Insight
Centre for Data Analytics

DEEP
LEARNING
WORKSHOP

Dublin City University
21-22 May 2018

DCU

#InsightDL2018

Day 1 Lecture 6

Recurrent Neural Networks

Xavier Giro-i-Nieto
xavier.giro@upc.edu

Associate Professor
Intelligent Data Science and Artificial Intelligence Center
Universitat Politecnica de Catalunya (UPC)

Acknowledgements

Santiago Pascual

Feed-forward Neural Network

Feed Forward

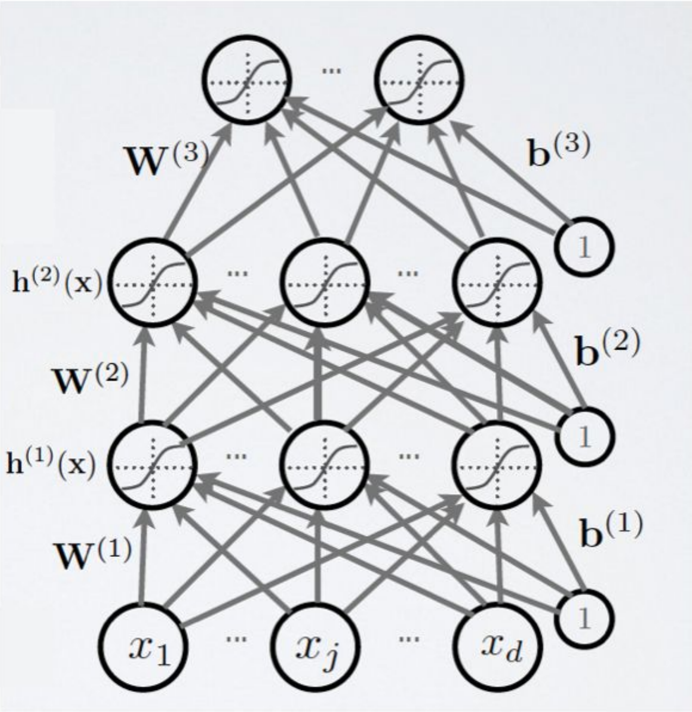


Figure: Hugo Larochelle

OUTPUT (y)

Feed-forward Weights (W_i)

$$y = f(\boxed{W_3} \cdot h_2 + b_3)$$

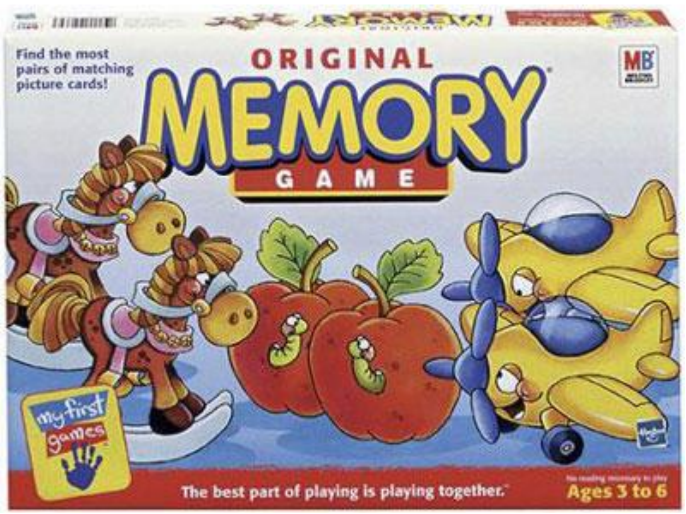
Hidden States h_1 & h_2

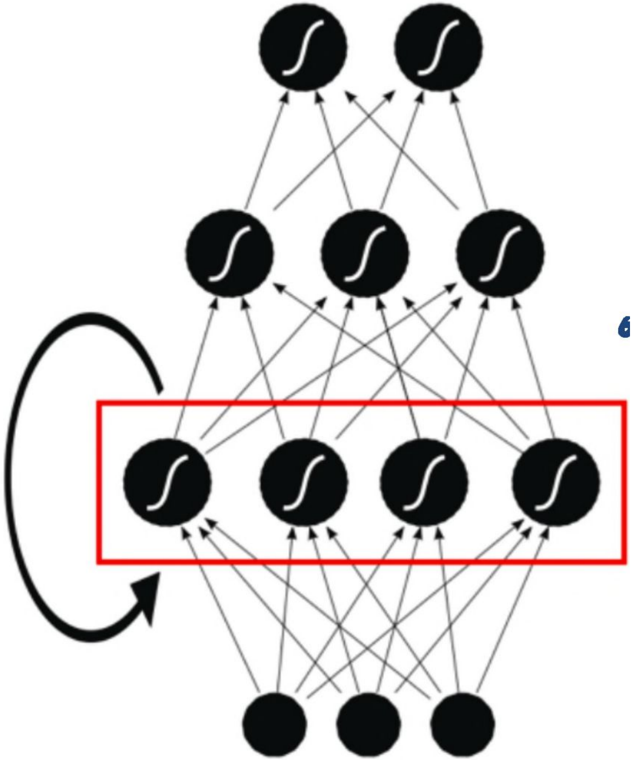
$$h_2 = f(\boxed{W_2} \cdot h_1 + b_2)$$

$$h_1 = f(\boxed{W_1} \cdot x + b_1)$$

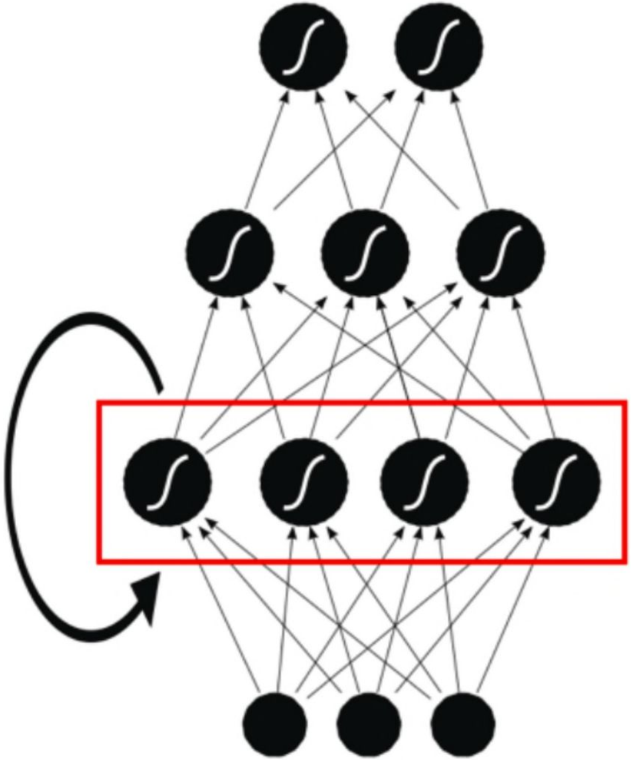
INPUT (x)

Recurrent Neural Network (RNN)





Recurrent Neural Network (RNN)

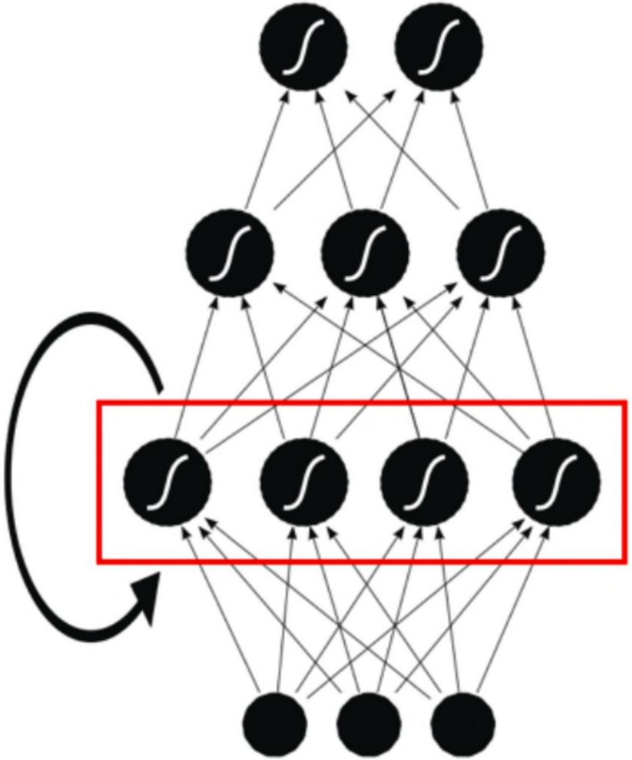


Feed-forward
Weights (W)

$$h_t = f(\boxed{\mathbf{W}} \cdot x_t + \boxed{U} \cdot h_{t-1} + b)$$

Recurrent
Weights (U)

Recurrent Neural Network (RNN)



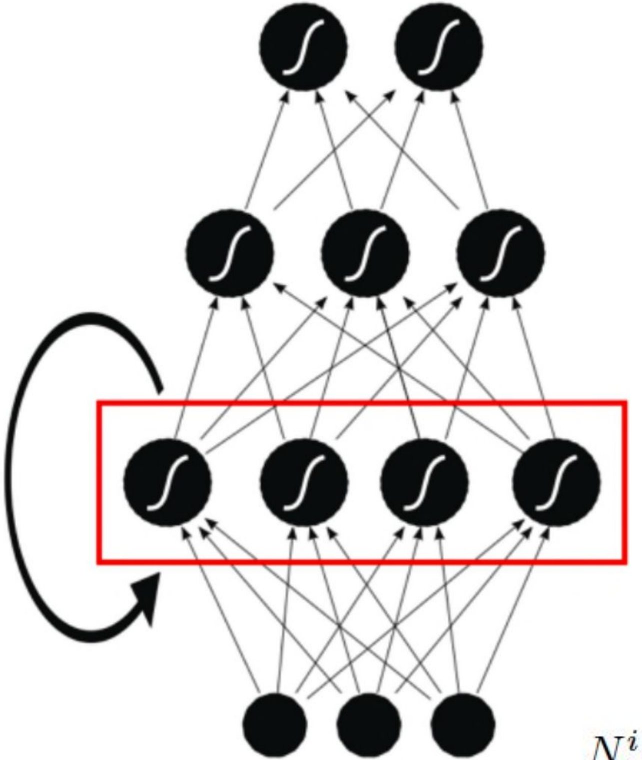
Updated state

Previous state

$$\boxed{h_t} = f(W \cdot x_t + U \cdot \boxed{h_{t-1}} + b)$$

INPUT (x)

Recurrent Neural Network (RNN)

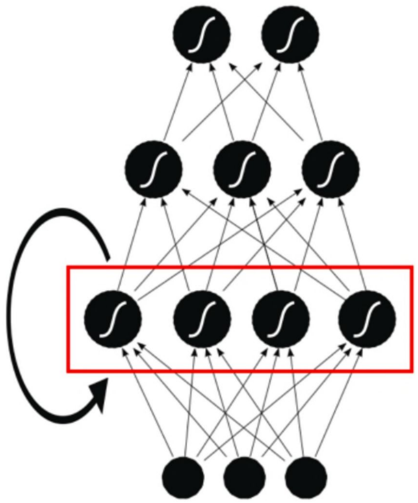


$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

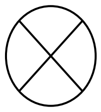
$$N^i_{params} = N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units}$$

Recurrent Neural Network (RNN)

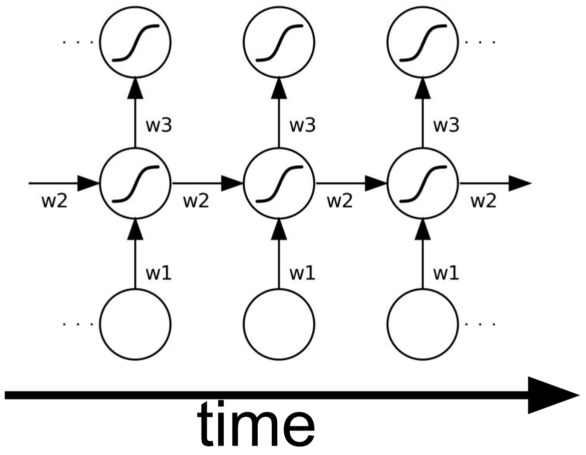
Unfold
(Rotation 90º)



time



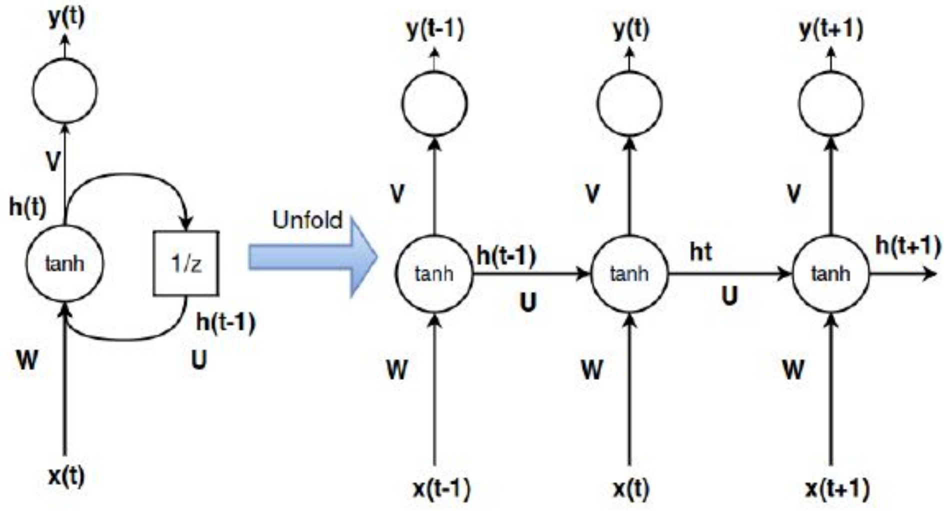
Unfold
(Rotation 90º)



Inference with a RNN

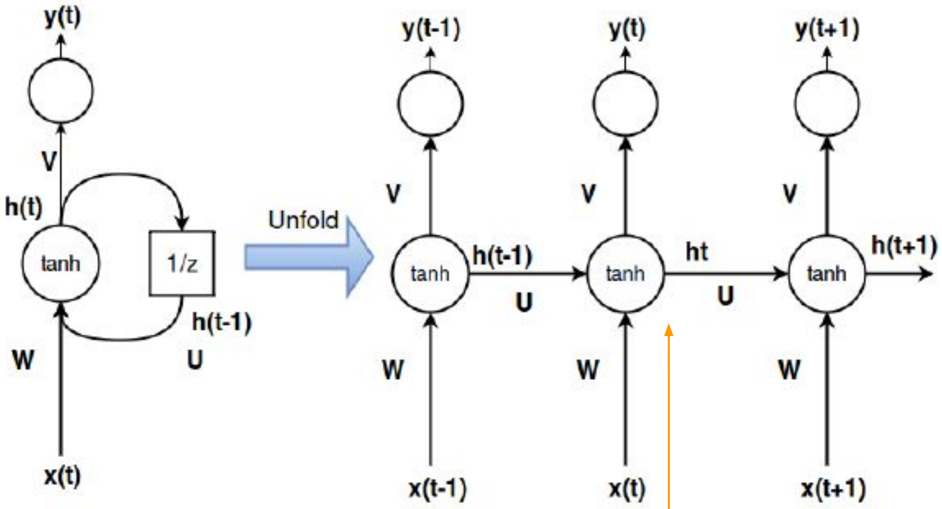
Hence we have two data flows: **Forward in neural layers + time** propagation

BEWARE: We have <u>extra depth</u> now! Every time-step is an extra level of depth (as a deeper stack of layers in a feed-forward fashion!)



Inference with a RNN

Hence we have two data flows: **Forward in layers + time** propagation



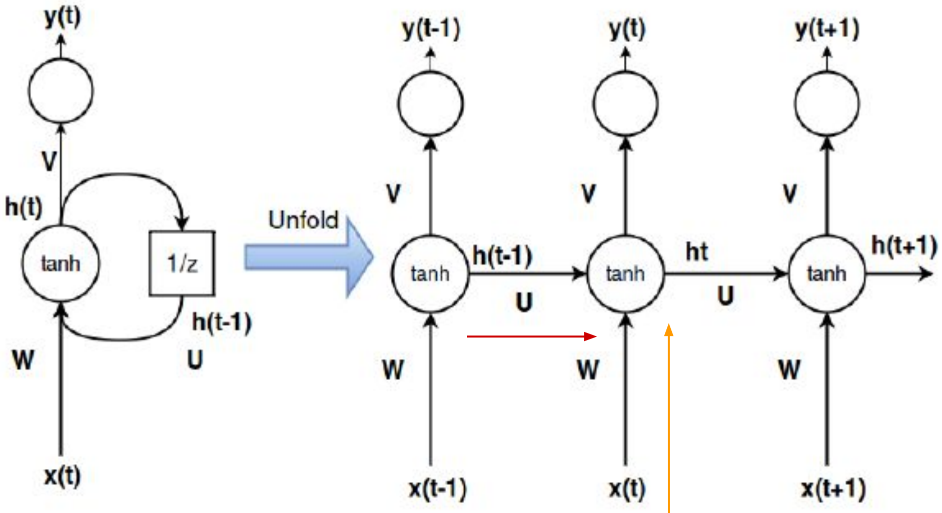
$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$\mathbf{x}_t \in \mathbb{R}^n$

Inference with a RNN

Hence we have two data flows: **Forward in layers + time** propagation

- Last time-step includes the context of our decisions recursively



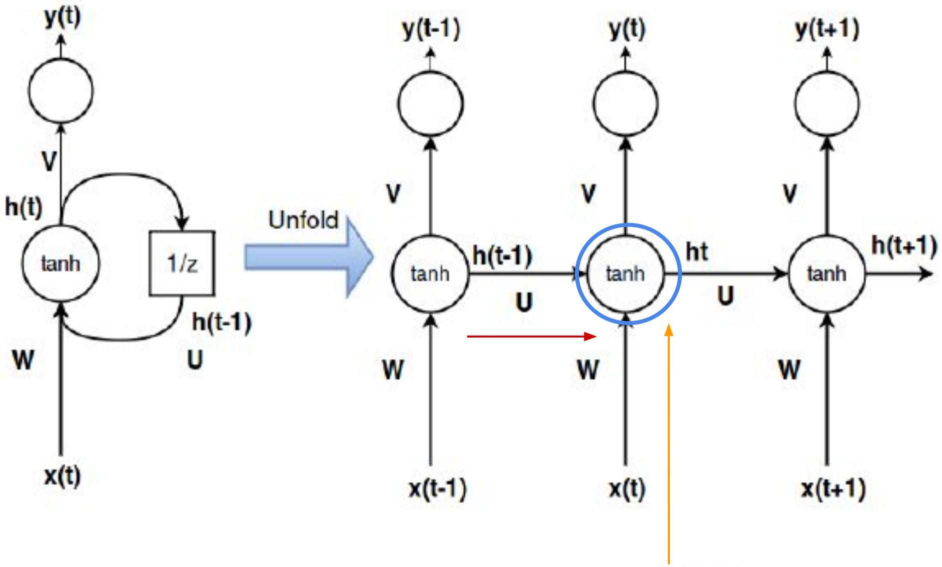
$$\mathbf{x}_t \in \mathbb{R}^n$$

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

Inference with a RNN

Hence we have two data flows: **Forward in layers + time** propagation

- Last time-step includes the context of our decisions recursively



$\mathbf{x}_t \in \mathbb{R}^n$

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

Training a RNN

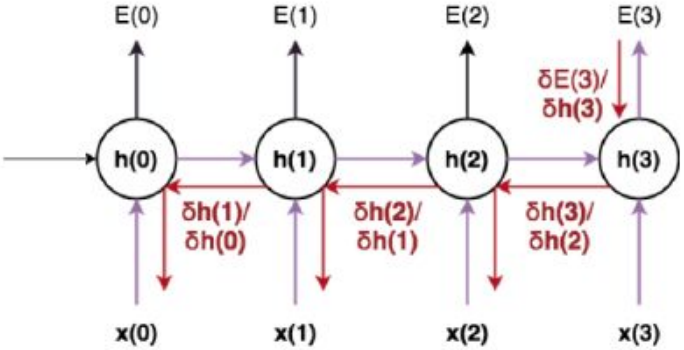
Back Propagation Through Time (BPTT): The training method has to take into account the time operations:

Total error at the output is the sum of errors at each time-step

$$E(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{t=1}^{T} E_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$$

Total gradient is the sum of gradients at each time-step

$$\frac{\partial E}{\partial \mathbf{W}} = \sum_{t=0}^{T-1} \frac{\partial E_t}{\partial \mathbf{W}}$$



Example back-prop in time with 3 time-steps

Training a RNN

Back Propagation Through Time (BPTT): The training method has to take into account the time operations:

Total error at the output is the sum of errors at each time-step

$$E(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{t=1}^{T} E_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$$

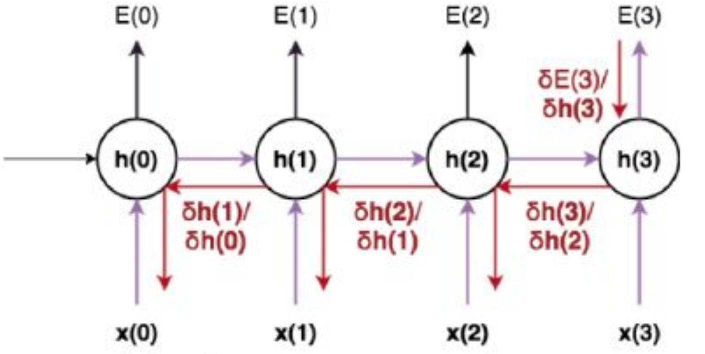
Total gradient is the sum of gradients at each time-step

$$\frac{\partial E}{\partial \mathbf{W}} = \sum_{t=0}^{T-1} \frac{\partial E_t}{\partial \mathbf{W}}$$

T: **max amount of time-steps to do back-prop**. In Keras this is specified when defining the “input shape” to the RNN layer, by means of:
(batch size, sequence length (T), input_dim)

Input shape

3D tensor with shape `(nb_samples, timesteps, input_dim)`.



Example back-prop in time with 3 time-steps

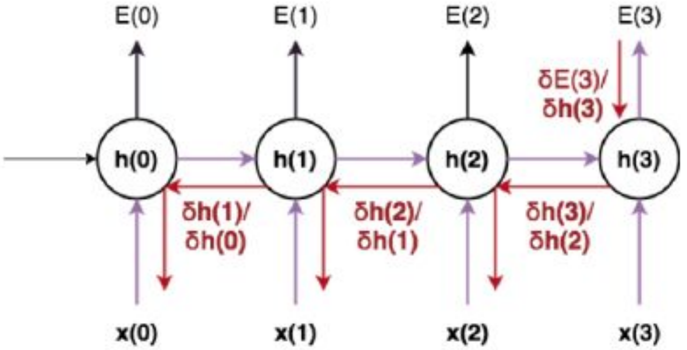
Training a RNN

Main problems:

- **Long-term memory** (remembering quite far time-steps) **vanishes quickly** because of the recursive operation with **U**

$$\mathbf{h}_t = g(\mathbf{W}\cdot\mathbf{x}_t + \mathbf{U}\cdot g(\cdots g(\mathbf{W}\cdot\mathbf{x}_{t-T} + \mathbf{U}\cdot\mathbf{h}_{t-T} + \mathbf{b}_h)\cdots) + \mathbf{b}_h)$$

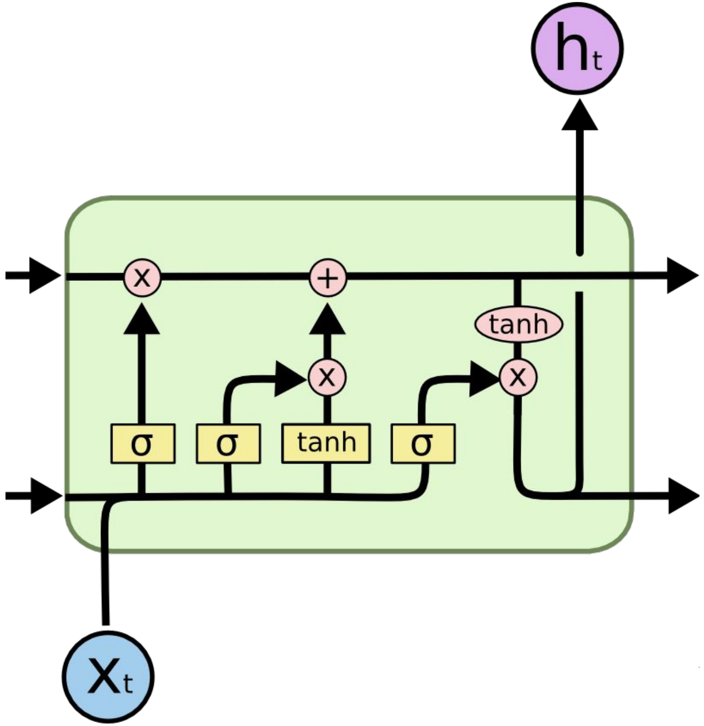
- **During training gradients explode/vanish easily because of depth-in-time** → Exploding/Vanishing gradients!



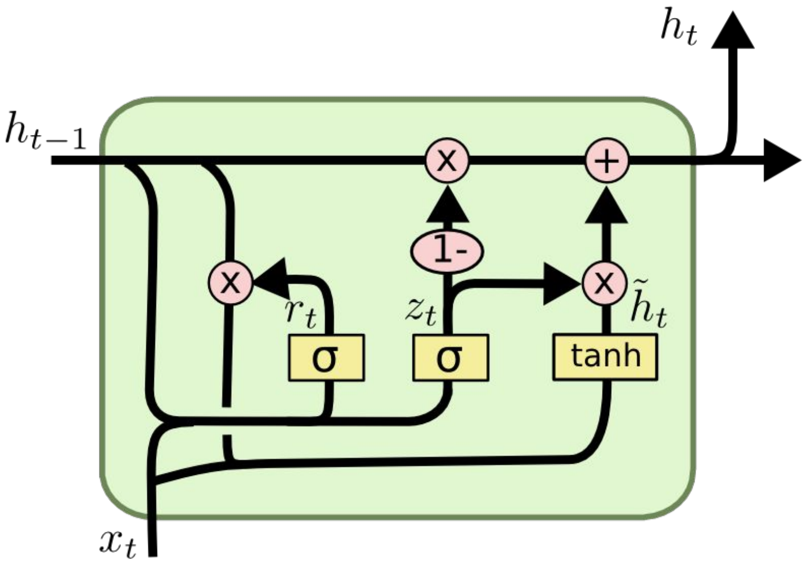
Example back-prop in time with 3 time-steps

Gated RNNs

Gated RNNs



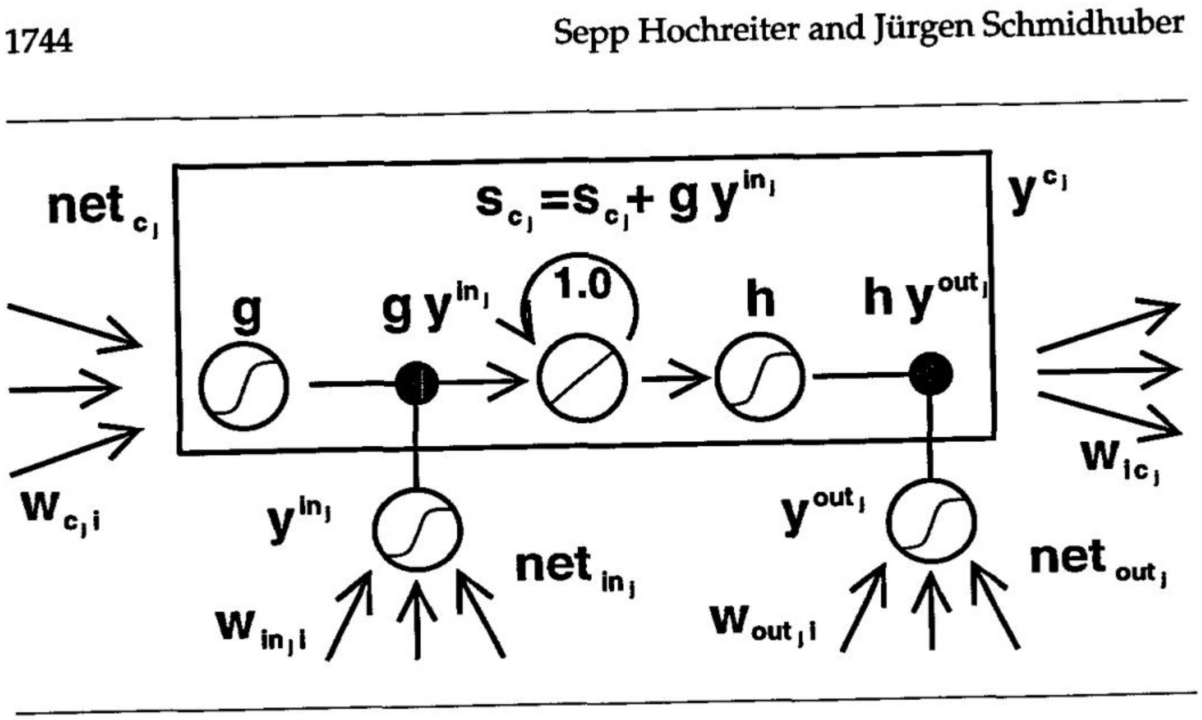
LSTM



GRU

bit.ly/InsightDL2018
#InsightDL2018

Long Short-term Memory (LSTM)



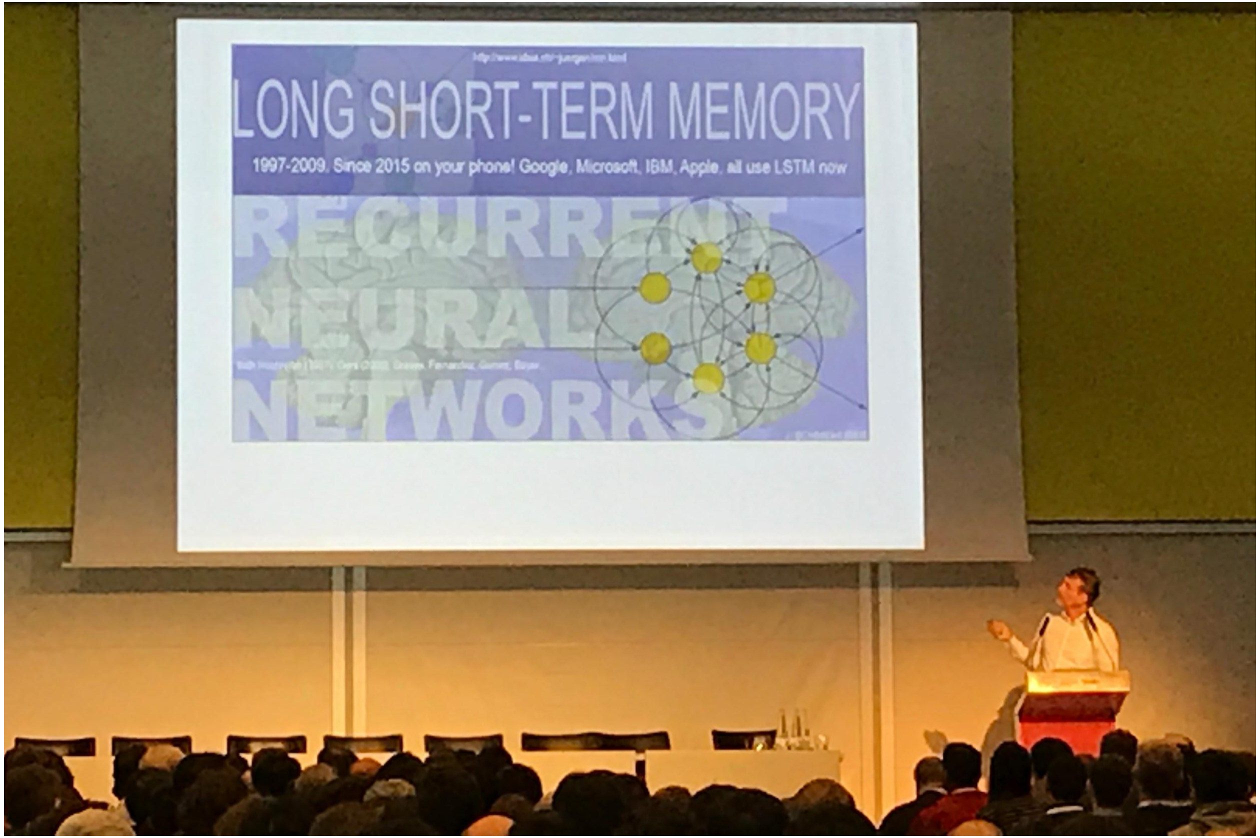
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

Long Short-term Memory (LSTM)

The New York Times, [“When A.I. Matures, It May Call Jürgen Schmidhuber ‘Dad’”] (November 2016)

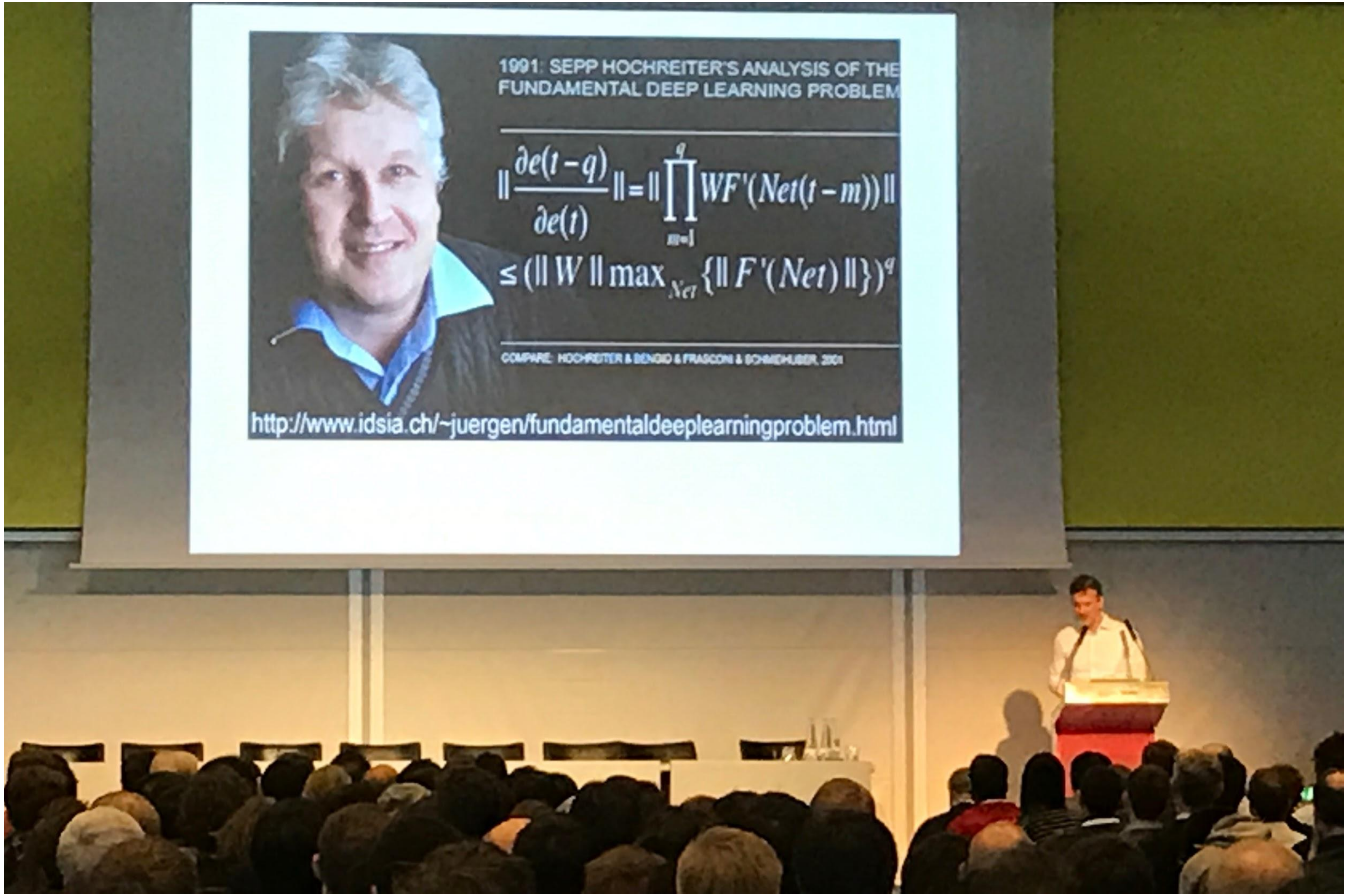
USI/SUPSI
Istituto Dalle Molle di studi sull’intelligenza artificiale
IDSIA

Long Short-term Memory (LSTM)



[Jürgen Schmidhuber]() @ NIPS 2016 Barcelona

Long Short-term Memory (LSTM)



[Jürgen Schmidhuber]() @ NIPS 2016 Barcelona

Long Short-term Memory (LSTM)

[Jürgen Schmidhuber]() @ NIPS 2016 Barcelona

Long Short-term Memory (LSTM)

Solutions:

1. Change the way in which past information is kept → create the notion of **cell state: a memory** unit that keeps long-term information in a safer way by protecting it from recursive operations
2. Make every RNN unit able to **forget whatever may not be useful anymore** by clearing that info from the cell state (optimized clearing mechanism)
3. Make every RNN unit able to decide whether **the current time-step input matters or not**, to accept or discard (optimized reading mechanism)
4. Make every RNN unit able to **output the decisions whenever it is ready to do so** (optimized output mechanism)

Long Short-term Memory (LSTM)

The **cell state** contains the information and is only modified by simple linear operations at each time step.

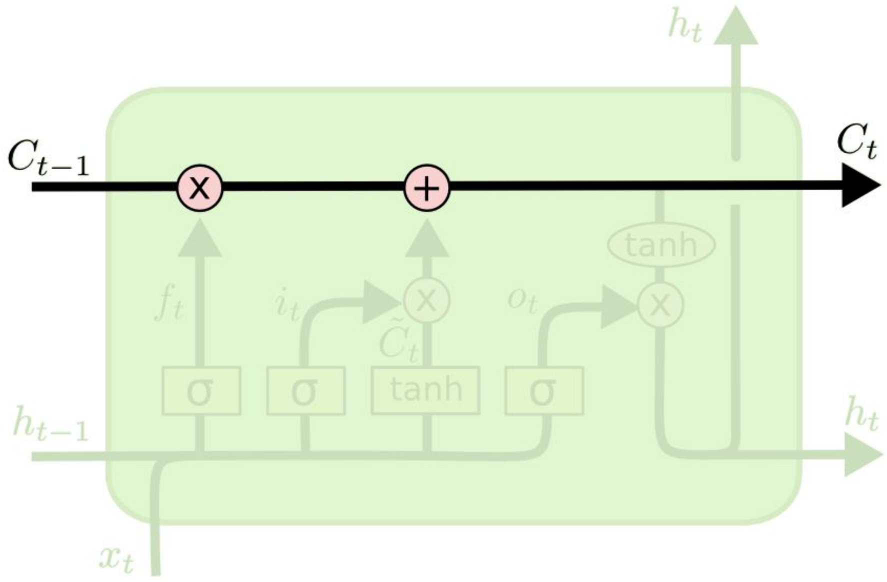


Figure: Cristopher Olah, [“Understanding LSTM Networks”](#) (2015)

Long Short-term Memory (LSTM)

Three **gates** are governed by *sigmoid* units (btw [0,1]) define the control of in & out information with a product.

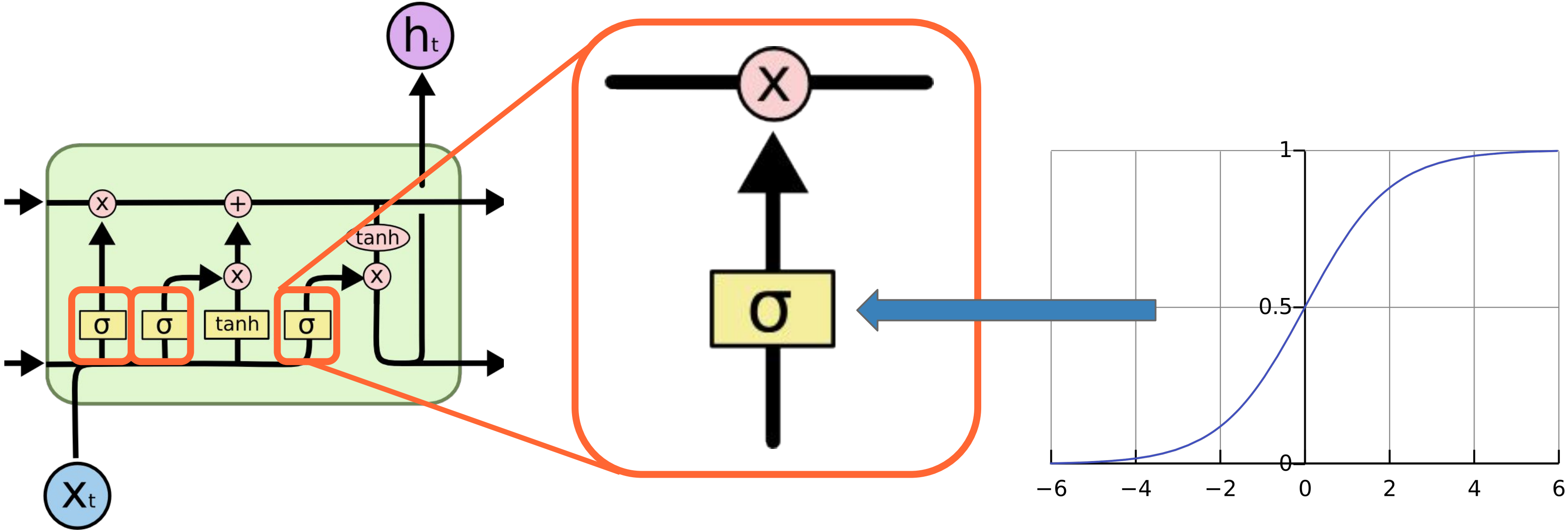
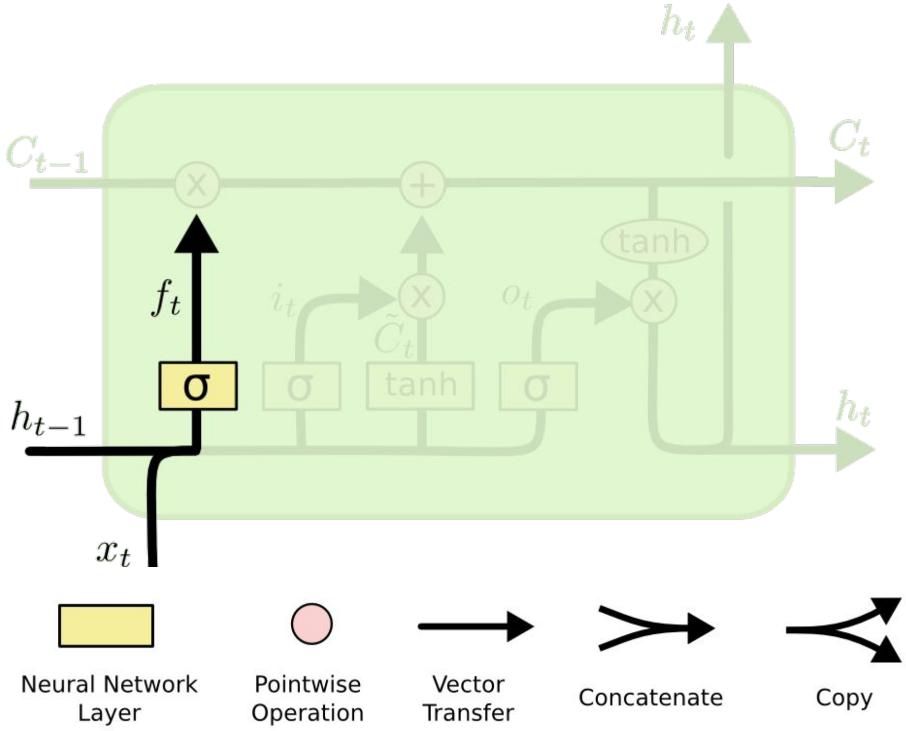


Figure: Cristopher Olah, "Understanding LSTM Networks" (2015)

Long Short-term Memory (LSTM)

Make every RNN unit able to **forget whatever may not be useful anymore** by clearing that info from the cell state (optimized clearing mechanism)



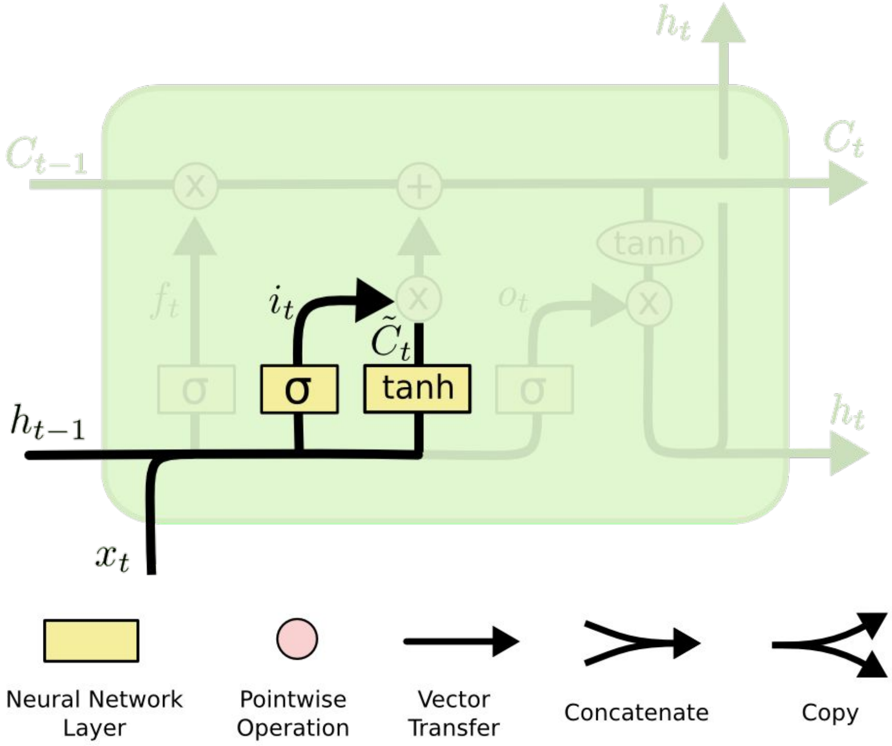
Forget Gate:

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

Figure: Cristopher Olah, “Understanding LSTM Networks” (2015) / Slide: Alberto Montes

Long Short-term Memory (LSTM)

Make every RNN unit able to decide whether **the current time-step information matters or not**, to accept or discard (optimized reading mechanism)



Input Gate Layer

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

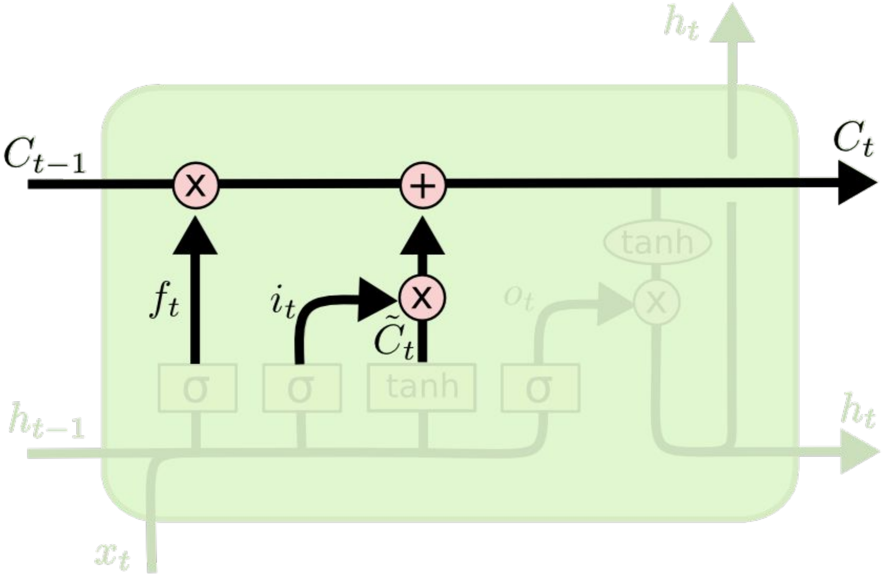
New candidate values

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

Figure: Cristopher Olah, “Understanding LSTM Networks” (2015) / Slide: Alberto Montes

Long Short-term Memory (LSTM)

Make every RNN unit able to decide whether **the current time-step information matters or not**, to accept or discard (optimized reading mechanism)



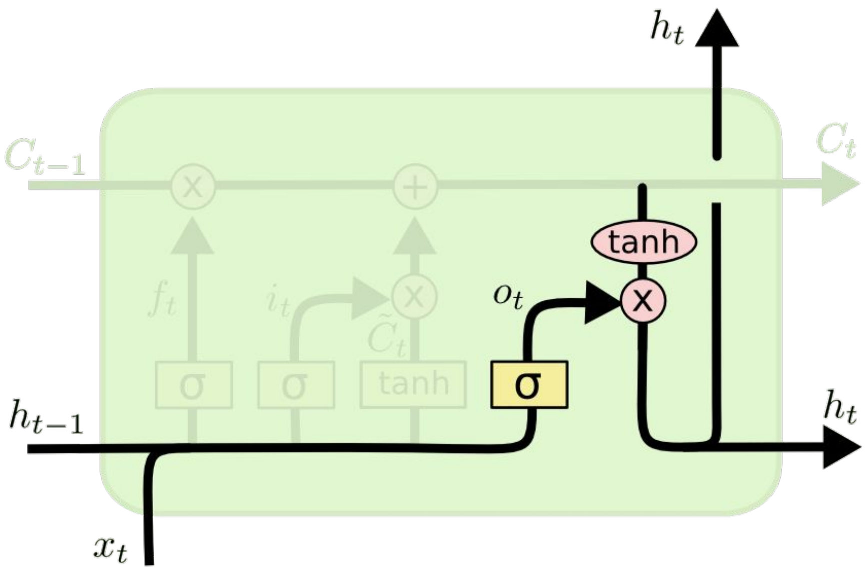
Forget + Input Gates = Update Cell State (memory):

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

Figure: Cristopher Olah, "Understanding LSTM Networks" (2015) / Slide: Alberto Montes

Long Short-term Memory (LSTM)

Make every RNN unit able to **output the decisions whenever it is ready to do so** (optimized output mechanism)



Output Gate Layer

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

Output to next layer & timestep

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

Figure: Cristopher Olah, "Understanding LSTM Networks" (2015) / Slide: Alberto Montes

Long Short-term Memory (LSTM)

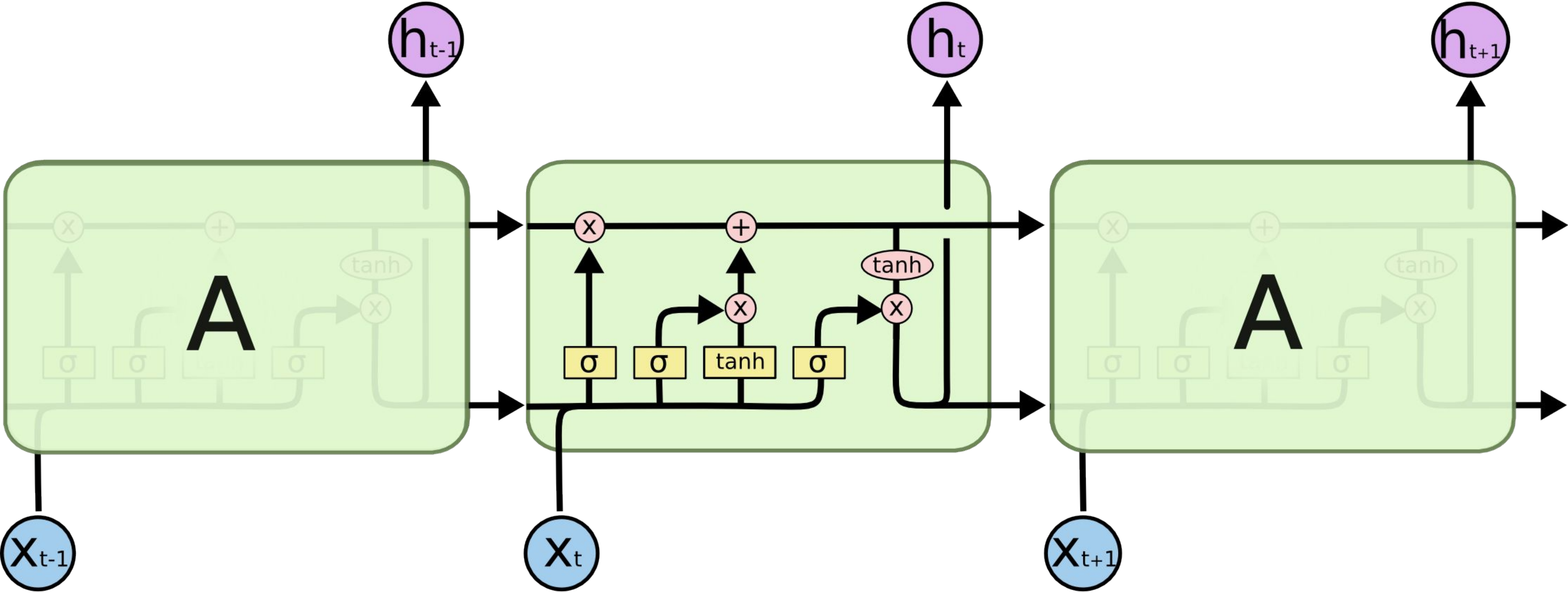
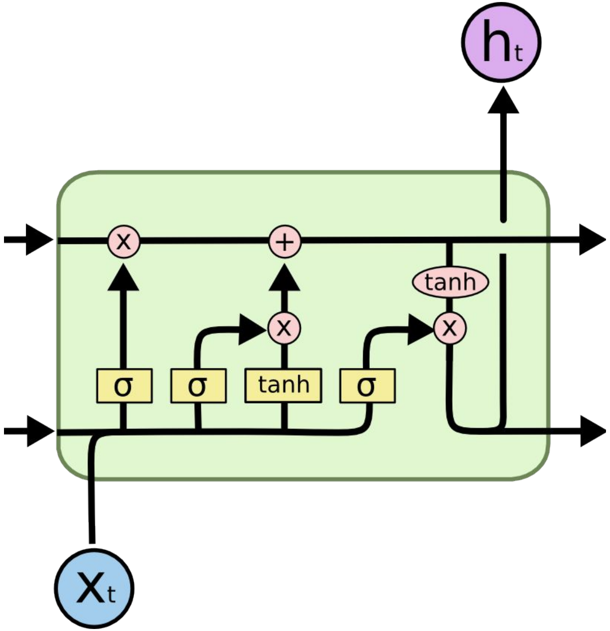


Figure: Cristopher Olah, "Understanding LSTM Networks" (2015)

Long Short-term Memory (LSTM)



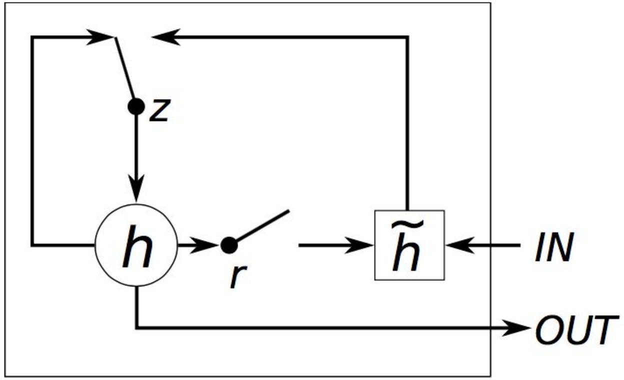
Compared to a non-gated RNN, an LSTM has four times more parameters because of the additional neurons that govern the gates:

$$N^i_{params} = 4 \times (N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units})$$

3 gates + input activation

Gated Recurrent Unit (GRU)

GRU obtain a similar performance as LSTM with one gate less.



$$u_i = \sigma\left(W^{(u)}x_i + U^{(u)}h_{i-1} + b^{(u)}\right) \quad (1)$$

$$r_i = \sigma\left(W^{(r)}x_i + U^{(r)}h_{i-1} + b^{(r)}\right) \quad (2)$$

$$\tilde{h}_i = \tanh\left(Wx_i + r_i \circ Uh_{i-1} + b^{(h)}\right) \quad (3)$$

$$h_i = u_i \circ \tilde{h}_i + (1 - u_i) \circ h_{i-1} \quad (4)$$

$$N^i_{params} = 3 \times (N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units})$$

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." AMNLP 2014.

Questions?