UniGe | MaLGa

Recurrent Neural Networks

Deep Learning: A hands-on introduction

A course offered in the PhD program in Computer Science and Systems Engineering

Nicoletta Noceti

Machine Learning Genoa Center – University of Genoa

Sources of inspiration:

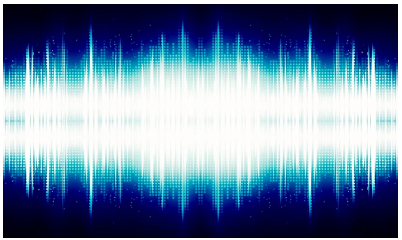
https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L14_intro-rnn/L14_intro-rnn-part1_slides.pdf

https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

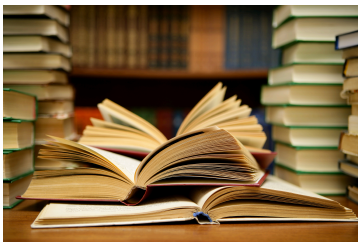
http://introtodeeplearning.com/

Dealing with sequential data

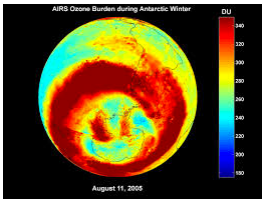
- Today we consider data in the form of sequences or time series



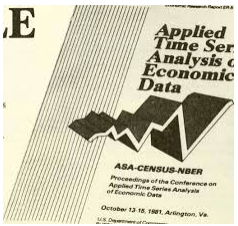
Audio



Text



IoT



An example: predicting the next word

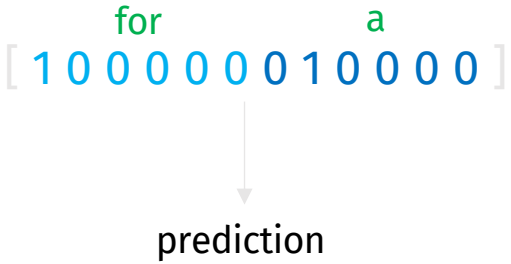
“This morning I took my dog for a walk”

- Predicting the last word in the sentence given all or some of the previous words
- Let’s reason on possible solutions

Solution 1

"This morning I took my dog for a walk"

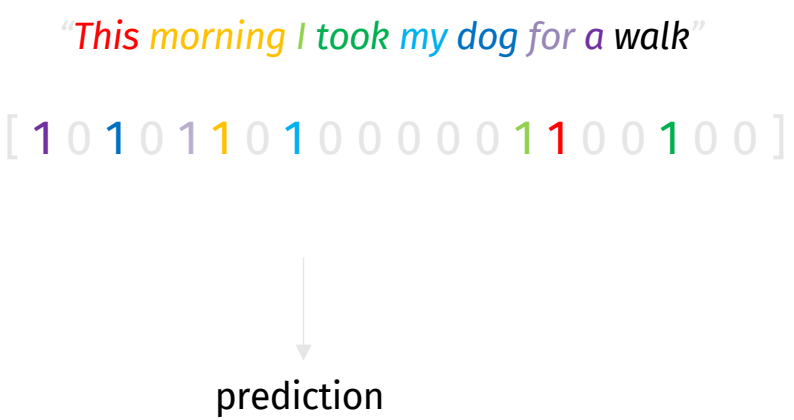
- Using small fixed windows and one-hot encoding



Problem: what if the few words are not informative?
e.g. *"Italy is where I grew up but now I live in London. I speak fluent _____"*
Longer-term dependences may be needed!

Solution 2

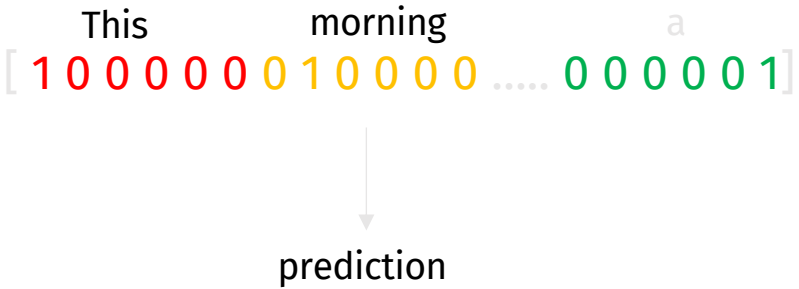
- Using all the available words to form an embedding



Problem: order is not preserved
e.g. *"The food was good, not bad at all" vs "The food was bad, not good at all"*

Solution 3

- Going back to solution 1 with larger windows



Problem: no parameter sharing, and the knowledge we may derive does not transfer is words appear elsewhere

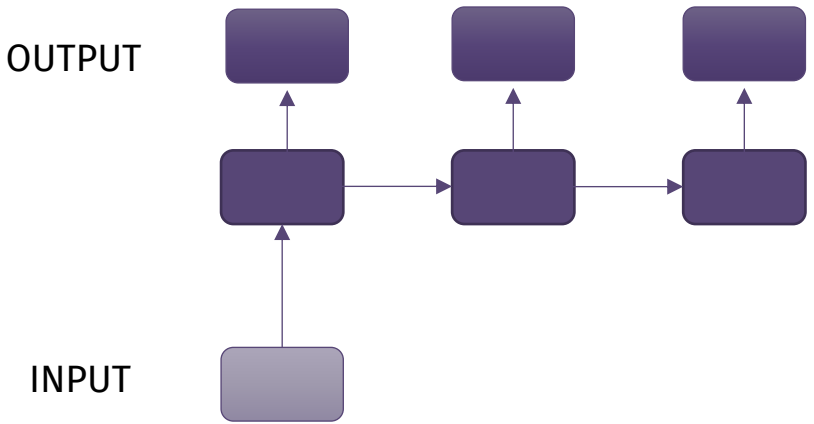
How to approach sequence modelling

- Handling sequences of **different lengths**
- Taking into account **short** and **long term** dependences
- Considering **order** between elements
- **Sharing parameters** across the sequence

Different formulations of the problem

One-to-may: the input is in a standard format (not a sequence!), the output is a sequence

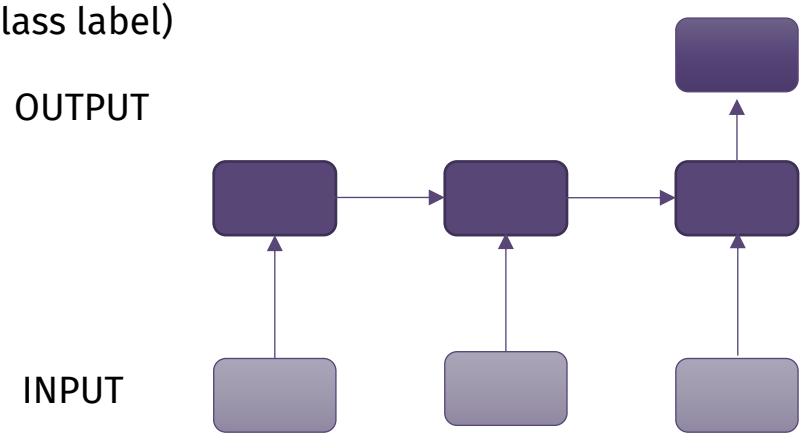
Example of applications: image captioning (input: image, output: text describing the image content)



Different formulations of the problem

Many-to-one: the input is a sequence, the output is a fixed-size vector (not a sequence!)

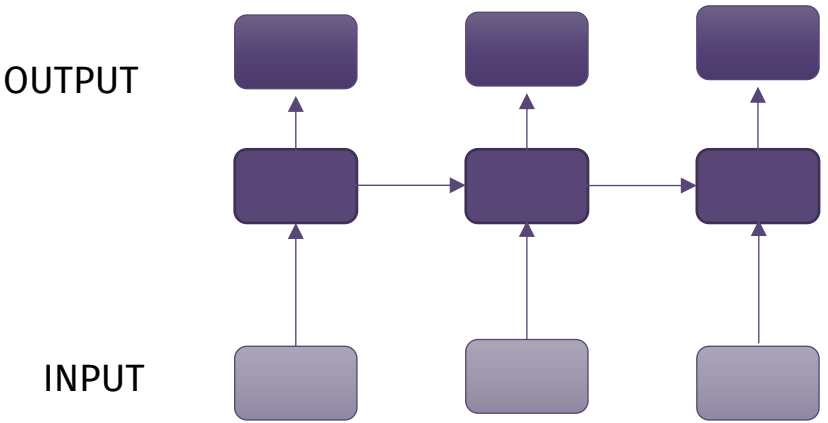
Example of applications: sentiment analysis (input: text, output: class label)



Different formulations of the problem

Direct Many-to-many: input and output are both sequences

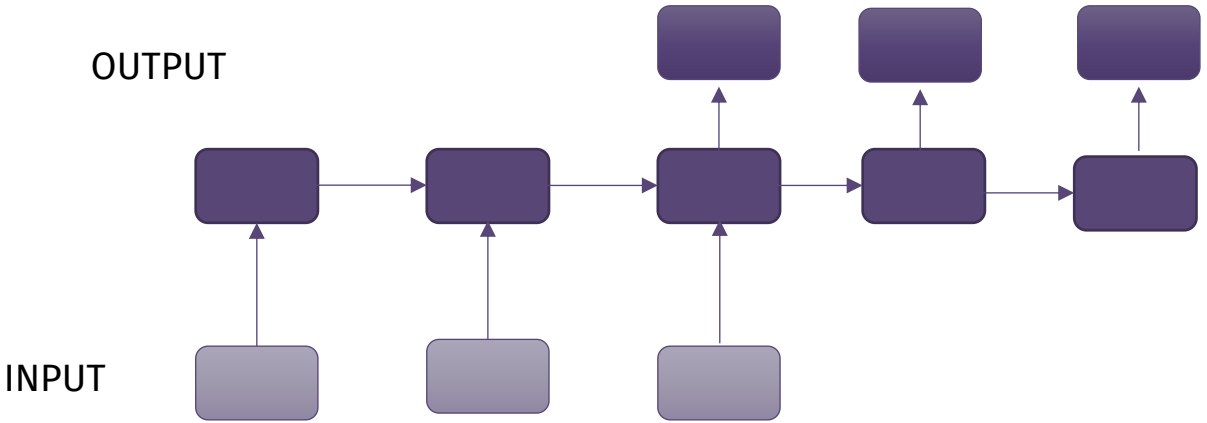
Example of applications: video captioning (input is a sequence of images, output is text)



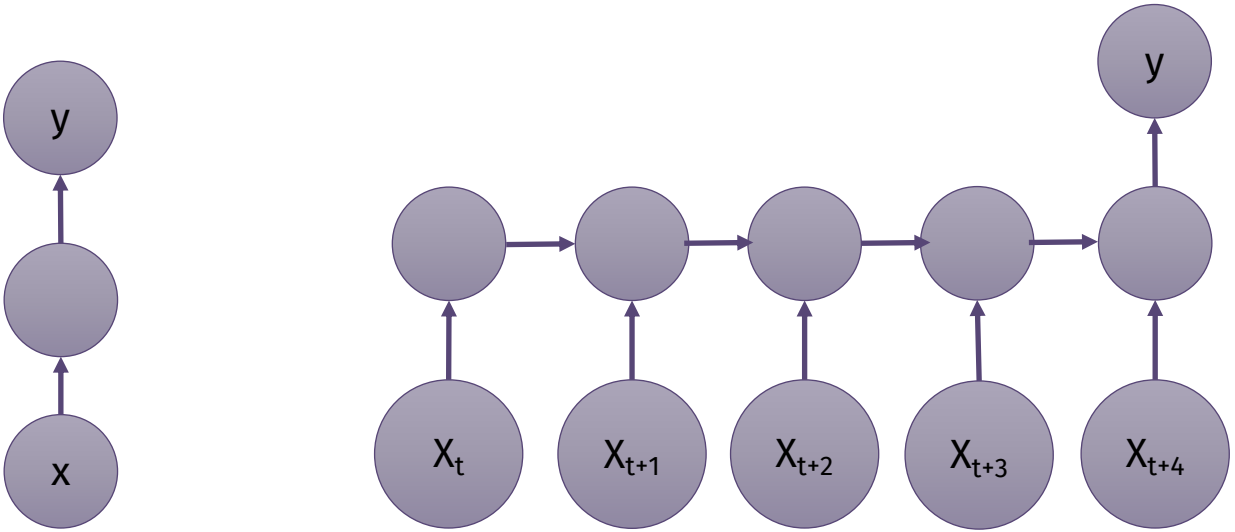
Different formulations of the problem

Delayed Many-to-many: input and output are both sequences

Example of applications: language translation (input is a text, output is a text)



Modelling sequences

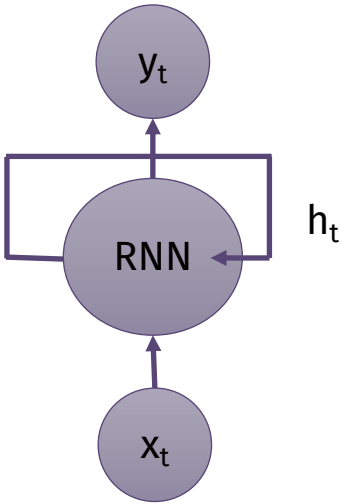


Standard one-to-one vanilla network

UniGe | MaLGa

Recurrent Neural Networks

Recurrent Neural Networks



A recurrence relation is applied at each time step to model the sequence

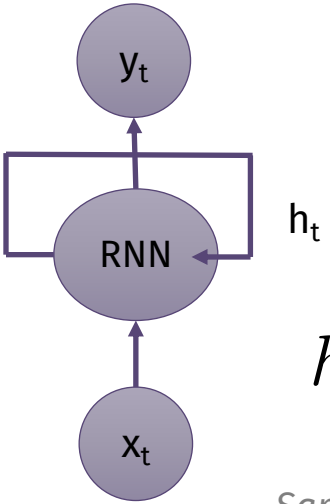
$$h_t = f_W(h_{t-1}, x_t)$$

Same function and weights are used for each time step, so they are a way to share weights over time

Recurrent Neural Networks (1986)

- Recurrence adds memory to the NN
- It also provides a way to model causal relationships between observations: the decision a recurrent net reached at time step t-1 affects the decision it will reach at time step t
- RNNs have two sources of input: the present and the recent past, which combine to determine how they respond to new data
- It is finding correlations between events separated by many moments, and these correlations are called “*long-term dependencies*”

RNNs: Forward propagation



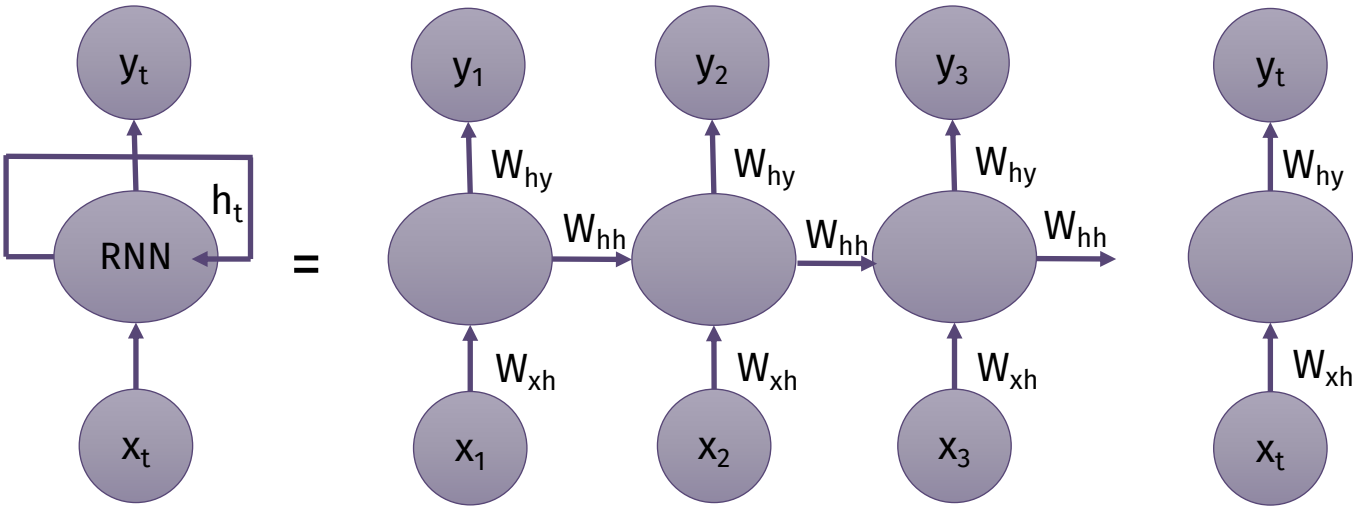
$$y_t = W_{hy} h_t$$

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

Same function and weights are used for each time step, so they are a way to share weights over time

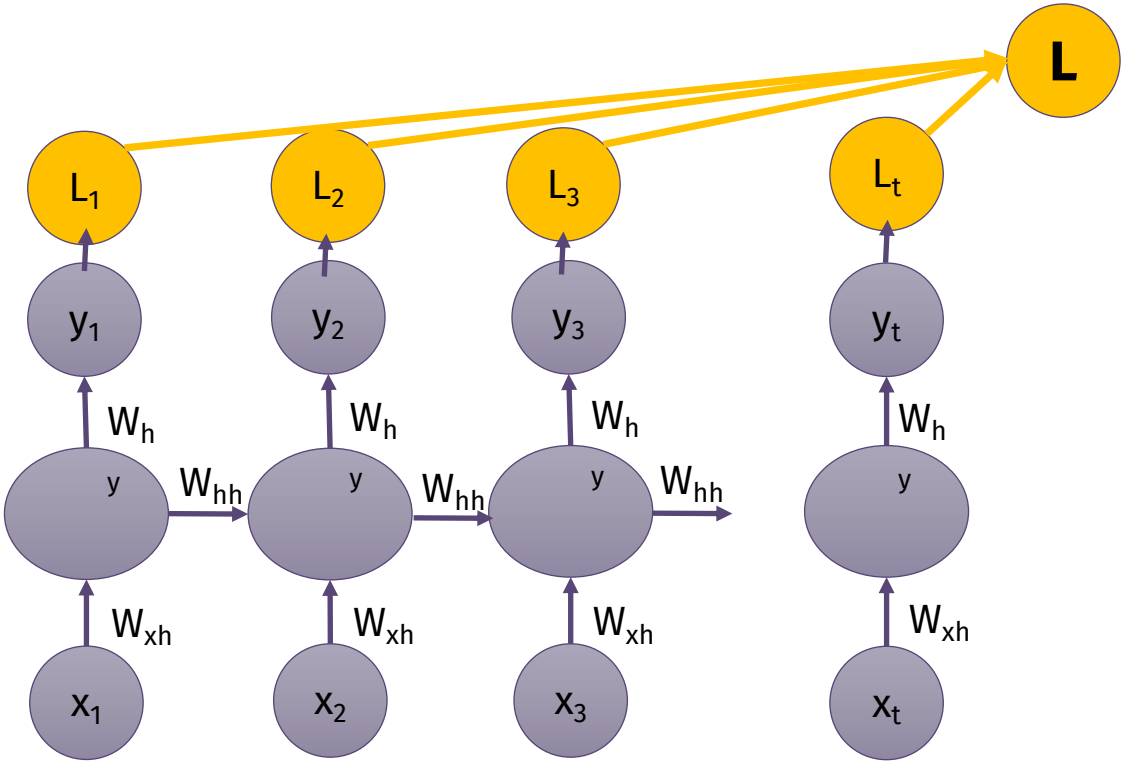
Unrolling RNNs over time

A RNN can be seen as a sequence of multiple, communicating copies of the same network

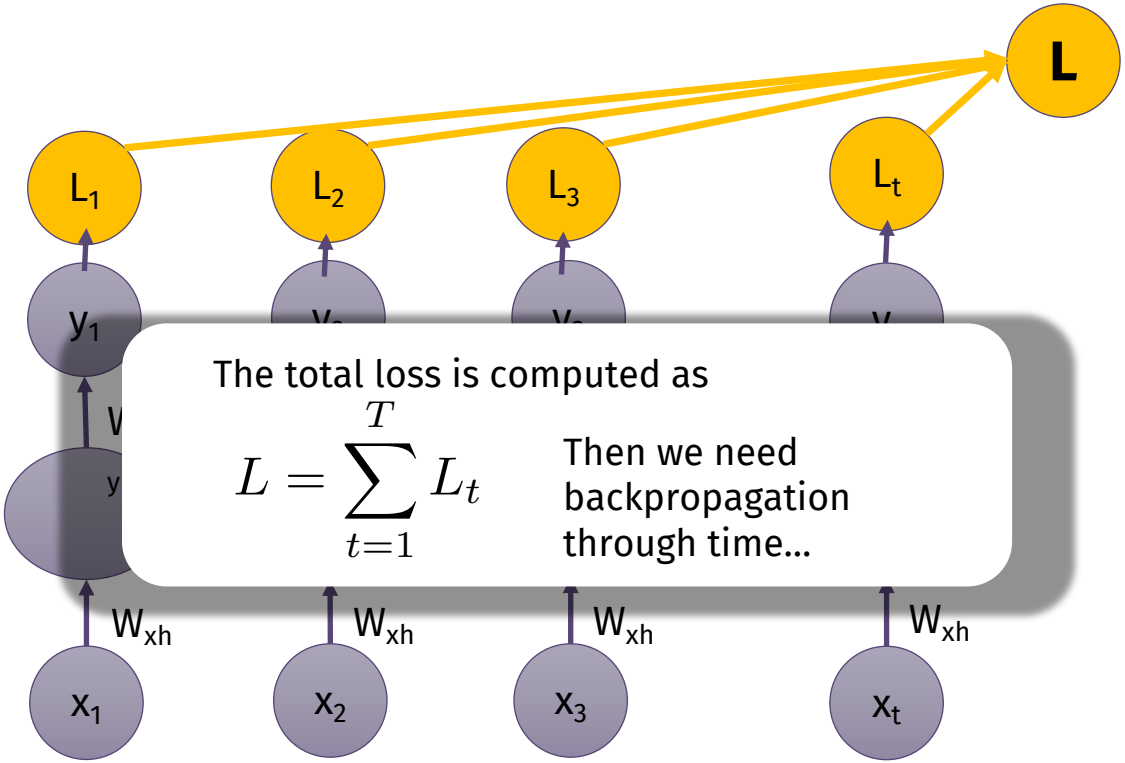


The weight matrices are filters that determine ***how much importance to give to both the present input and the past hidden state***

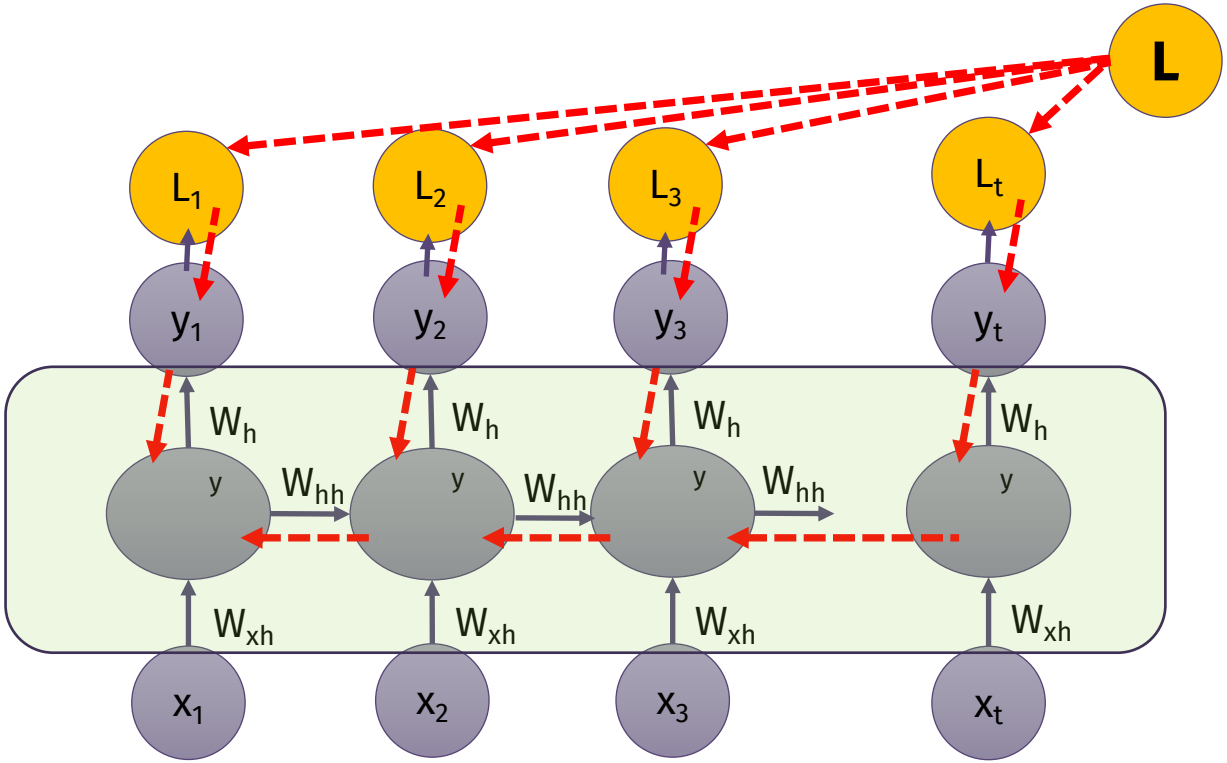
Again on forward propagation



Again on forward propagation



Backward propagation



Loss and backpropagation

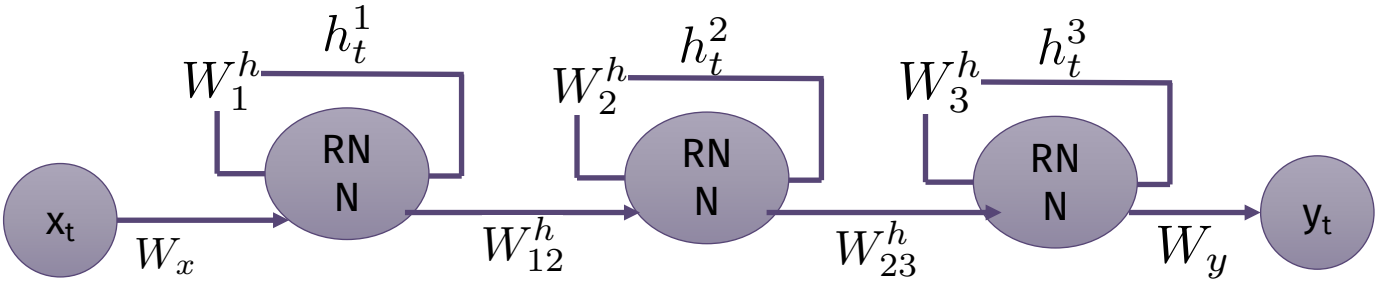
$$L = \sum_{t=1}^{T} L_t$$

$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \right)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Computed by multiplying consecutive time steps

What about having multiple hidden layers?



Gradients-related issues for long-term dependences

- The computation of the loss gradient as successive multiplication leads to instability of the gradient and may take very long training times
- Many values < 1 lead to **vanishing** gradient problems
- Many values > 1 lead to **exploding** gradient problems

Exploding gradient

- Many values > 1 lead to **exploding** gradient problems: the update with SGD is done with very large steps, leading to bad results

$$\Theta_{new} = \Theta_{old} - \alpha \nabla_{\Theta} L(\Theta)$$

- A possible solution is gradient clipping: if the gradient is greater than some threshold, scale it down before applying SGD update

- You make a step in the same direction but with a smaller step

Vanishing gradient

- Many values < 1 lead to **vanishing** gradient problems: gradient signal far over time is lost because it's much smaller than gradient signal from closer times

- Model weights are updated only with respect to near effects, not long-term effects

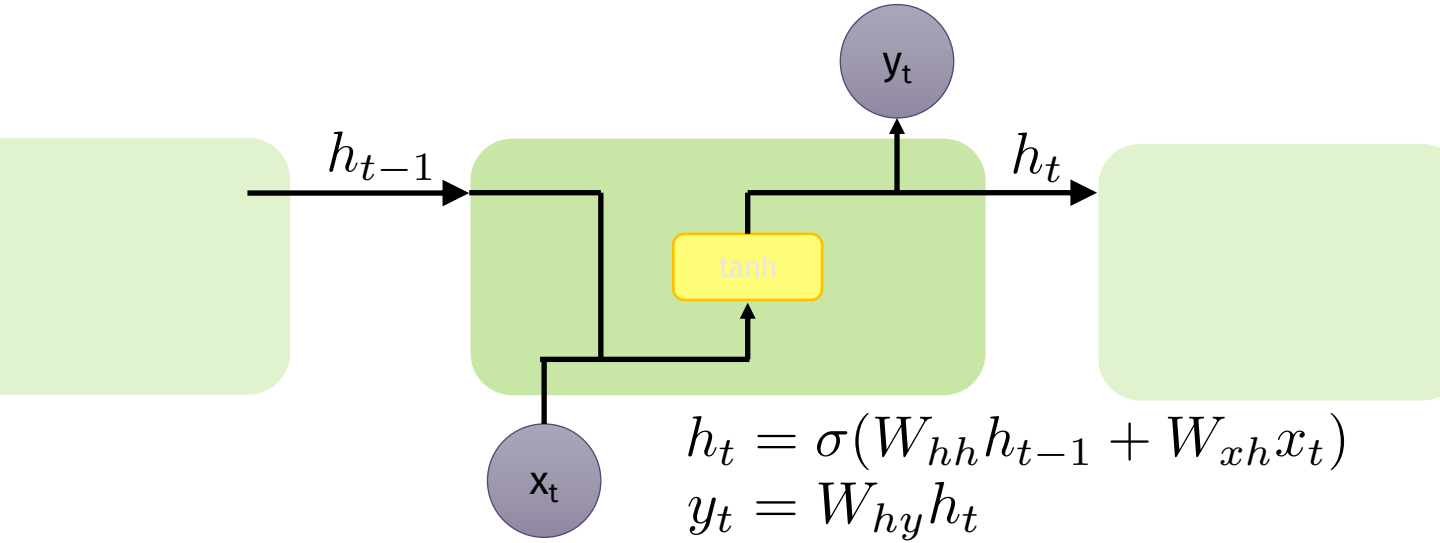
- A possible solution to learn long-term dependences in the data is to use **gated cells**

UniGe | MaLGa

Long-Short Term Memory

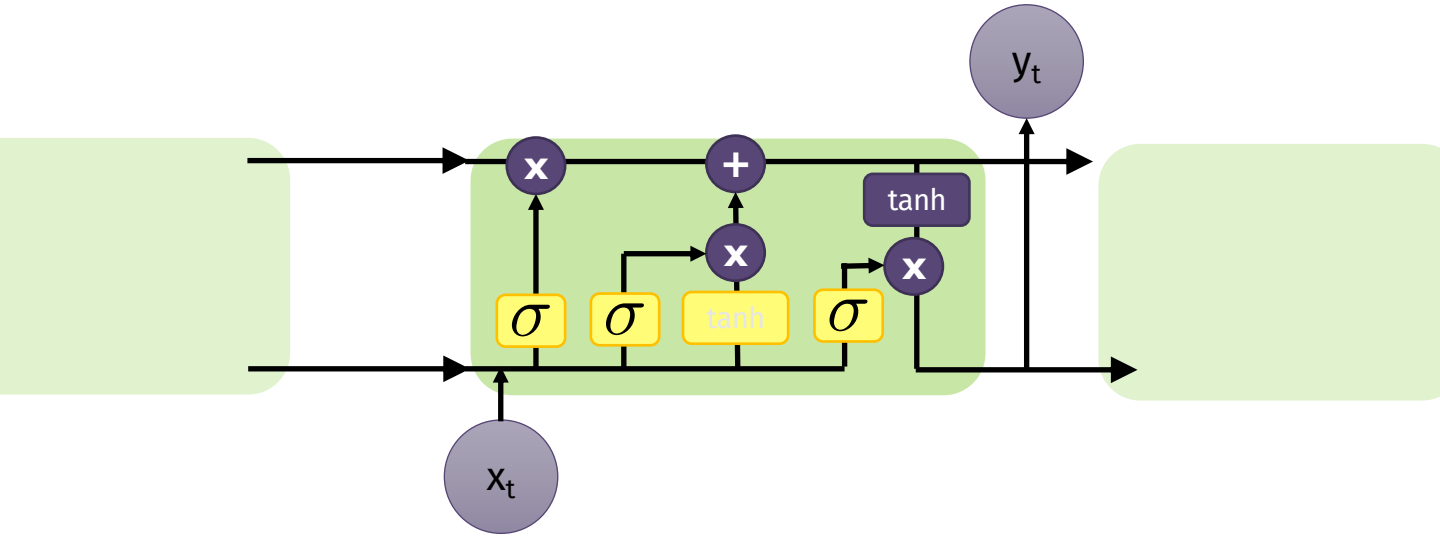
RNNs cells

- In standard RNNs, the cells contain a simple computation and their state is constantly re-written



Gated cells

- Gated cells contain computational blocks that control information flow



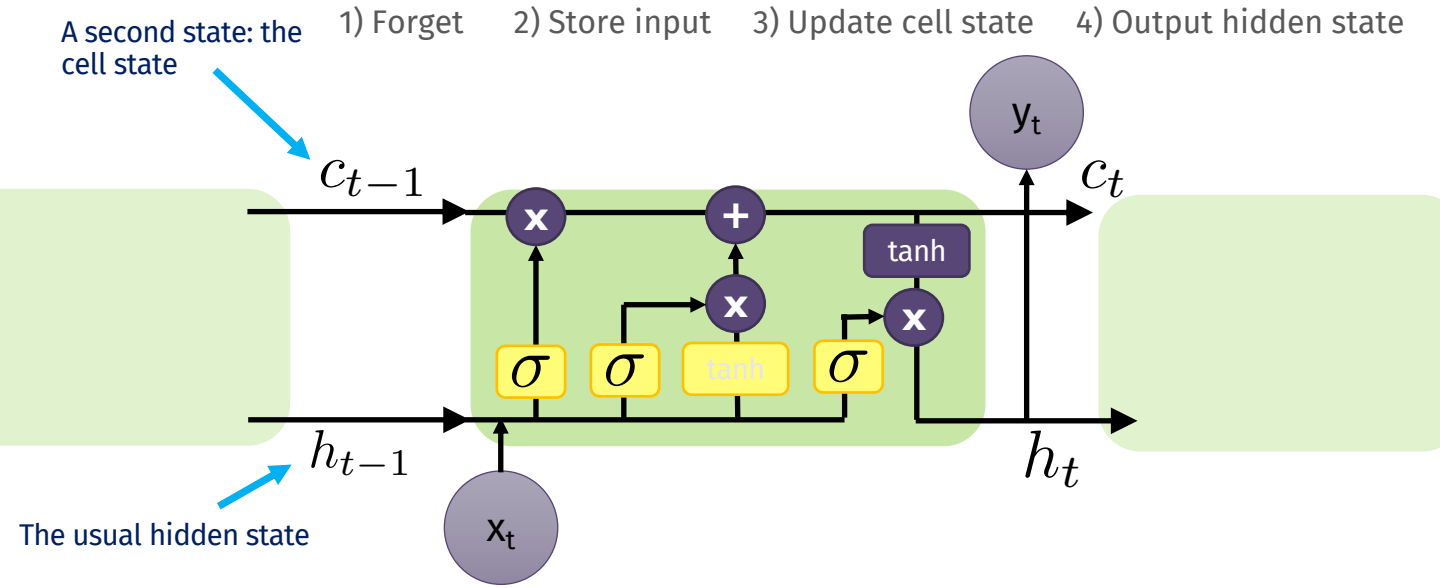
Long-short term memory (LSTM, 1997)

- They consider **connection weights that may change at each time step**
- Information is accumulated over a long duration
- Once the information has been used, it may be useful for the RNN to forget the old state or keep the information
- The LSTM learns how to decide when to do that (this is in fact the role of gated units)

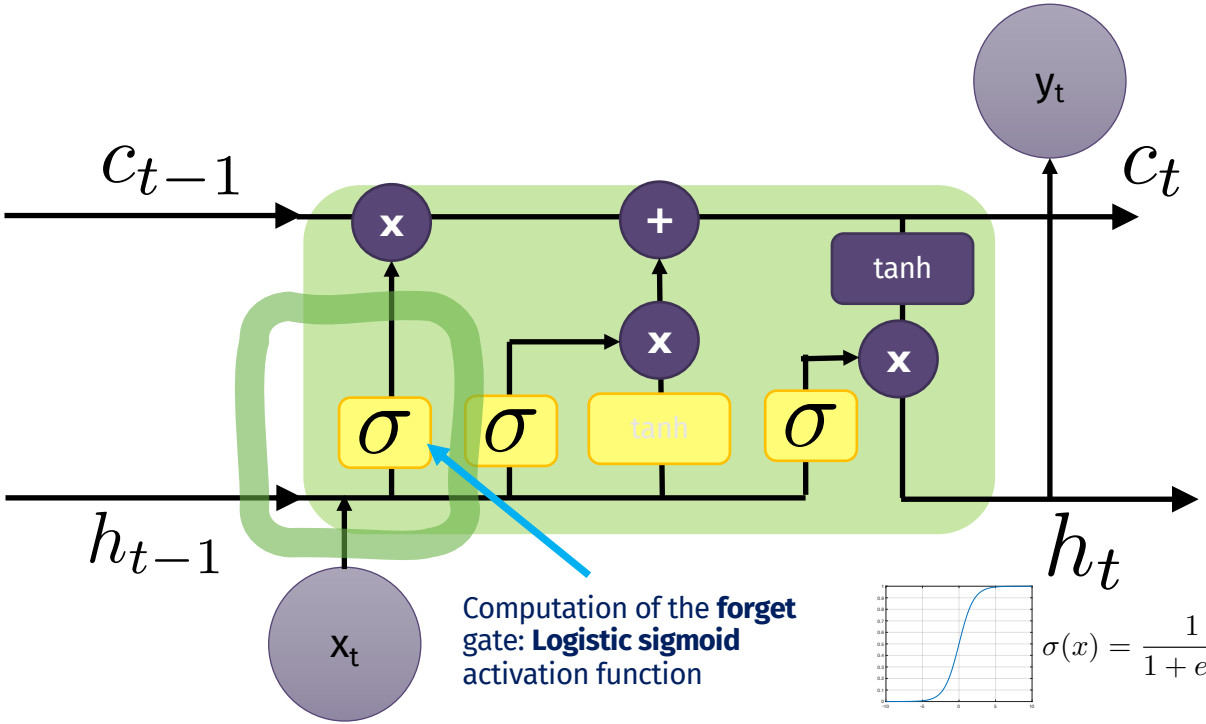
LSTM cell

- It includes two states: a **hidden state** and a **cell state**, both vectors of length n
- The cell stores long-term information. The LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**, vectors again of length n
- Each element of the gates can be open (1), closed (0), or somewhere in-between
- The gates are dynamic: their value is computed based on the current context

LSTM cell: how does it work



LSTM cell – 1) Forget

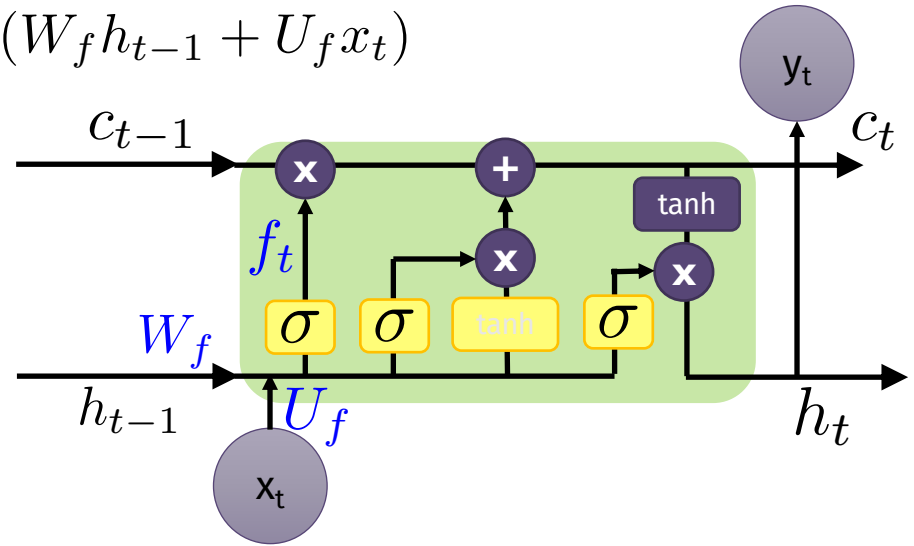


Forget gate

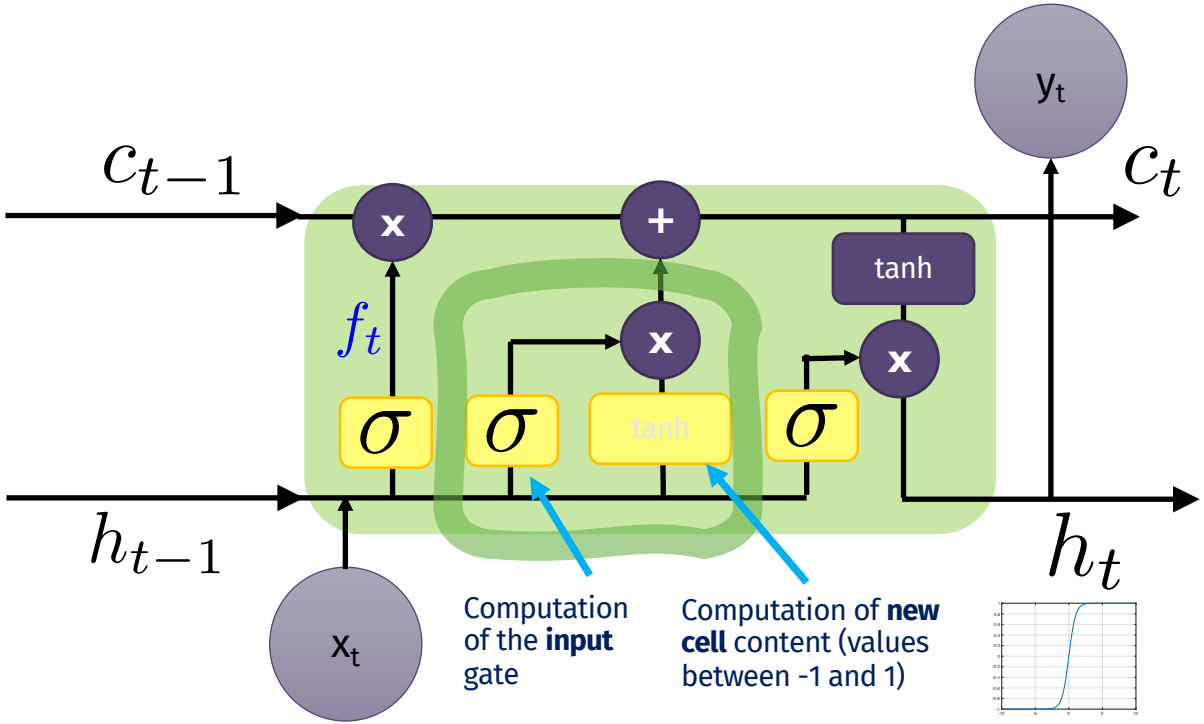
- It decides which information to keep and what to forget from the previous cell state

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

Sigmoid gate: all values are between 0 and 1



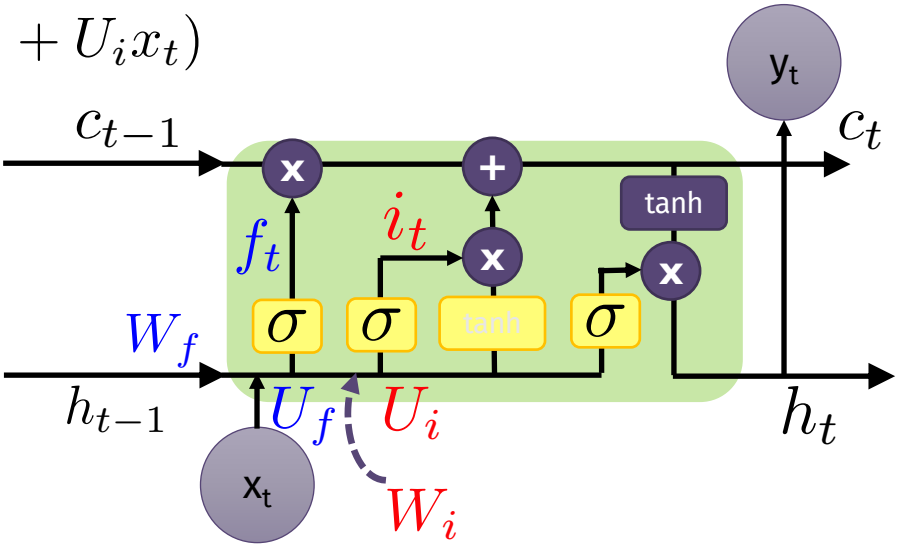
LSTM cell – 2) Store input



Input gate

- It decides what new information storing in the cell states

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

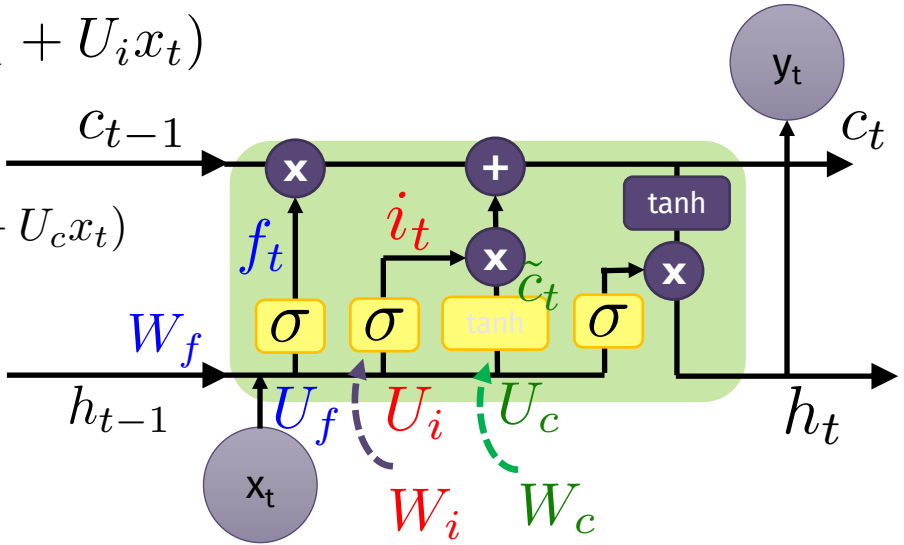


Input gate

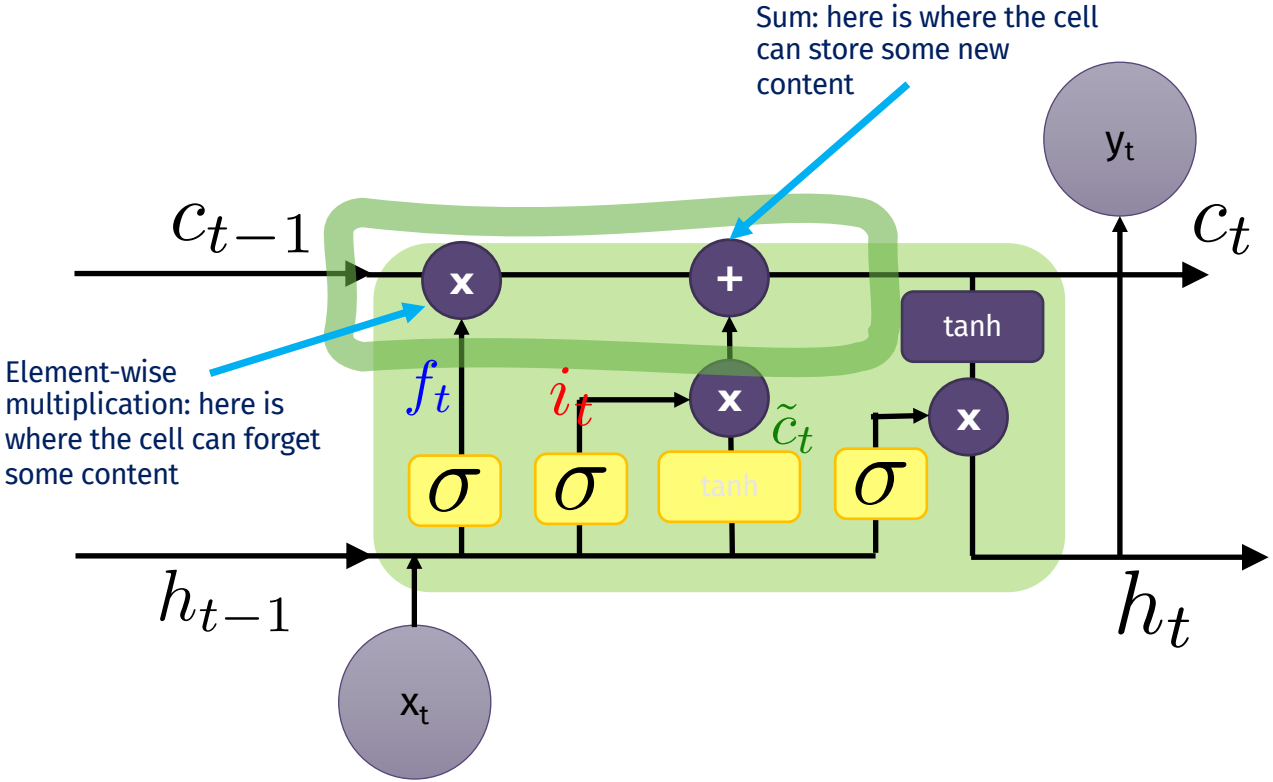
- It decides what new information storing in the cell states

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

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



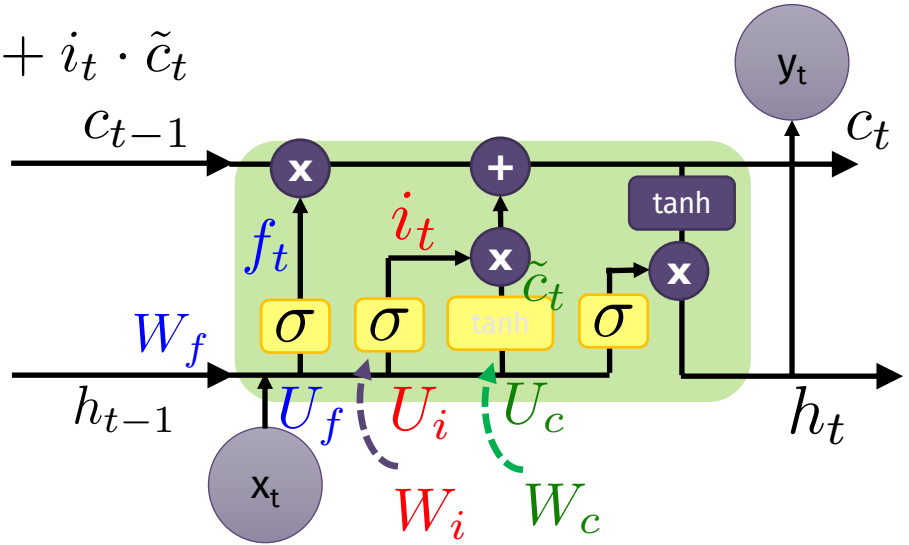
LSTM cell – 3) Update



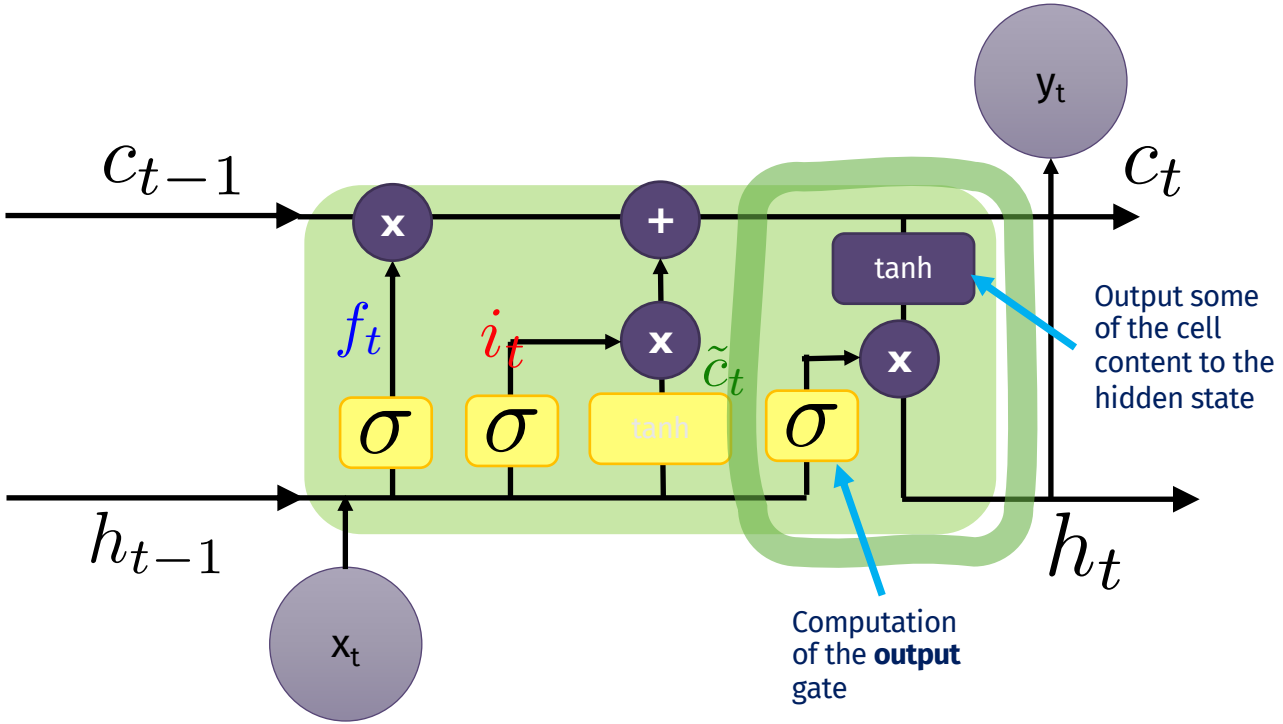
Update

- The cell state values are selectively updated

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



LSTM cell – 4) Output



UniGe