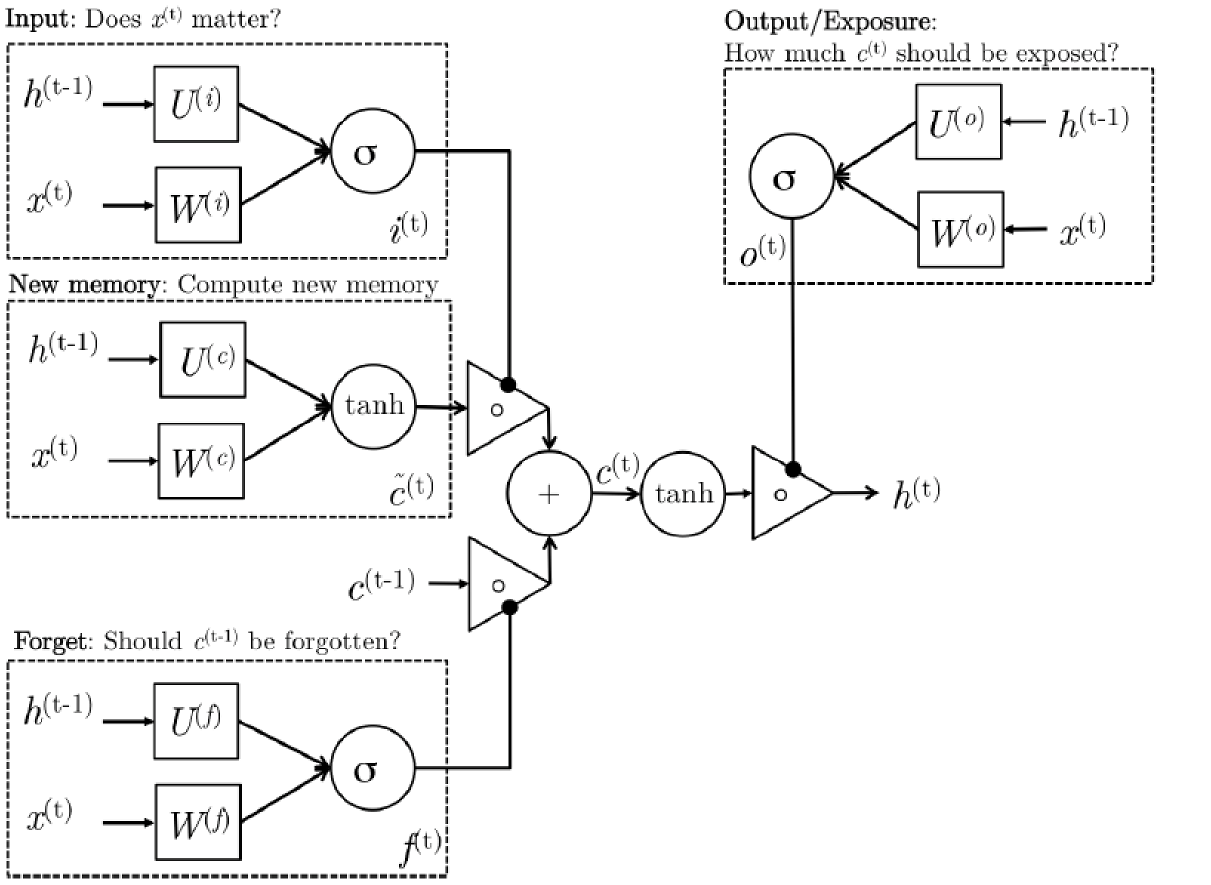
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

LSTM: as feedforward layer

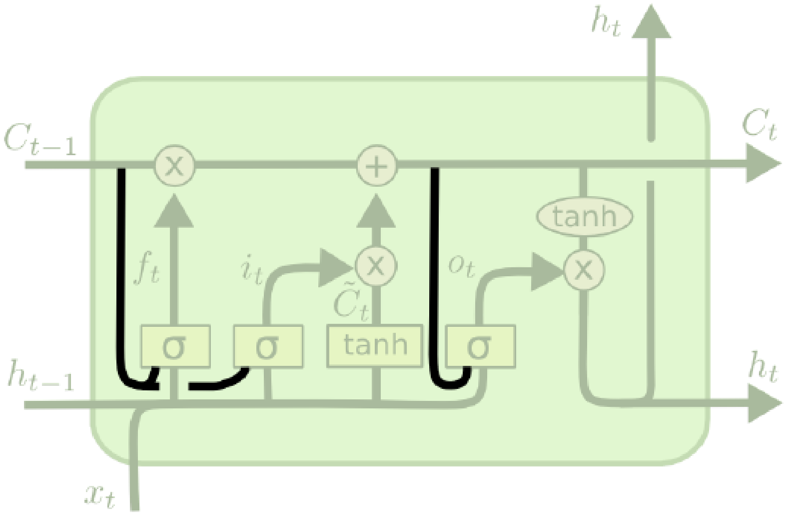
- As a gated feedforward network



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

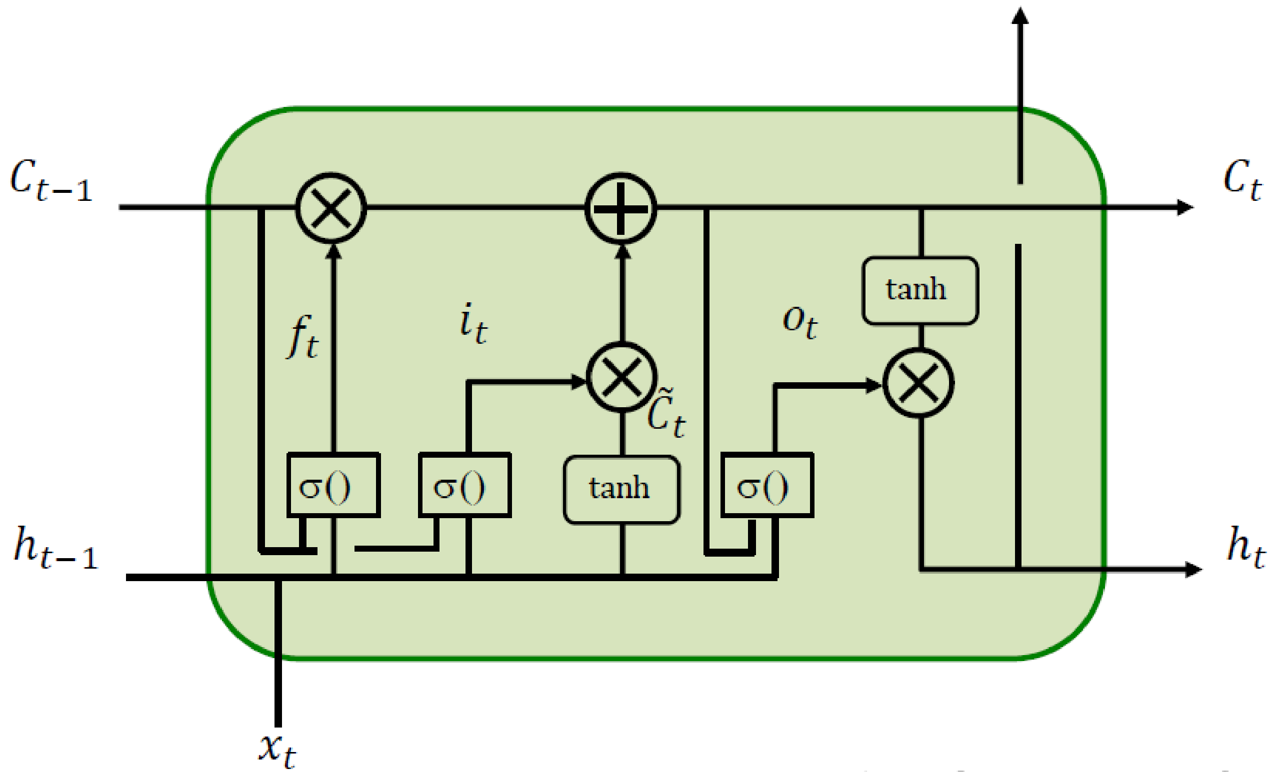


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

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

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

Computation: forward in full model



- Forward rules:

Gates

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

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

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

Variables

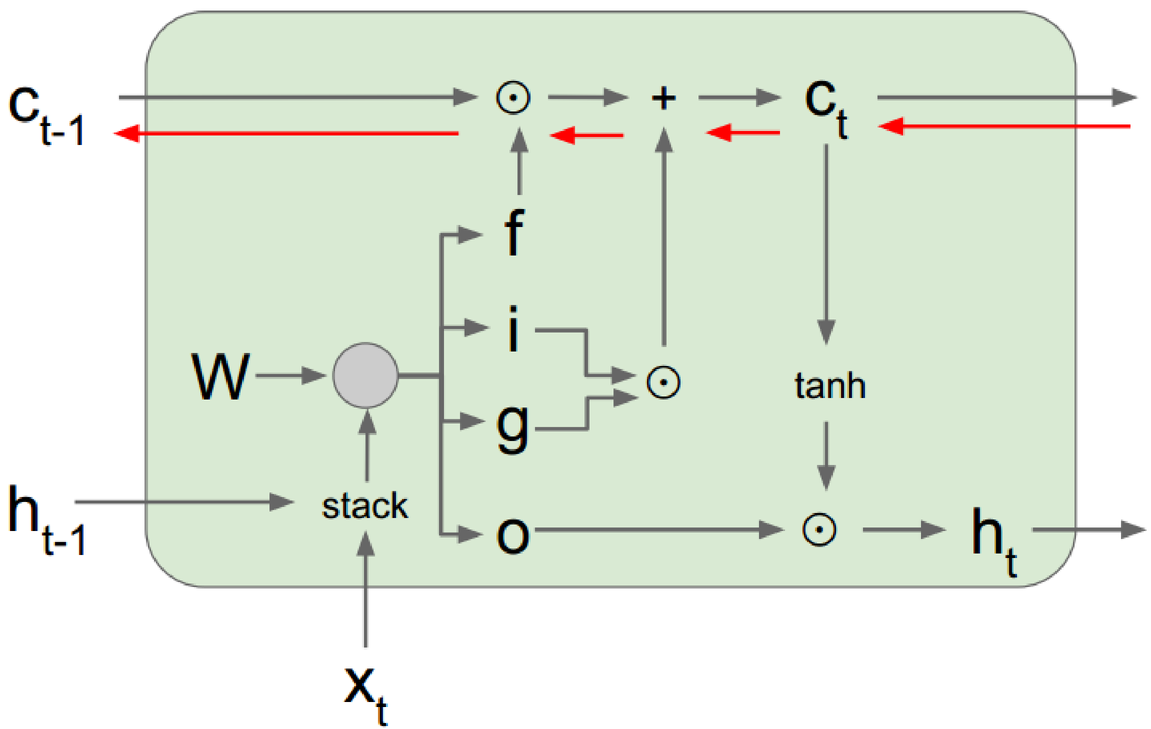
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

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

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

LSTM: Backpropagation

[Hochreiter et al., 1997]



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

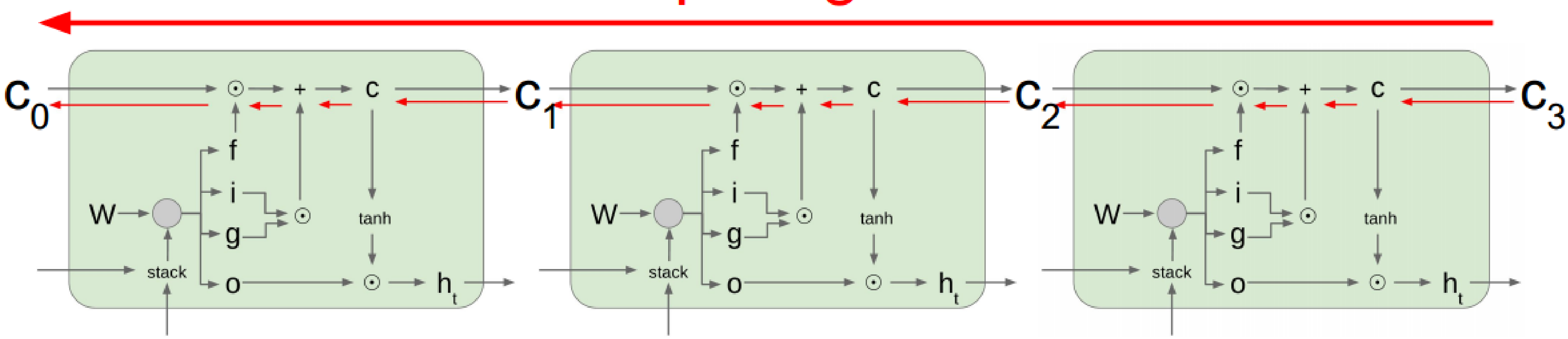
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

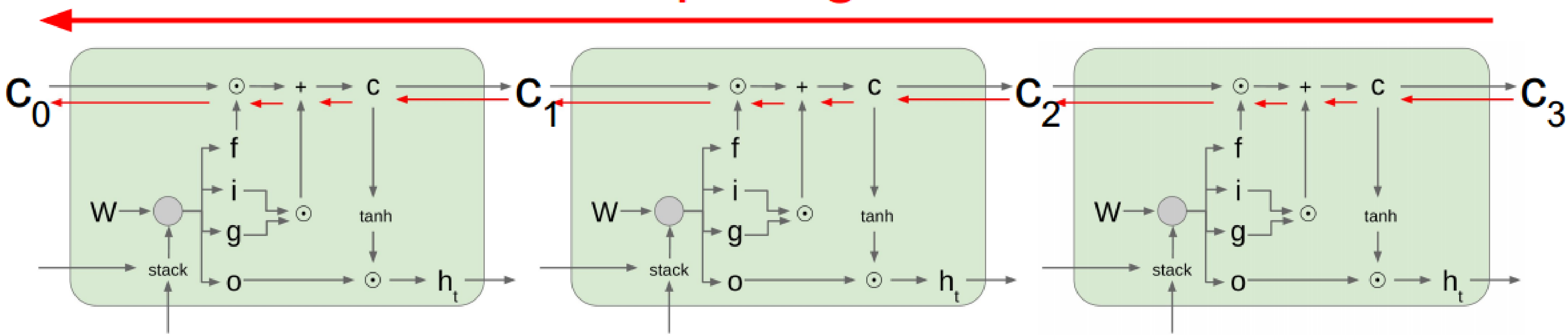
LSTM: Backpropagation

Uninterrupted gradient flow!

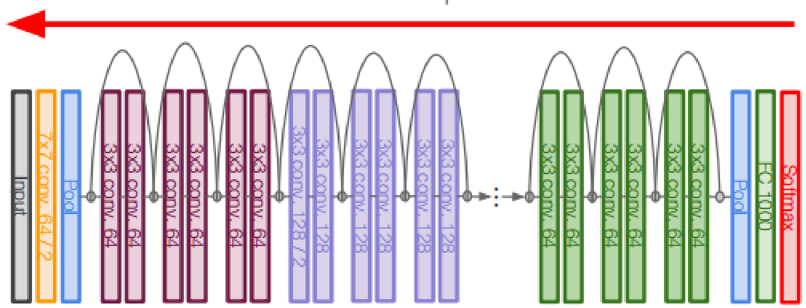


LSTM: Backpropagation

Uninterrupted gradient flow!



Similar to ResNet!



In between:
Highway Networks

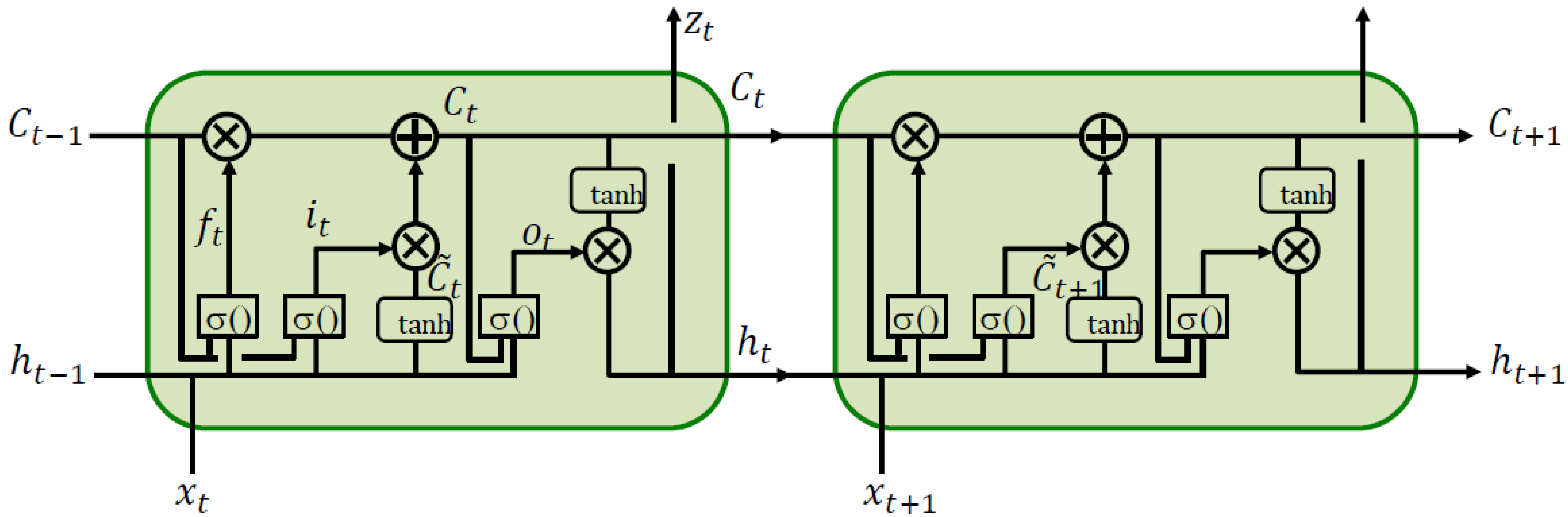
$$g = T(x, W_T)$$

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

Srivastava et al, “Highway Networks”, ICML DL Workshop 2015

LSTM: Backpropagation

- Full model version

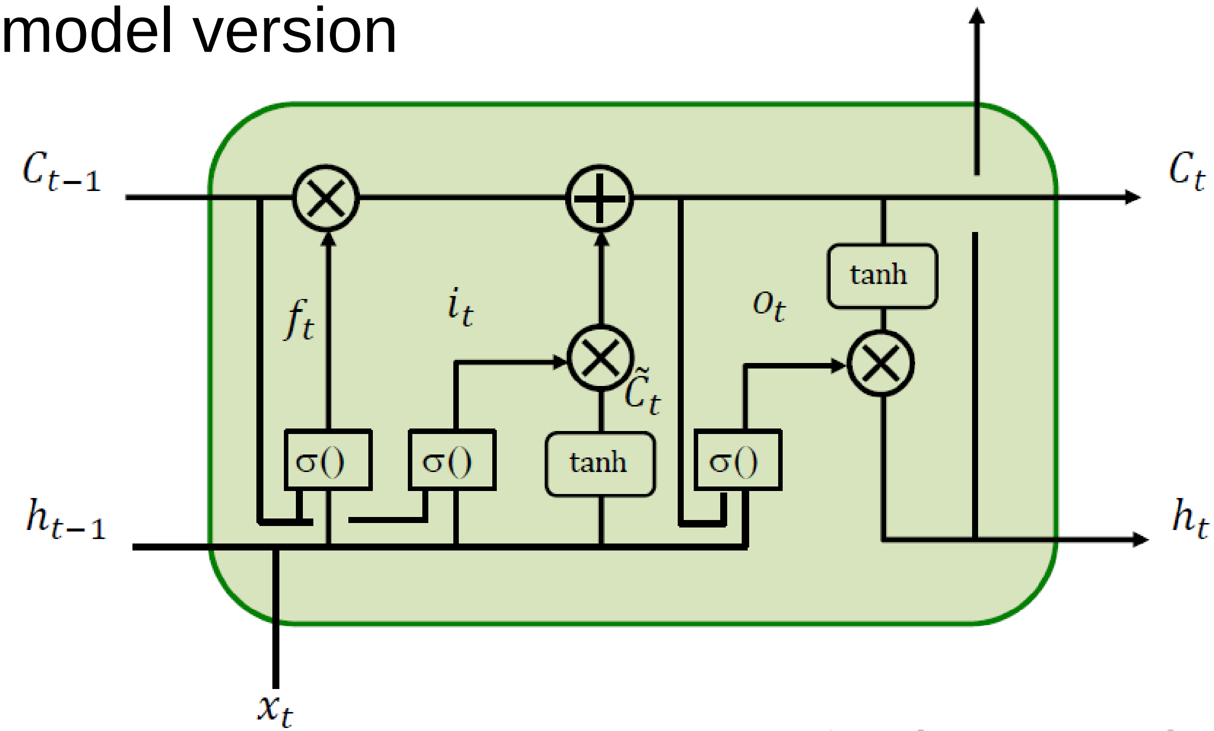


$$\nabla_{C_t} L =$$

$$\nabla_{h_t} L =$$

Computation: forward in full model

- Full model version



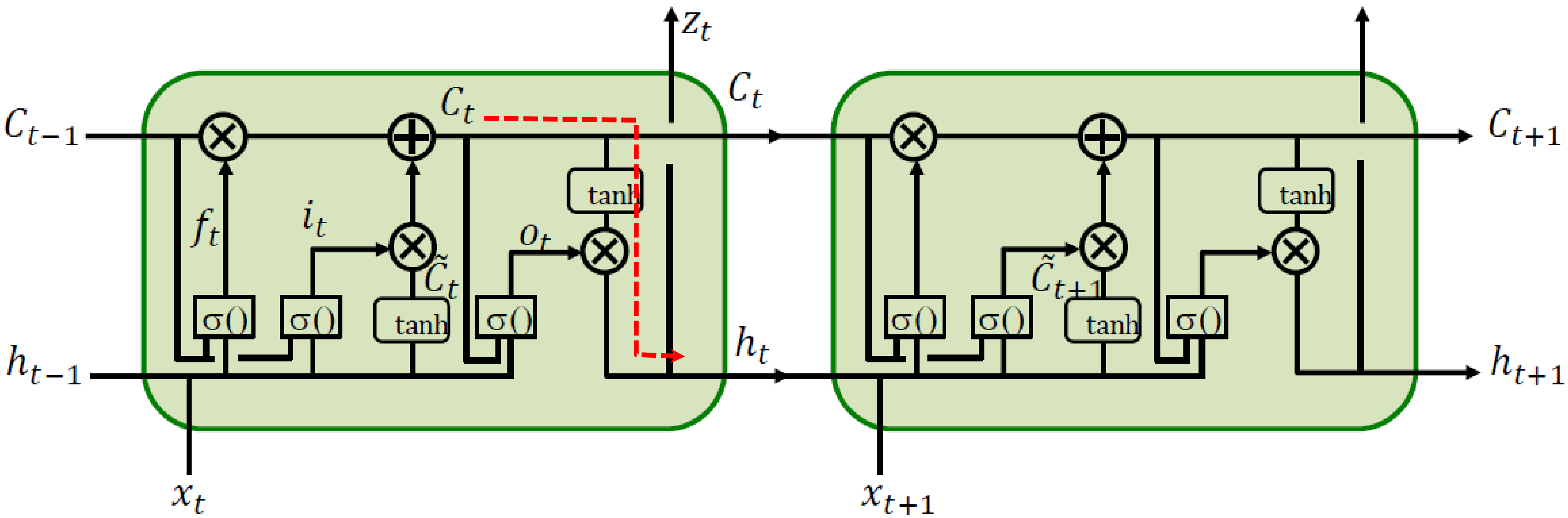
- Forward rules:

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

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

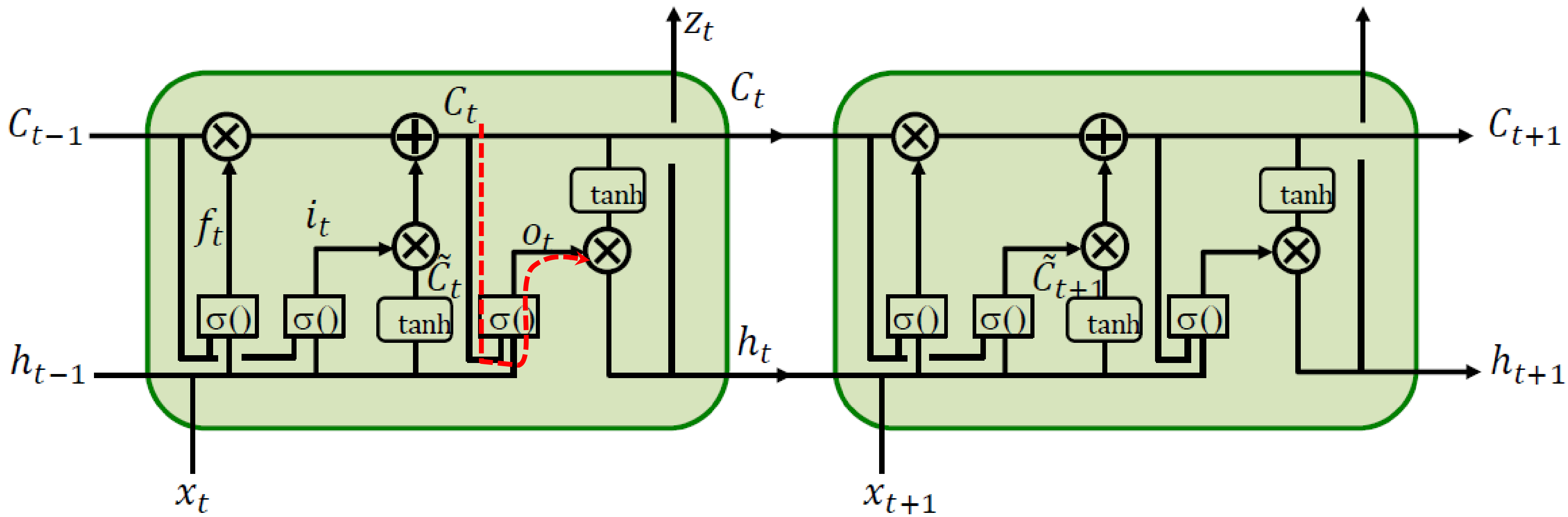
- Full model version



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

LSTM: Backpropagation

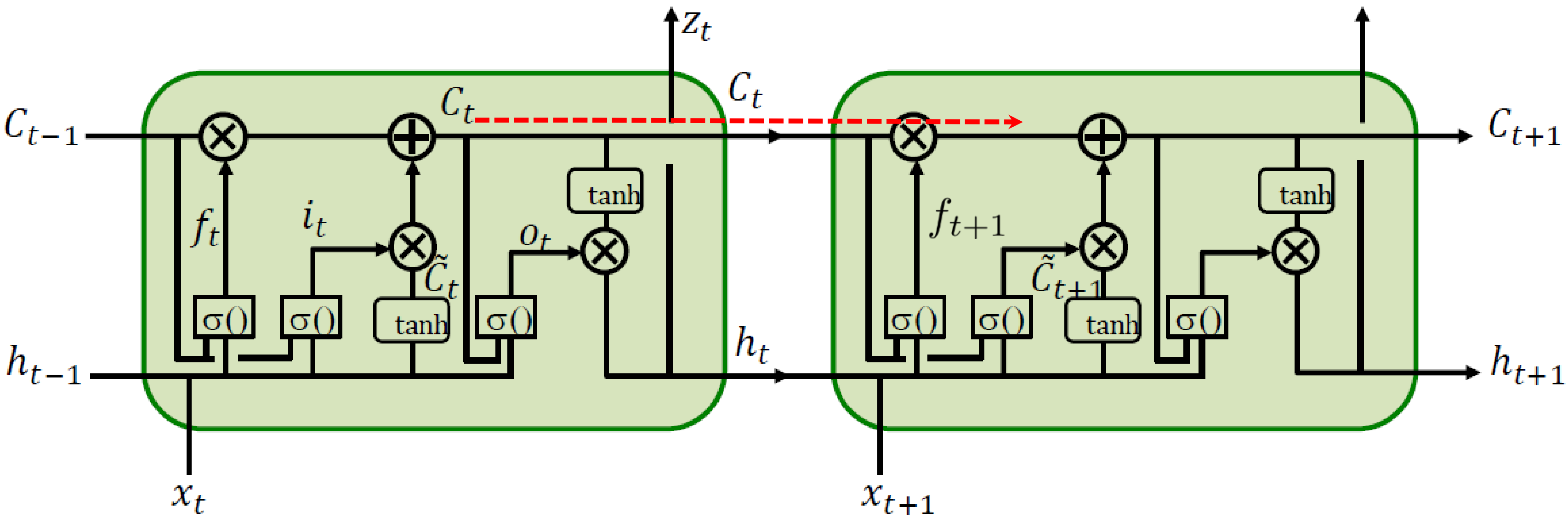
- Full model version



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

LSTM: Backpropagation

- Full model version

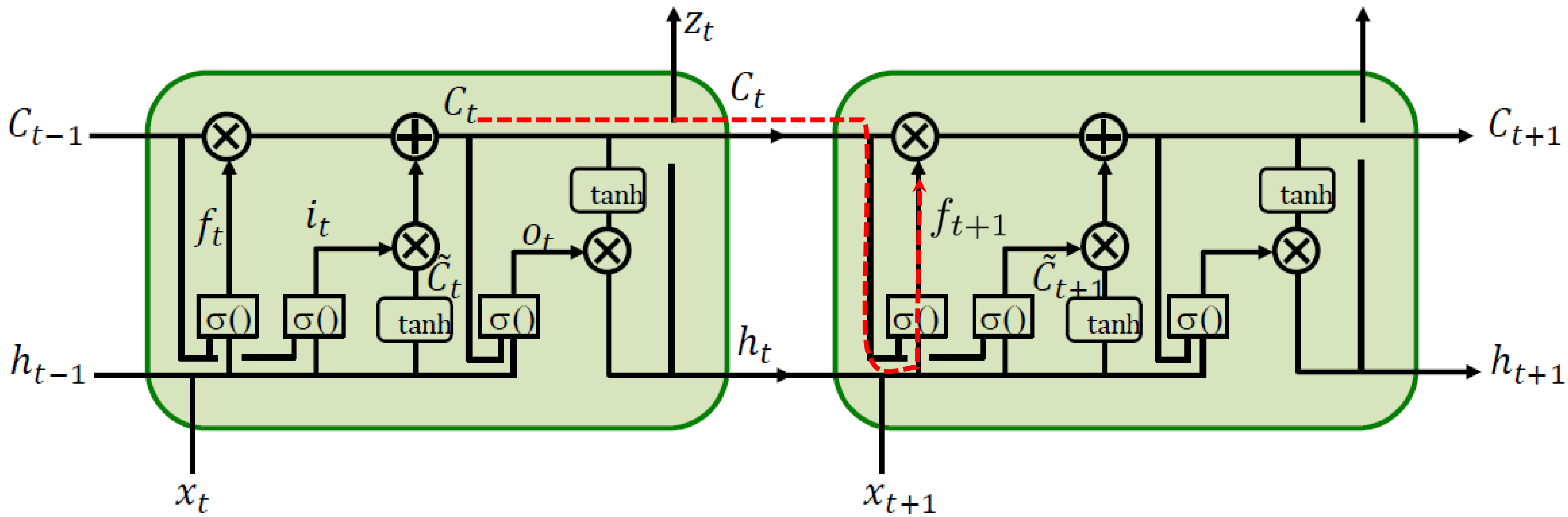


$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

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

LSTM: Backpropagation

- Full model version

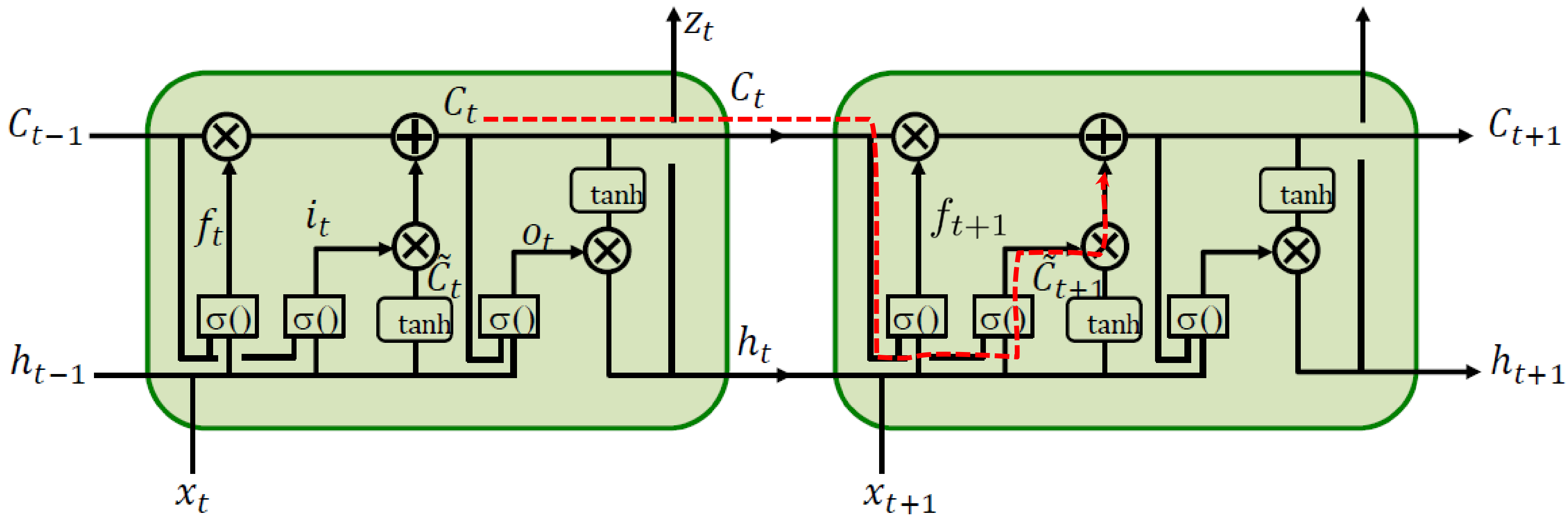


$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

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

LSTM: Backpropagation

- Full model version

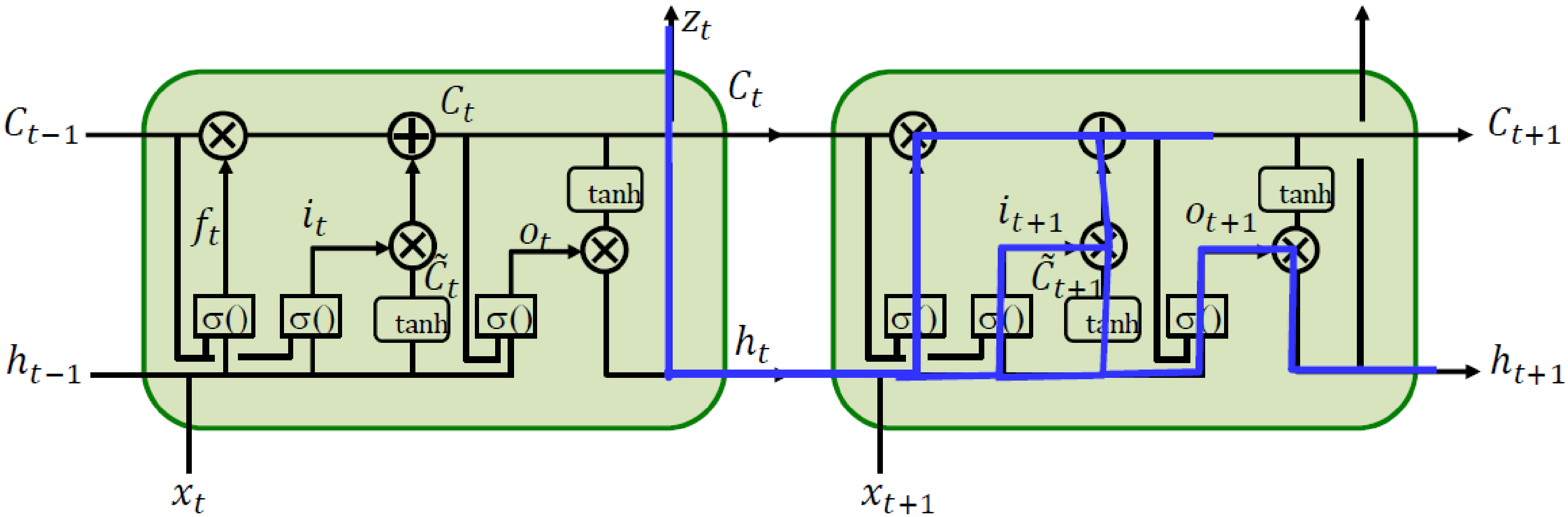


$$\nabla_{C_t} L = \nabla_{h_t} L \circ (o_t \circ \tanh'(\cdot) W_{Ch} + \tanh(\cdot) \circ \sigma'(\cdot) W_{Co})$$

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

LSTM: Backpropagation

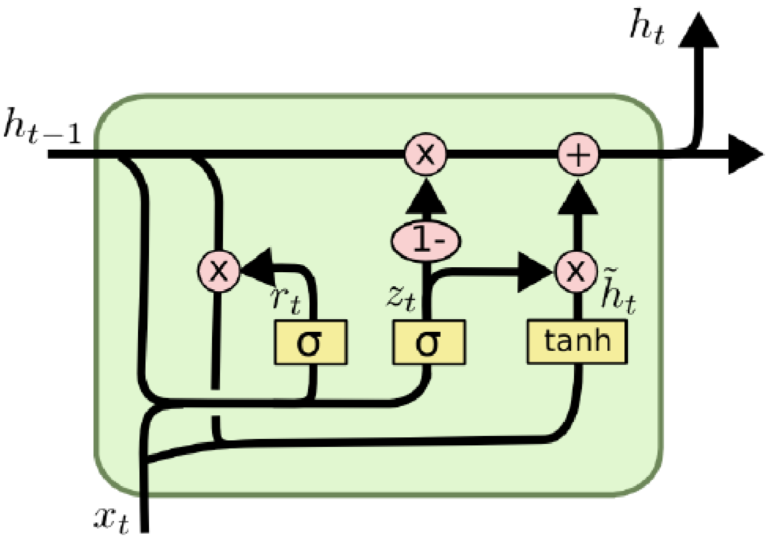
- Full model version



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

Gated Recurrent Units

- Simplified LSTM
  - Can we merge some operations?



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

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

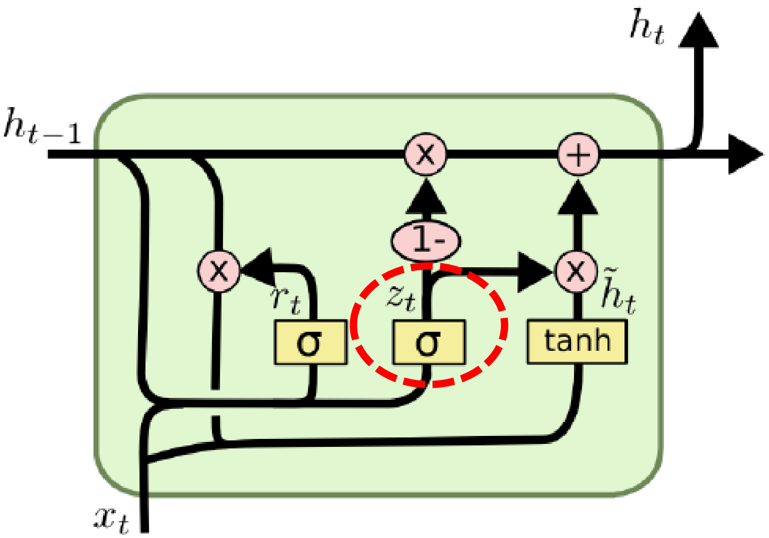
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio, “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”, SSST-8 2014

Gated Recurrent Units

- Simplified LSTM
  - Combine the forget and input gates



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

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

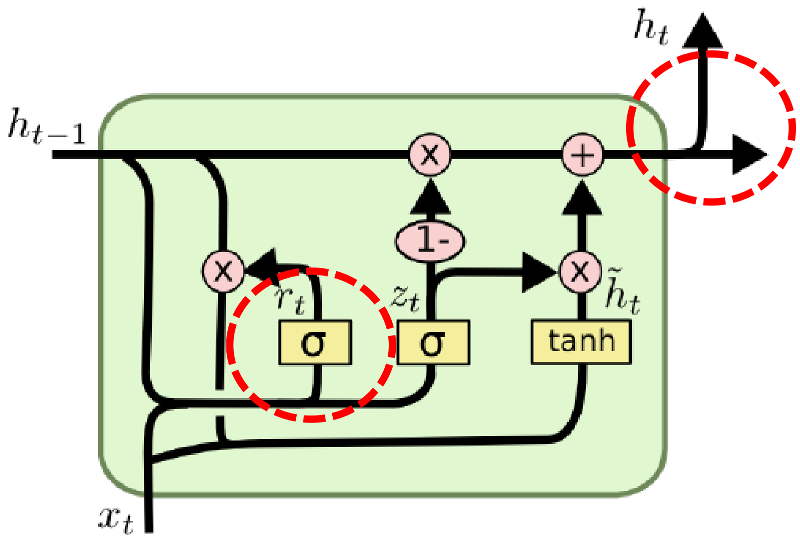
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio, “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”, SSST-8 2014

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



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

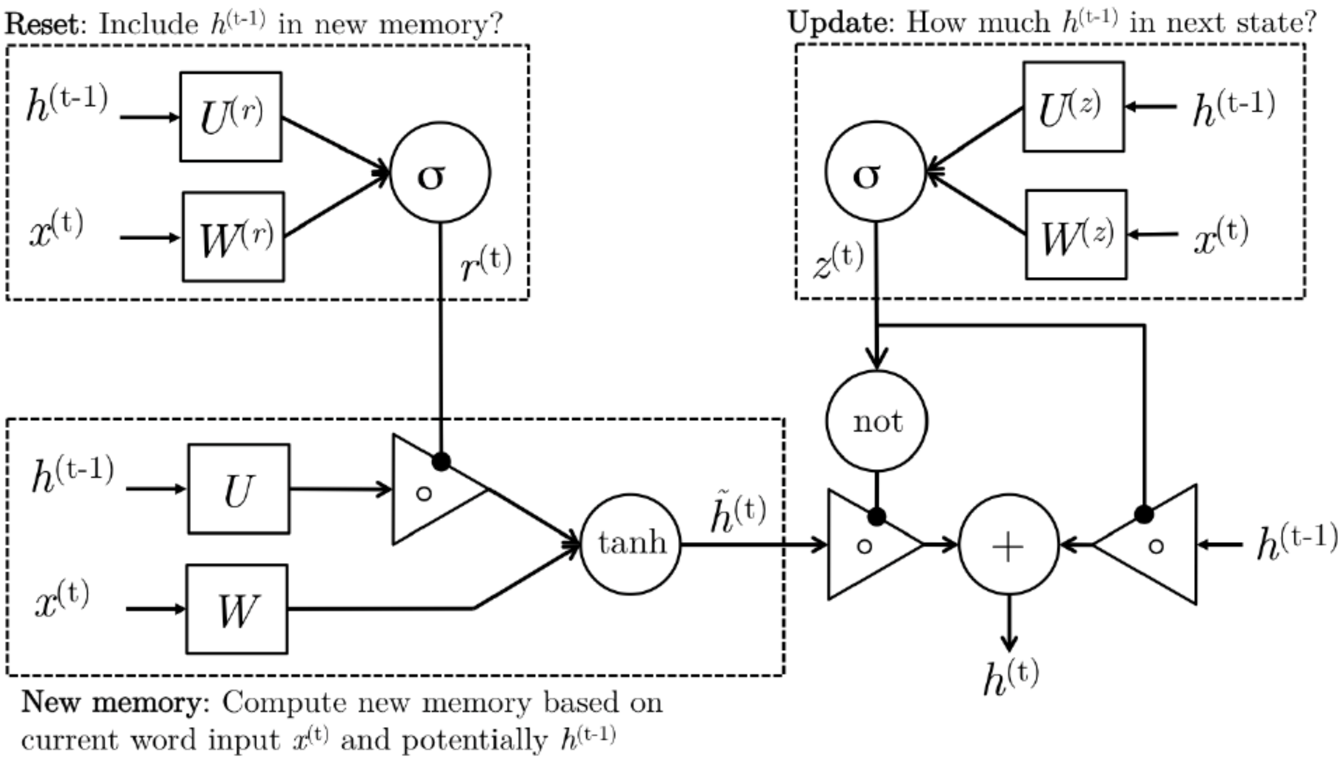
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

GRU: As a feedforward layer

- As a gated feedforward network



$$z^{(t)} = \sigma(W^{(z)}x^{(t)} + U^{(z)}h^{(t-1)}) \qquad \text{(Update gate)}$$

$$r^{(t)} = \sigma(W^{(r)}x^{(t)} + U^{(r)}h^{(t-1)}) \qquad \text{(Reset gate)}$$

$$\tilde{h}^{(t)} = \tanh(r^{(t)} \circ Uh^{(t-1)} + Wx^{(t)}) \qquad \text{(New memory)}$$

$$h^{(t)} = (1 - z^{(t)}) \circ \tilde{h}^{(t)} + z^{(t)} \circ h^{(t-1)} \qquad \text{(Hidden state)}$$

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:

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

MUT2:

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

MUT3:

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

Multi-Layer RNNs

Multilayer RNNs

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

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

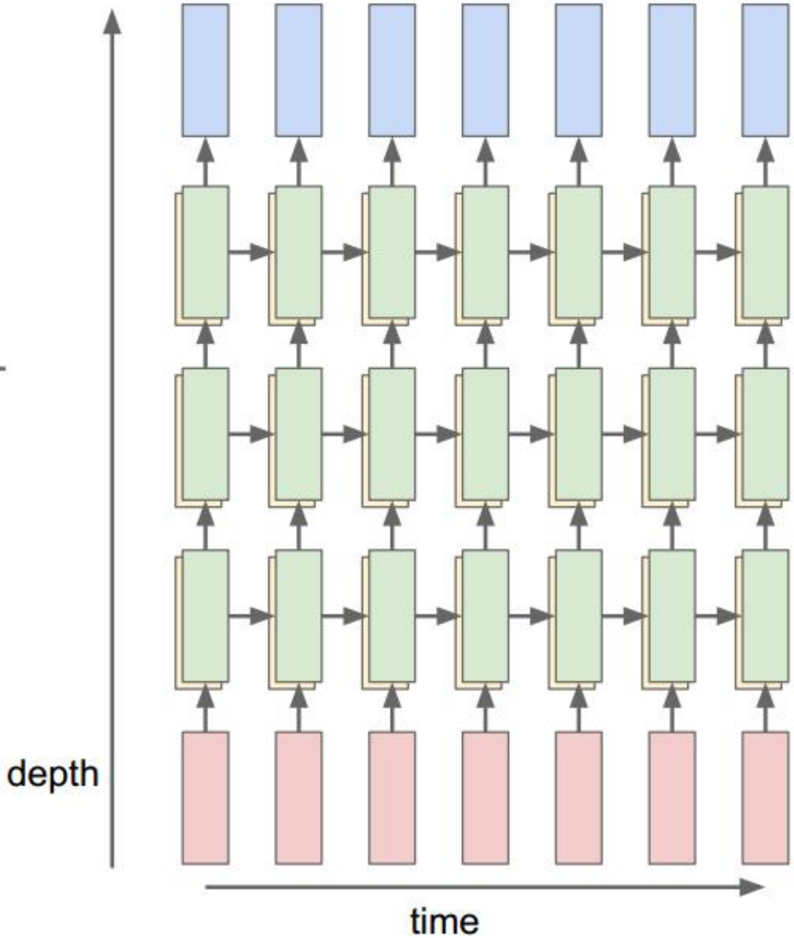
LSTM:

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

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

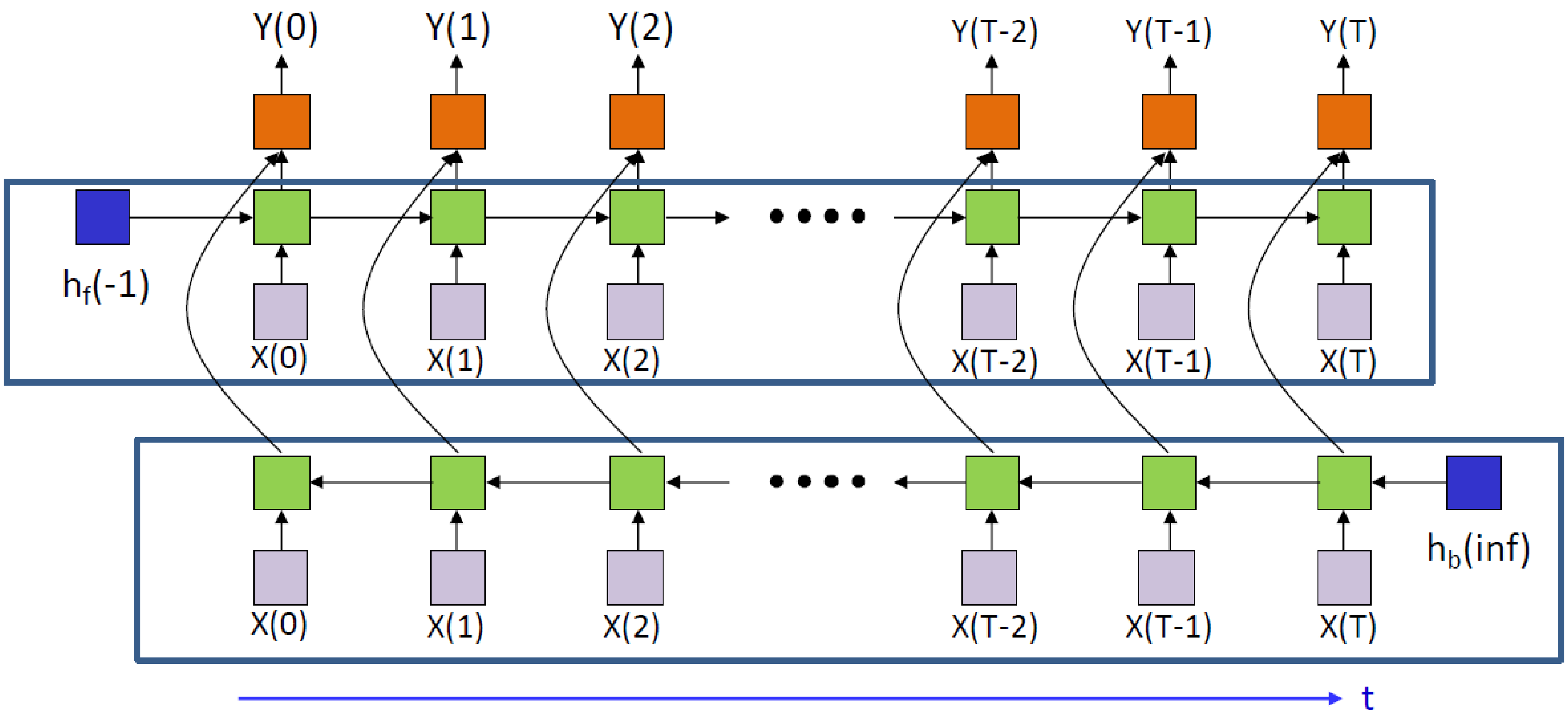
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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



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
- Reading materials:
  - http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/readings/L14%20Exploding%20and%20Vanishing%20Gradients.pdf
  - http://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes05-LM_RNN.pdf