

LSTM-GRU



Heung-II Suk

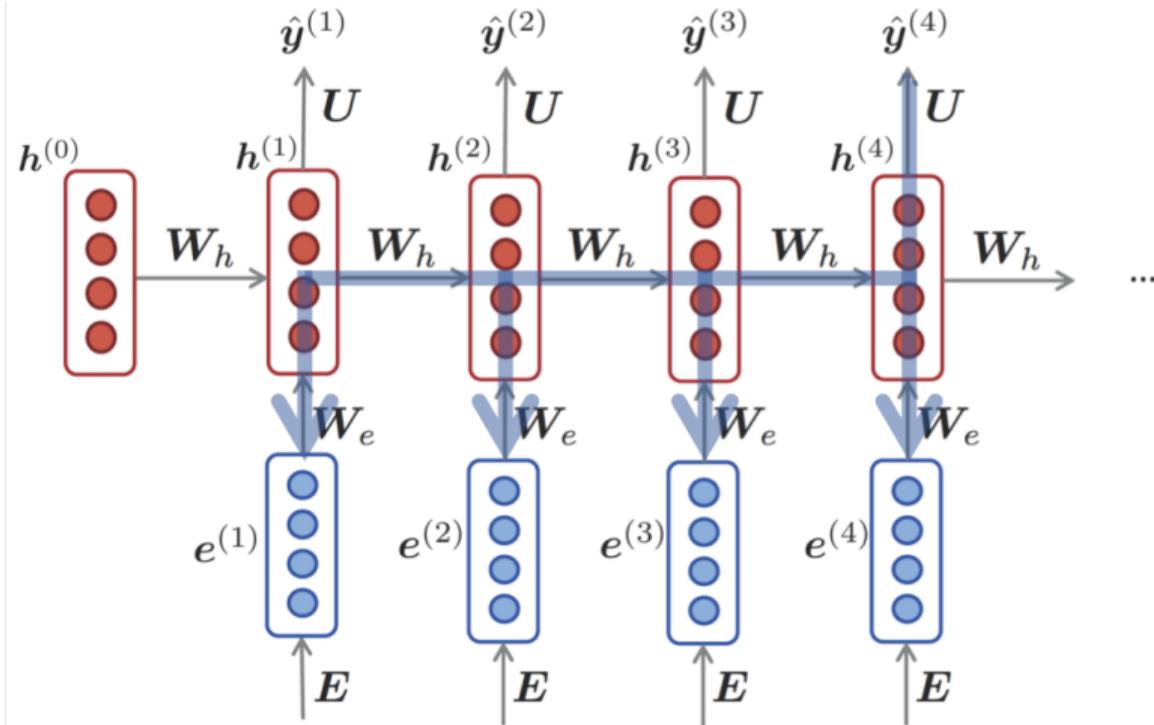
hisuk@korea.ac.kr

<http://milab.korea.ac.kr>



Department of Brain and Cognitive Engineering,
Korea University

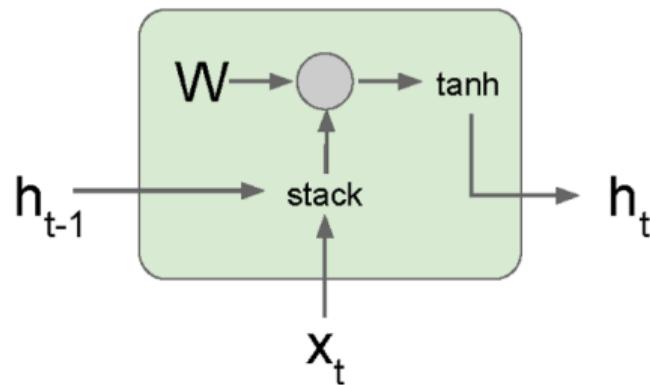
Vanishing/Exploding Gradient Problem in RNNs



Modified from CS224n-2018-Lecture8 Richard Socher

Vanilla RNN Gradient Flow

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

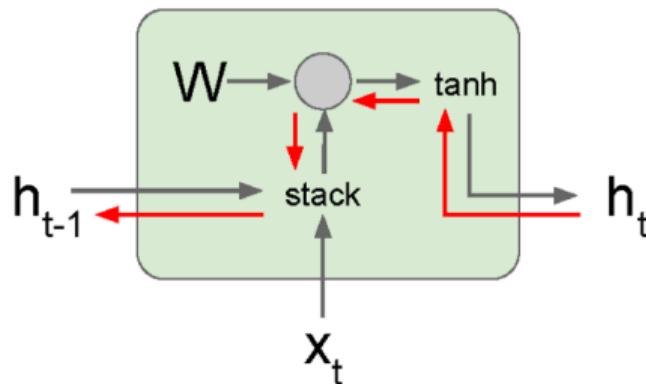


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

Vanilla RNN Gradient Flow

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

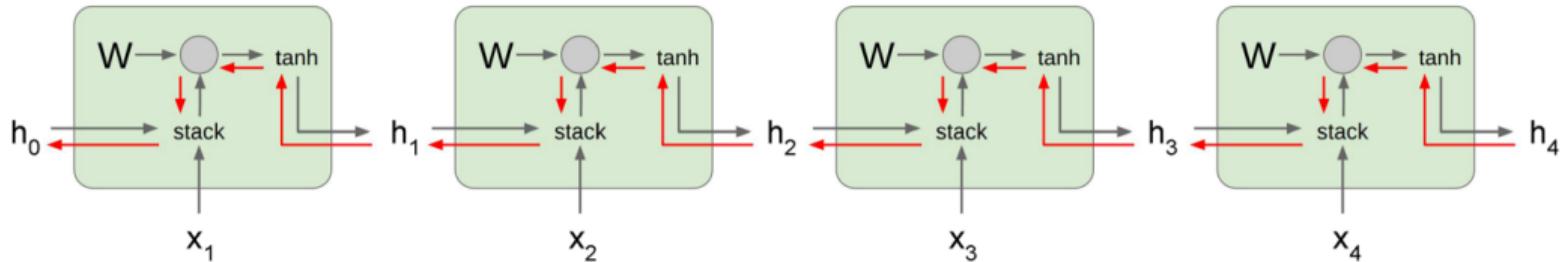
Backpropagation from h_t
to h_{t-1} multiplies by W
(actually W_{hh}^T)



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

Vanilla RNN Gradient Flow

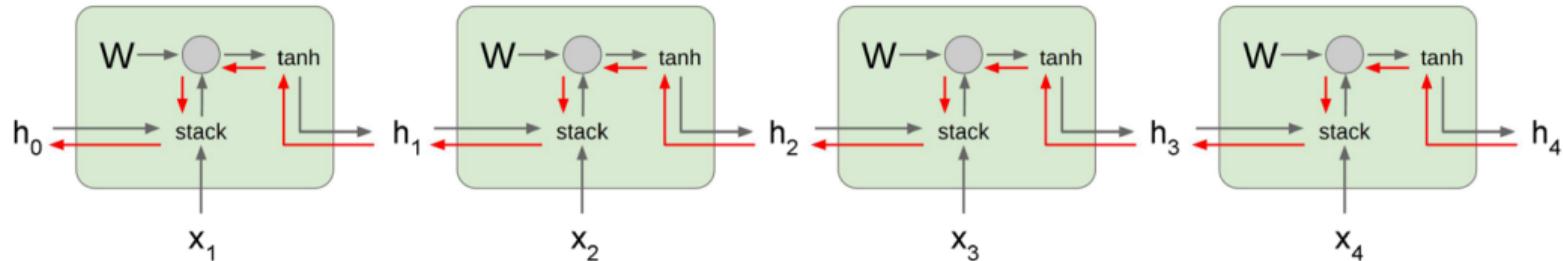
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

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



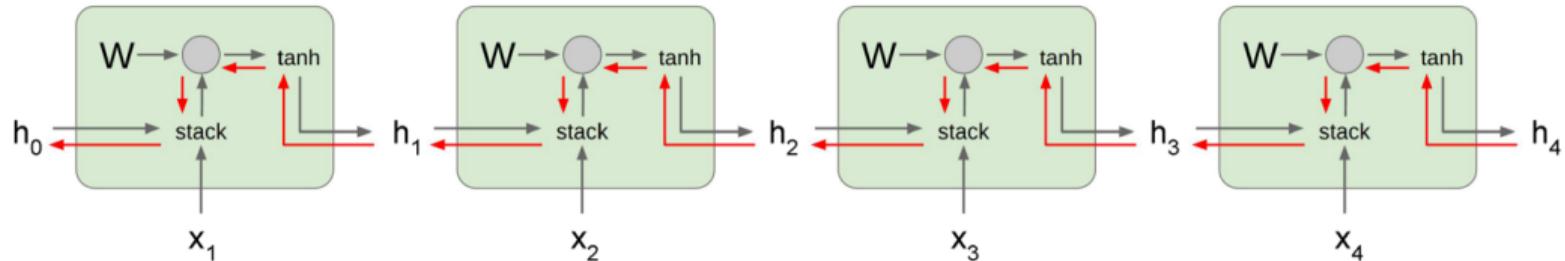
Computing gradient of h_0 involves many factors of W (and repeated \tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

Vanilla RNN Gradient Flow

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



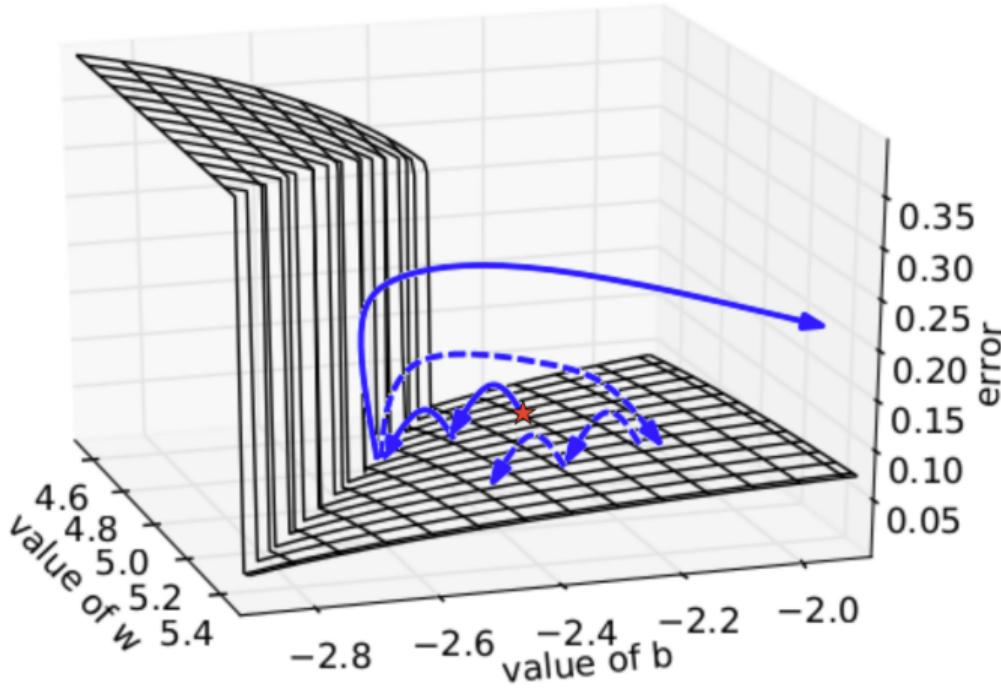
Computing gradient of h_0 involves many factors of W (and repeated \tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

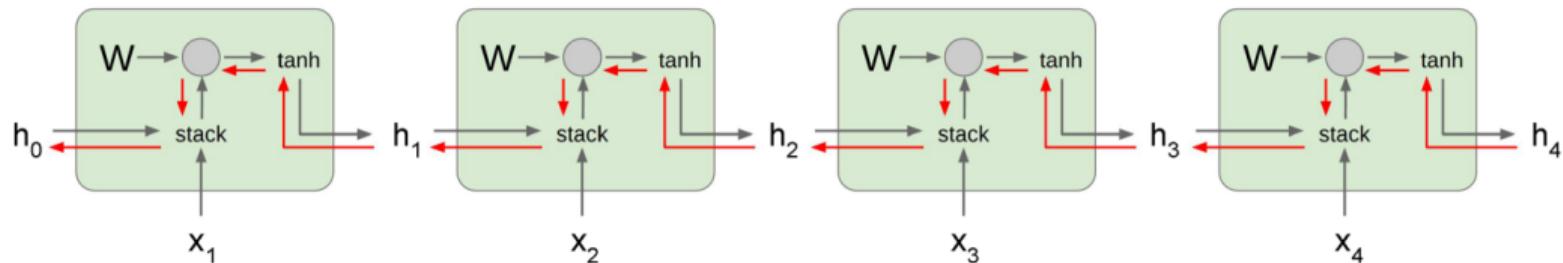
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```



Gradient clipping

Vanilla RNN Gradient Flow

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



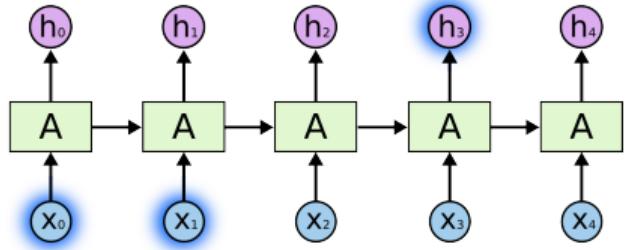
Computing gradient of h_0 involves many factors of W (and repeated \tanh)

Largest singular value > 1 :
Exploding gradients

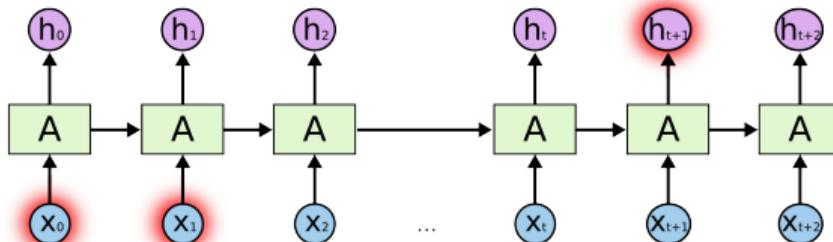
Largest singular value < 1 :
Vanishing gradients

→ Change RNN architecture

Long Dependency Problem



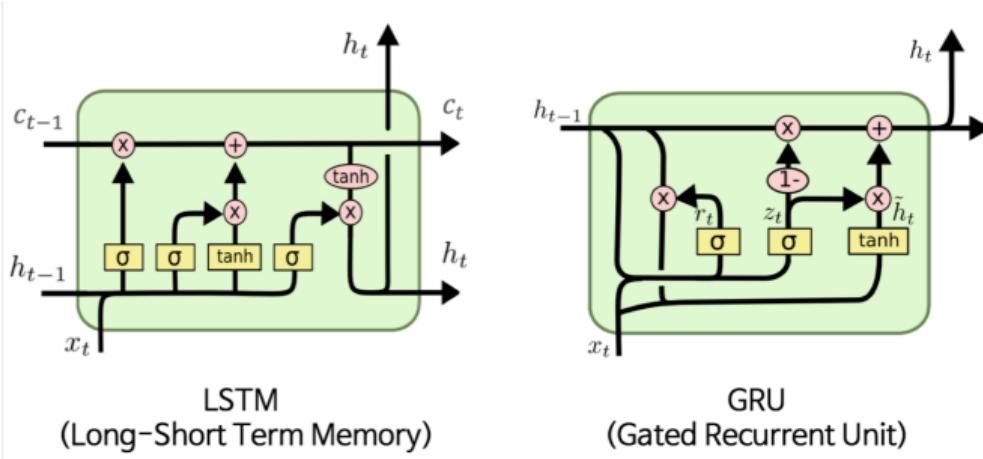
Short-term dependency



Long-term dependency

LSTM and GRU as a Rescue

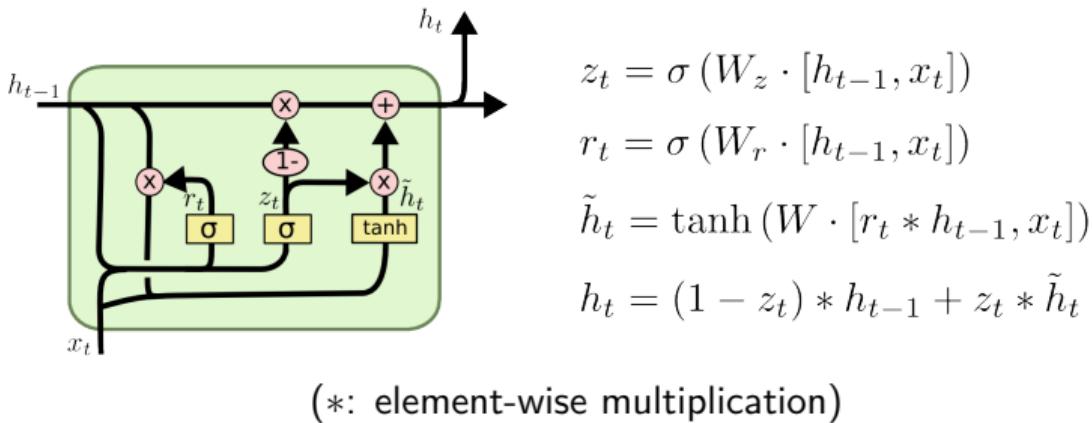
Solution to the VG problem: using better units

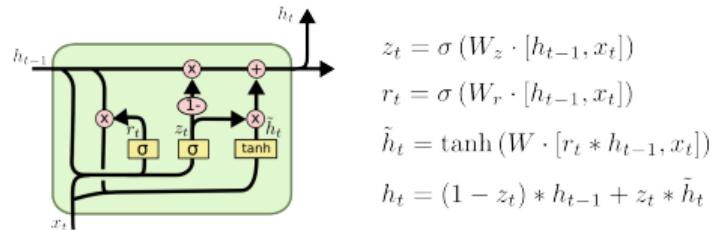


- LSTMs [Hochreiter & Schmidhuber, 1999], GRU [Cho *et al.*, 2014]
- Keep around memories to capture long distance dependencies
- Allow error messages to flow at different strengths depending on the inputs

Gated Recurrent Unit (GRU) [Cho et al., 2014]

- **update gate z_t , reset gate r_t :** current input x_t and previous hidden state h_{t-1}
- **new memory content \tilde{h}_t :** when reset gate unit is ~ 0 , ignores previous memory and only stores the new information
- **final memory h_t :** combines current and previous contents

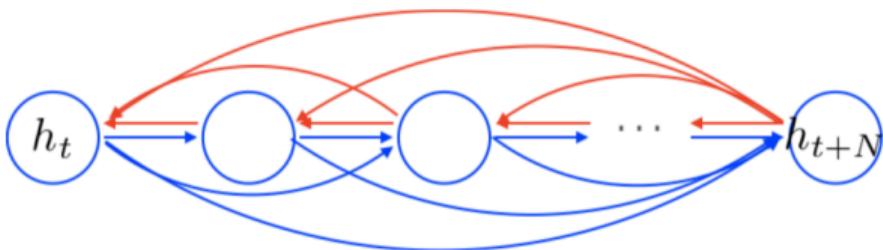




GRU intuition

- If reset gate r is close to 0, ignore previous hidden state → allowing model to drop information that is irrelevant in the future
- Update gate z controls how much of past state should matter now
 - ▶ If z close to 1, then we can copy information in that unit through many time steps! (**Less vanishing gradient!**)
- Units with **short-term dependencies** often have **reset gates very active**
- Units with **long-term dependencies** have **active update gates z**

Fixing vanishing gradient problems in GRU



Let the net prune unnecessary connections adaptively.

Adaptive shortcut connections

c.f.) ResNet: $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$

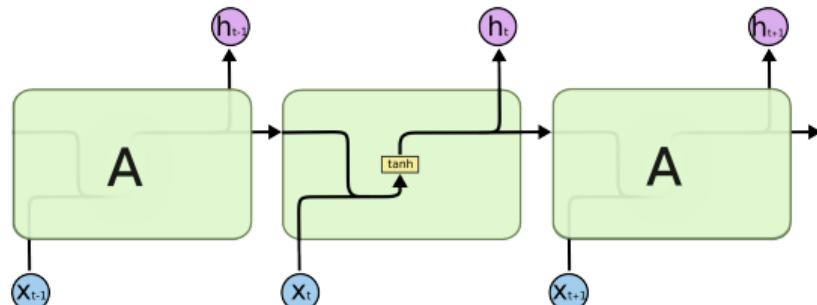
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

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

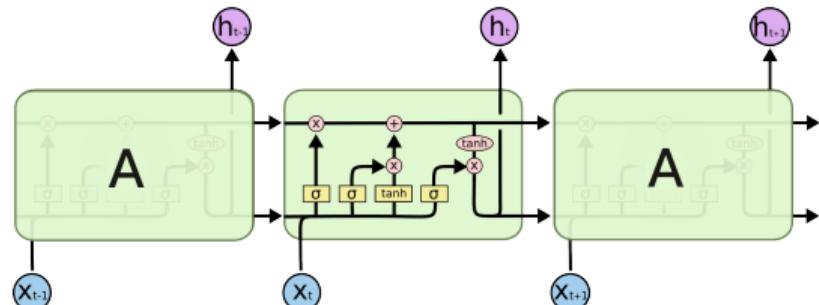
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

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

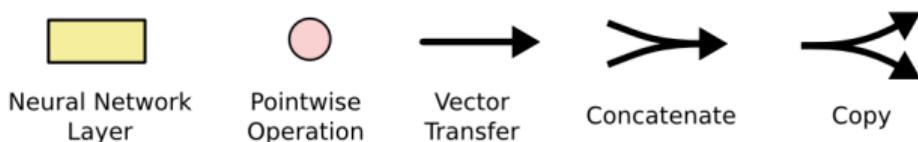
Long-Short Term Memory [Hochreiter and Schmidhuber, 1997]



The repeating module in a standard RNN contains a single layer

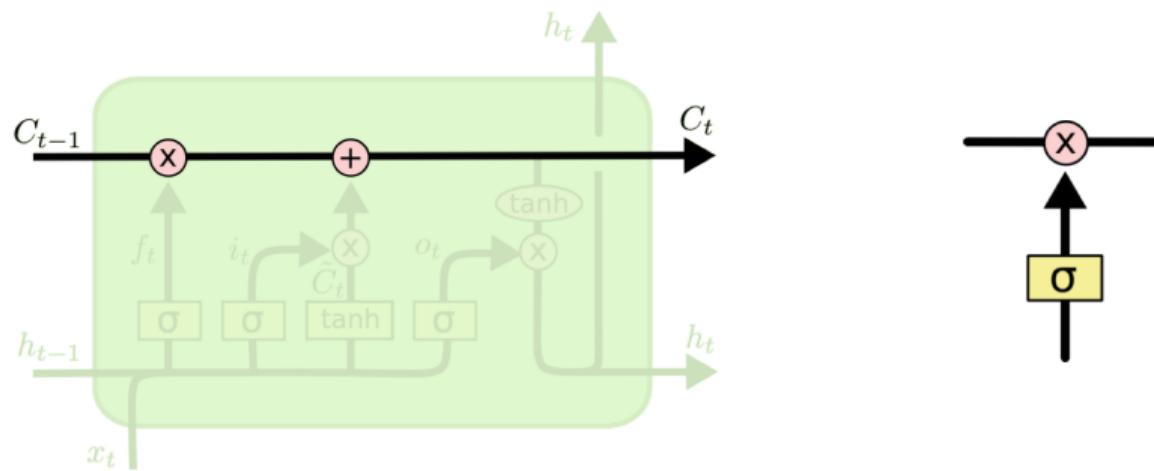


The repeating module in an LSTM contains four interacting layers



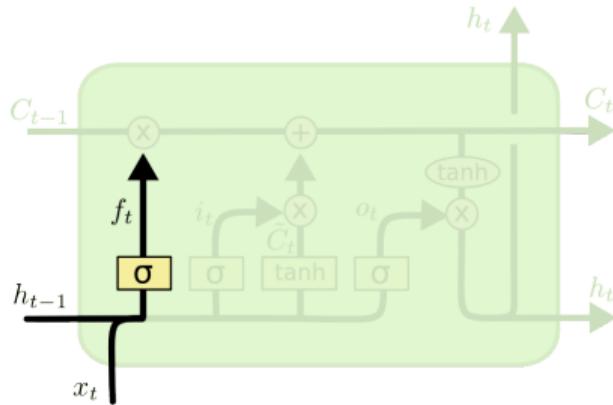
- “Cell state”

- ▶ removing or adding information, carefully regulated by structures called ‘gates’
- ▶ Gates: a way to optionally let information through



- “Forget gate layer”

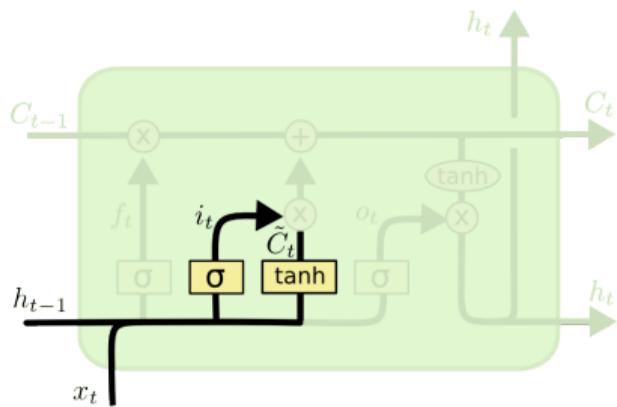
- ▶ to decide what information we’re going to throw away from the cell state



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

• “Input gate layer”

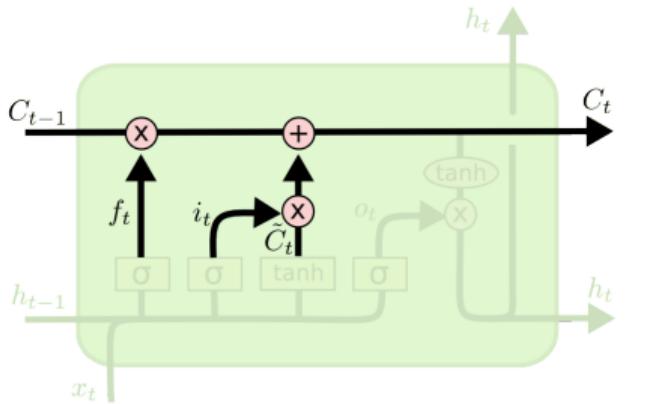
- ▶ a sigmoid layer to decide what new information we’re going to store in the cell state
- ▶ a tanh layer: to create a vector of new candidate values \tilde{C}_t that could be added to the cell state



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

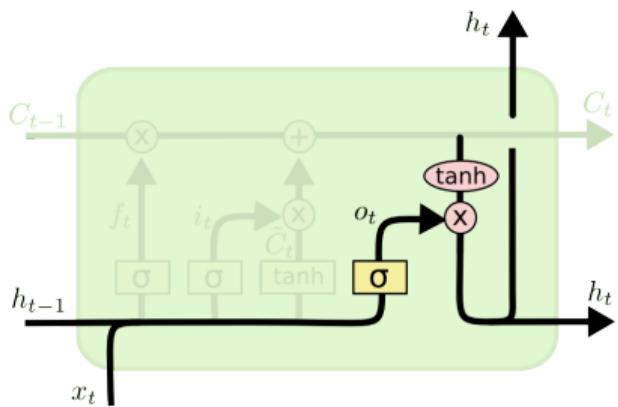
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- multiply the old state by f_t , forgetting the things we decided to forget earlier
- $i_t * \tilde{C}_t$: the new candidate values, scaled by how much we decided to update each state value



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

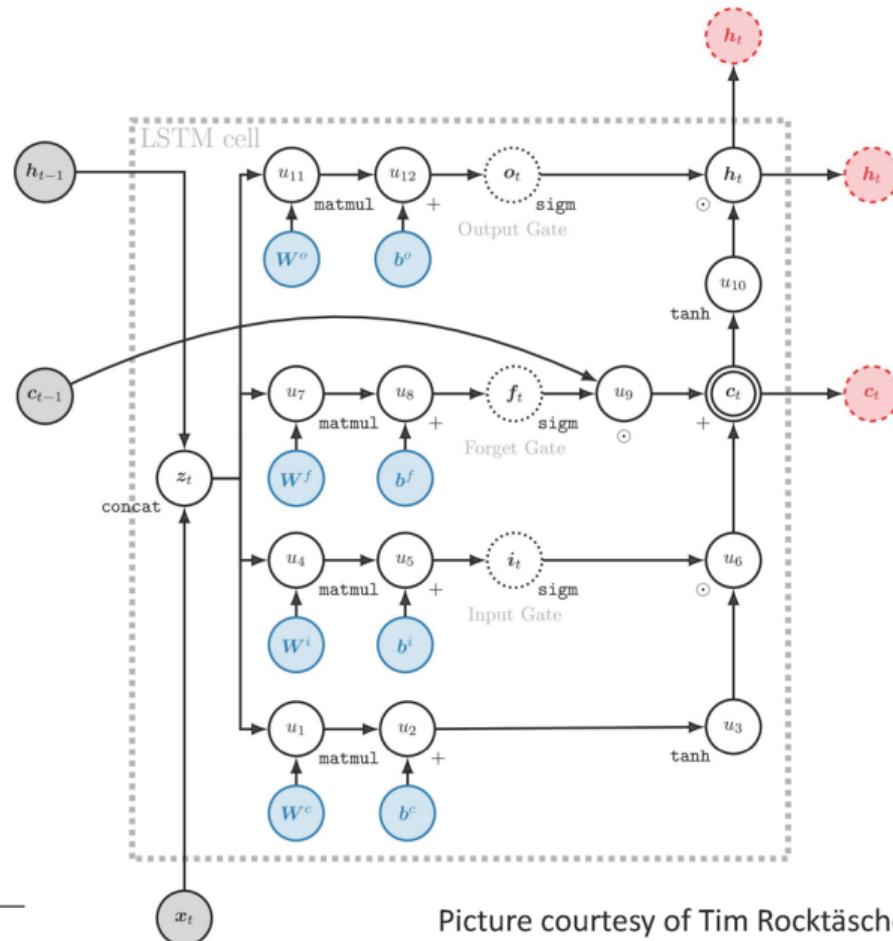
- To decide what we're going to output. This output will be based on our cell state, but will be a filtered version.
- a sigmoid layer which decides what parts of the cell state we're going to output
- put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to



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

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

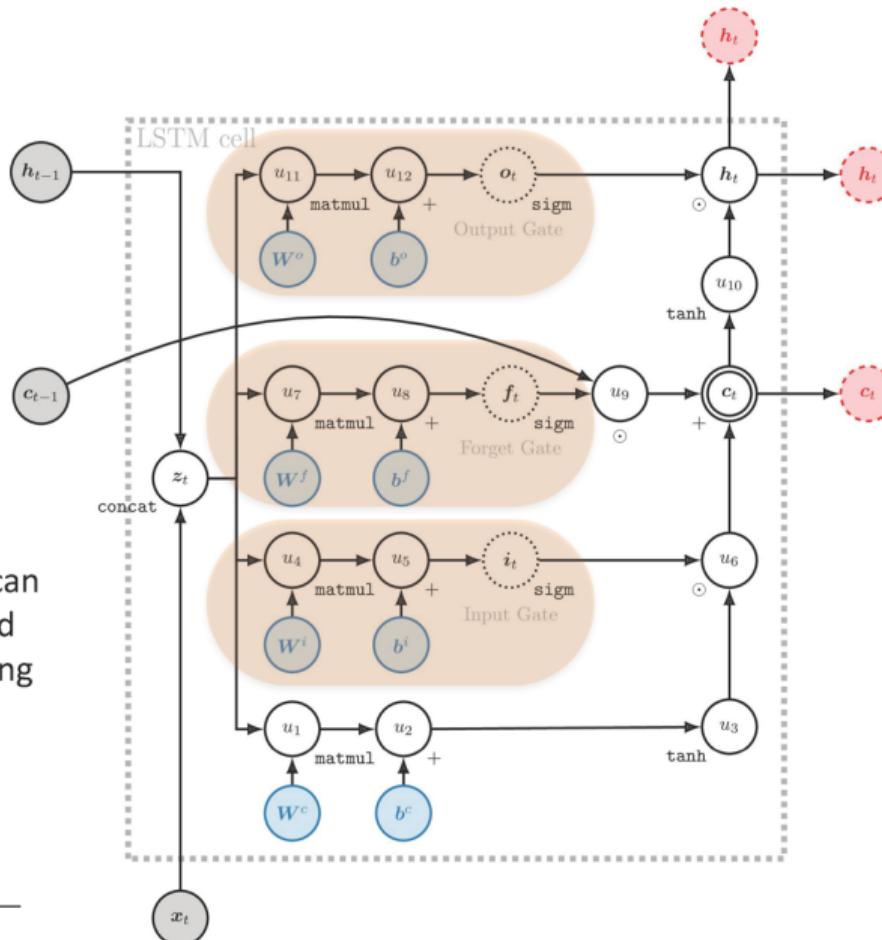
Another LSTM visualization inspired by code



Picture courtesy of Tim Rockäschel

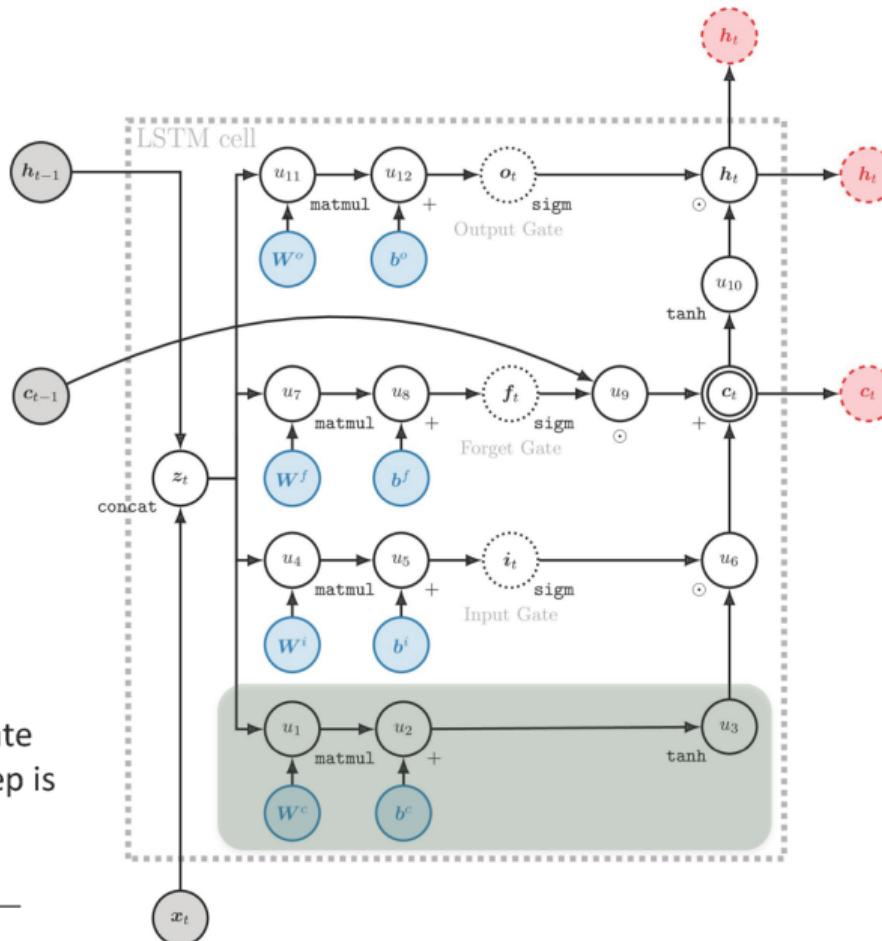
The LSTM

The LSTM gates all operations so stuff can be forgotten/ignored rather than it all being crammed on top of everything else



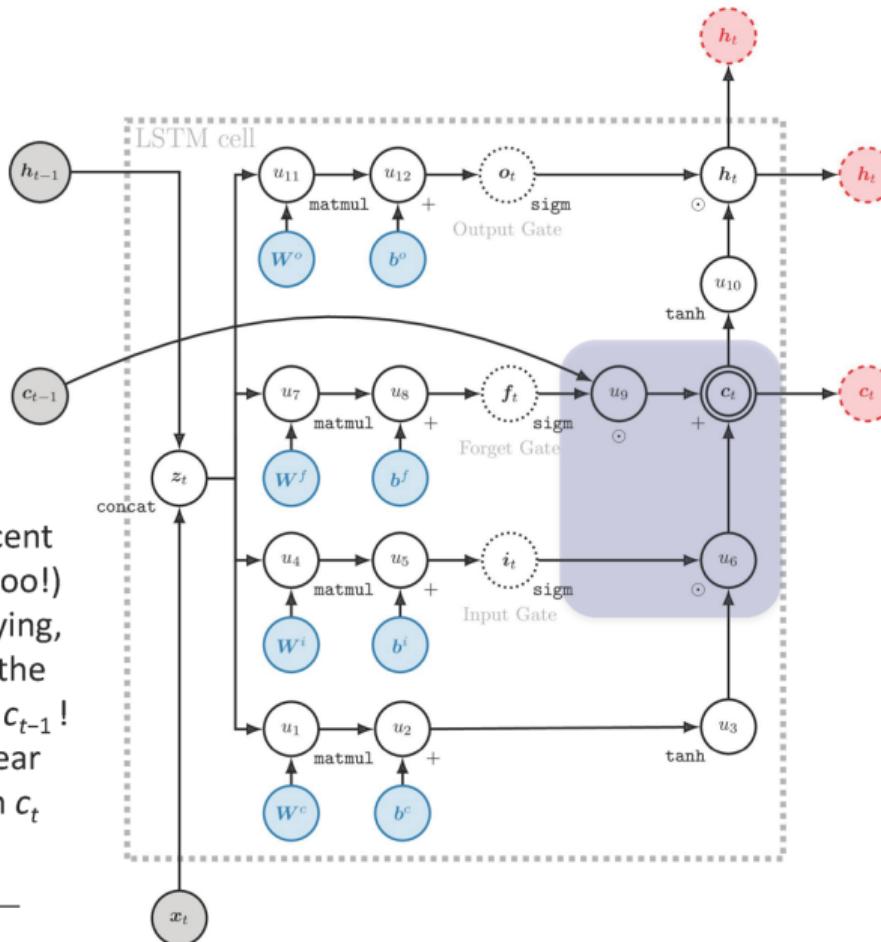
The LSTM

The non-linear update
for the next time step is
just like an RNN

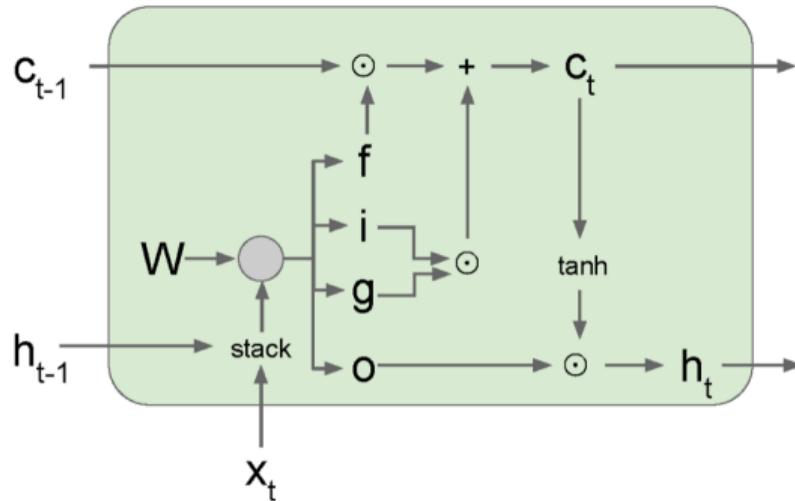


The LSTM

This part is the the secret! (Of other recent things like ResNets too!) Rather than multiplying, we get c_t by adding the non-linear stuff and c_{t-1} ! There is a direct, linear connection between c_t and c_{t-1} .



Gradient flow in LSTM

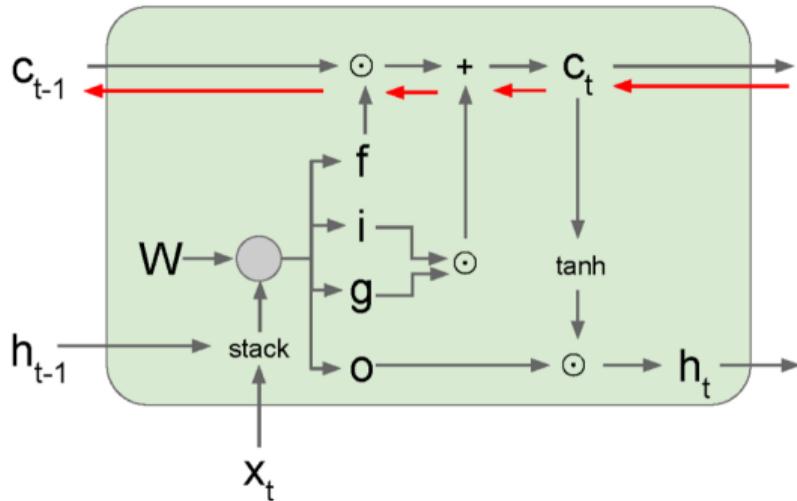


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Gradient flow in LSTM



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

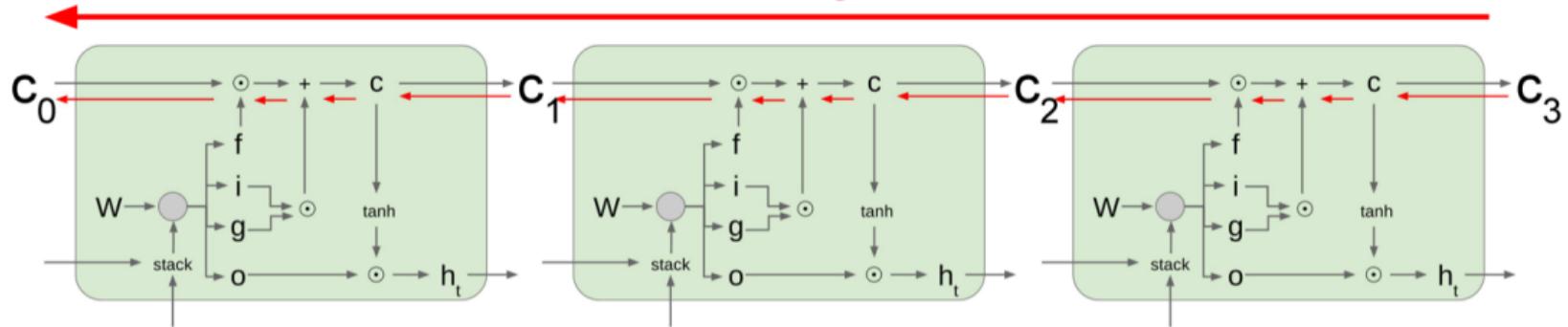
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

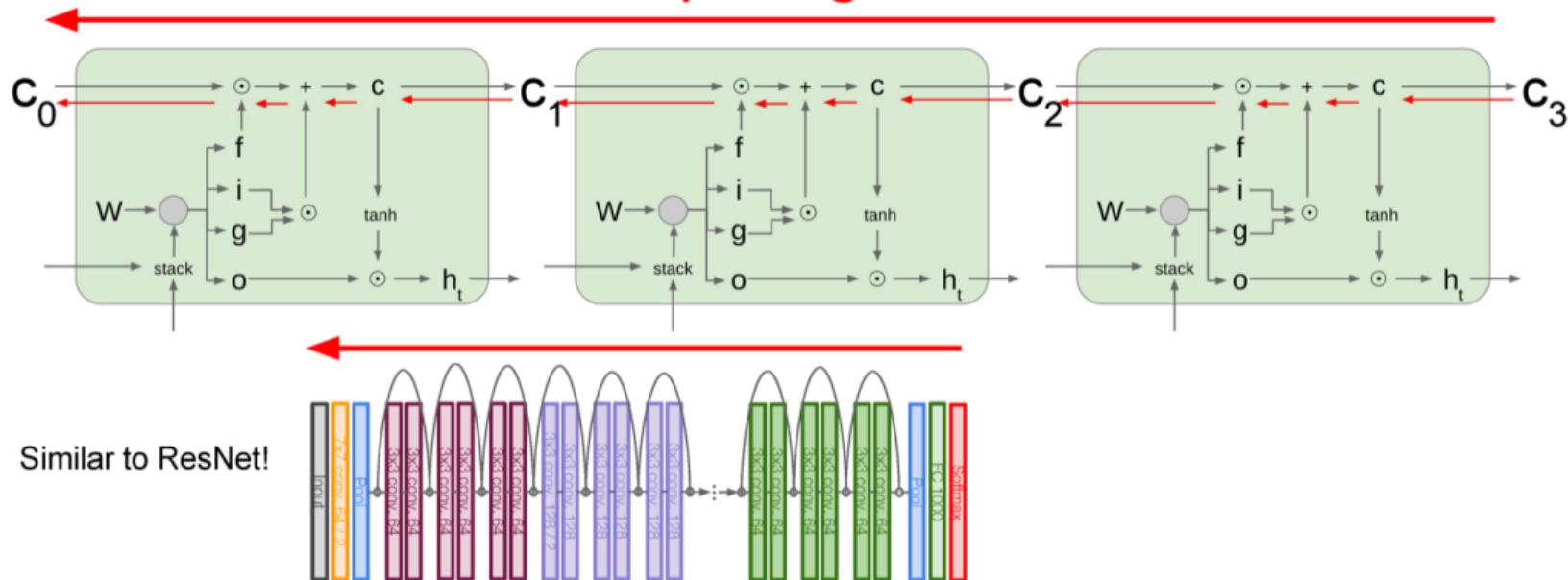
Gradient flow in LSTM

Uninterrupted gradient flow!



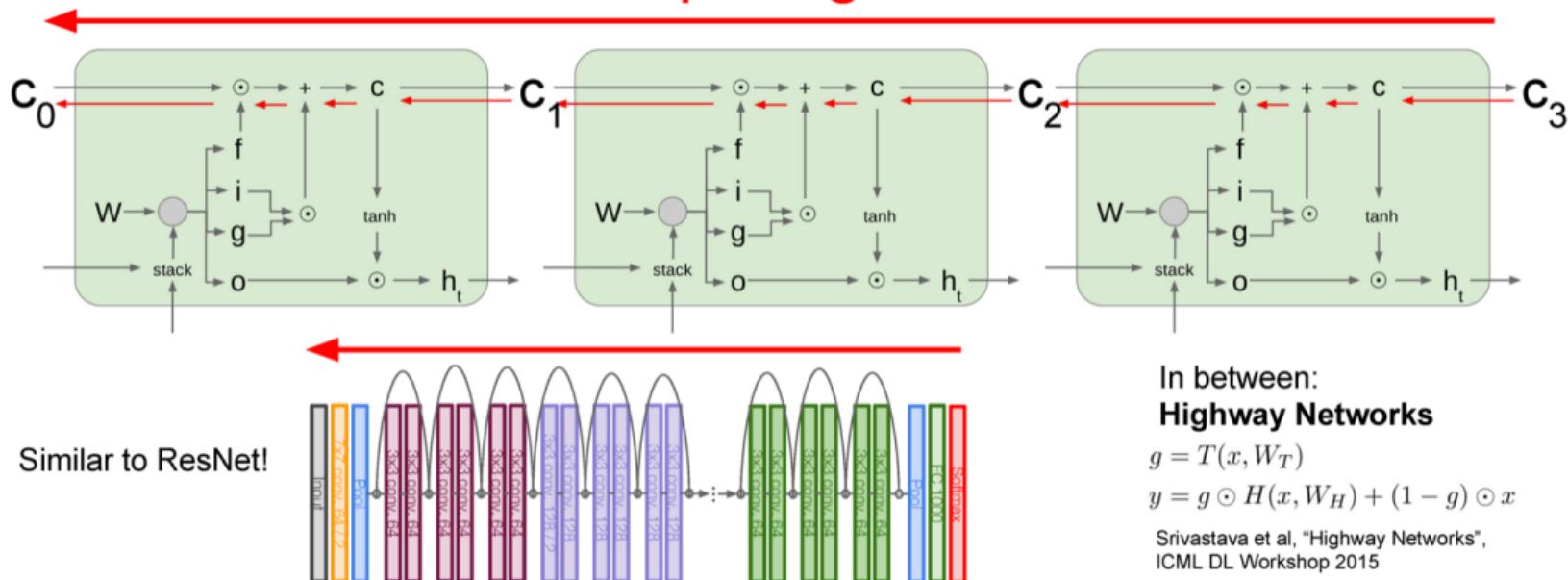
Gradient flow in LSTM

Uninterrupted gradient flow!



Gradient flow in LSTM

Uninterrupted gradient flow!



In between:
Highway Networks

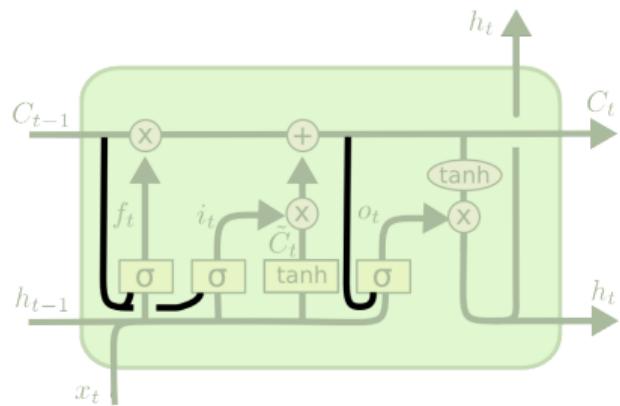
$$g = T(x, W_T)$$

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

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

Variants of LSTM

- [Gers & Schmidhuber, 2000]: to let the gate layers look at the cell state (called ‘peephole LSTM’)



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

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

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

- Coupled forget and input gates
 - ▶ Only forget when we're going to input something in its place
 - ▶ Only input new values to the state when we forget something older

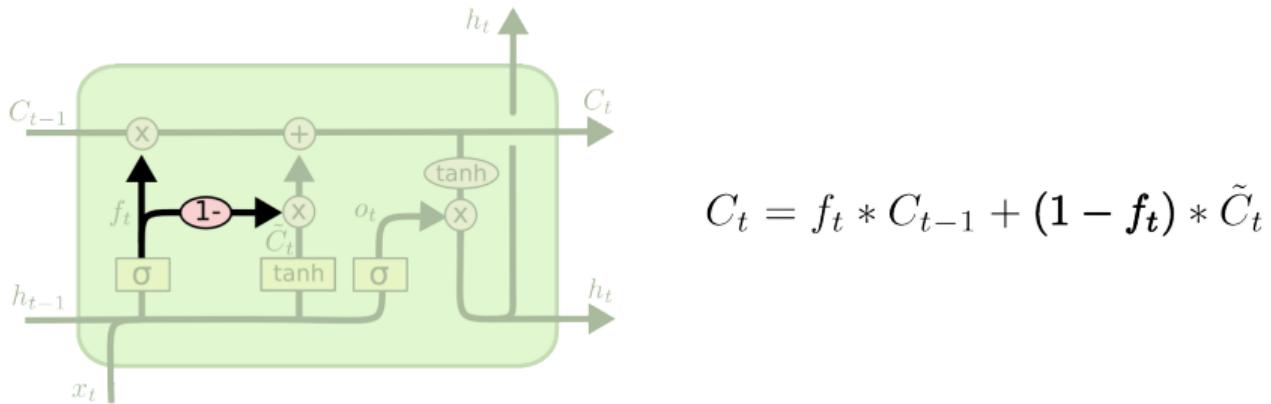
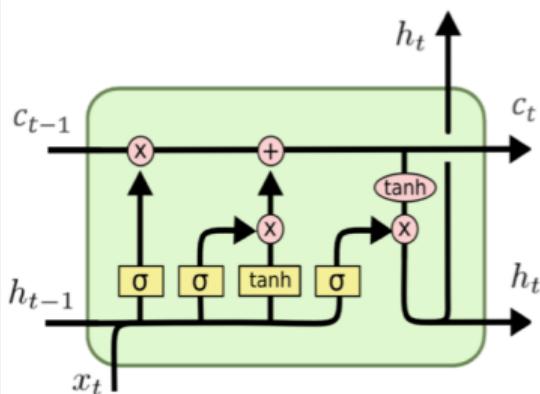


Image source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



LSTM
(Long-Short Term Memory)

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

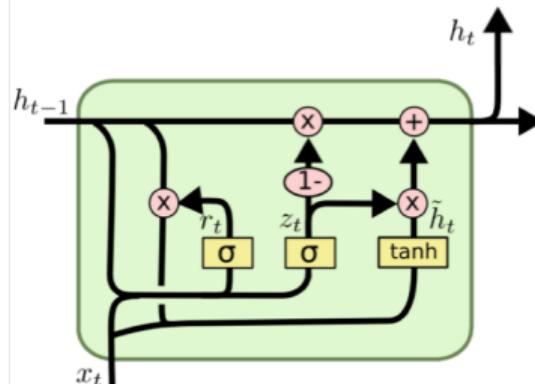
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$\tilde{c}_t = \tanh(W_c [x_t] + U_c h_{t-1} + b_c)$$

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

$$i_t = \sigma(W_i [x_t] + U_i h_{t-1} + b_i)$$

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



GRU
(Gated Recurrent Unit)

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

$$\tilde{h}_t = \tanh(W [x_t] + U(r_t \odot h_{t-1}) + b)$$

$$u_t = \sigma(W_u [x_t] + U_u h_{t-1} + b_u)$$

$$r_t = \sigma(W_r [x_t] + U_r h_{t-1} + b_r)$$

**Thank you
for your attention!!!**

(Q & A)

hisuk (AT) korea.ac.kr

<http://milab.korea.ac.kr>