ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

4 - Long-Short-Term Memory

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 25-26-27, 2023

Introduction

Why LSTM?

- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame.
- If RNNs could do this, they'd be extremely useful.
- But can they? It depends.
- Sometimes, we only need to look at recent information to perform the present task...

Why LSTM?

- For example, consider a language model trying to predict the next word based on the previous ones.
- If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context - it's pretty obvious the next word is going to be sky.
- In such cases, where the gap between the relevant information and the place that it"s needed is small, RNNs can learn to use the past information.

Why LSTM?

- But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France... (a lot of words) ... I speak fluent ???."
- Recent information suggests that the next word is probably the name of a language, French in this case, but if we want to narrow down which language, we need the context of France, from further back.
- It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

What is LSTM

- Long Short Term Memory networks - usually just called "LSTMs" - are a special kind of RNN, capable of learning long-term dependencies.
- They were introduced by Hochreiter and Schmidhuber (1997), and were refined and popularized by many people in following work.
- They work very well on a large variety of problems, and are now widely used.

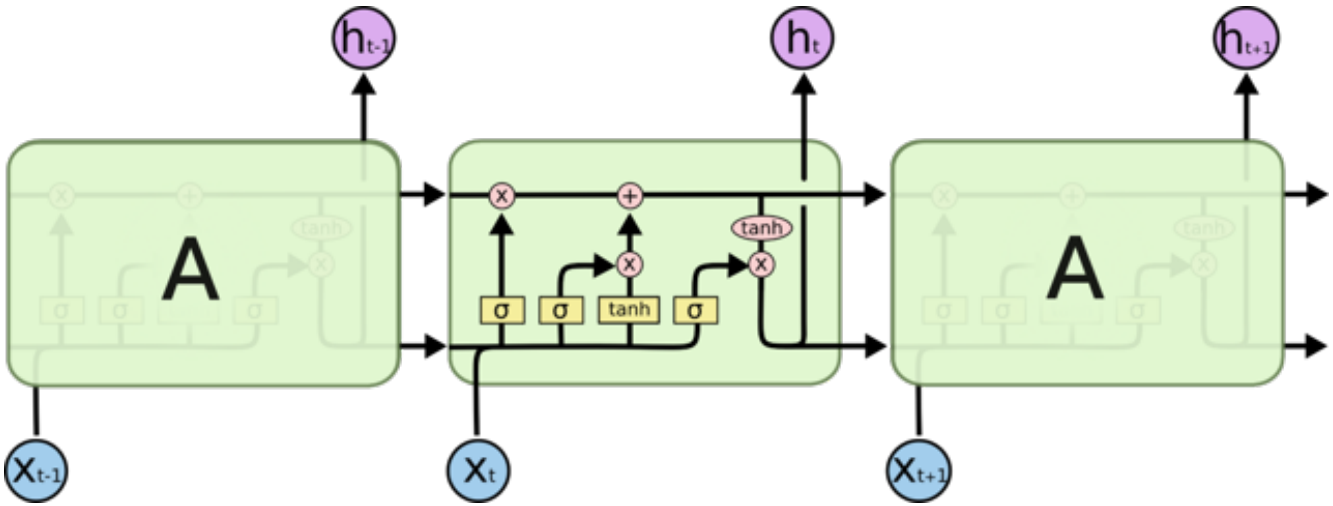
What is LSTM

- LSTMs are explicitly designed to avoid the long-term dependency problem.
- All recurrent neural networks have the form of a chain of repeating modules (Memory Cell) of neural network.
- As we have seen, in standard RNNs, this repeating module has a very simple structure.
- The GRU Network is slightly more complex, let‘s see the LSTM Memory Cell and its fundamental equations ...

LSTM Memory Cell and Equations

LSTM

- The repeating module has a different structure with respect to the GRU case.
- Now we have **three gates** interacting in a very special way.



LSTM: Difference Between LSTM and GRU

GRU

$$\tilde{c}^{<t>} = tanh(W_{cc}[\Gamma_r * c^{<t-1>}] + W_{cx}x^{<t>} + b_c)$$

$$\Gamma_u = \sigma(W_{uc}c^{<t-1>} + W_{ux}x^{<t>} + b_u)$$

$$\Gamma_r = \sigma(W_{rc}c^{<t-1>} + W_{rx}x^{<t>} + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

LSTM

$$\tilde{c}^{<t>} = tanh(W_{ca}a^{<t-1>} + W_{cx}x^{<t>} + b_c)$$

$$\Gamma_u = \sigma(W_{ua}a^{<t-1>} + W_{ux}x^{<t>} + b_u)$$

$$\Gamma_f = \sigma(W_{fa}a^{<t-1>} + W_{fx}x^{<t>} + b_f)$$

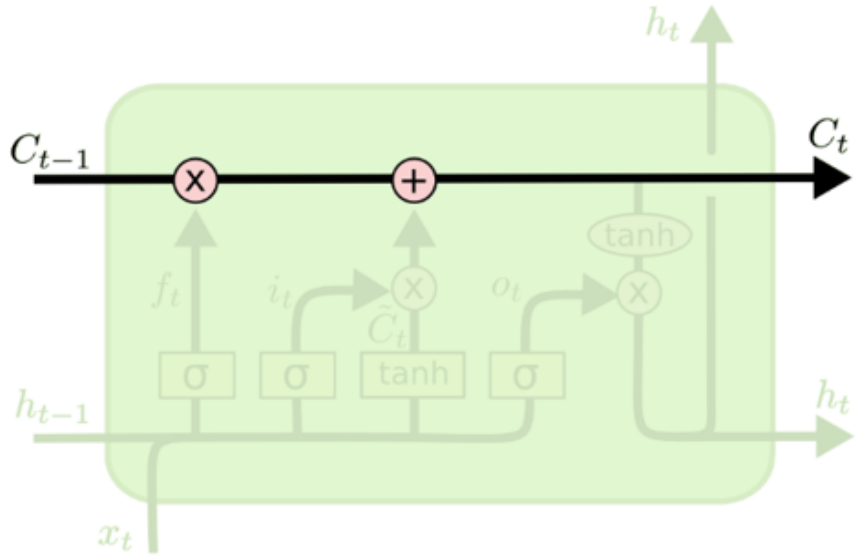
$$\Gamma_o = \sigma(W_{oa}a^{<t-1>} + W_{ox}x^{<t>} + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

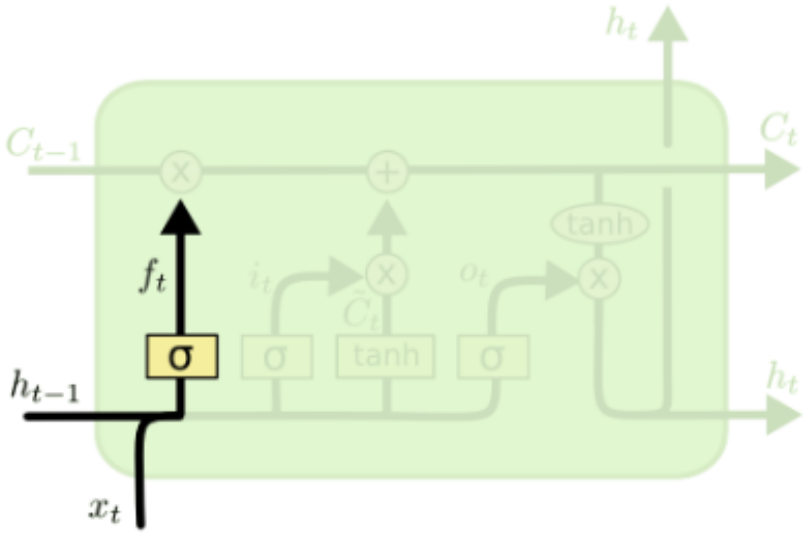
LSTM: Cell State

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



Step-by-Step LSTM Walk Through

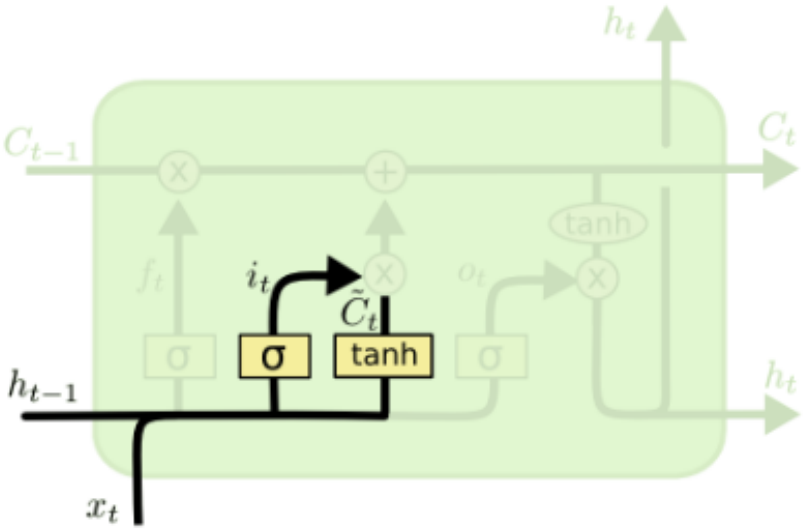
- The first step in our LSTM is to decide what information we‘re going to throw away from the cell state.
- This decision is made by a sigmoid layer called the "forget gate layer." It looks at $h^{<t-1>}$ and x^t, and outputs a number between 0 and 1 for each number in the cell state $c^{<t-1>}$. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

Step-by-Step LSTM Walk Through

- The next step is to decide what new information we're going to store in the cell state. This has two parts.
- First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a *tanh* layer creates a vector of new candidate values, $\tilde{c}^{<t>}$, that could be added to the state.
- In the next step, we'll combine these two to create an update to the state.

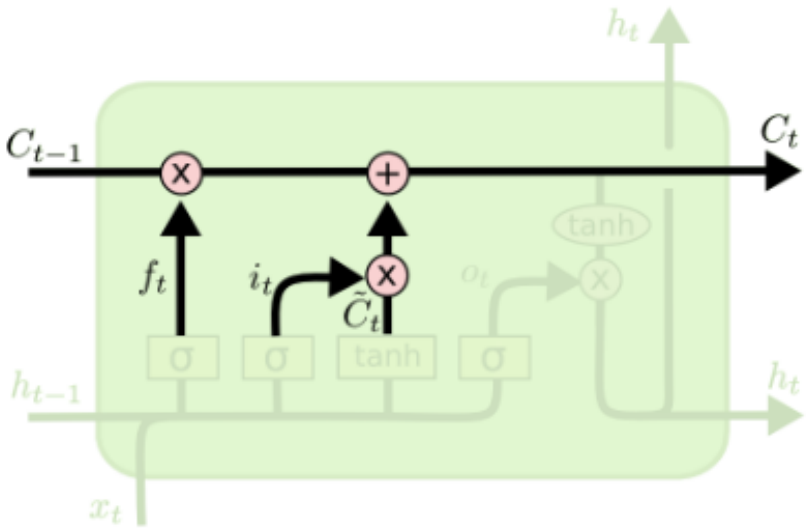


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Step-by-Step LSTM Walk Through

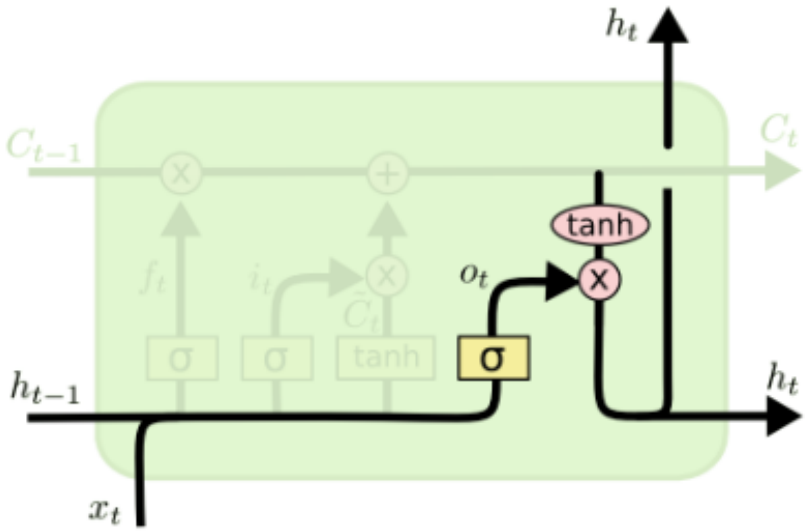
It's now time to update the old cell state, $C^{<t-1>}$, into the new cell state C^t. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by $f^{<t>}$, then we add $i_t \cdot \tilde{C}^{<t>}$. This is the new candidate values.



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

Step-by-Step LSTM Walk Through

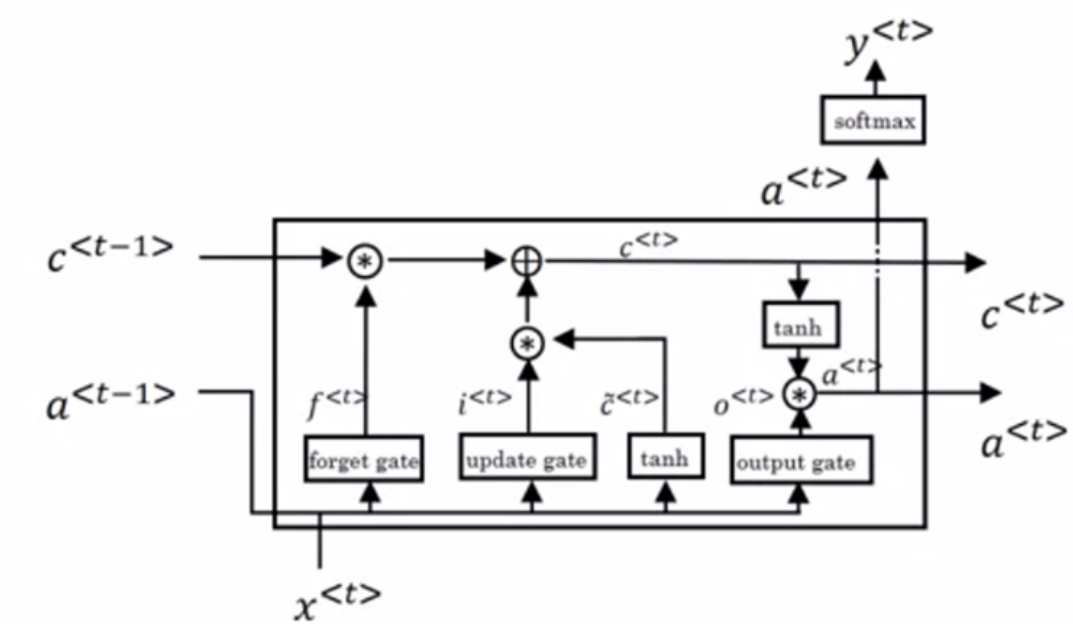
Finally, we need to decide what we‘re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we‘re going to output. Then, we put the cell state through *tanh* and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



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

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

LSTM: Recap



$$\tilde{c}^{<t>} = tanh(W_{ca}a^{<t-1>} + W_{cx}x^{<t>} + b_c)$$

$$\Gamma_u = \sigma(W_{ua}a^{<t-1>} + W_{ux}x^{<t>} + b_u)$$

$$\Gamma_f = \sigma(W_{fa}a^{<t-1>} + W_{fx}x^{<t>} + b_f)$$

$$\Gamma_o = \sigma(W_{oa}a^{<t-1>} + W_{ox}x^{<t>} + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

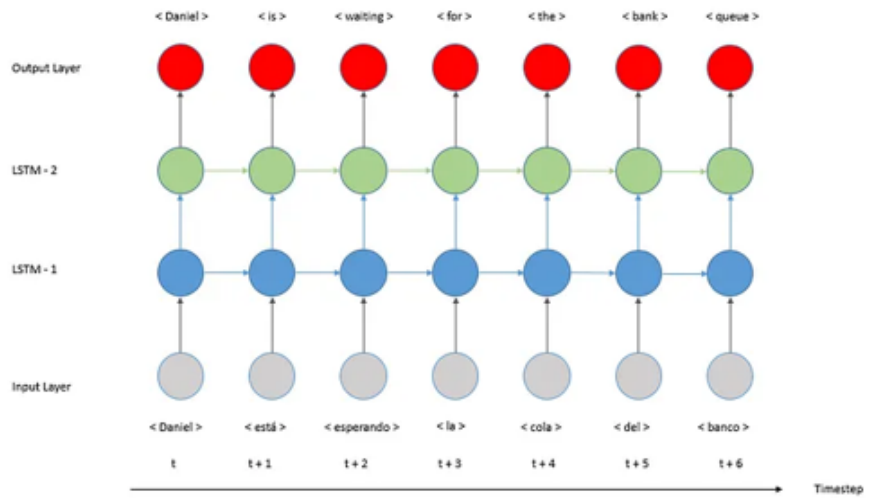
Conclusions

Chollet on LSTM

If you want to get philosophical, you can interpret what each of these operations is meant to do. For instance, you can say that multiplying `c_t` and `f_t` is a way to deliberately forget irrelevant information in the carry dataflow. Meanwhile, `i_t` and `k_t` provide information about the present, updating the carry track with new information. But at the end of the day, these interpretations don't mean much, because what these operations *actually* do is determined by the contents of the weights parameterizing them; and the weights are learned in an end-to-end fashion, starting over with each training round, making it impossible to credit this or that operation with a specific purpose. The specification of an RNN cell (as just described) determines your hypothesis space—the space in which you'll search for a good model configuration during training—but it doesn't determine what the cell does; that is up to the cell weights. The same cell with different weights can be doing very different things. So the combination of operations making up an RNN cell is better interpreted as a set of *constraints* on your search, not as a *design* in an engineering sense.

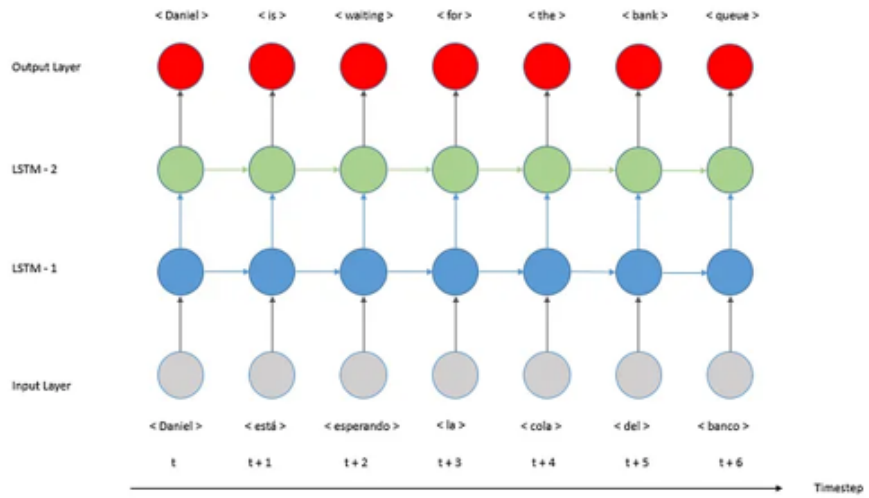
Limitations of the approach seen so far

- Every architecture seen so far has one important limitation: the sequence length.
- As we can see in the image, **the input sequence and the output sequence must have the same length**.



Limitations of the approach seen so far

- What if we need to have different lengths?
- Models with different sequences lengths are, for example, sentiment analysis that receives a sequence of words and outputs a number, or Image captioning models where the input is an image and the output is a sequence of words.



Limitations of the approach seen so far

- If we want to develop models were **inputs and outputs lengths are different** we need to develop an encoder decoder model.
- This is the subject of the next chapter.