



COS 484

Natural Language Processing

# L12: Recurrent neural networks (cont'd)

Spring 2023

# This lecture

- **Recurrent neural networks cont'd** 45 mins
- **PyTorch basics** 30mins (by Howard Chen)

# Recap: Recurrent neural networks

$\mathbf{h}_0 \in \mathbb{R}^h$  is an initial state

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

$\mathbf{h}_t$  : hidden states which store information from  $\mathbf{x}_1$  to  $\mathbf{x}_t$

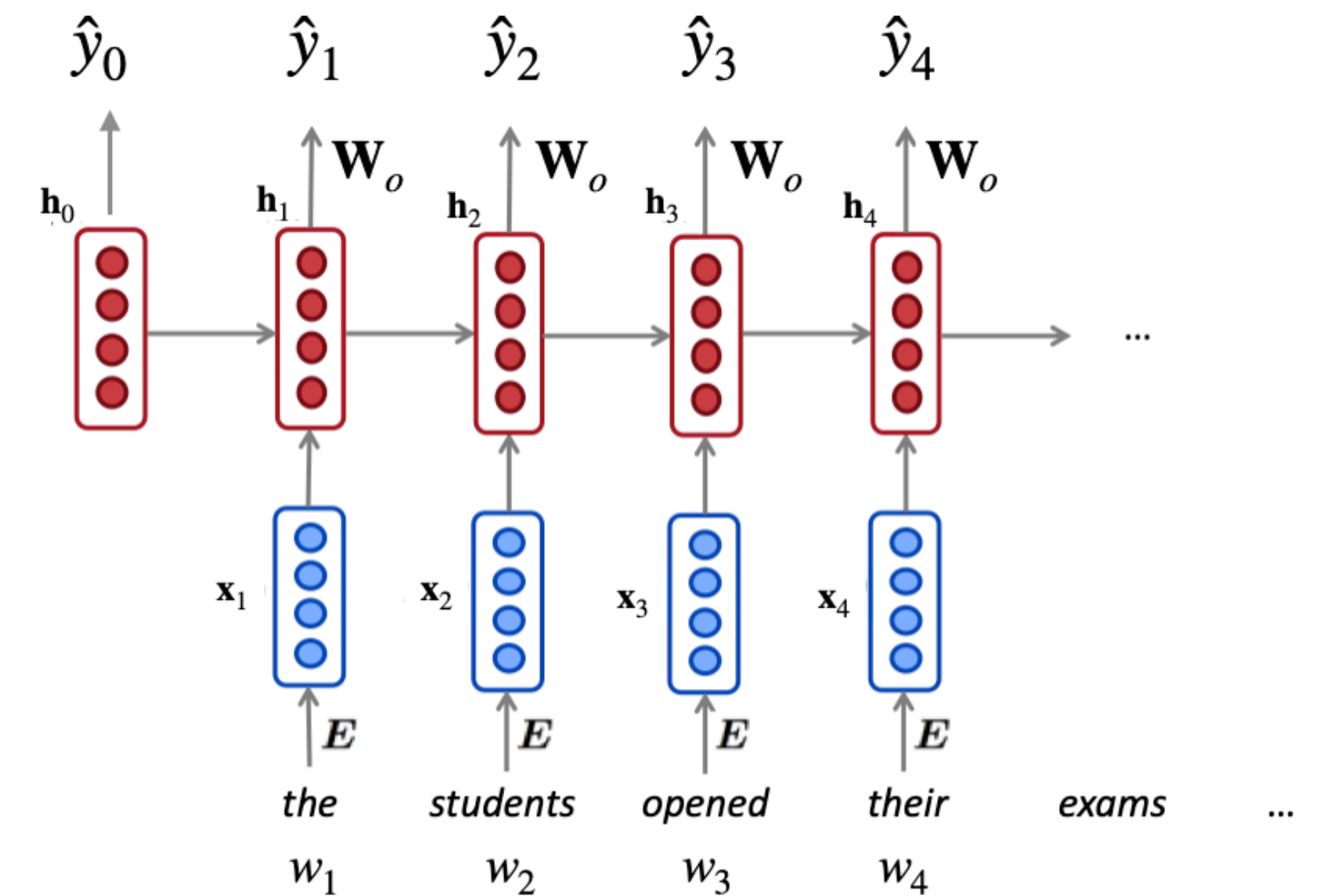
**Simple RNNs:**

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

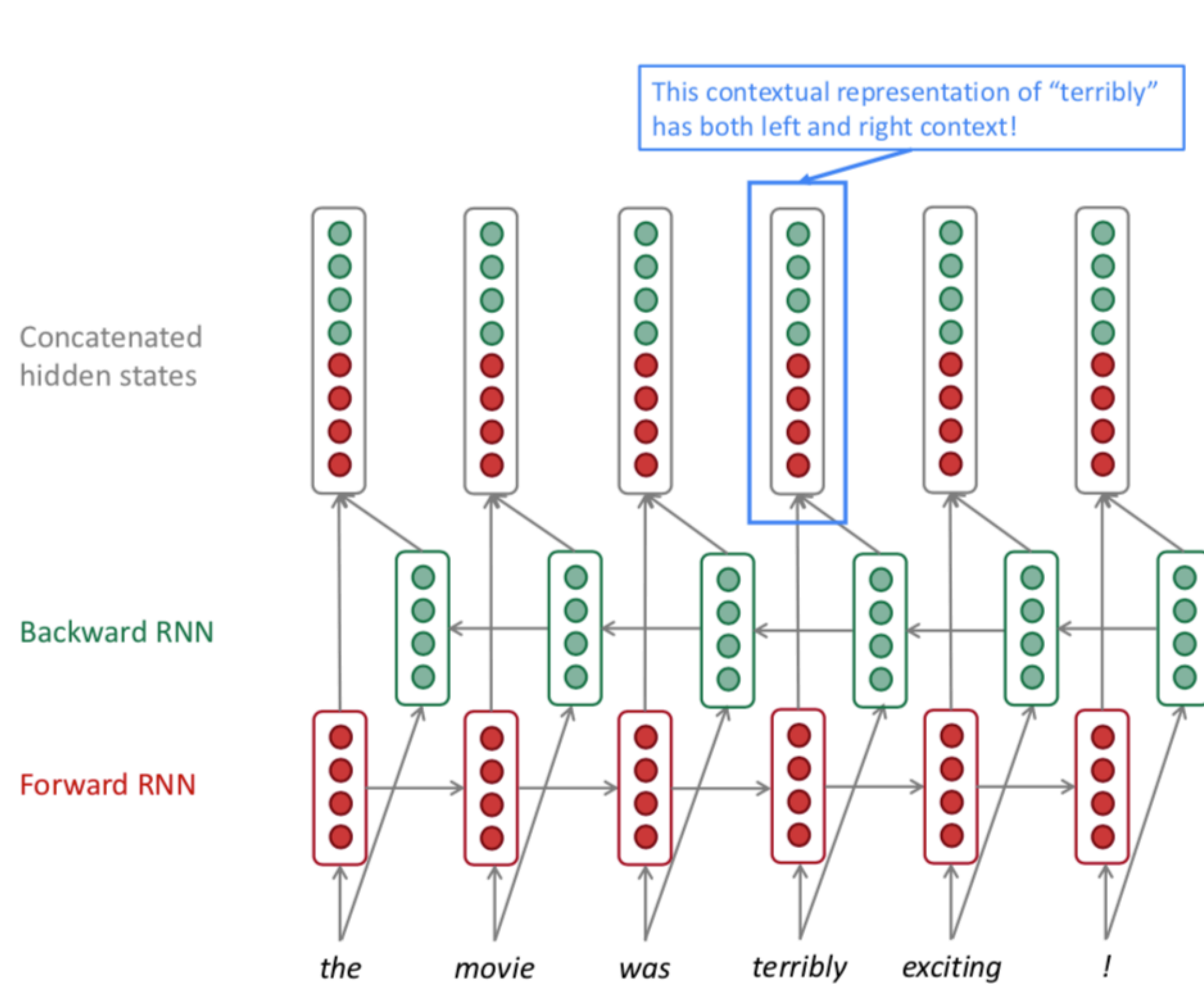
$g$ : nonlinearity (e.g. tanh, ReLU),

$$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \mathbf{b} \in \mathbb{R}^h$$

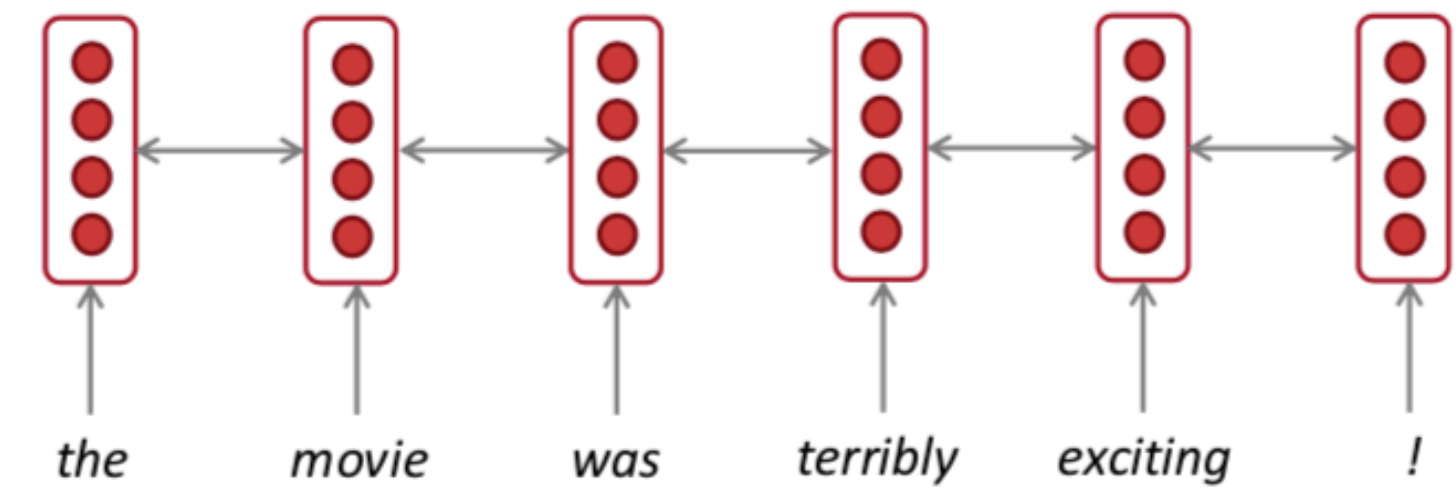
**RNNLMs:**



# Bidirectional RNNs



=



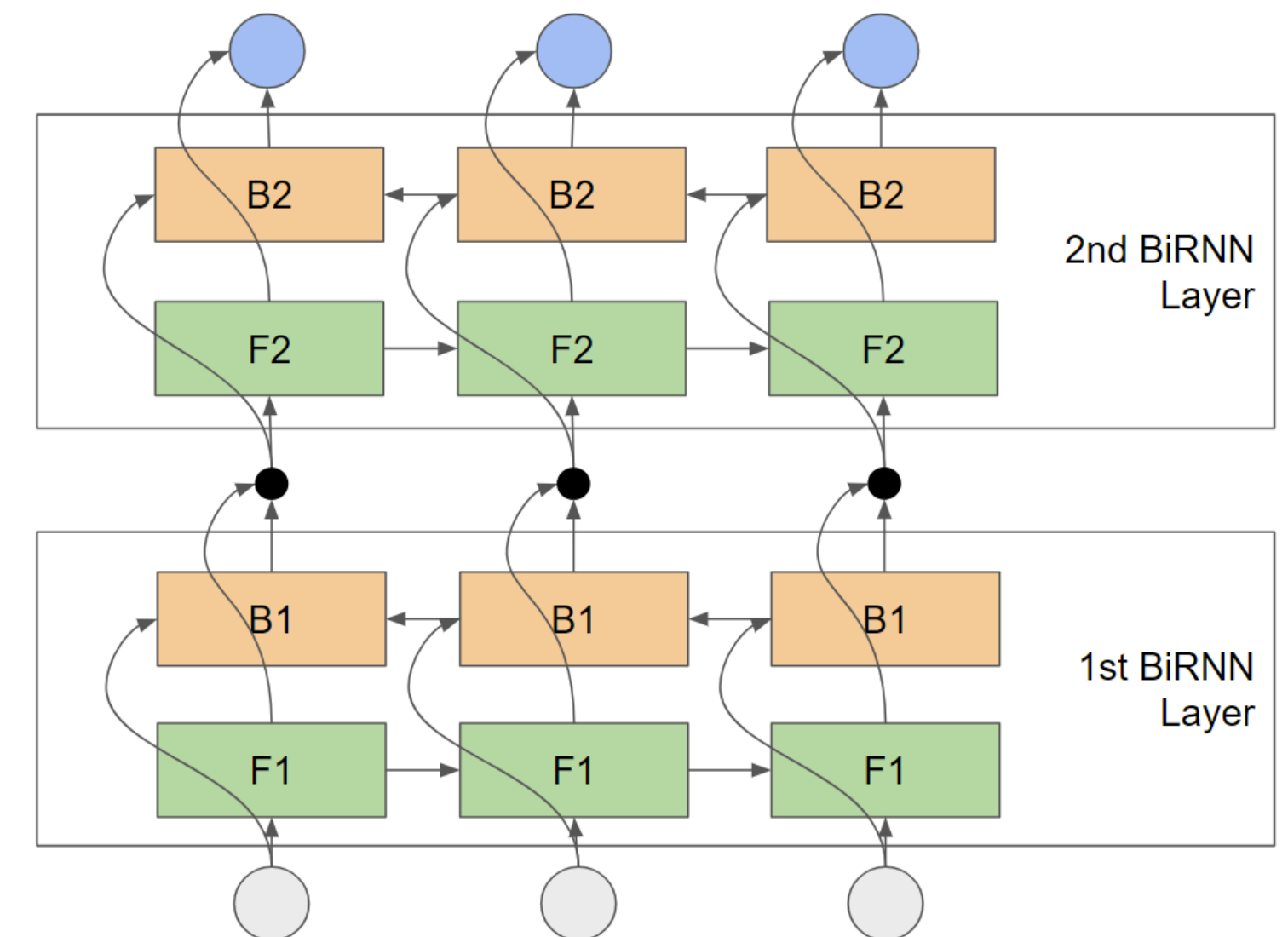
$$\vec{\mathbf{h}}_t = f_1(\vec{\mathbf{h}}_{t-1}, \mathbf{x}_t), t = 1, 2, \dots, n$$

$$\overleftarrow{\mathbf{h}}_t = f_2(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t), t = n, n-1, \dots, 1$$

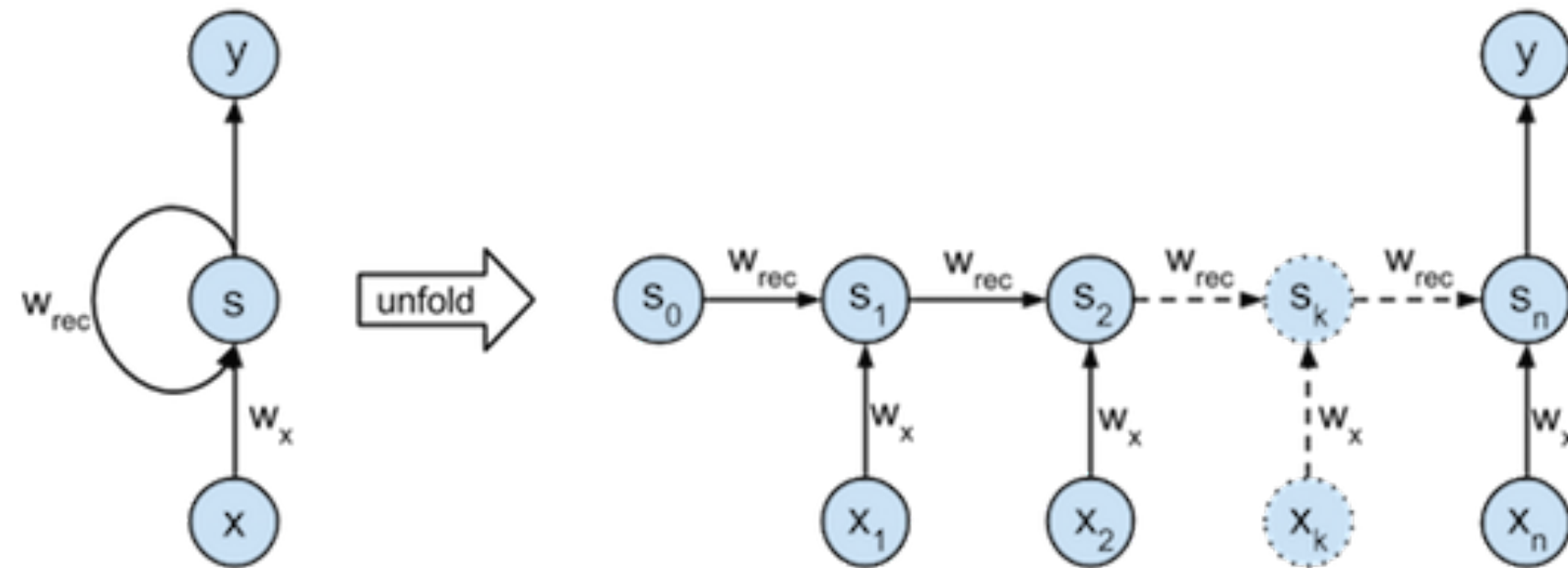
$$\mathbf{h}_t = [\overleftarrow{\mathbf{h}}_t, \vec{\mathbf{h}}_t] \in \mathbb{R}^{2h}$$

# Bidirectional RNNs

- Bidirectional RNNs are only applicable if you have access to the **entire input sequence** (= they can't do text generation!)
- If you do have entire input sequence, bidirectionality is powerful (and you should use it by default)
- Modeling the bidirectionality is the key idea behind BERT (BERT = **Bidirectional** Encoder Representations from Transformers)
  - We will learn Transformers and BERT in a few weeks!
- A very common choice for sentence/document modeling: multi-layer bidirectional RNNs



# Advanced RNN variants



$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

## LSTMs

$$\mathbf{i}_t = \sigma(\mathbf{W}^i \mathbf{h}_{t-1} + \mathbf{U}^i \mathbf{x}_t + \mathbf{b}^i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^f \mathbf{h}_{t-1} + \mathbf{U}^f \mathbf{x}_t + \mathbf{b}^f)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o \mathbf{h}_{t-1} + \mathbf{U}^o \mathbf{x}_t + \mathbf{b}^o)$$

$$\mathbf{g}_t = \tanh(\mathbf{W}^g \mathbf{h}_{t-1} + \mathbf{U}^g \mathbf{x}_t + \mathbf{b}^g)$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{g}_t \odot \mathbf{i}_t$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t) \odot \mathbf{o}_t$$

## GRUs

$$\mathbf{r}_t = \sigma(\mathbf{W}^r \mathbf{h}_{t-1} + \mathbf{U}^r \mathbf{x}_t + \mathbf{b}^r)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}^z \mathbf{h}_{t-1} + \mathbf{U}^z \mathbf{x}_t + \mathbf{b}^z)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$



# Long Short-Term Memory RNNs (LSTMs)

A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the **vanishing gradients problem**.

- Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000)

## LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter  
Fakultät für Informatik  
Technische Universität München  
80290 München, Germany  
hochreit@informatik.tu-muenchen.de  
<http://www7.informatik.tu-muenchen.de/~hochreit>

Jürgen Schmidhuber  
IDSIA  
Corso Elvezia 36  
6900 Lugano, Switzerland  
juergen@idsia.ch  
<http://www.idsia.ch/~juergen>

## Learning to Forget: Continual Prediction with LSTM

**Felix A. Gers**  
**Jürgen Schmidhuber**  
**Fred Cummins**  
*IDSIA, 6900 Lugano, Switzerland*

# Recap: Vanishing Gradient Problem

$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$

$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$

$$L_3 = -\log \hat{y}_3(w_4)$$

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

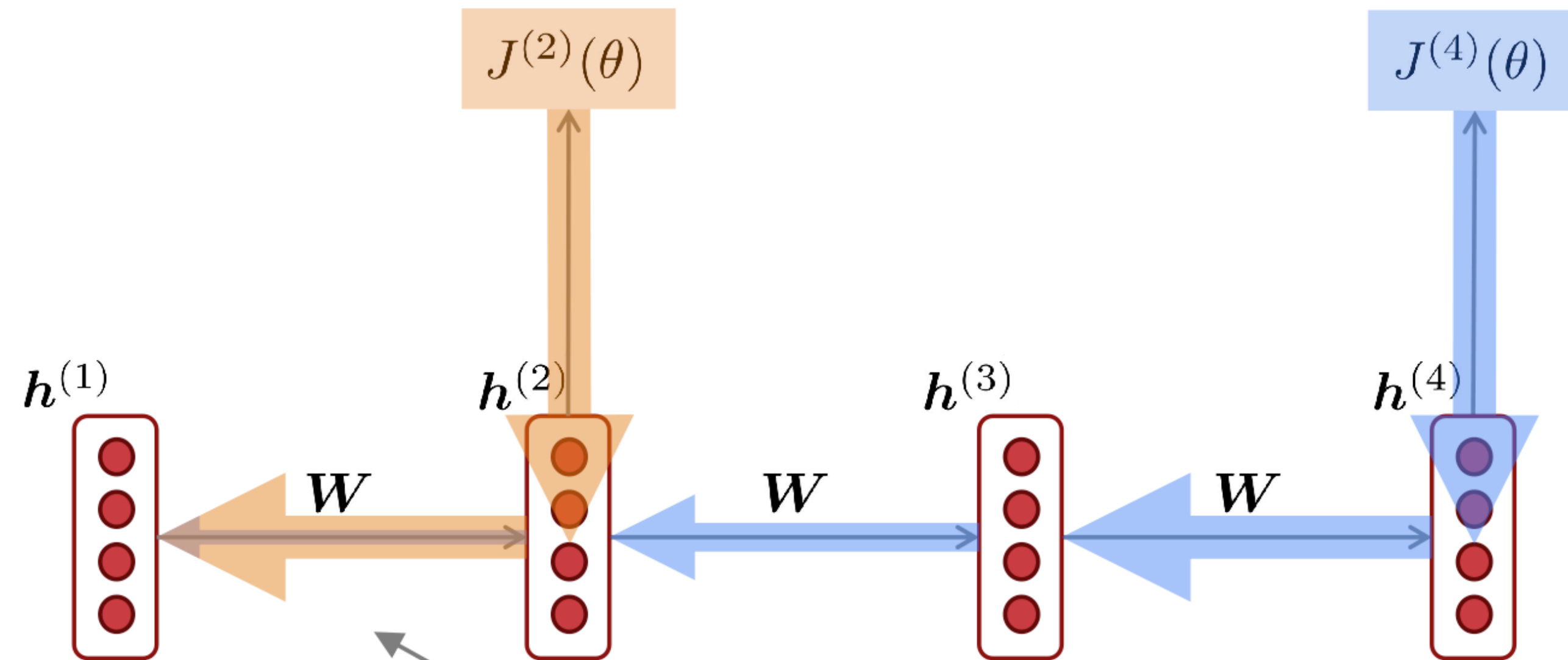
**Vanishing gradient problem:**  
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

$$\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n} \sum_{t=1}^n \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{h}_t} \left( \prod_{j=k+1}^t \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right) \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

If  $k$  and  $t$  are far away, the gradients are very easy to grow/shrink exponentially (called the gradient exploding or gradient vanishing problem)



# Recap: Vanishing Gradient Problem

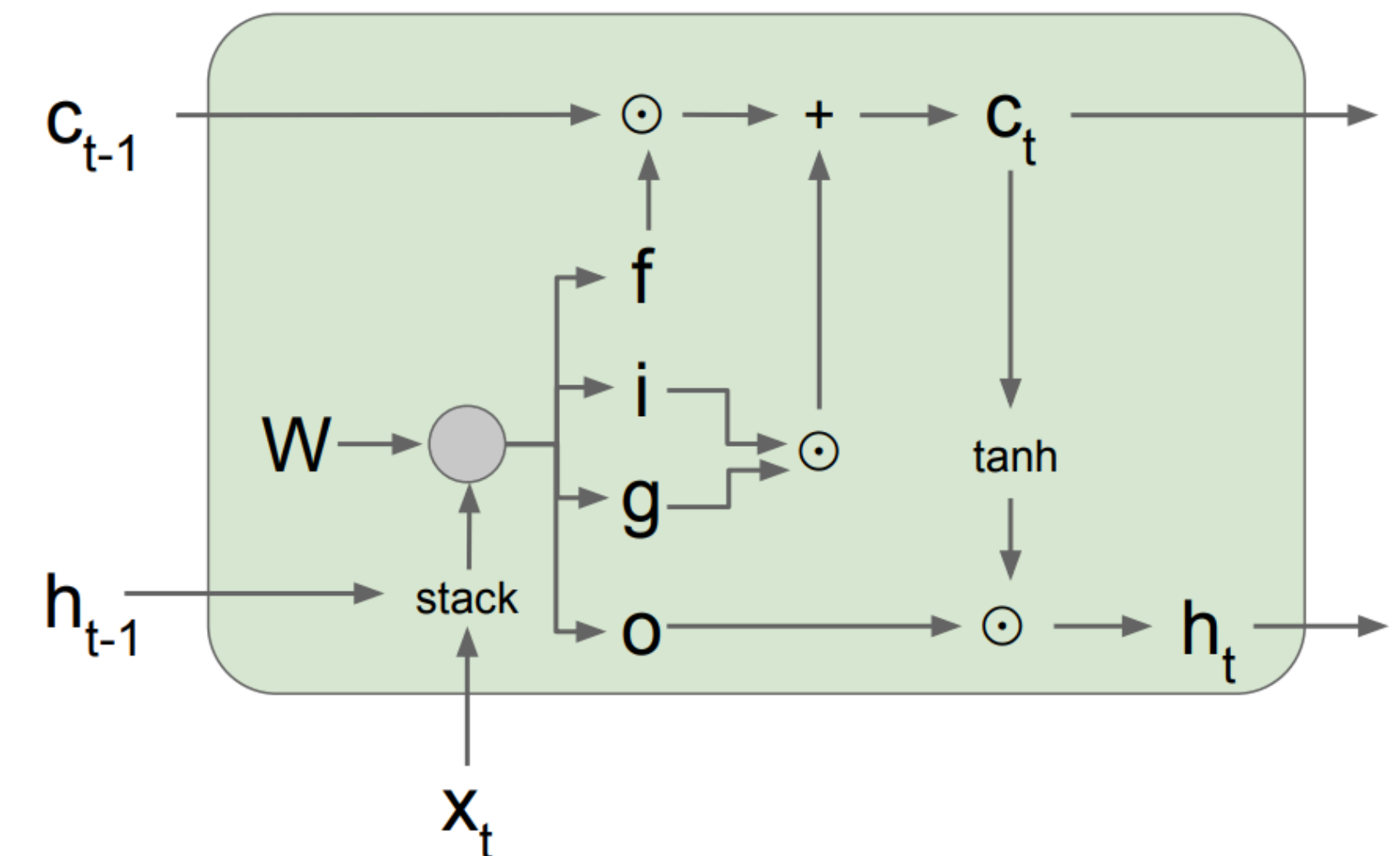


Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are basically updated only with respect to near effects, not long-term effects.

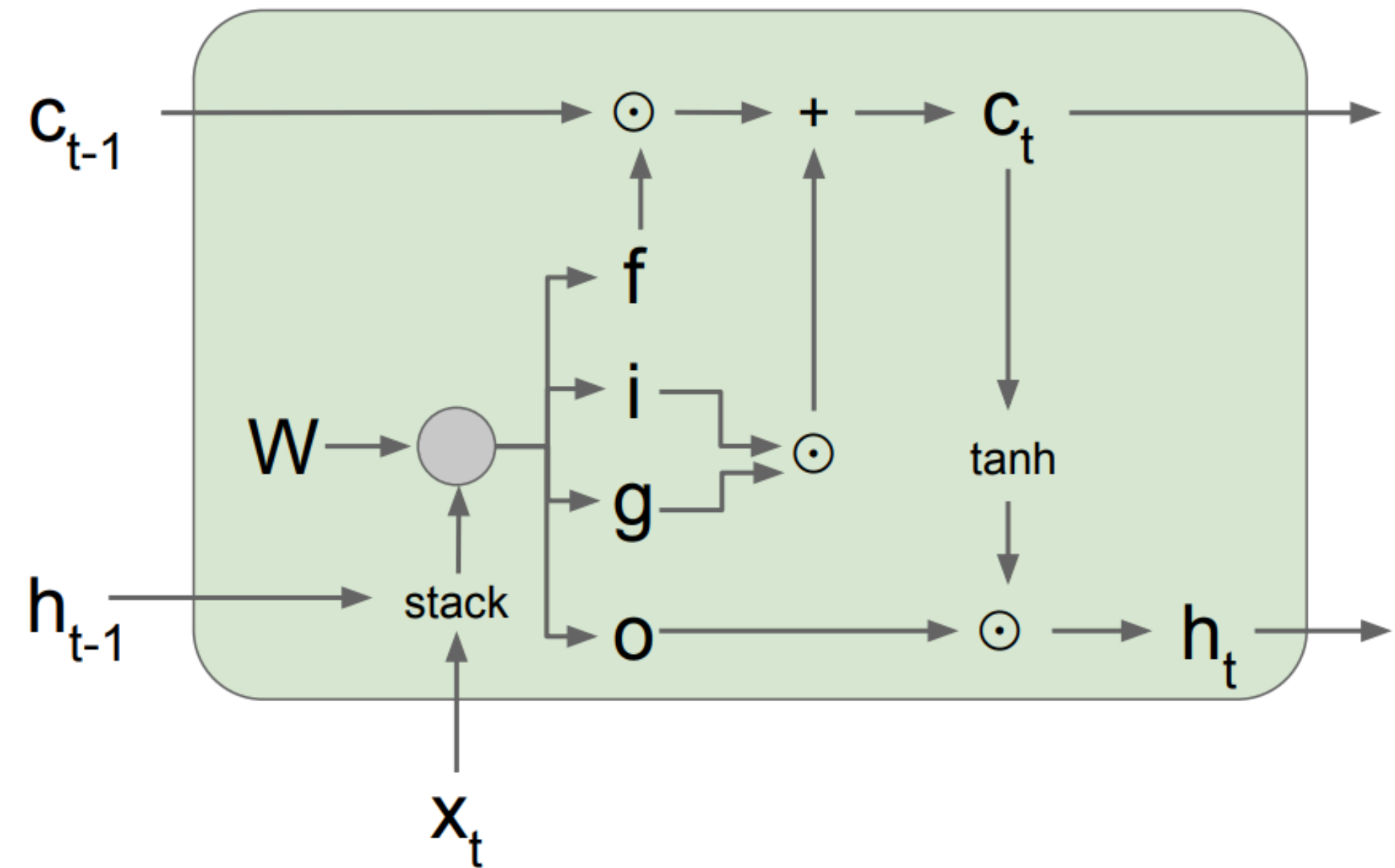
# LSTMs: The intuition

- Key idea: turning **multiplication** into **addition** and using “**gates**” to control how much information to add/erase
- At each time step, instead of re-writing the hidden state  $\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$ , there is also a cell state  $\mathbf{c}_t \in \mathbb{R}^h$  which stores **long-term information**
  - We write to/erase information from  $\mathbf{c}_t$  after each step
  - We read  $\mathbf{h}_t$  from  $\mathbf{c}_t$



# LSTMs: the formulation

- Input gate (**how much to write**):  
 $\mathbf{i}_t = \sigma(\mathbf{W}^i \mathbf{h}_{t-1} + \mathbf{U}^i \mathbf{x}_t + \mathbf{b}^i) \in \mathbb{R}^h$
- Forget gate (**how much to erase**):  
 $\mathbf{f}_t = \sigma(\mathbf{W}^f \mathbf{h}_{t-1} + \mathbf{U}^f \mathbf{x}_t + \mathbf{b}^f) \in \mathbb{R}^h$
- Output gate (**how much to reveal**):  
 $\mathbf{o}_t = \sigma(\mathbf{W}^o \mathbf{h}_{t-1} + \mathbf{U}^o \mathbf{x}_t + \mathbf{b}^{(o)}) \in \mathbb{R}^h$
- New memory cell (**what to write**):  
 $\mathbf{g}_t = \tanh(\mathbf{W}^g \mathbf{h}_{t-1} + \mathbf{U}^g \mathbf{x}_t + \mathbf{b}^g) \in \mathbb{R}^h$
- Final memory cell:  $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$   
← element-wise product
- Final hidden cell:  $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$



$\mathbf{h}_0, \mathbf{c}_0 \in \mathbb{R}^h$  are initial states (usually set to  $\mathbf{0}$ )

# LSTMs: the formulation

LSTMs has 4x parameters compared to simple RNNs:

Input dimension:  $d$ , hidden size:  $h$

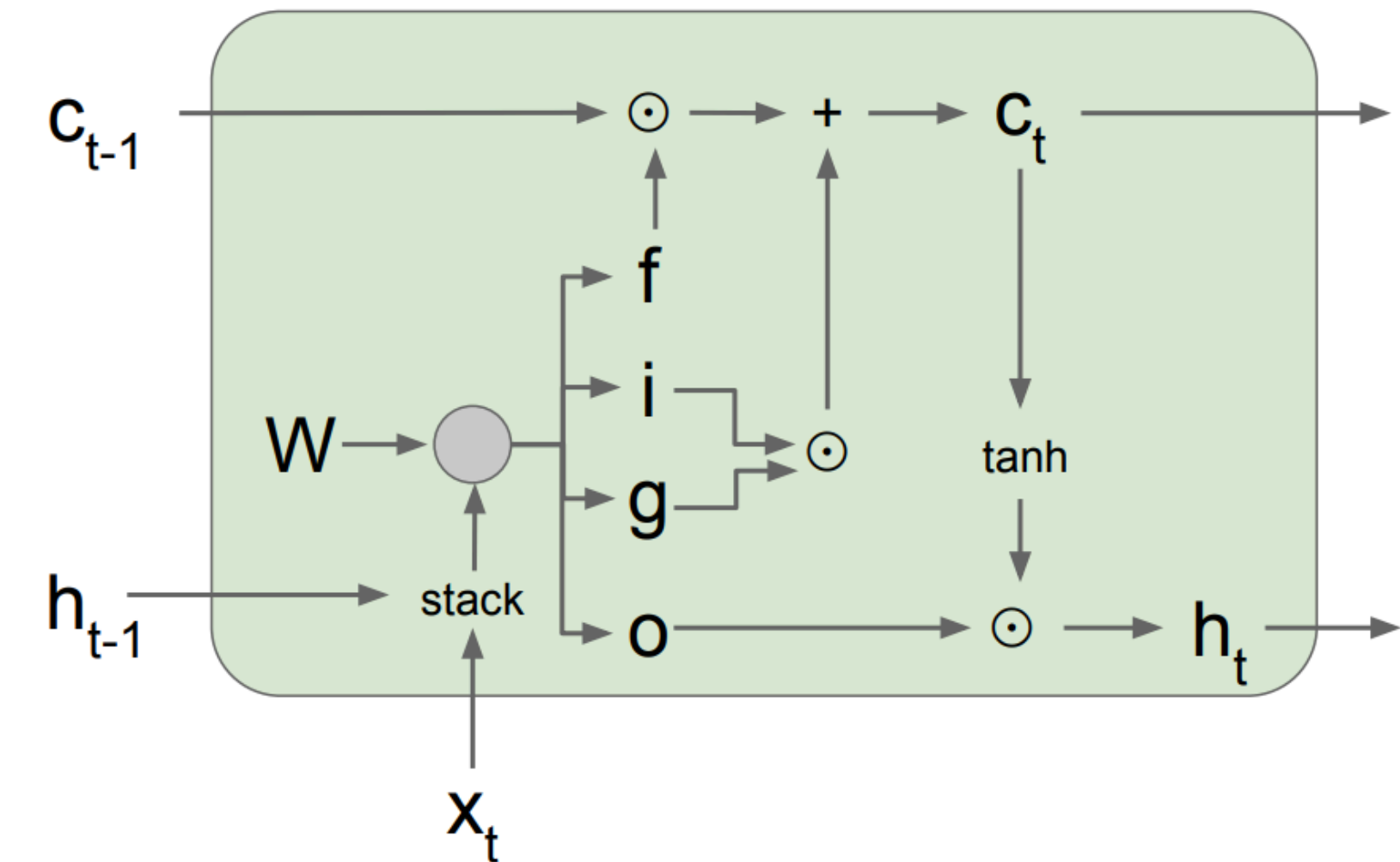
$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

$$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \mathbf{b} \in \mathbb{R}^h$$

$$\mathbf{W}^i, \mathbf{W}^f, \mathbf{W}^g, \mathbf{W}^o \in \mathbb{R}^{h \times h}$$

$$\mathbf{U}^i, \mathbf{U}^f, \mathbf{U}^g, \mathbf{U}^o \in \mathbb{R}^{h \times d}$$

$$\mathbf{b}^i, \mathbf{b}^f, \mathbf{b}^g, \mathbf{b}^o \in \mathbb{R}^h$$



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

$\mathbb{R}^{4h \times (h+d)}$

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

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

Simple RNNs:

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



# What is the range of values?

Q: What is the range of values for each element in the hidden representations  $\mathbf{h}_t$ ?

- (a) 0 to  $\infty$
- (b) 0 to 1
- (c) -1 to 1
- (d)  $-\infty$  to  $\infty$

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

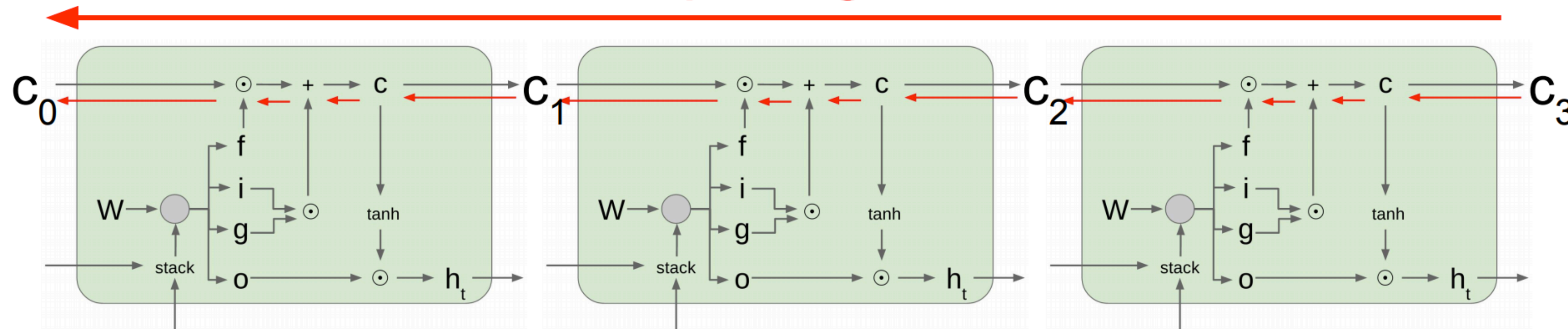
$$c_t = f \odot c_{t-1} + i \odot g$$

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



# LSTMs: the formulation

Uninterrupted gradient flow!



- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies
- LSTMs were invented in 1997 but finally got working from 2013-2015.



# Visualization of LSTMs

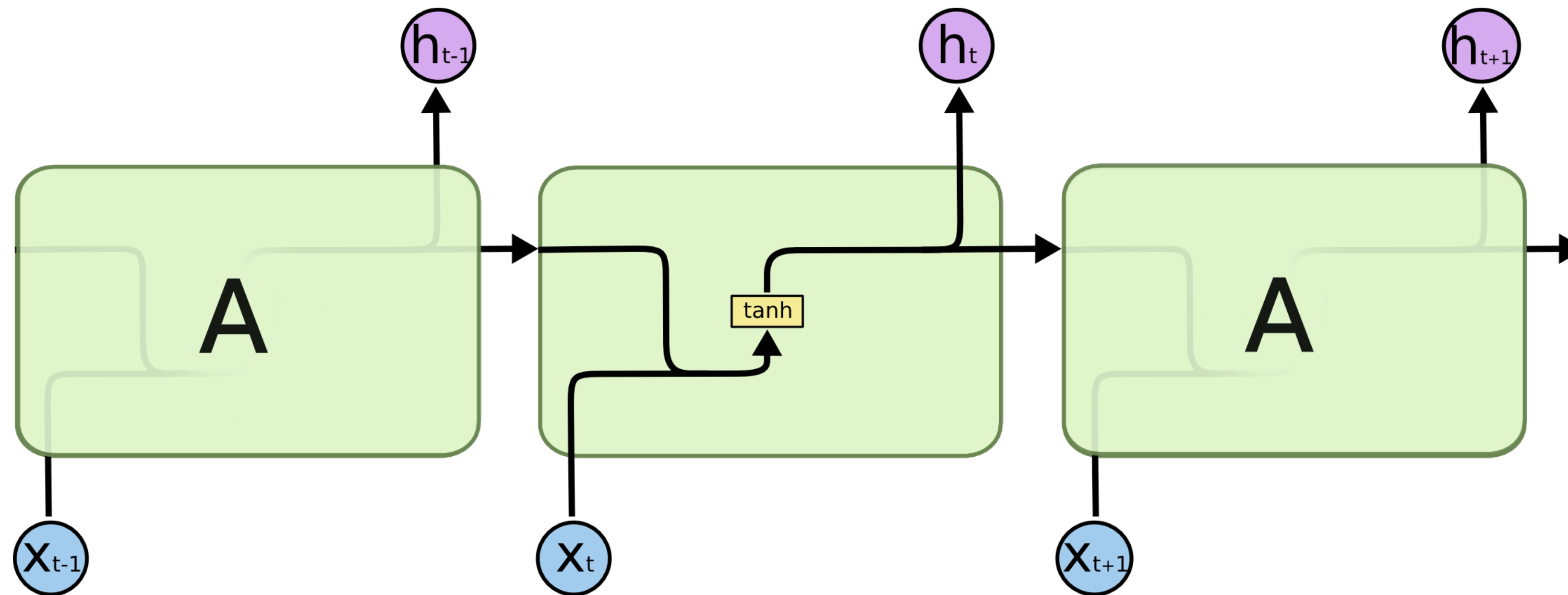
## Understanding LSTM Networks

*Posted on August 27, 2015*

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



Christopher Olah

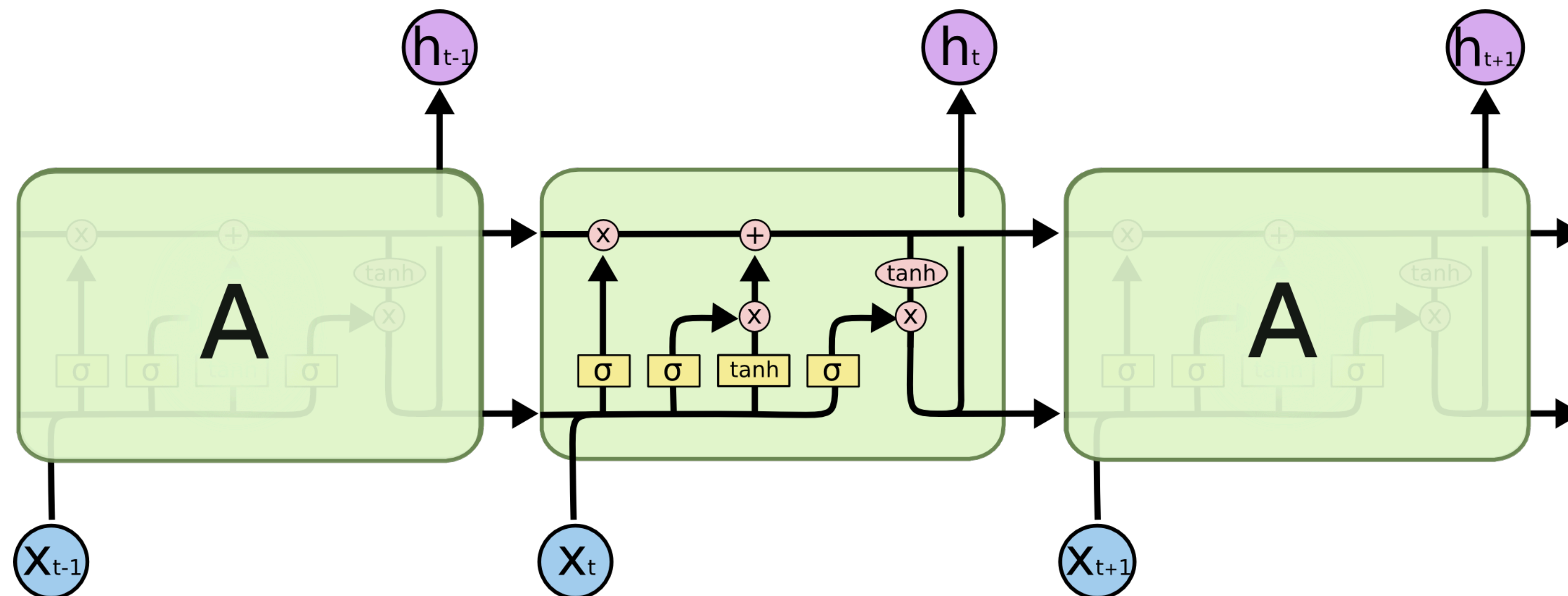


# Visualization of LSTMs

## Understanding LSTM Networks

Posted on August 27, 2015

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

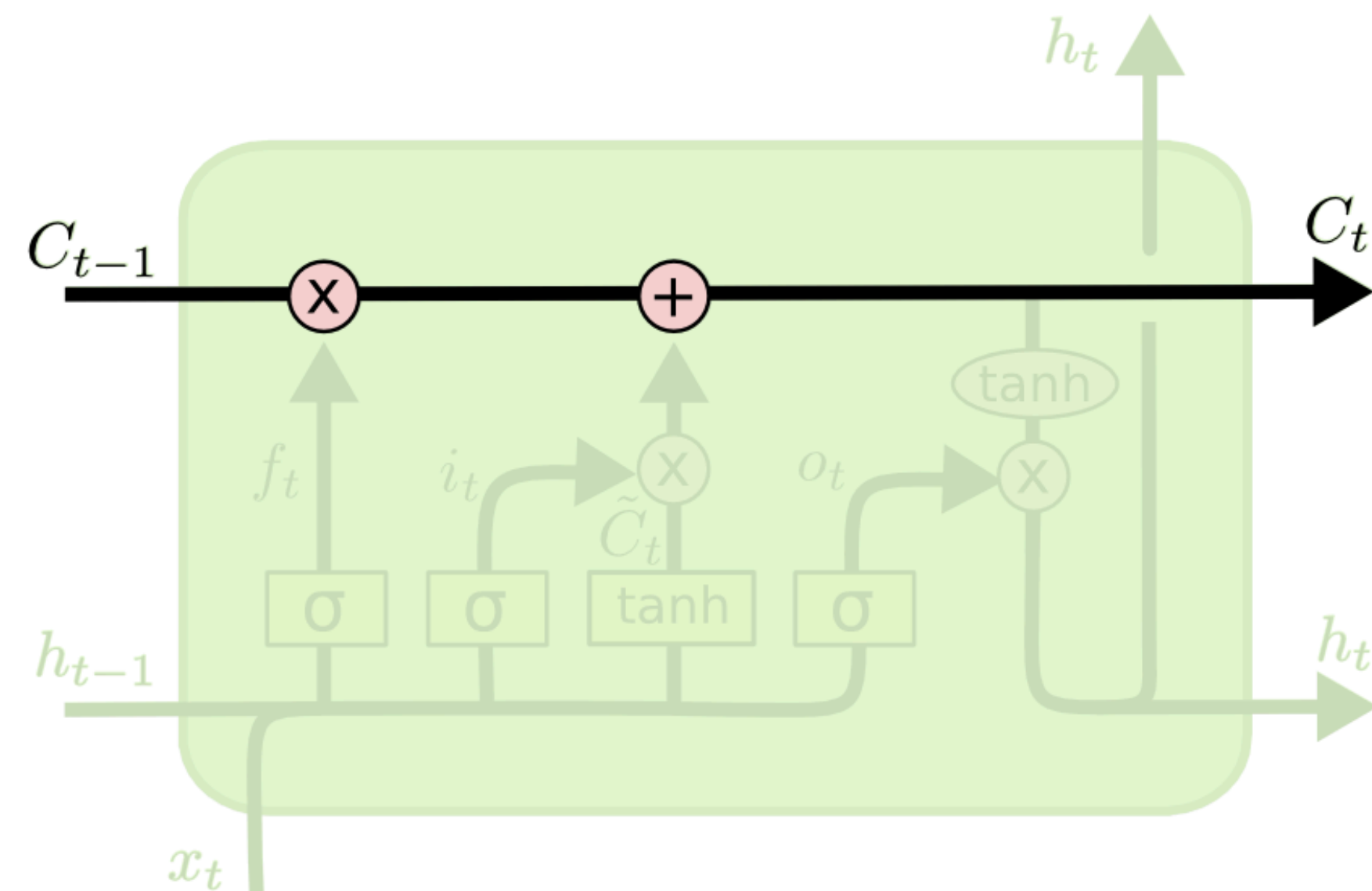


# Visualization of LSTMs

## Understanding LSTM Networks

*Posted on August 27, 2015*

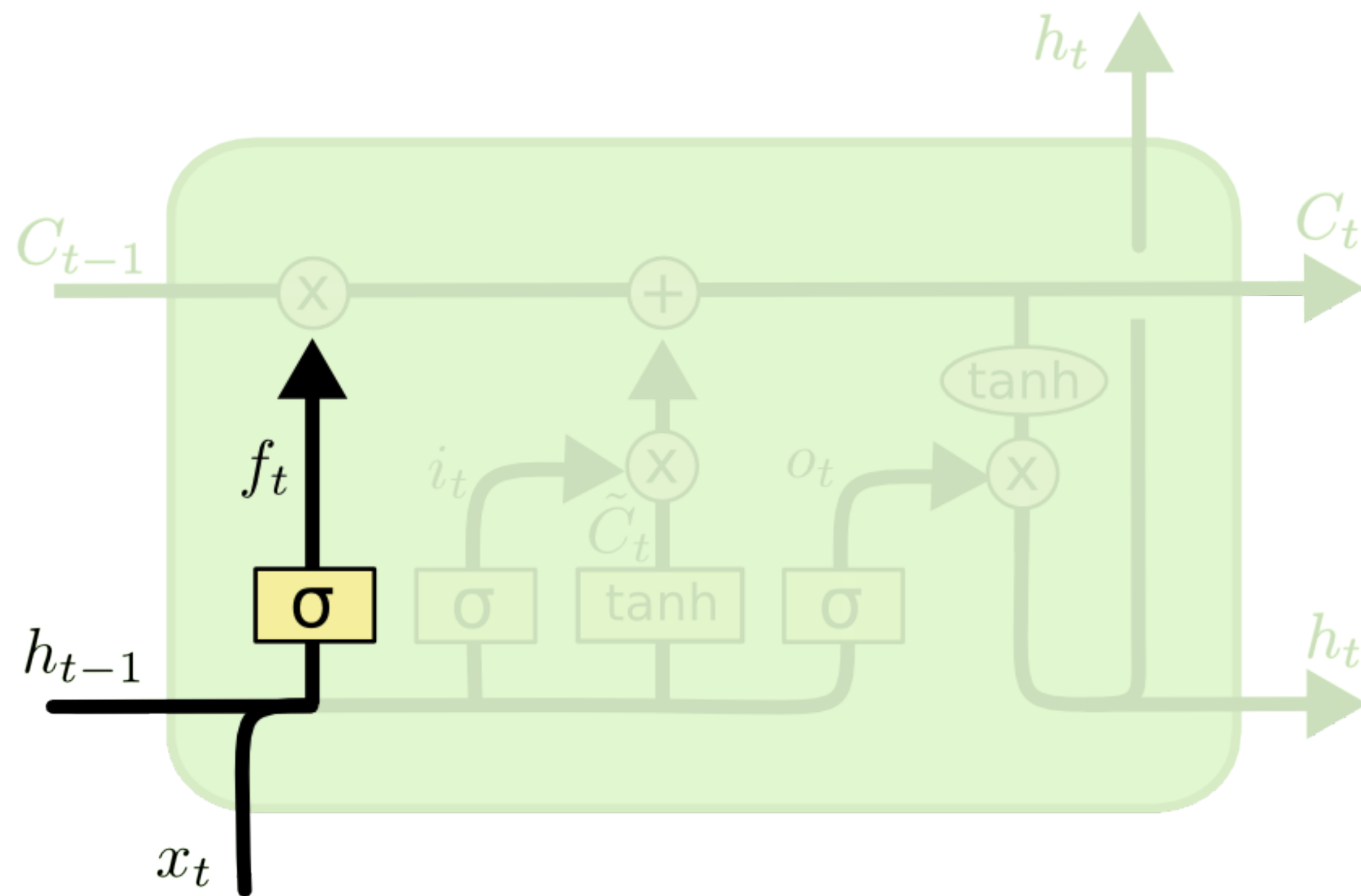
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



Cell state = a conveyor belt

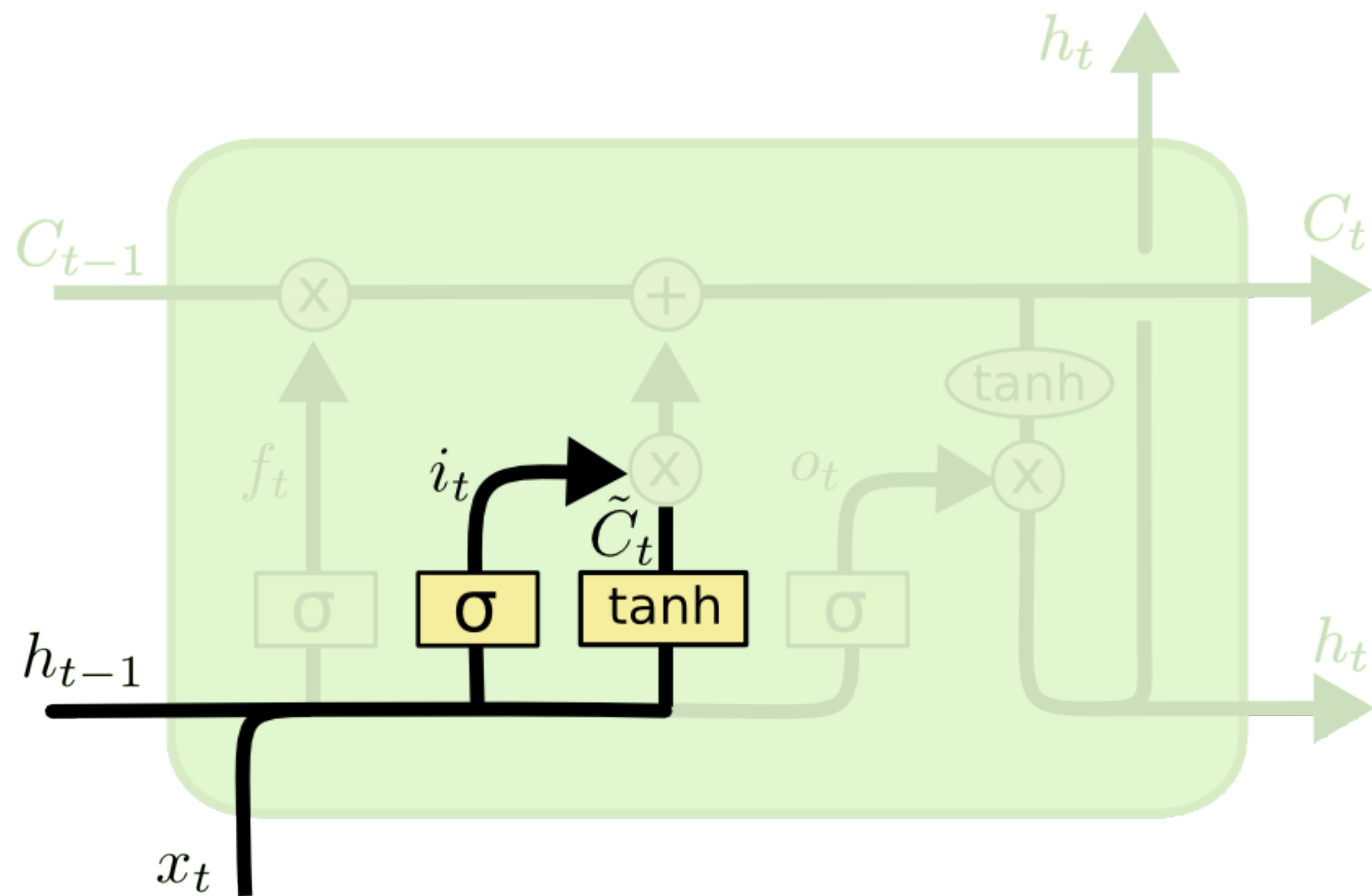
It allows **adding** or **removing** information, carefully regulated by gates

# Visualization of LSTMs (Warning: notation change!)



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

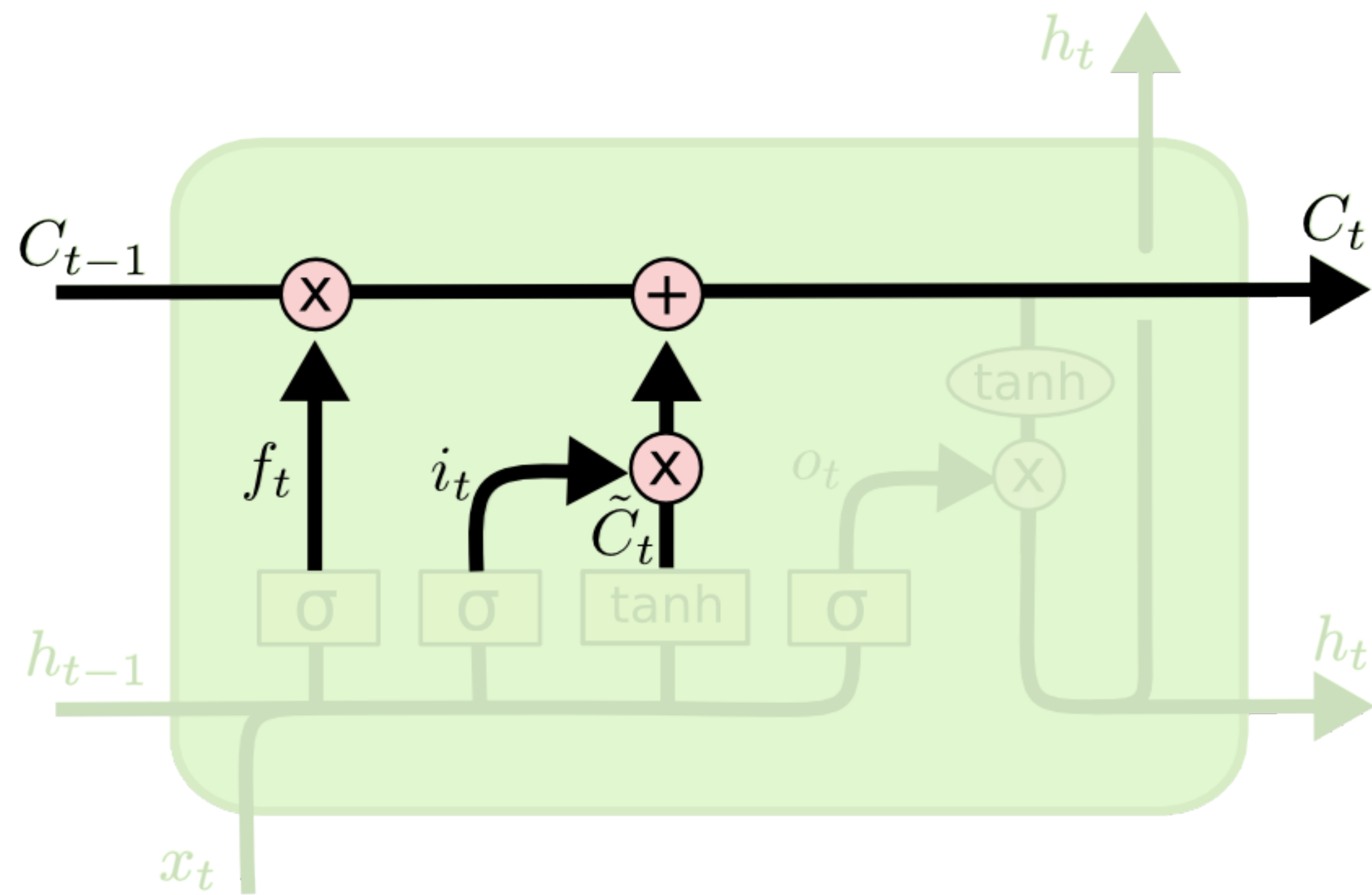
# Visualization of LSTMs (Warning: notation change!)



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



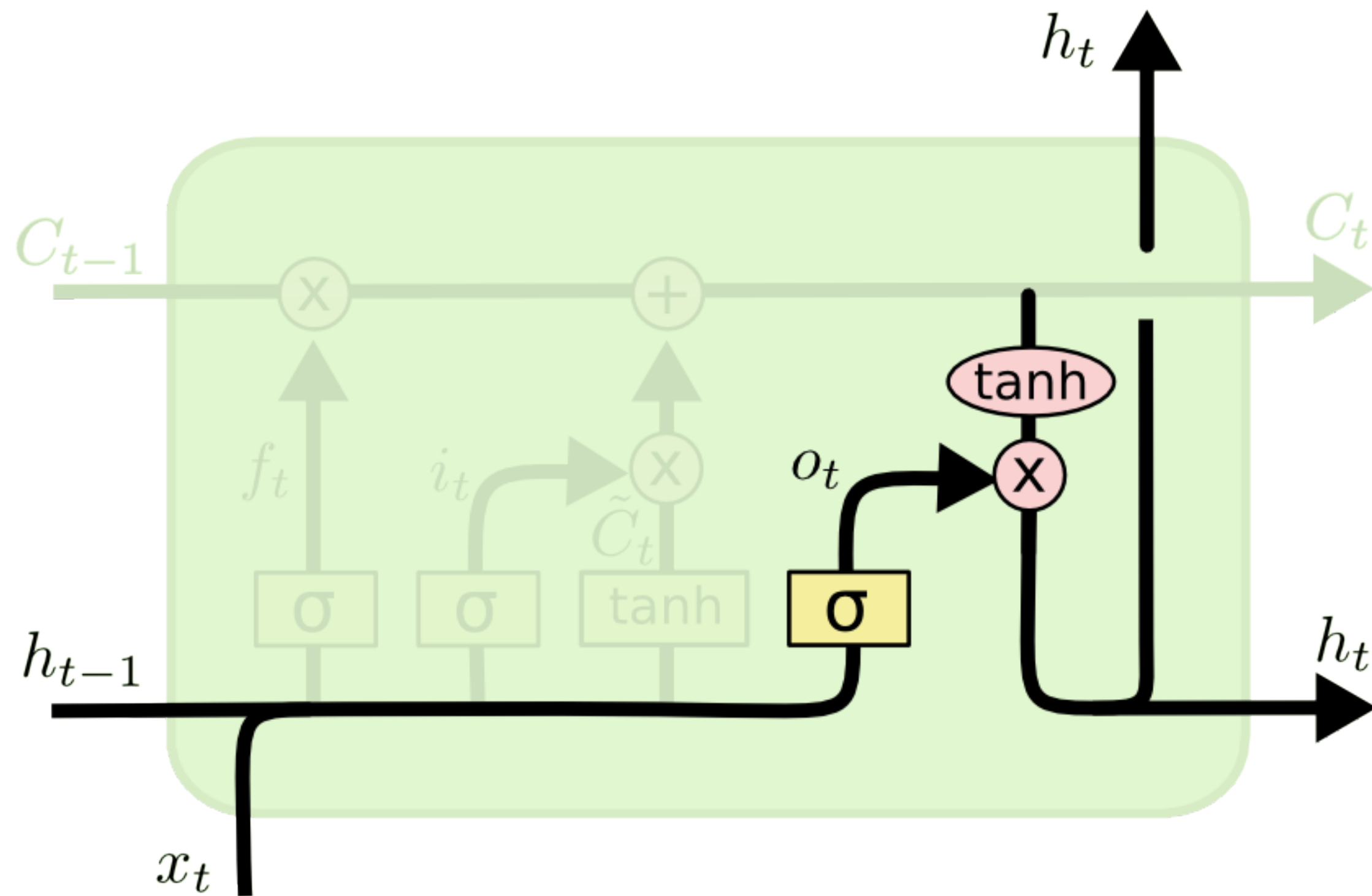
# Visualization of LSTMs (Warning: notation change!)



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



# Visualization of LSTMs (Warning: notation change!)



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

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

# Gated Recurrent Units (GRUs)

- Introduced by Kyunghyun Cho et al. in 2014:

## Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

**Kyunghyun Cho**

**Bart van Merriënboer Caglar Gulcehre**

Université de Montréal

`firstname.lastname@umontreal.ca`

**Dzmitry Bahdanau**

Jacobs University, Germany

`d.bahdanau@jacobs-university.de`

**Fethi Bougares Holger Schwenk**

Université du Maine, France

`firstname.lastname@lium.univ-lemans.fr`

**Yoshua Bengio**

Université de Montréal, CIFAR Senior Fellow

`find.me@on.the.web`



- Simplified 3 gates to 2 gates: **reset** gate and **update** gate, without an explicit cell state

# Gated Recurrent Units (GRUs)

- Reset gate:

$$\mathbf{r}_t = \sigma(\mathbf{W}^r \mathbf{h}_{t-1} + \mathbf{U}^r \mathbf{x}_t + \mathbf{b}^r)$$

- Update gate:

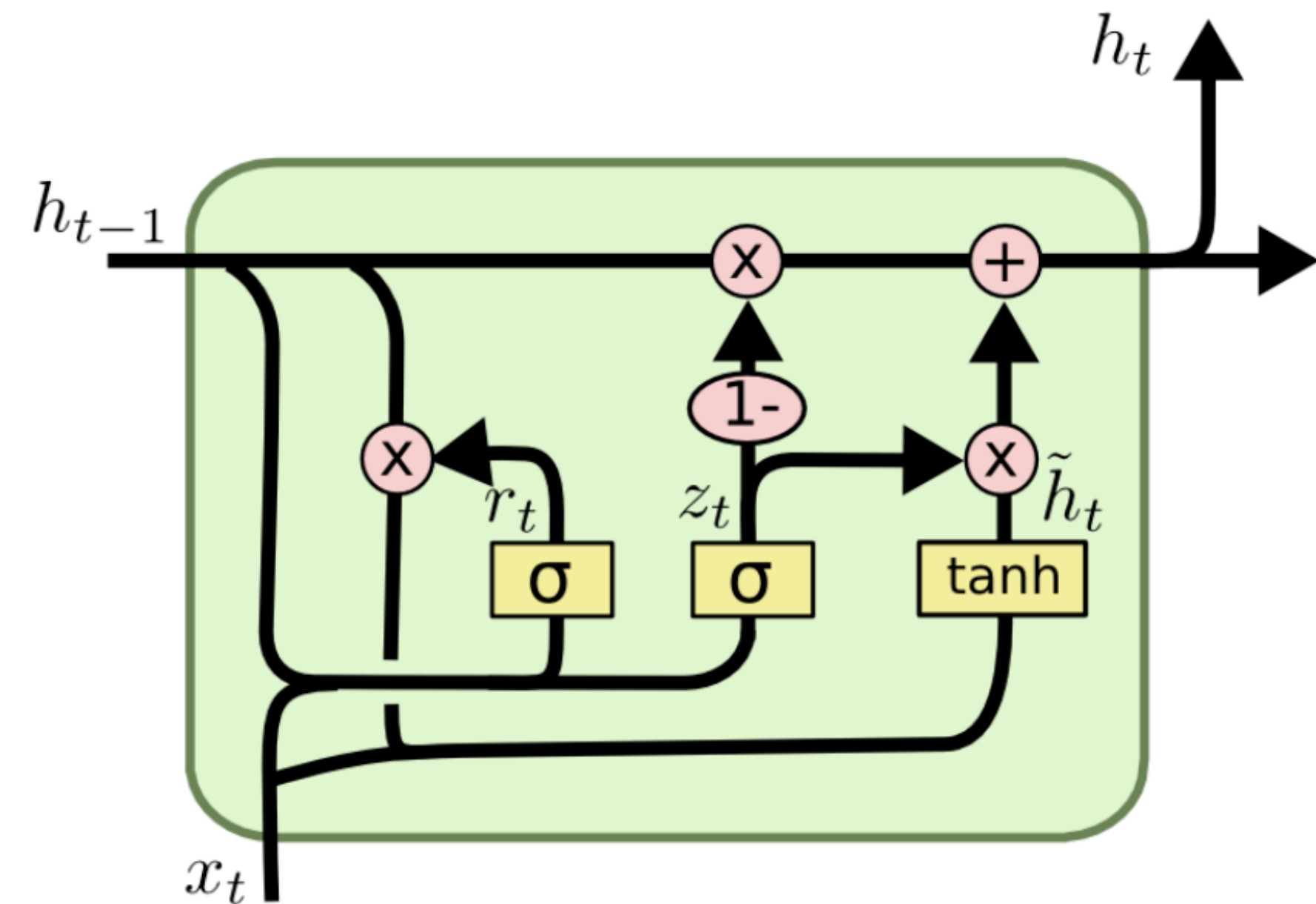
$$\mathbf{z}_t = \sigma(\mathbf{W}^z \mathbf{h}_{t-1} + \mathbf{U}^z \mathbf{x}_t + \mathbf{b}^z)$$

- New hidden state:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

merge input and forget gate!



Q: What is the range of the hidden representations  $\mathbf{h}_t$ ?

Q: How many parameters are there compared to simple RNNs?

# Comparison of LSTMs and GRUs

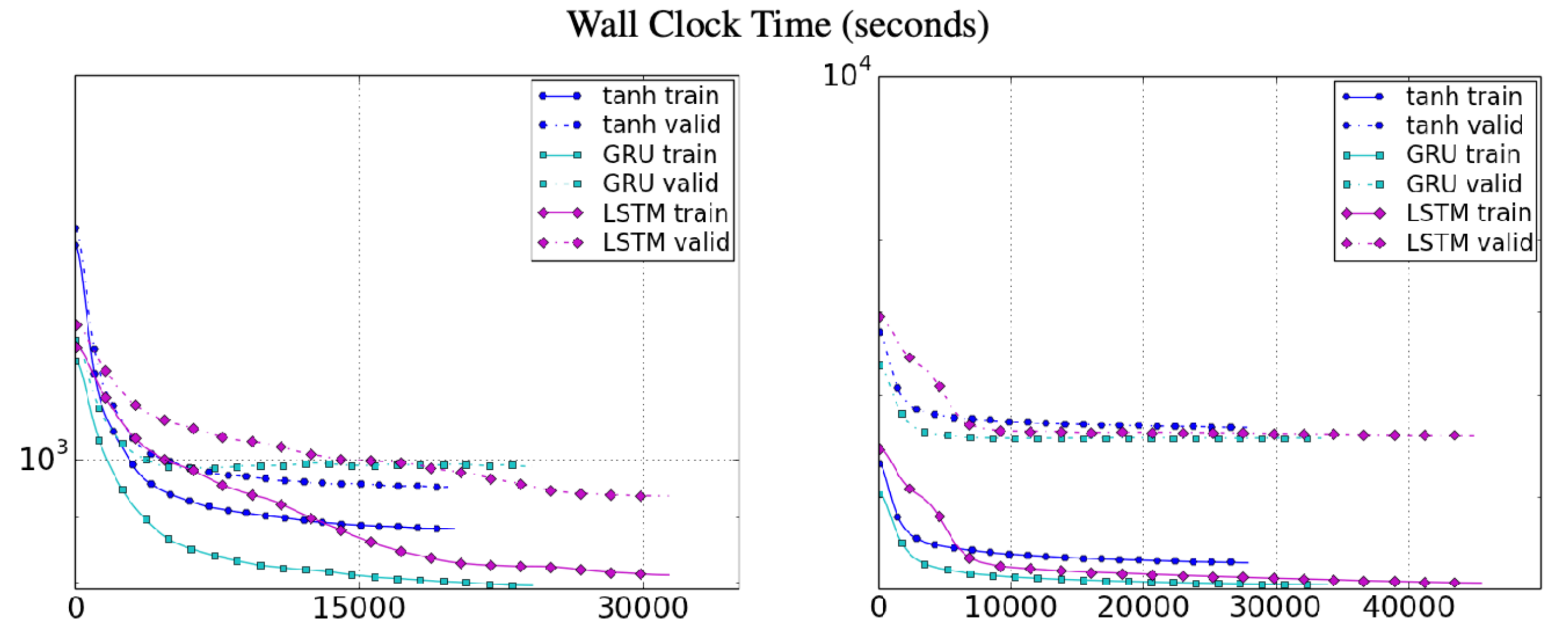
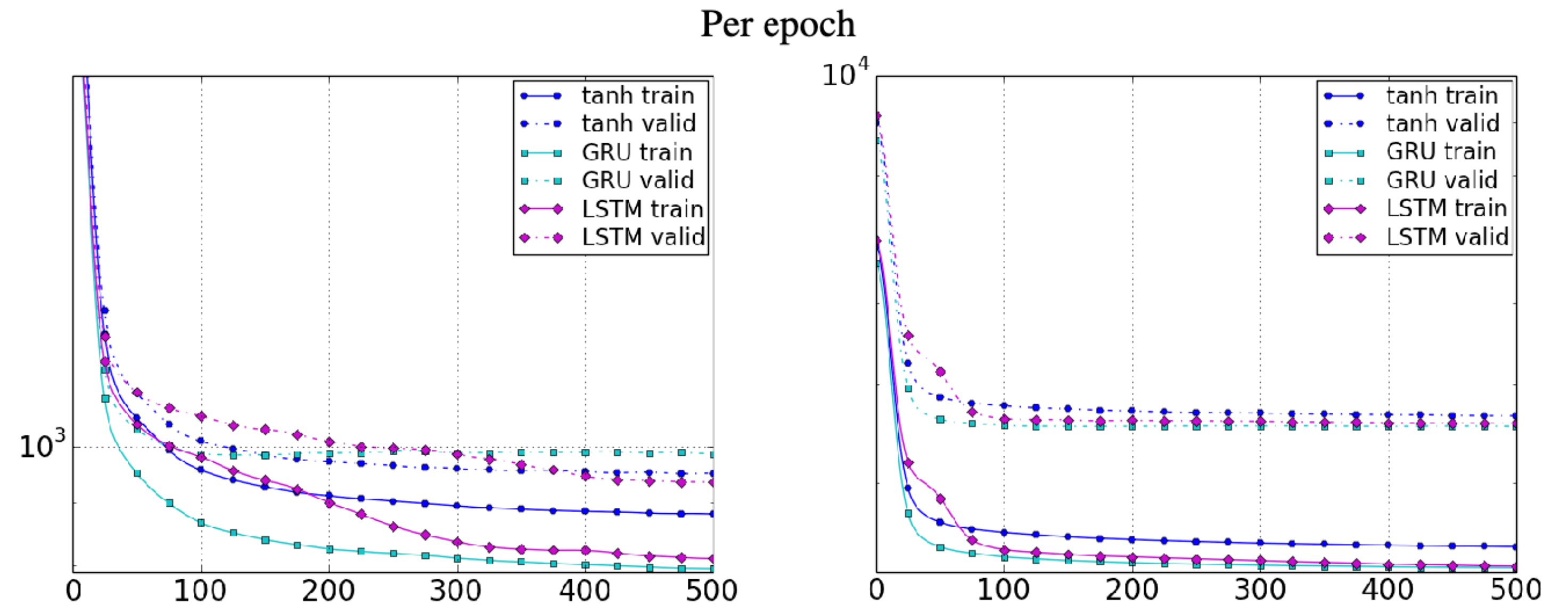


Let's compare LSTMs and GRUs. Which of the following statements is correct?

- (a) GRUs can be trained faster
- (b) In theory LSTMs can capture long-term dependencies better
- (c) LSTMs have a controlled exposure of memory content while GRUs don't
- (d) All of the above

# LSTMs vs GRUs

Music modeling



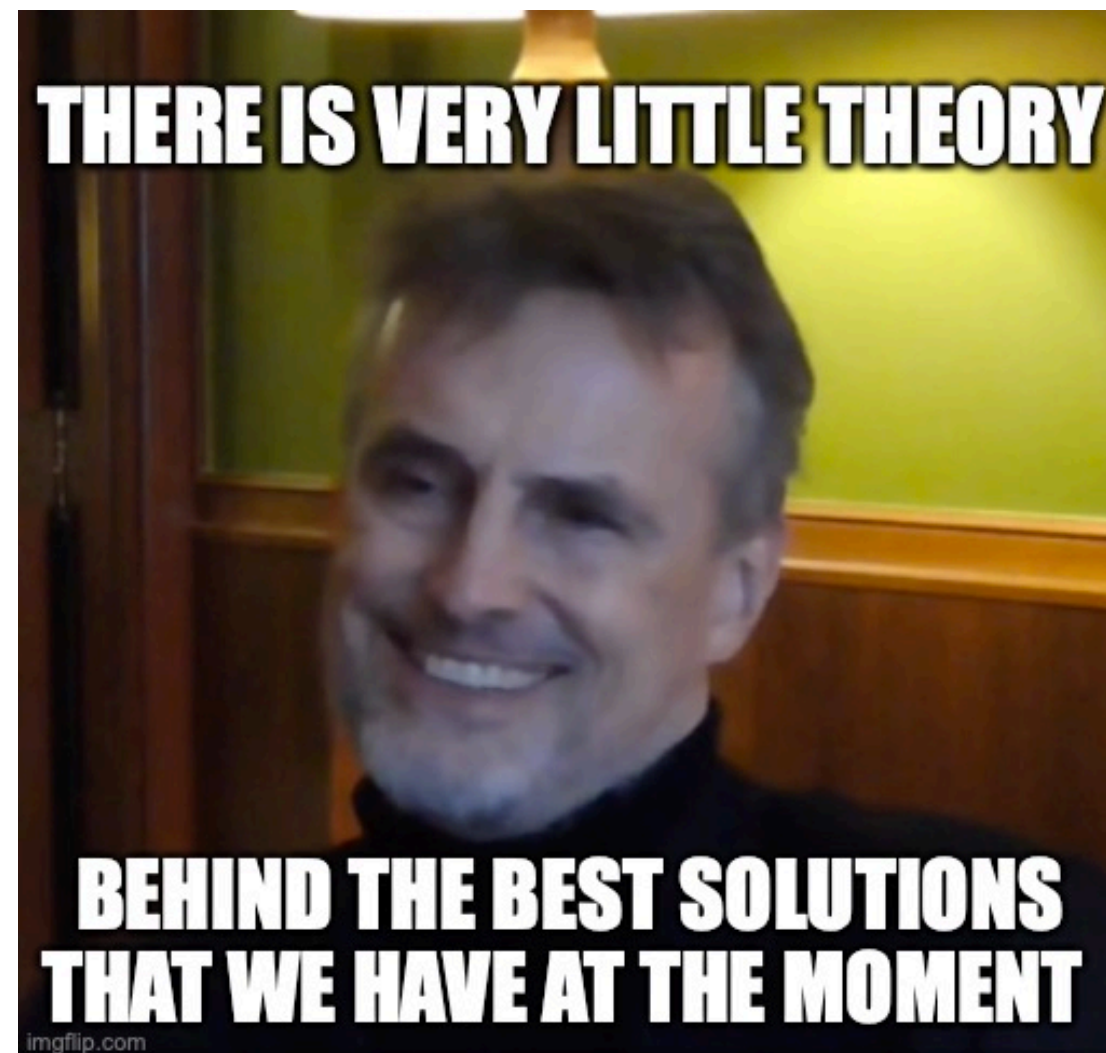
(a) Nottingham Dataset

(b) MuseData Dataset

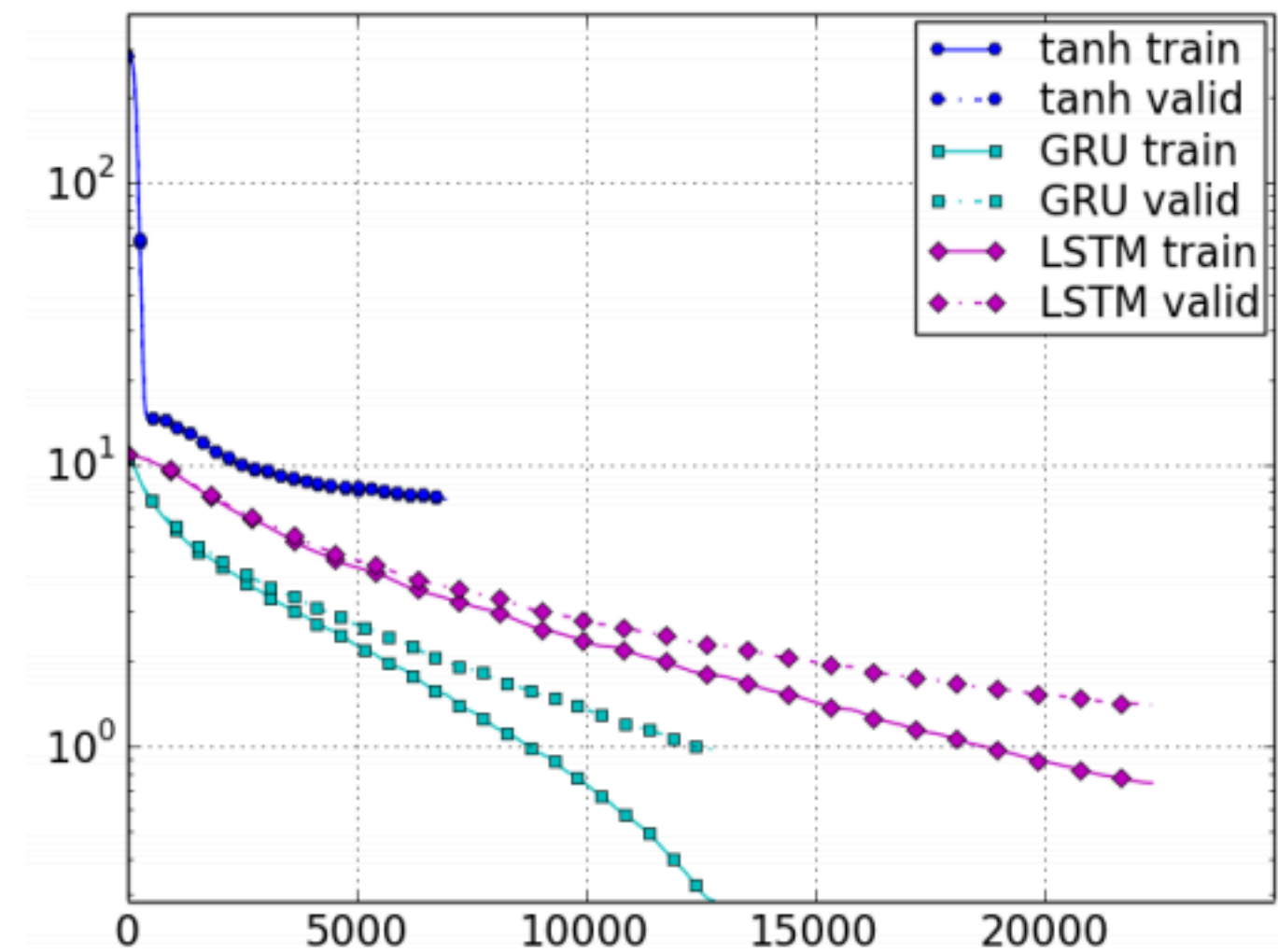
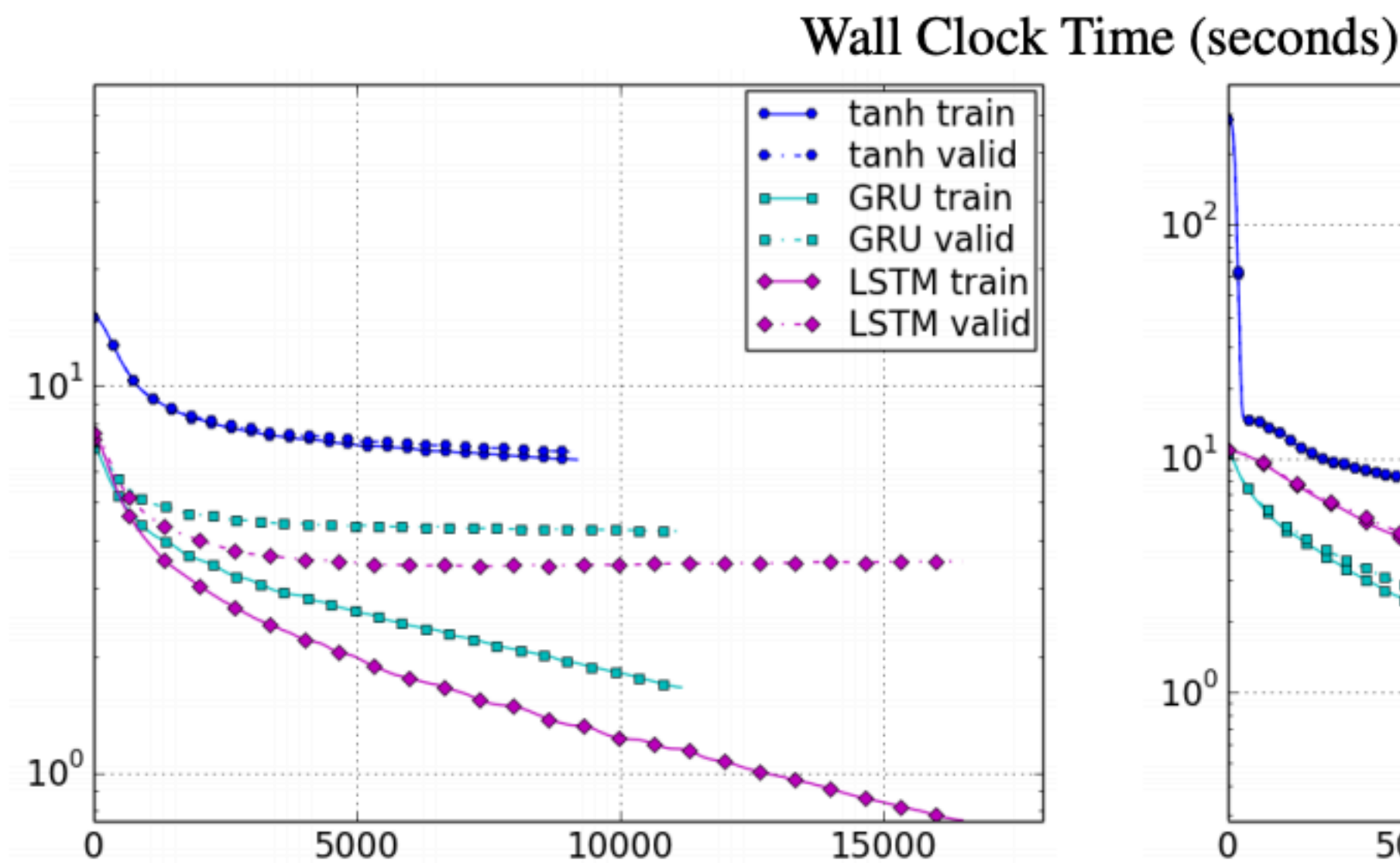
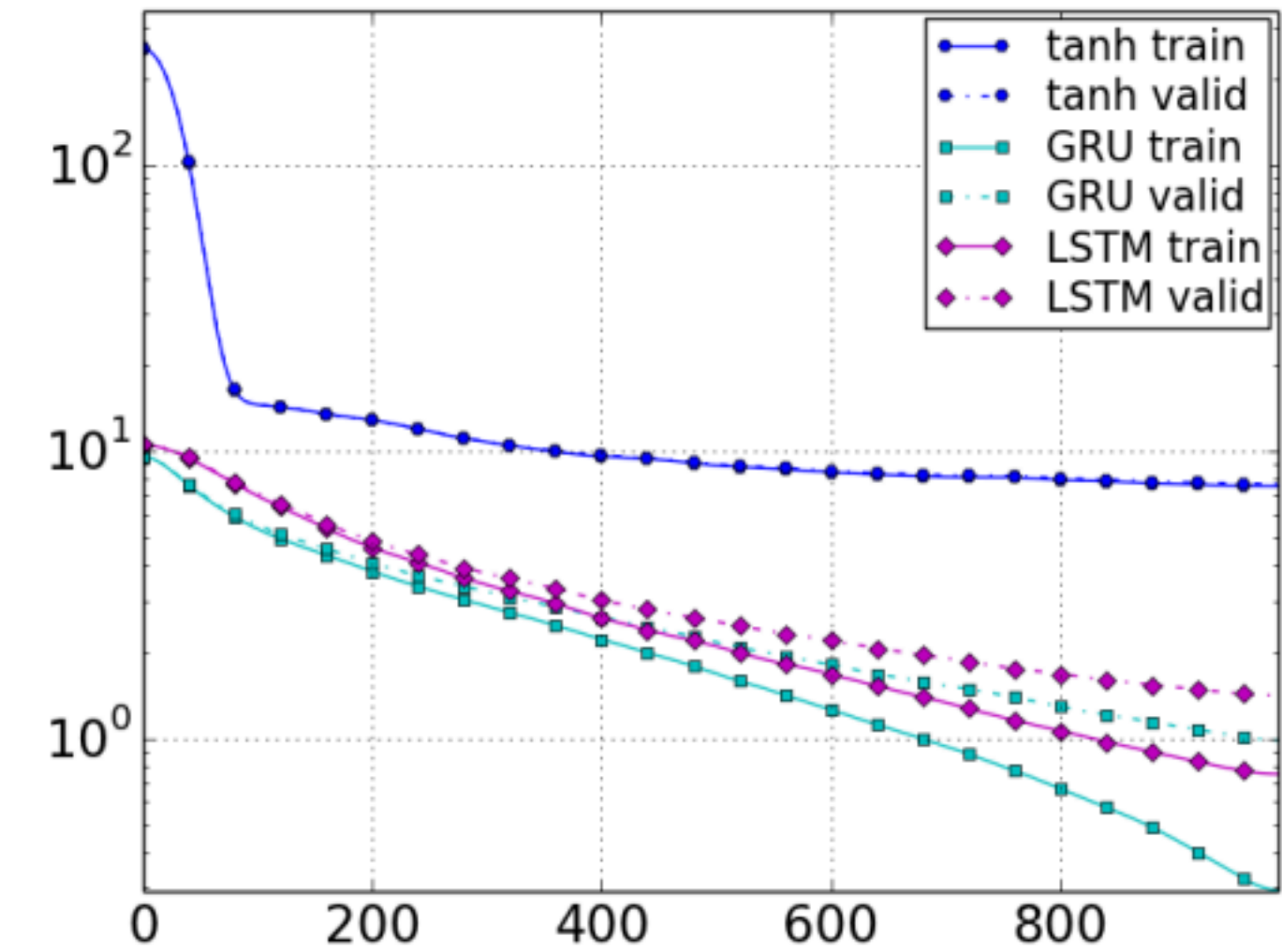
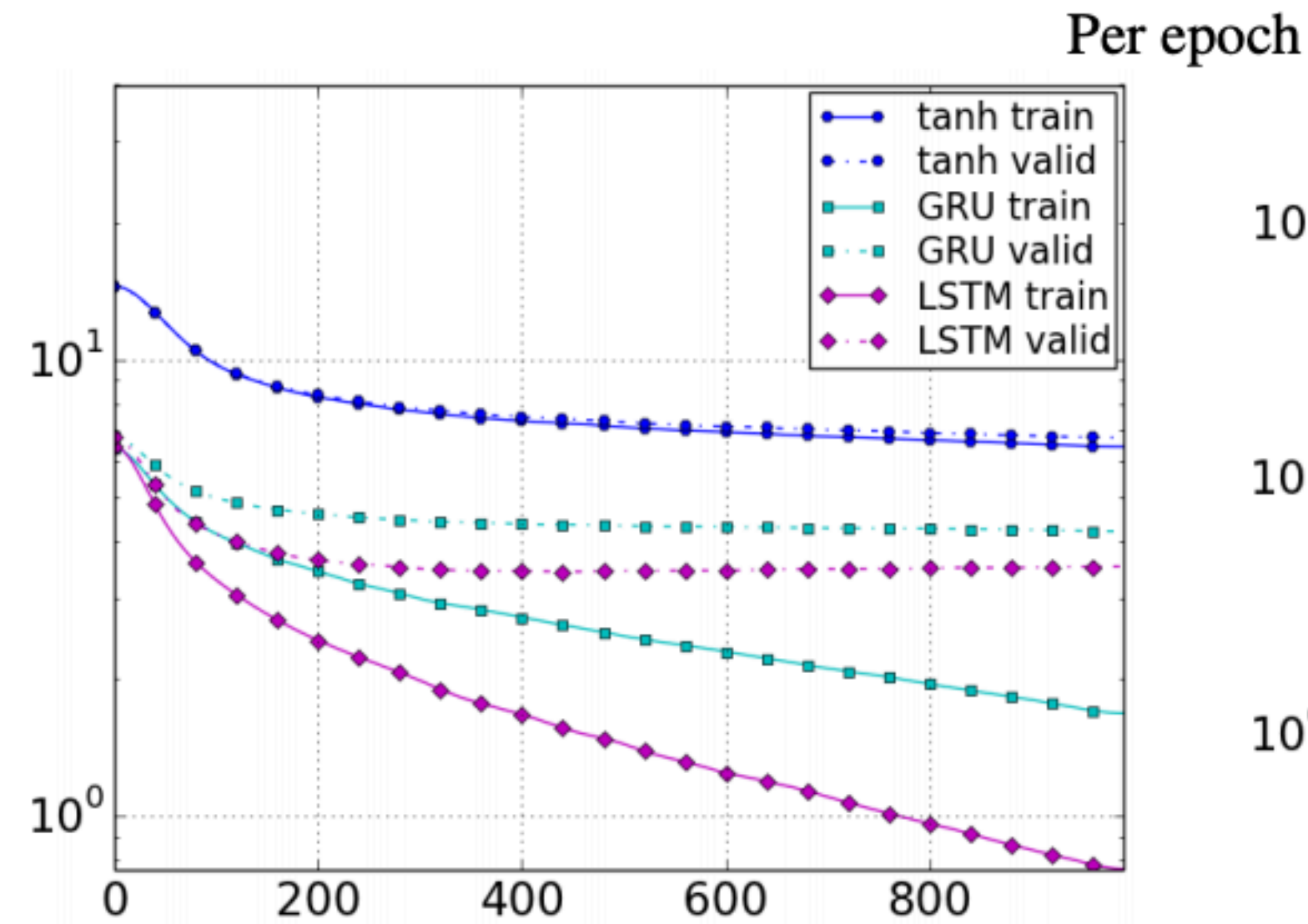


# LSTMs vs GRUs

Speech signal modeling



<https://imgflip.com/i/495iim>  
(only for fun!!!)



(a) Ubisoft Dataset A

(b) Ubisoft Dataset B



# Are LSTMs and GRUs optimal?

MUT1:

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

MUT2:

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

MUT3:

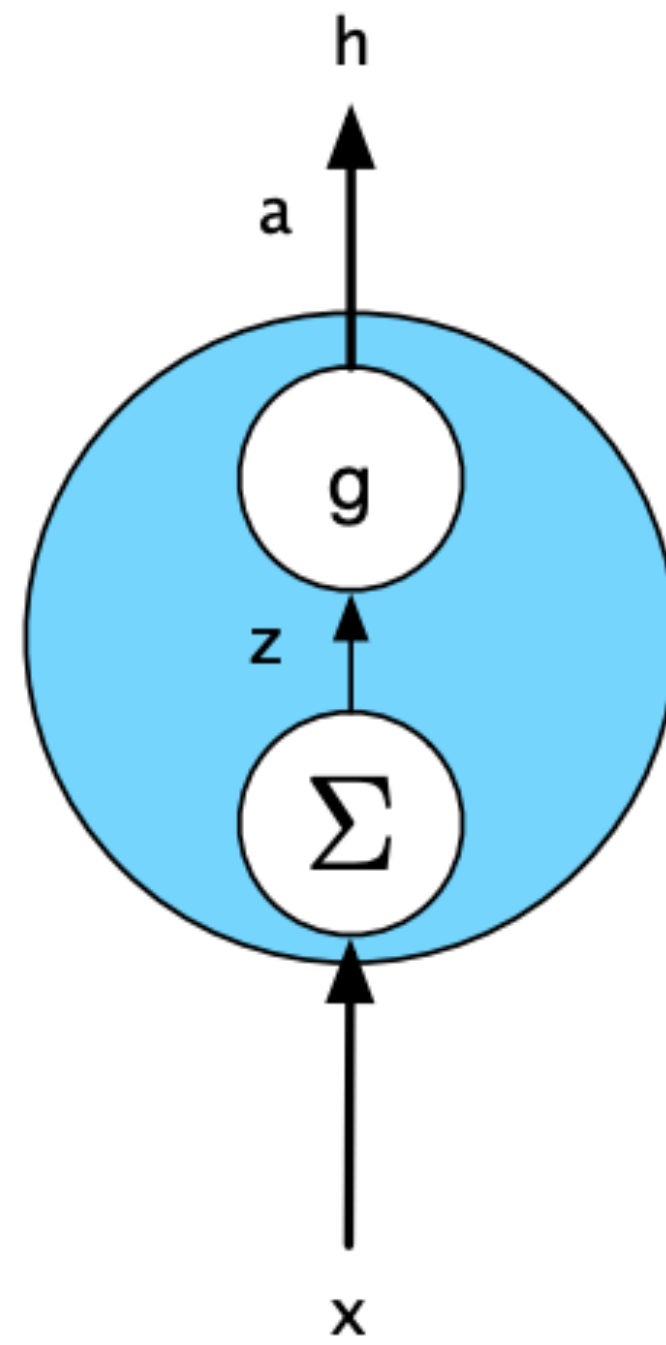
$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z) \\
 r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
 &+ h_t \odot (1 - z)
 \end{aligned}$$

Arch.	Arith.	XML	PTB
Tanh	0.29493	0.32050	0.08782
LSTM	0.89228	0.42470	0.08912
LSTM-f	0.29292	0.23356	0.08808
LSTM-i	0.75109	0.41371	0.08662
LSTM-o	0.86747	0.42117	0.08933
LSTM-b	0.90163	0.44434	0.08952
GRU	0.89565	0.45963	0.09069
MUT1	<b>0.92135</b>	<b>0.47483</b>	0.08968
MUT2	0.89735	<b>0.47324</b>	0.09036
MUT3	0.90728	0.46478	<b>0.09161</b>

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	<b>4.380 (79.83)</b>
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)

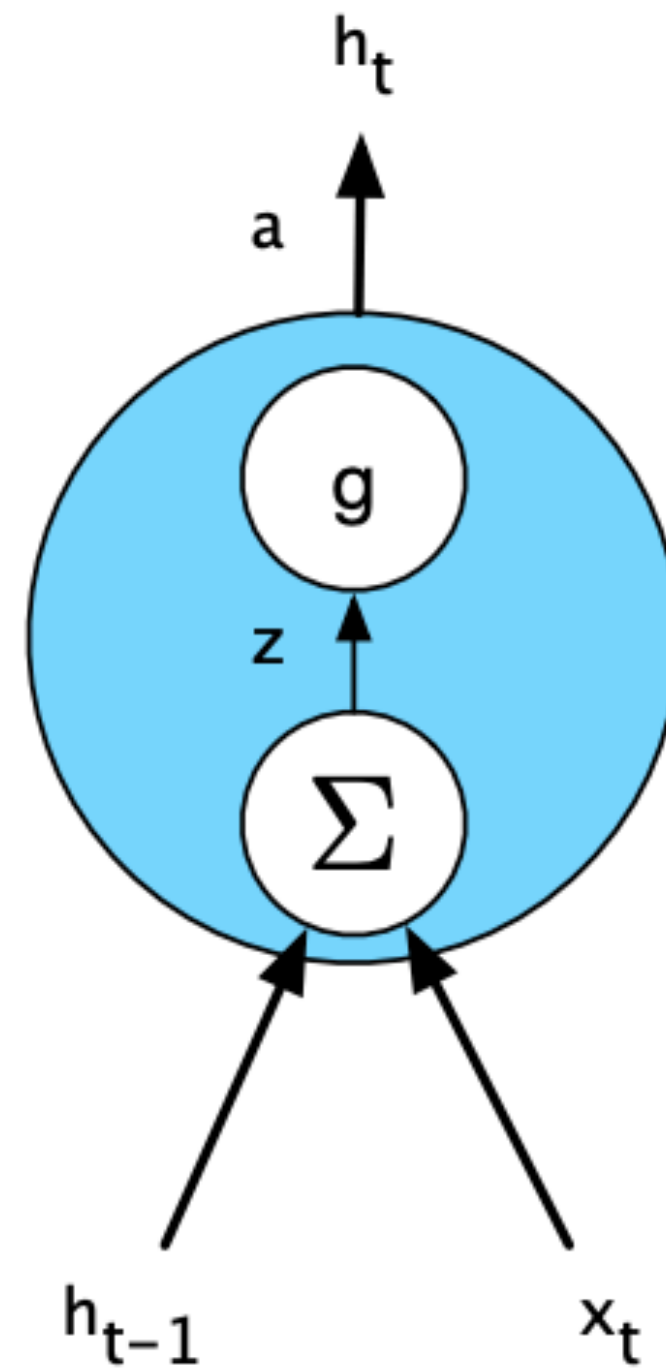
# Comparison: FFNNs vs simple RNNs vs LSTMs vs GRUs

Feedforward NNs



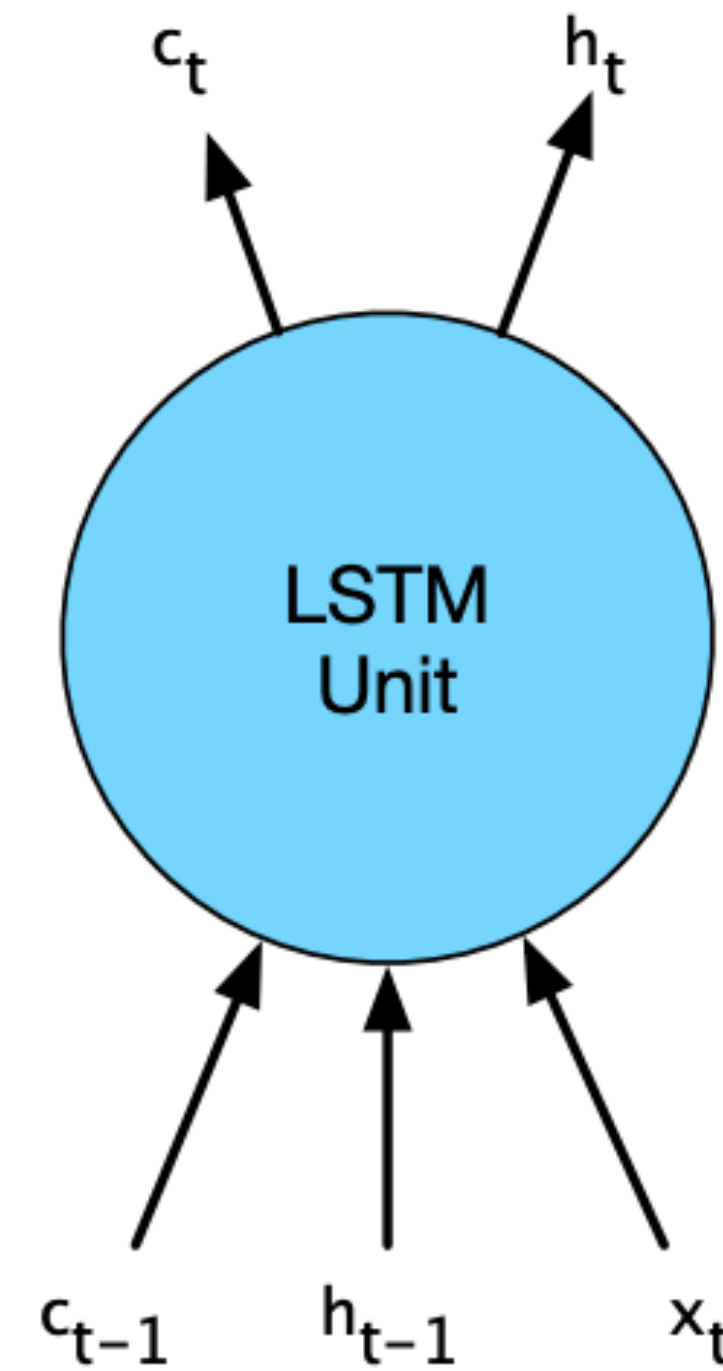
(a)

Simple RNNs



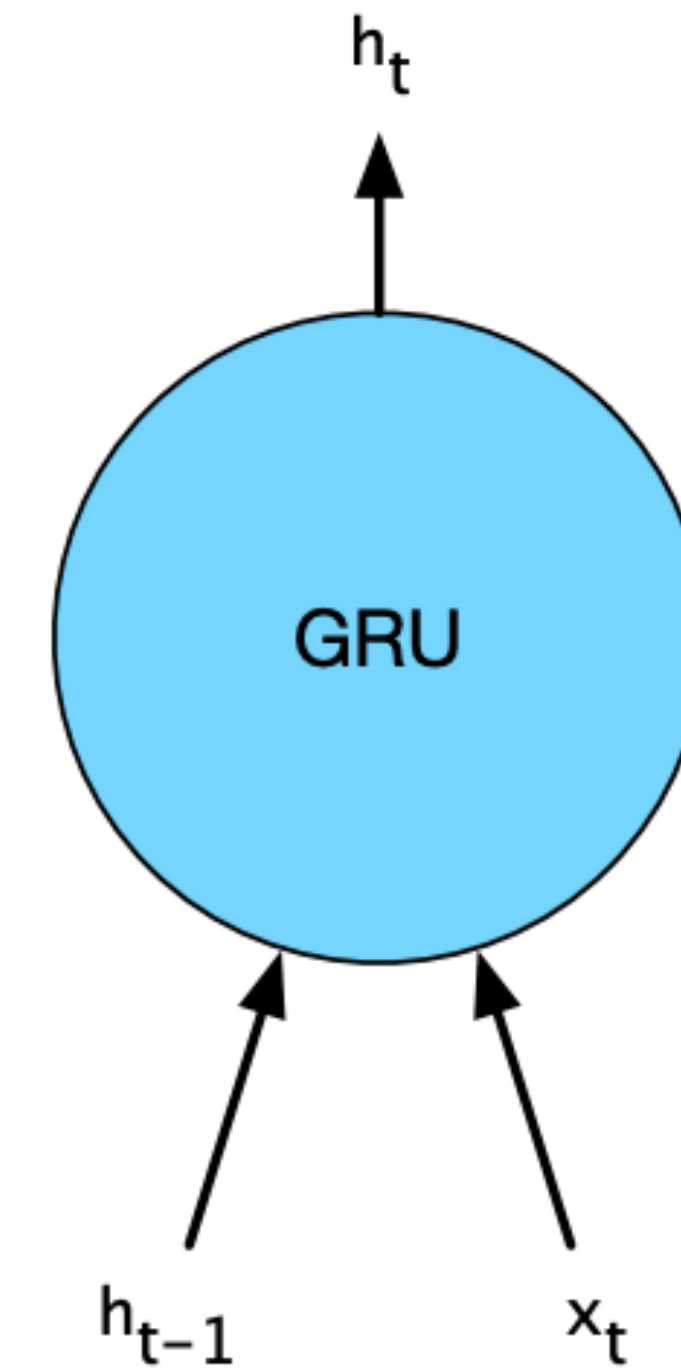
(b)

LSTMs



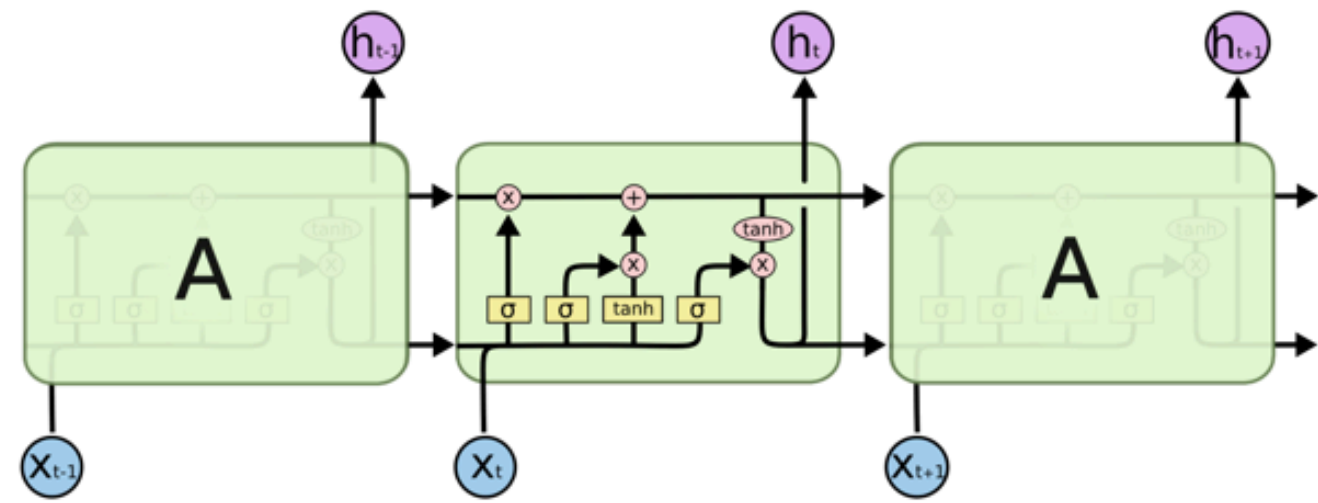
(c)

GRUs

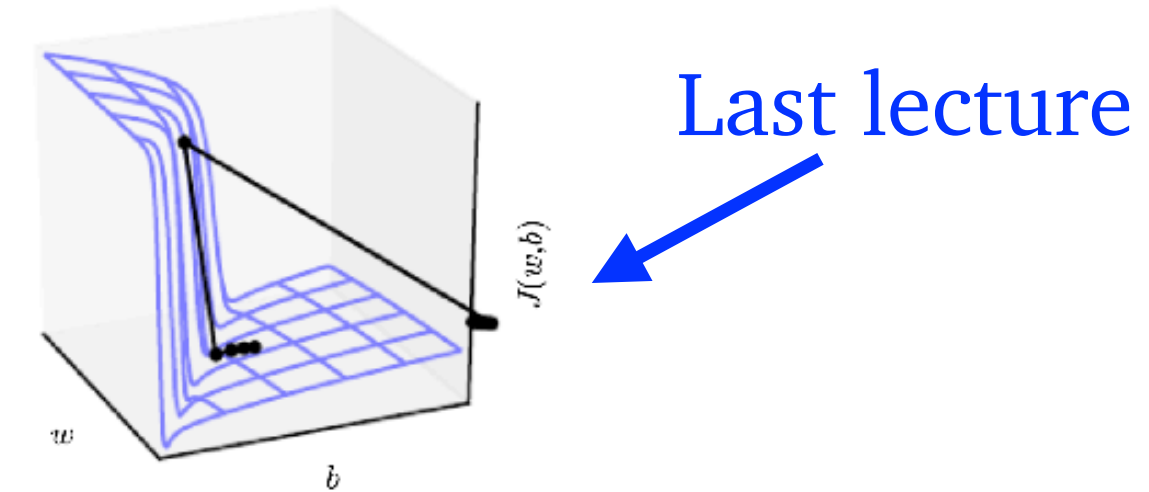


(d)

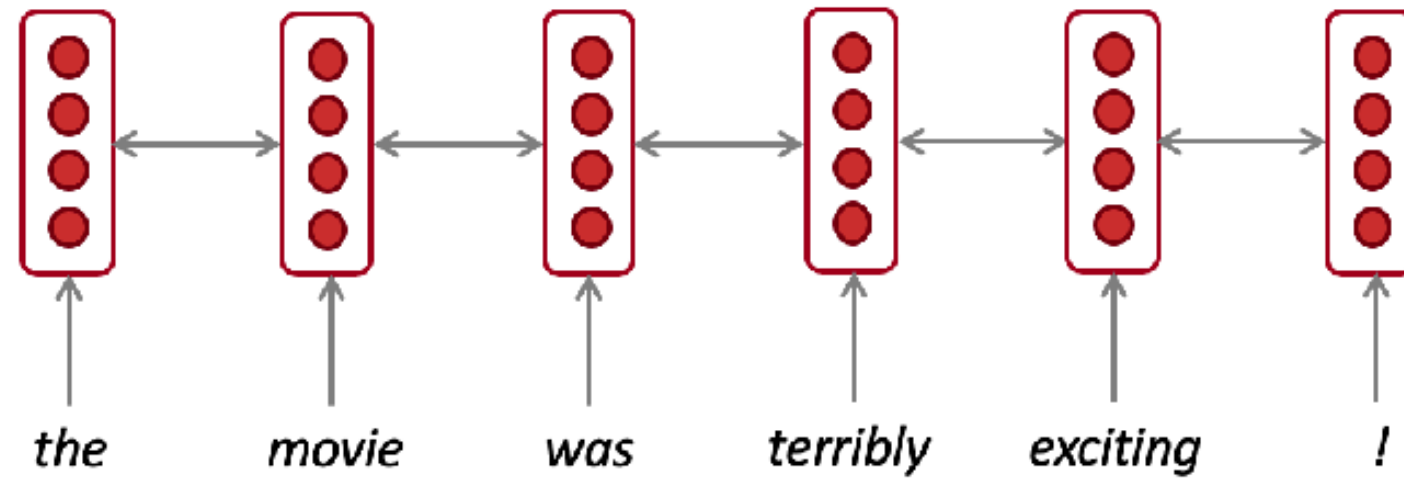
# Practical takeaways



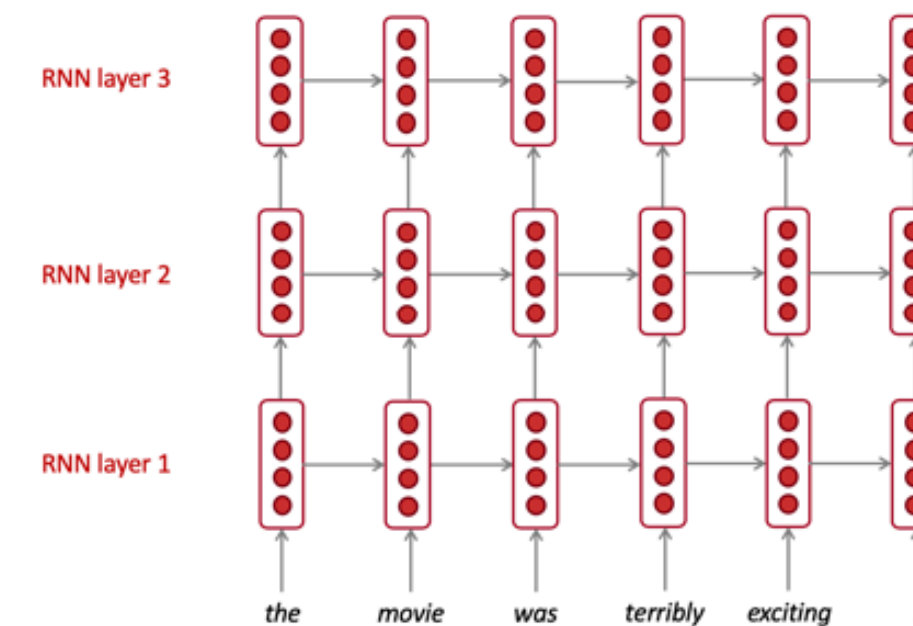
1. LSTMs are powerful



2. Clip your gradients



3. Use bidirectionality when possible



4. Multi-layer RNNs are more powerful, but you might need skip connections if it's deep

# Simple recurrent units (SRU)

## Simple Recurrent Units for Highly Parallelizable Recurrence

Tao Lei<sup>1</sup> Yu Zhang<sup>2</sup> Sida I. Wang<sup>1,3</sup> Hui Dai<sup>1</sup> Yoav Artzi<sup>1,4</sup> 2017  
<sup>1</sup>ASAPP Inc. <sup>2</sup>Google Brain <sup>3</sup>Princeton University <sup>4</sup>Cornell University  
<sup>1</sup>{tao, hd}@asapp.com <sup>2</sup>ngyuzh@google.com  
<sup>3</sup>sidaw@cs.princeton.edu <sup>4</sup>yoav@cs.cornell.edu

$$\begin{aligned}\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{v}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot (\mathbf{W} \mathbf{x}_t) \\ \mathbf{r}_t &= \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{v}_r \odot \mathbf{c}_{t-1} + \mathbf{b}_r) \\ \mathbf{h}_t &= \mathbf{r}_t \odot \mathbf{c}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t\end{aligned}$$

- Lighter form of recurrent neural networks
- Enable high amounts of parallelism in computation, while maintaining expressivity of recurrent computation
- Use of CUDA kernels to maximize parallel operations