Recurrent Neural Networks and LSTM

Anna Corazza
AIPA, DIETI, Università di Napoli Federico II

May 17, 2019

Outline

- Time representation: Recurrent Neural Networks (RNN).
- Most effective RNN: Long Short Term Memory (LSTM).
- seq2seq in machine translation.

Time representation

- Several problems require to process **sequences**:
  - Speech and natural language processing (NLP).
  - Videos.
  - Time series.

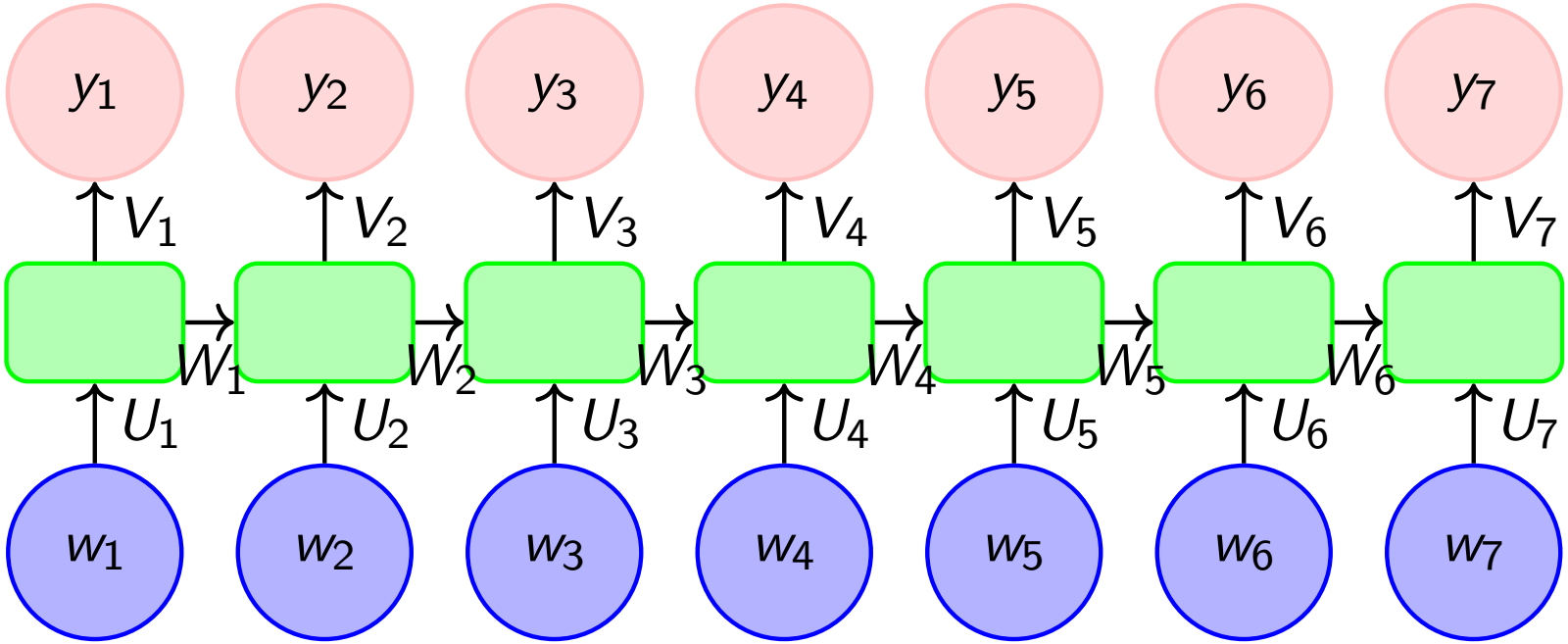
Time representation

- Several problems require to process **sequences**:
  - Speech and natural language processing (NLP).
  - Videos.
  - Time series.
- We can represent the sequence with a vector: we use **space** as a metaphor for time.

Time representation

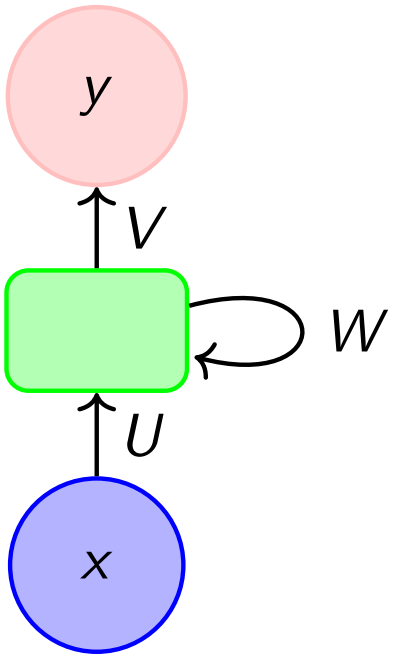
- Several problems require to process **sequences**:
  - Speech and natural language processing (NLP).
  - Videos.
  - Time series.
- We can represent the sequence with a vector: we use **space** as a metaphor for time.
- Problems with this representation:
  1. Implies **buffering**, with connected problems, eg.: how should a system know when a buffer's content should be examined?
  2. Rigid **limit** to duration, problematic, for example, in NLP.
  3. Difficult to distinguish **relative** temporal position from **absolute** temporal position.

Representing sequences as vectors



- Parameters are different for every position.
- Every word is represented by a vector.

Recurrent Neural Network



$$
\begin{aligned}
c_t &= f_W(c_{t-1}, x_t) \\
\text{e.g.} & \\
c_t &= tanh(Wc_{t-1} + Ux_t) \\
y_t &= f_V(c_t)
\end{aligned}
$$

- Parameters are independent of the position.
- Every word is again represented by a vector.

Simple Recurrent Networks

- Two variants of 3-layer RNNs: **Elman** networks and **Jordan** Networks.
- **Elman network**:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h \mathbf{h_{t-1}} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned}$$

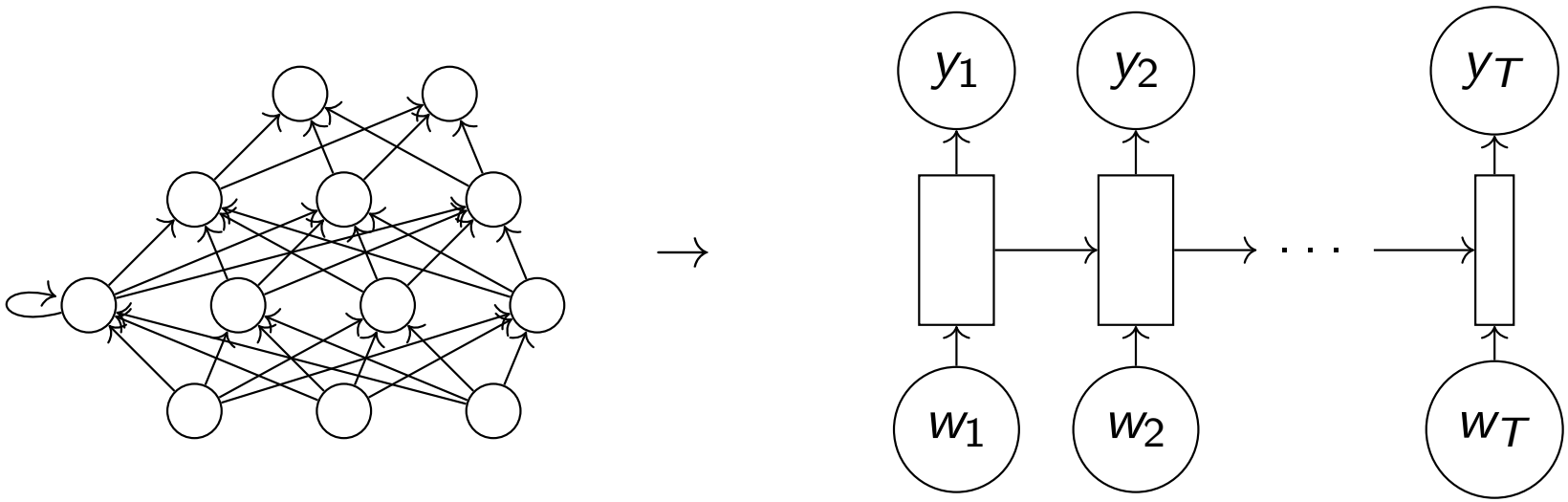
- **Jordan network**:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h \mathbf{y_{t-1}} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned}$$

- σ_h and σ_y are **activation** functions.

Recurrent Neural Network

Front view **Side view**



$\otimes$ time rotation 90^o $\longrightarrow$ time

Discussion

- They can model **dependencies from the past**, in principle in a **unlimited** way.

Discussion

- They can model **dependencies from the past**, in principle in a **unlimited** way.
- During training gradients may **explode** or **vanish** because of temporal depth.

Discussion

- They can model **dependencies from the past**, in principle in a **unlimited** way.
- During training gradients may **explode** or **vanish** because of temporal depth.
-

$$h_t = g\left(W \cdot x_t + U \cdot g\left(\ldots + U \cdot g\left(W \cdot x_T + U h_{t-T} + b_h\right)\ldots\right) + b_h\right)$$

Long term memory **vanishes** because of the nested multiplication by U.

Discussion

- They can model **dependencies from the past**, in principle in a **unlimited** way.
- During training gradients may **explode** or **vanish** because of temporal depth.
-

$$h_t = g\left(W \cdot x_t + U \cdot g\left(\ldots + U \cdot g\left(W \cdot x_T + U h_{t-T} + b_h\right)\ldots\right) + b_h\right)$$

  Long term memory **vanishes** because of the nested multiplication by U.
- Dependencies from unlimited past can be **very useful** or **very confusing**.

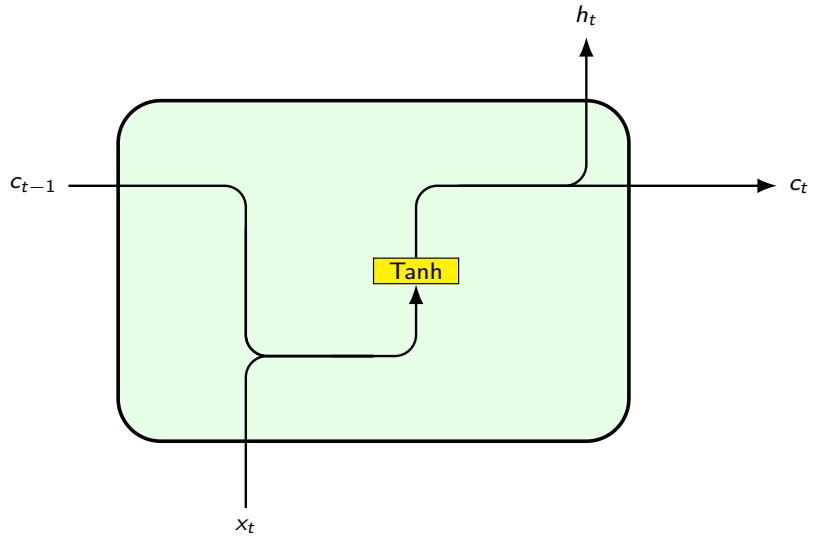
Discussion

- They can model **dependencies from the past**, in principle in a **unlimited** way.
- During training gradients may **explode** or **vanish** because of temporal depth.
-

$$h_t = g\left(W \cdot x_t + U \cdot g\left(\ldots + U \cdot g\left(W \cdot x_T + U h_{t-T} + b_h\right)\ldots\right) + b_h\right)$$

  Long term memory **vanishes** because of the nested multiplication by U.
- Dependencies from unlimited past can be **very useful** or **very confusing**.
- We need a model able to **focus** on the information which is important to our task.

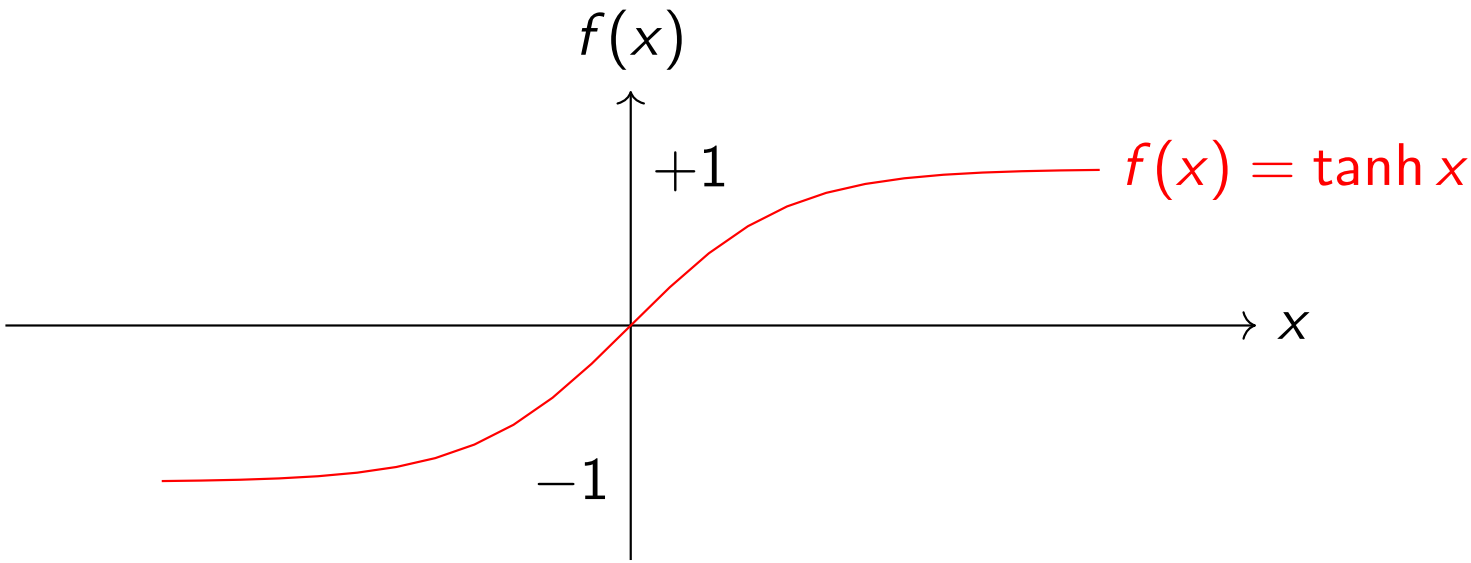
Standard recurrent network



tanh: neural network layer

x_t and c_{t-1}: concatenation

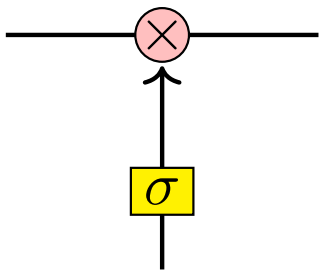
h_t and c_t: copy



Gates

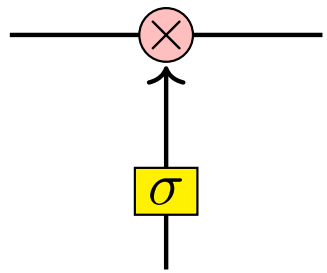
- Gates are a way to **optionally** let information through.

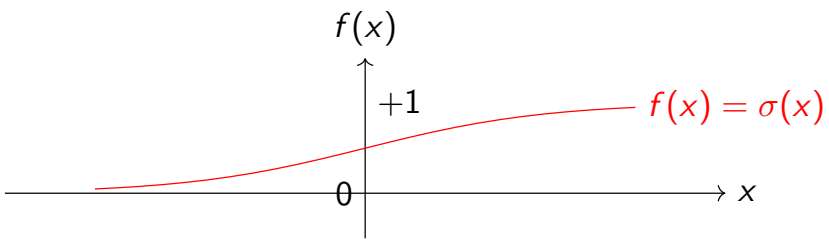
Gates

- Gates are a way to **optionally** let information through.
- 

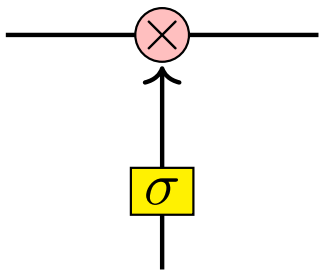
pink circles: pointwise operators

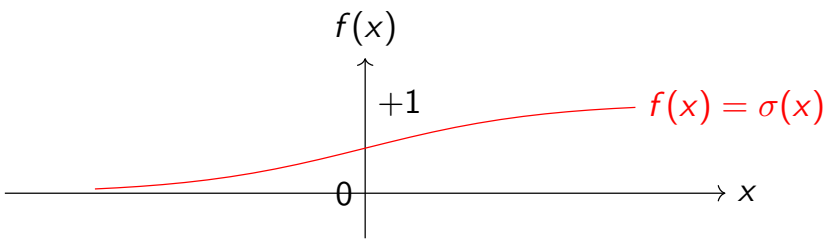
Gates

- Gates are a way to **optionally** let information through.
- 

pink circles: pointwise operators
- 

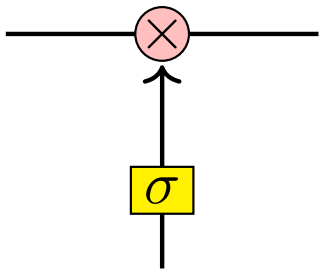
Gates

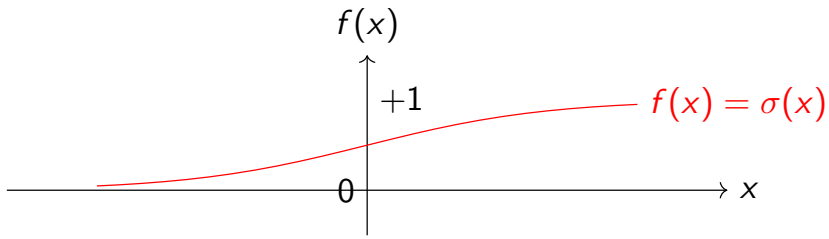
- Gates are a way to **optionally** let information through.
- 

pink circles: pointwise operators
- 

- $\tanh(x) \in [-1, 1]$

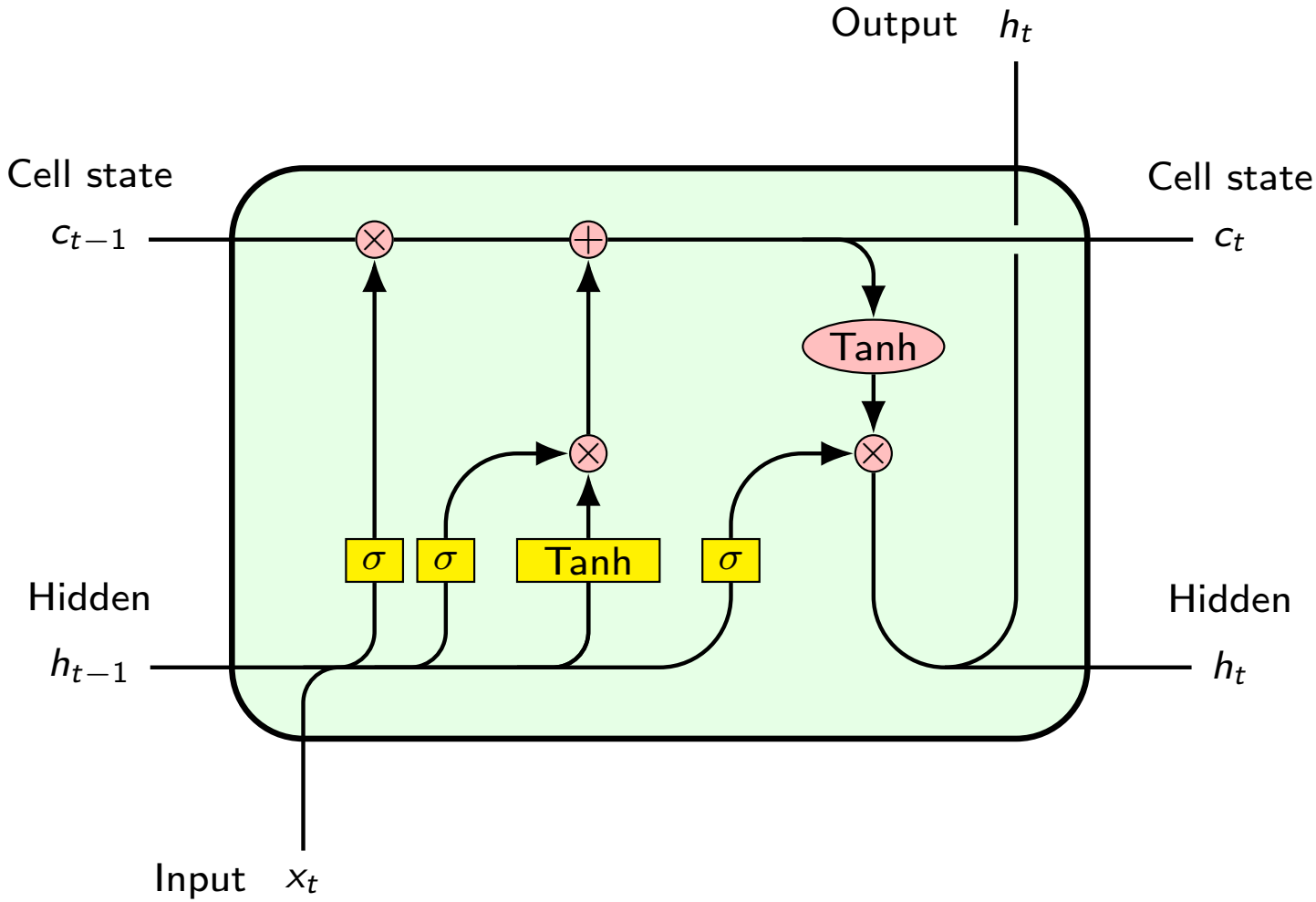
Gates

- Gates are a way to **optionally** let information through.
- 

pink circles: pointwise operators
- 

- $\tanh(x) \in [-1, 1]$
- $\sigma(x) \in [0, 1]$

Long Short Term Memory (LSTM)



Christopher Olah, "Understanding LSTM Networks" by `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Gates

Forget gate Whether to erase.

Gates

Forget gate Whether to erase.

Input gate Whether to write.

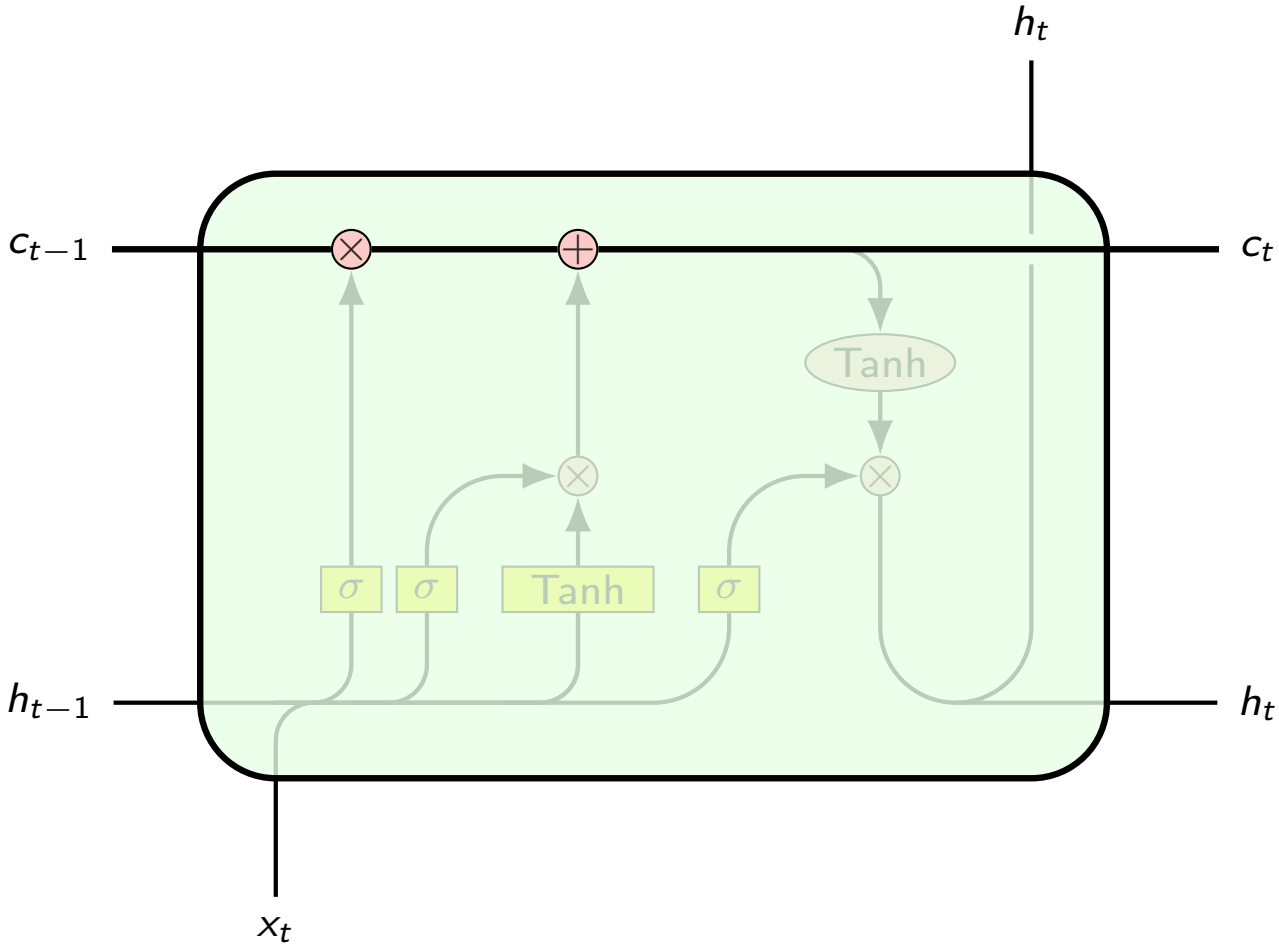
Gates

Forget gate	Whether to erase.
Input gate	Whether to write.
Update gate	How much to write.

Gates

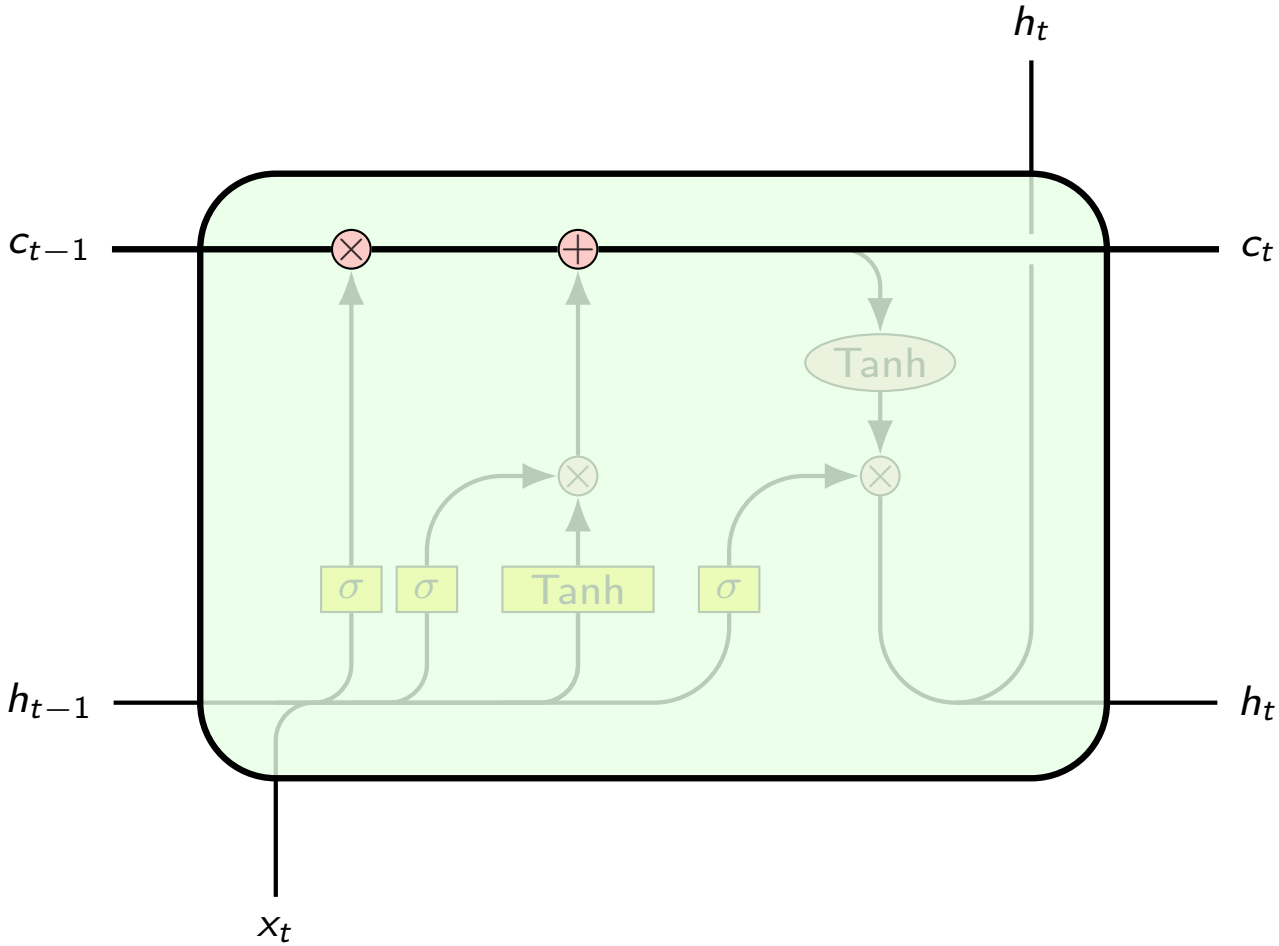
Forget gate	Whether to erase.
Input gate	Whether to write.
Update gate	How much to write.
Output gate	How much to reveal.

The cell state



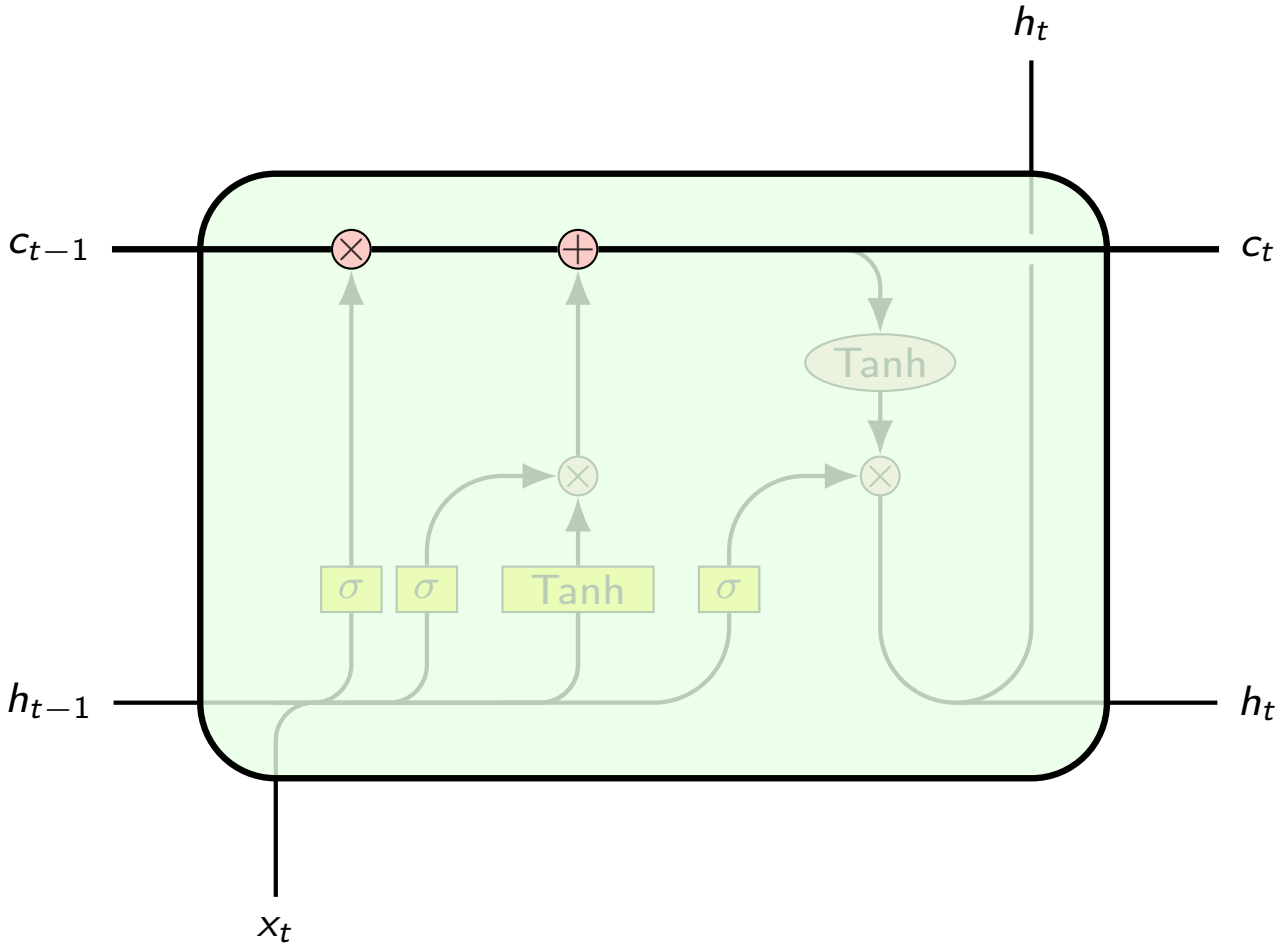
- Information could just flow along it unchanged.

The cell state



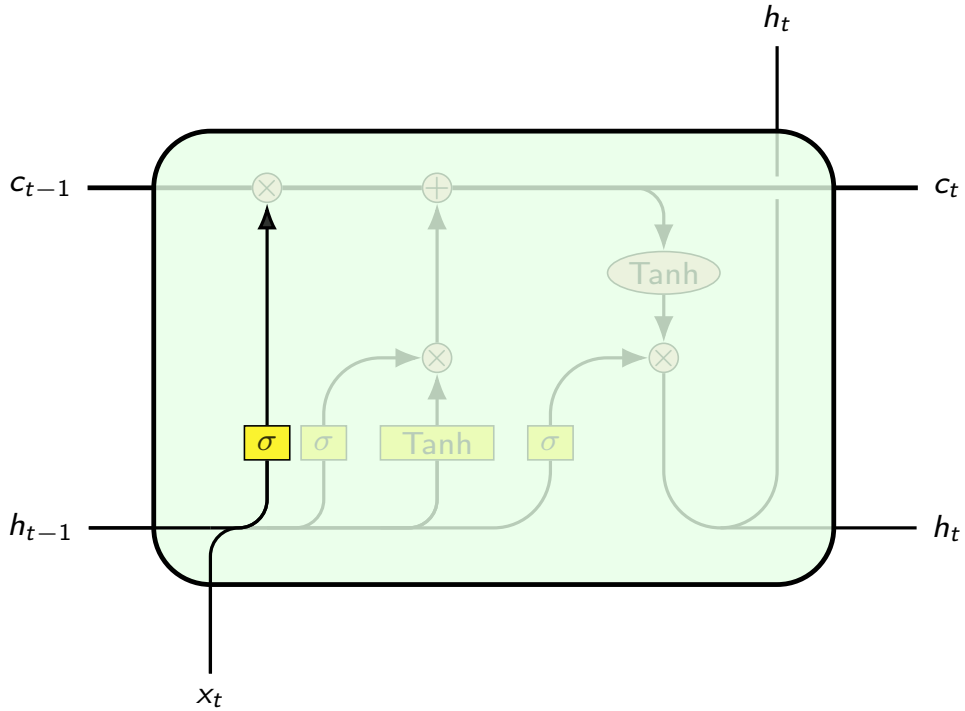
- Information could just flow along it unchanged.
- Information can be removed or added from the cell state.

The cell state



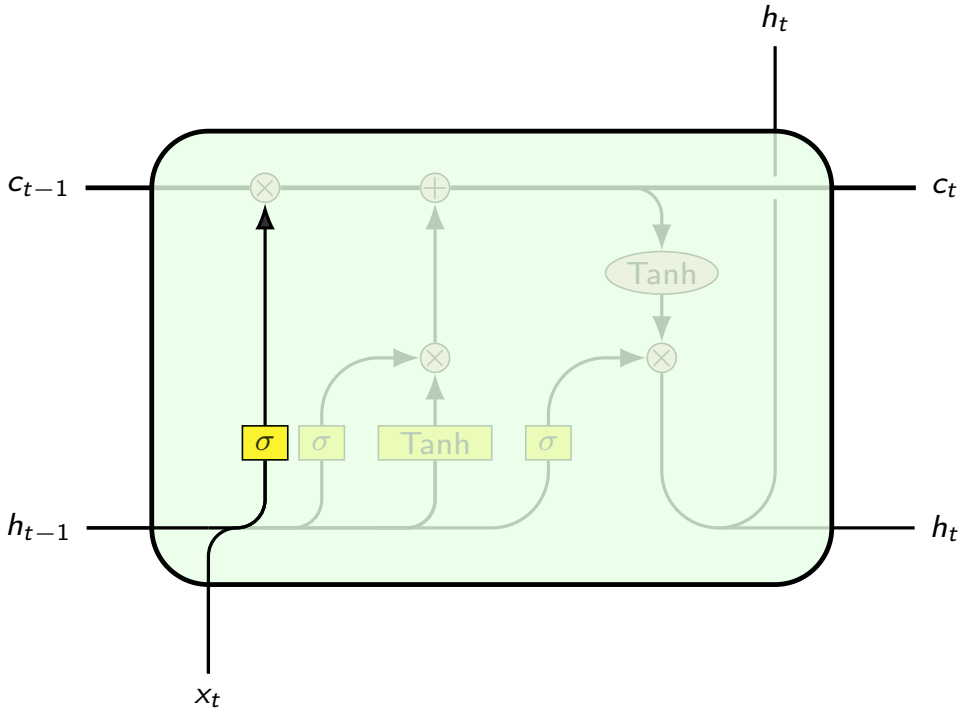
- Information could just flow along it unchanged.
- Information can be removed or added from the cell state.
- Regulation provided by **gates**.

The forget gate layer



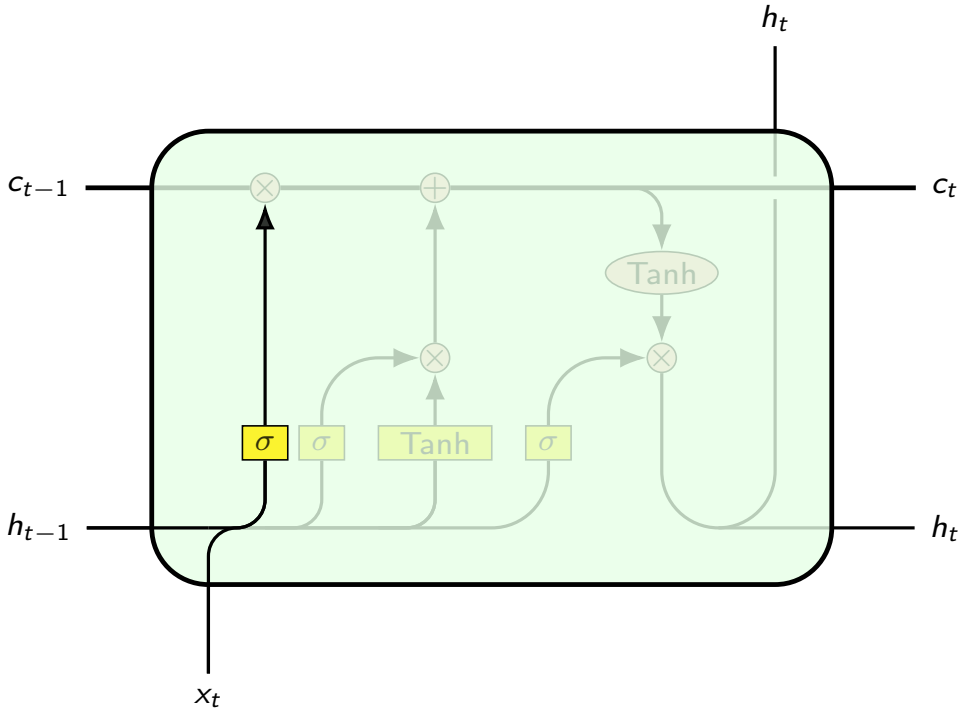
- h_{t-1} and x_t decide how much information in the cell state must be kept.

The forget gate layer



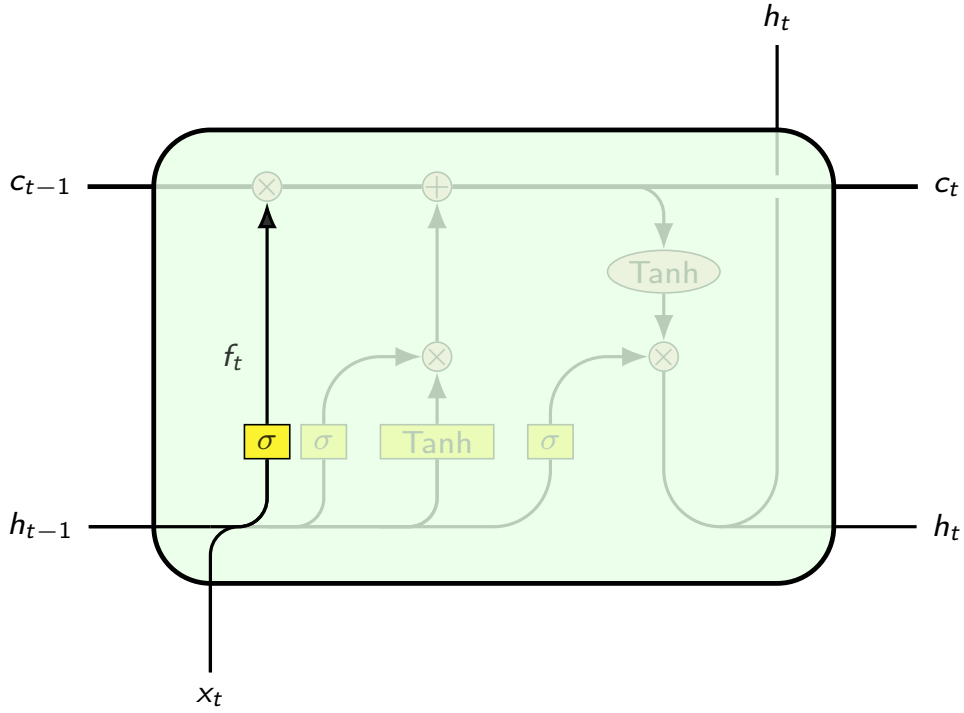
- h_{t-1} and x_t decide how much information in the cell state must be kept.
- The output of the forget gate is a number in $[0, 1]$: 1 means "completely keep", 0 "completely delete".

The forget gate layer



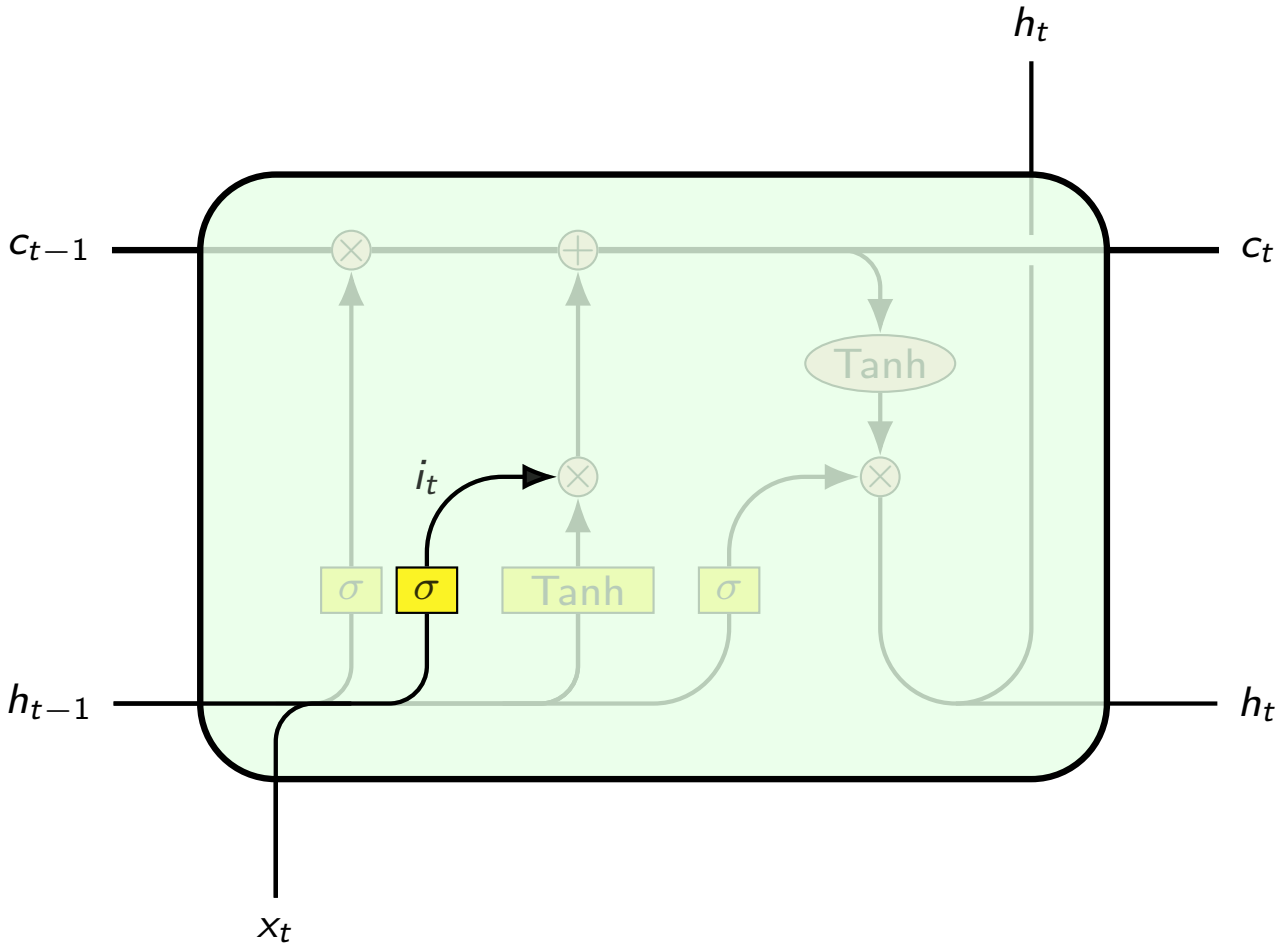
- h_{t-1} and x_t decide how much information in the cell state must be kept.
- The output of the forget gate is a number in $[0, 1]$: 1 means "completely keep", 0 "completely delete".
- It multiplies every element in the cell state.

The forget gate layer (2)



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

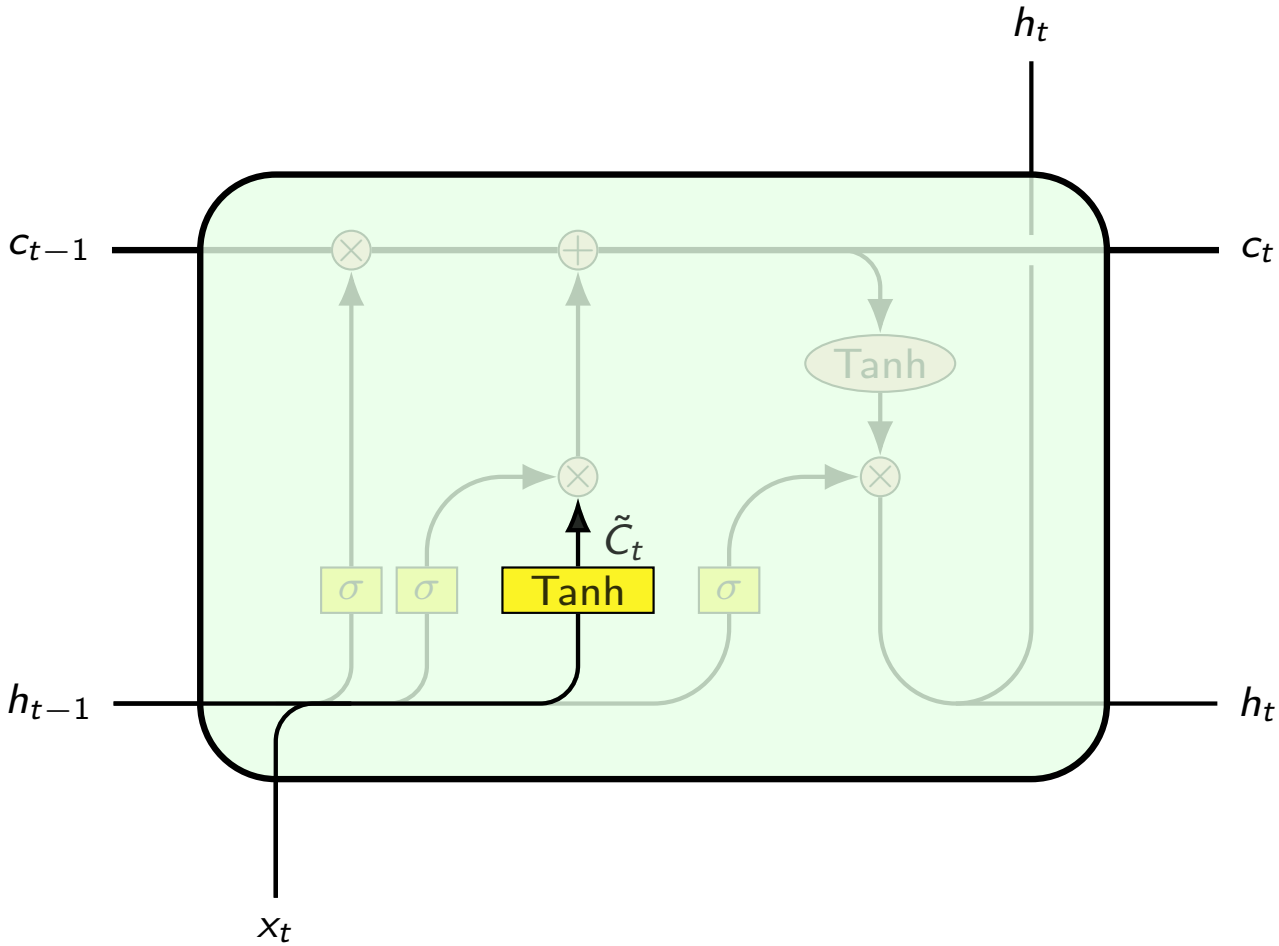
The input gate layer



- To decide **which** cell state values must be updated.
-

$$i_t = \sigma\left(W_i[h_{t-1}, x_t] + b_i\right) \tag{2}$$

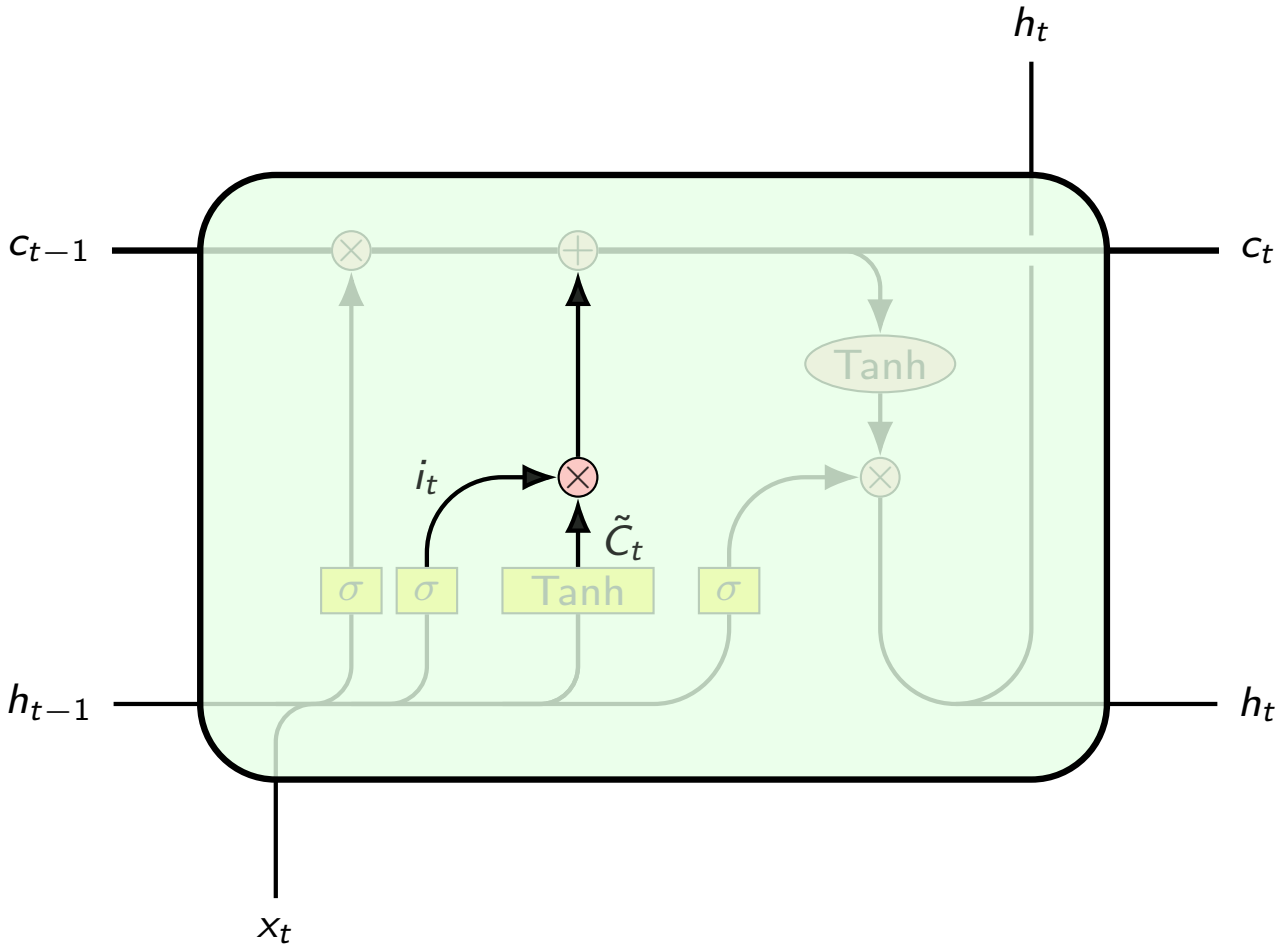
The update gate layer



- Tanh to decide a vector of new candidates for the cell state.
-

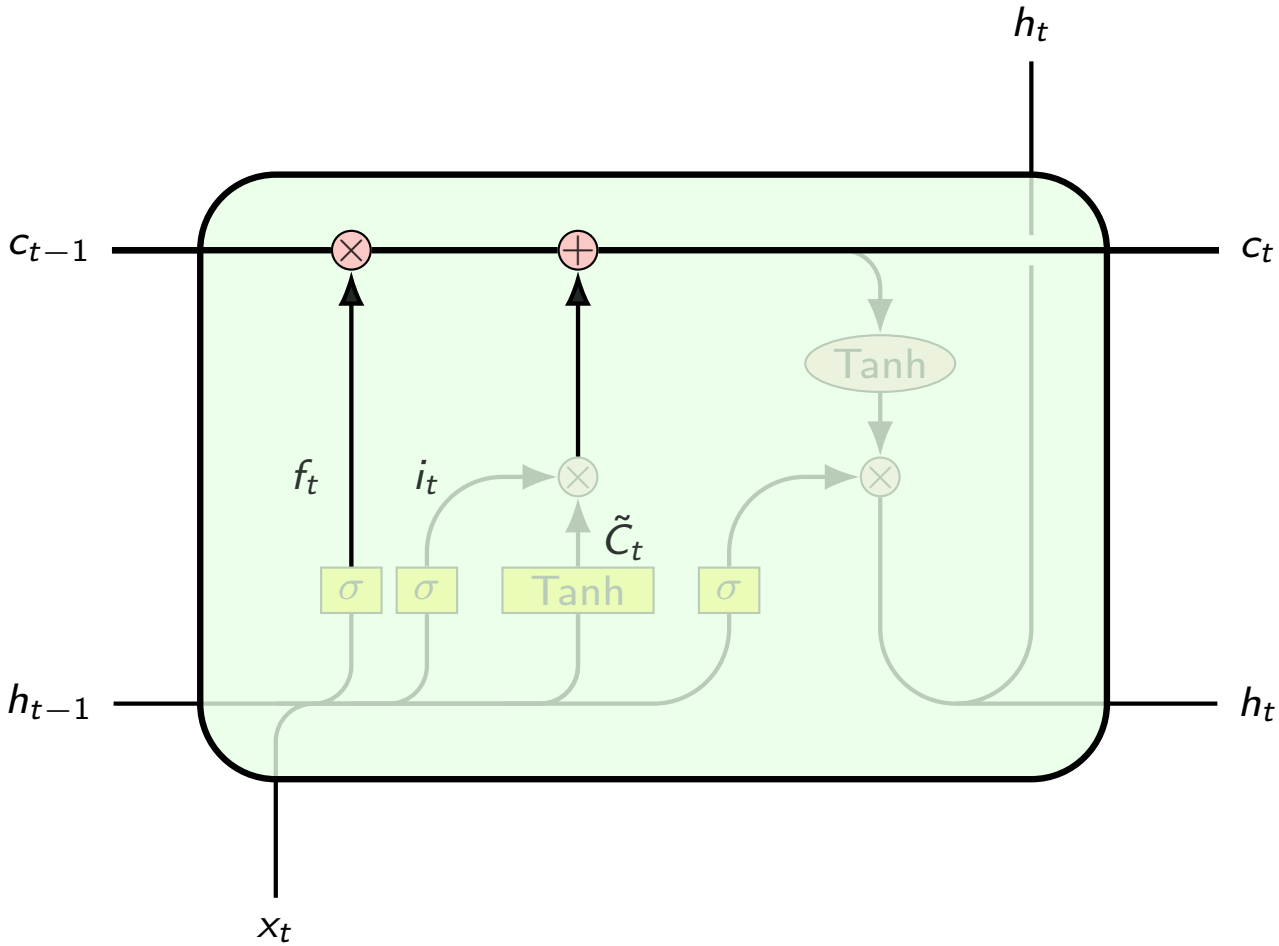
$$\tilde{C}_t = \tanh\left(W_C[h_{t-1}, x_t] + b_C\right) \tag{3}$$

Input and update gate layers combination



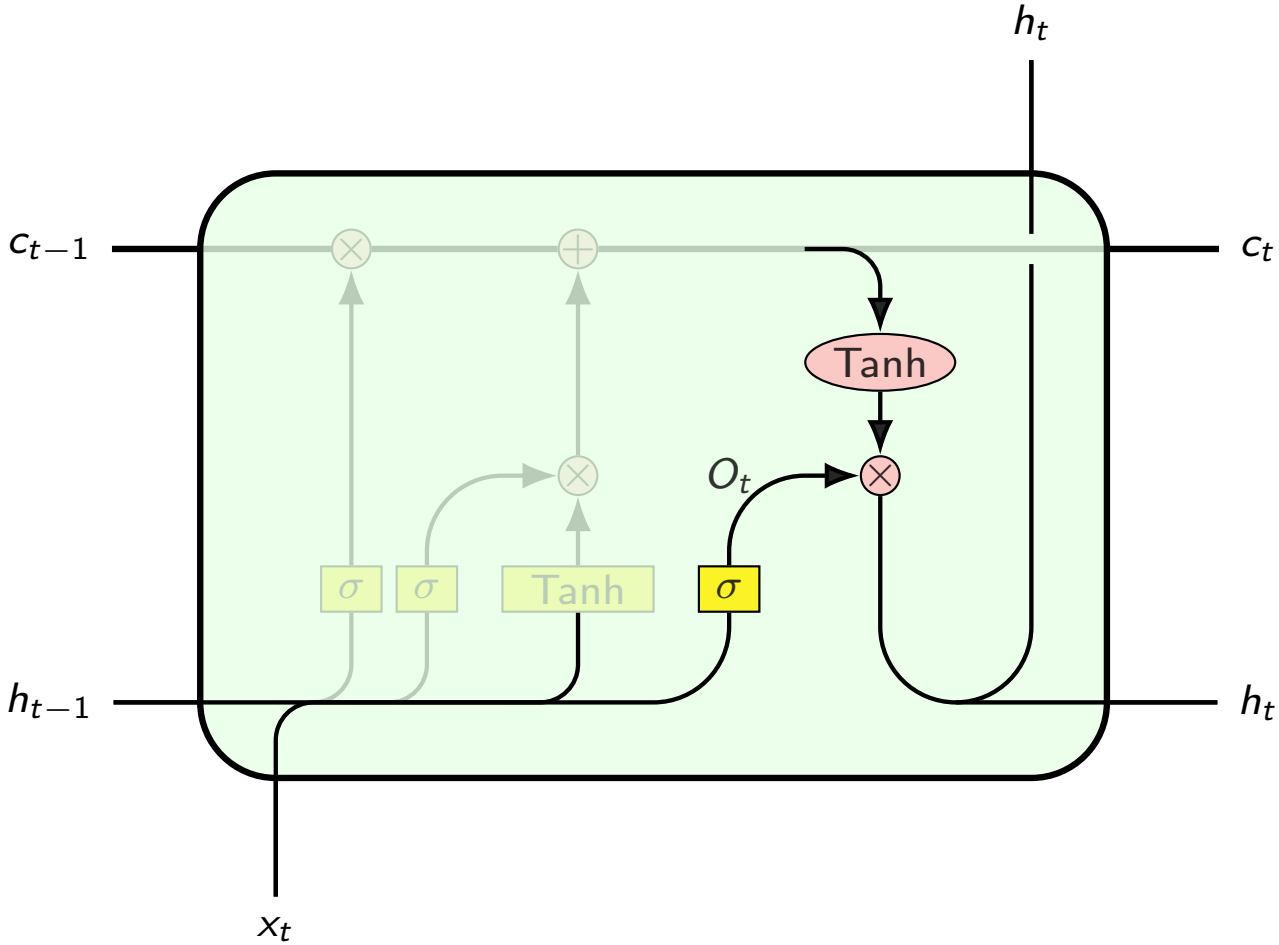
- Among the possible candidates, only the ones which will be updated are chosen.
- $$\tilde{C}_t \times i_t$$

Updating the cell state



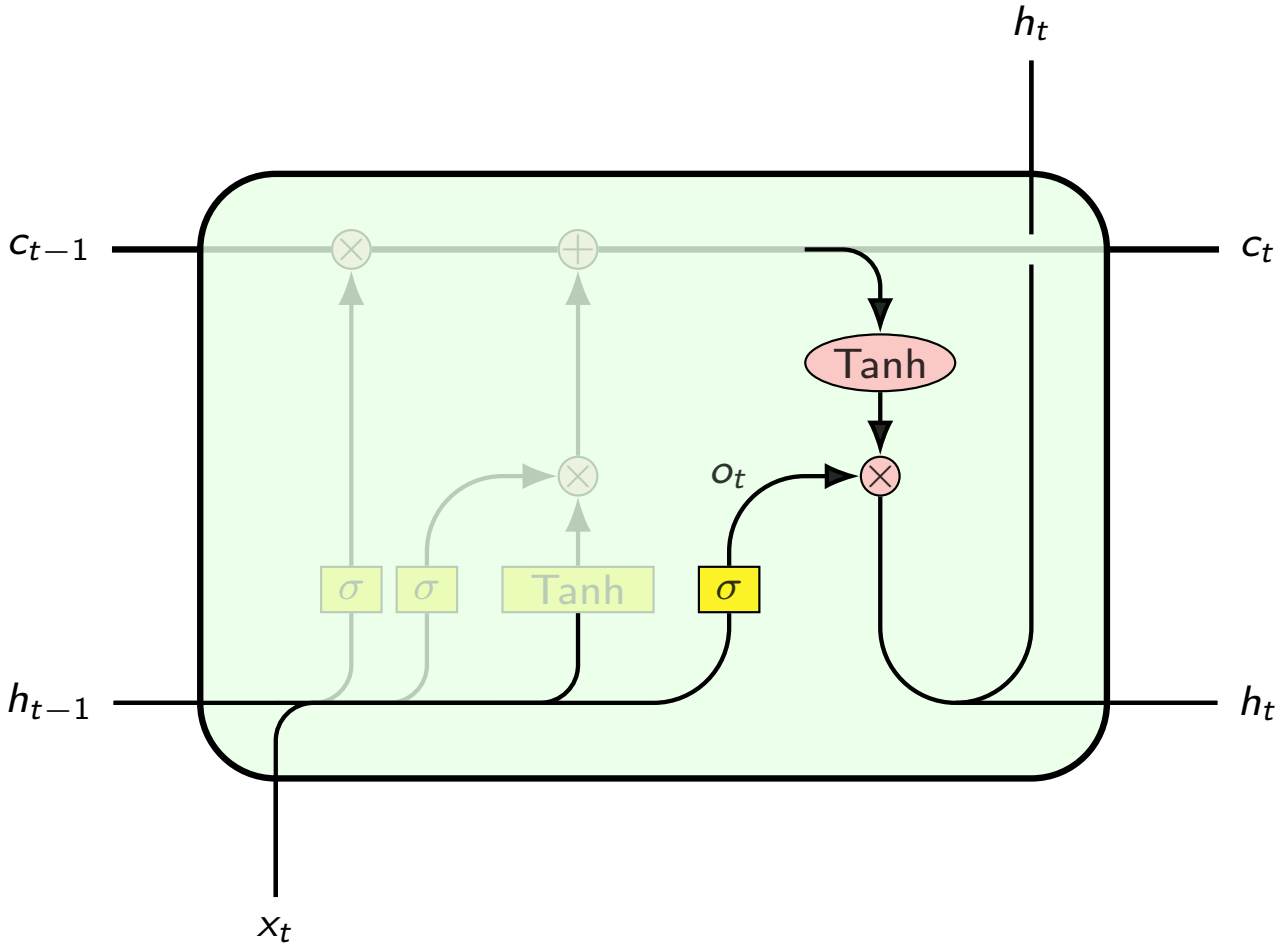
- Multiplication for the forget gate output.
- New cell state is then added.
- $C_t = f = t \times C_{t-1} + i_t \times \tilde{C}_t$

Output gate



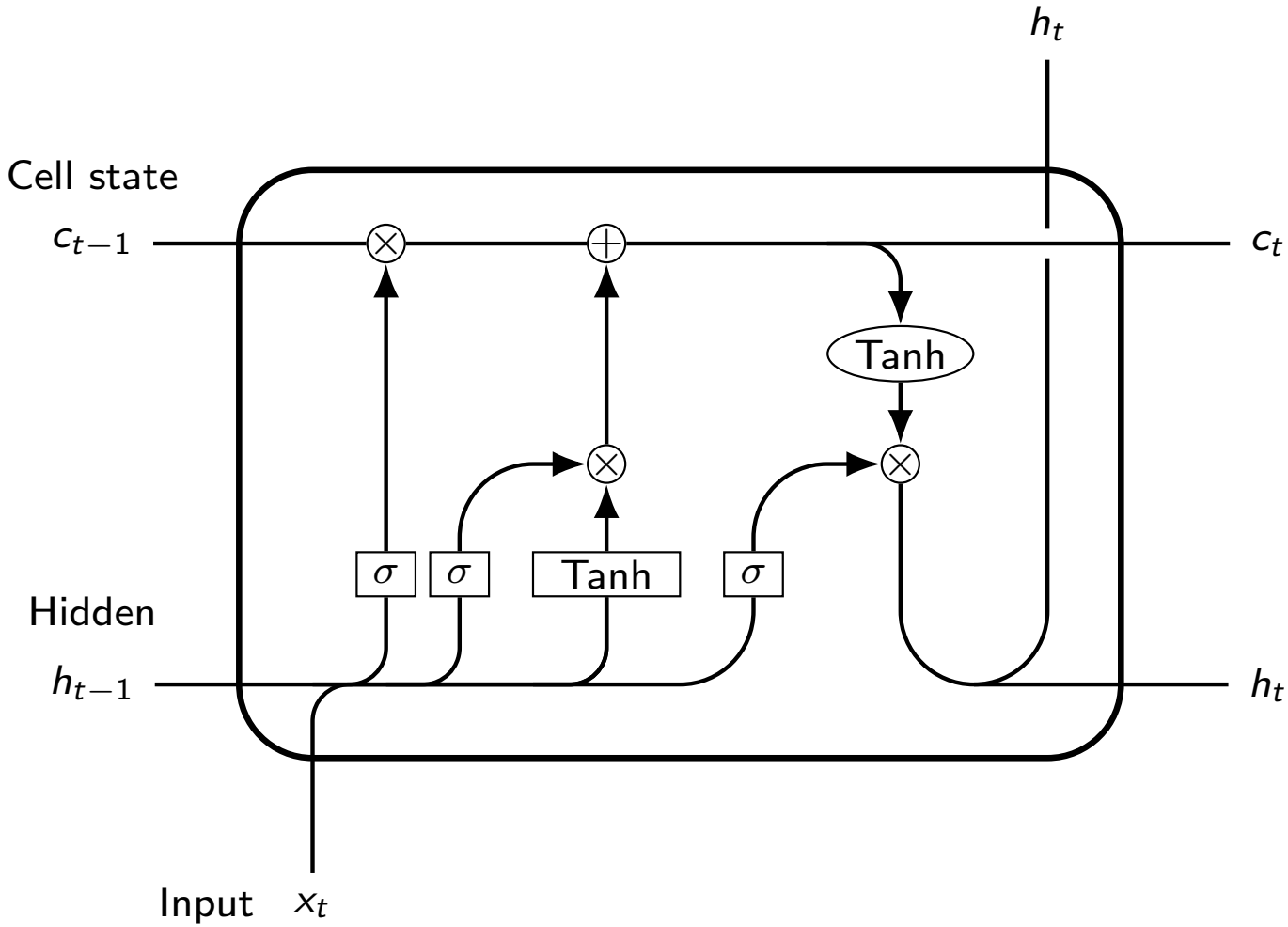
- The operator tanh reports the cell state in $[-1, +1]$.
- The sigmoid layer decides which values can be output.
- Their multiplication decides the actual output.

Output gate (2)



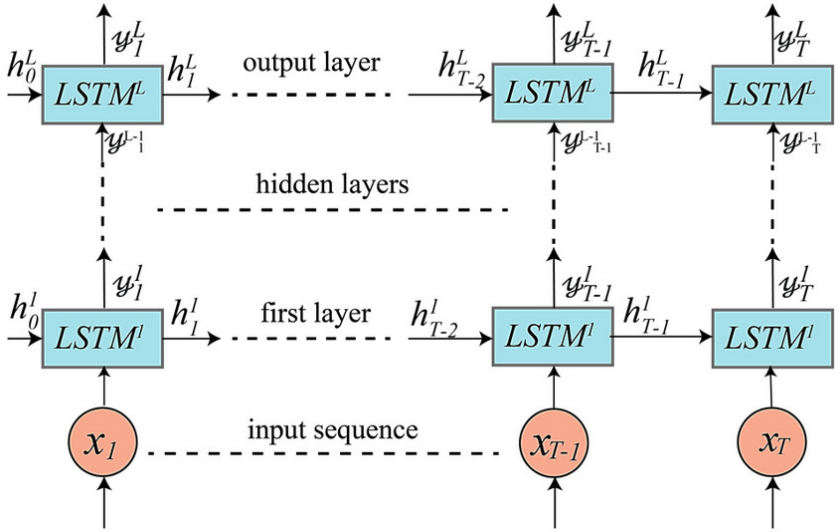
$$
\begin{aligned}
o_t &= \sigma\left(W_o[h_{t-1,x_t}] + b_o\right) \\
h_t &= o_t \times \tanh(C_t)
\end{aligned}
$$

All in all


h_t
Cell state
c_{t-1}
c_t
Tanh
σ
σ
Tanh
σ
Hidden
h_{t-1}
h_t
Input x_t

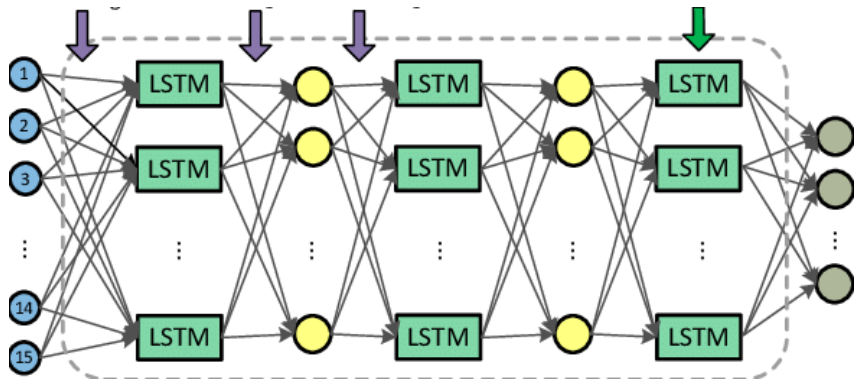
All in all (2)

Of course, this is just a brick in much larger architectures:



Unidirectional LSTM-based DRNN model consisting of an input layer, several hidden layers, and an output layer. The number of hidden layers is a hyperparameter that is tuned during training.

The proposed deep LSTM network with three LSTM layers and two feedforward layers. For clarity, the temporal recurrent structure is not shown.



(Two examples randomly taken from web)

Seq2Seq models

- Both input and output are sequences.
- A typical seq2seq model has 2 major components:
  1. an **encoder**;
  2. a **decoder**.
- Both are essentially two different RNNs combined into one network.
- Applications for seq2seq include:
  1. Machine translation.
  2. Speech Recognition.
  3. Named entities and Subject Extraction.
  4. Text Summarization.
  5. Question Answering.
- I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks". NIPS 2014

Machine translation

Encoder : an embedding layer + an LSTM layer.

Decoder : an LSTM layer + a dense layer.

"A Must-Read NLP Tutorial on Neural Machine Translation The Technique Powering Google Translate", by Prateek Joshi, January 31, 2019. `https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/`