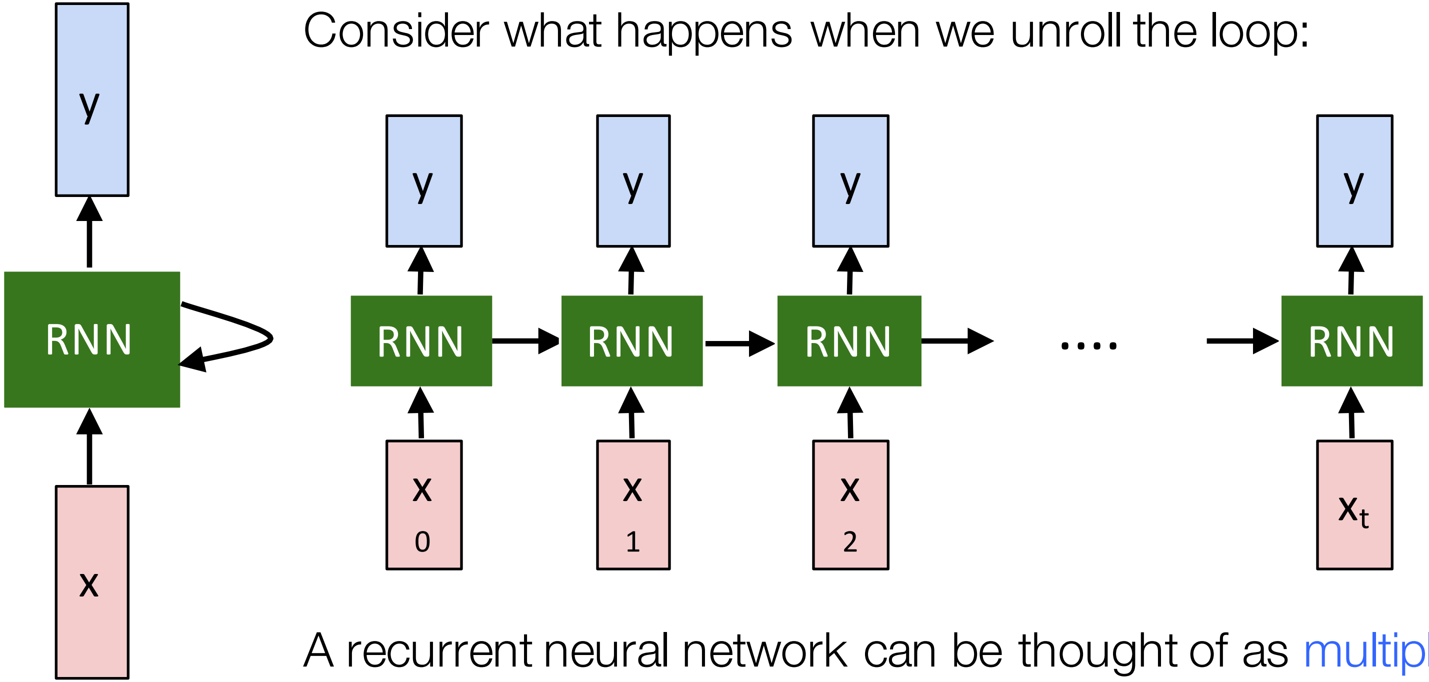
Recurrent Neural Network

Consider what happens when we unroll the loop:



A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

Recurrent Neural Network

We can process a sequence of vectors x by applying a recurrence formula at every time step:

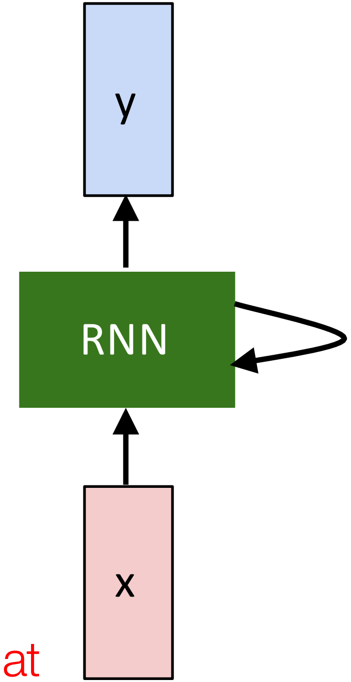
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

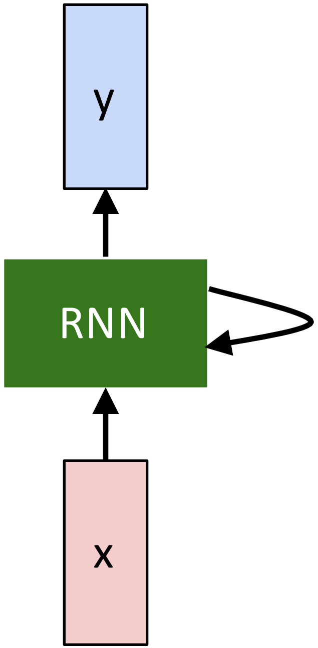
some function with parameters W



Important: the same function and the same set of parameters are used at every time step.

(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



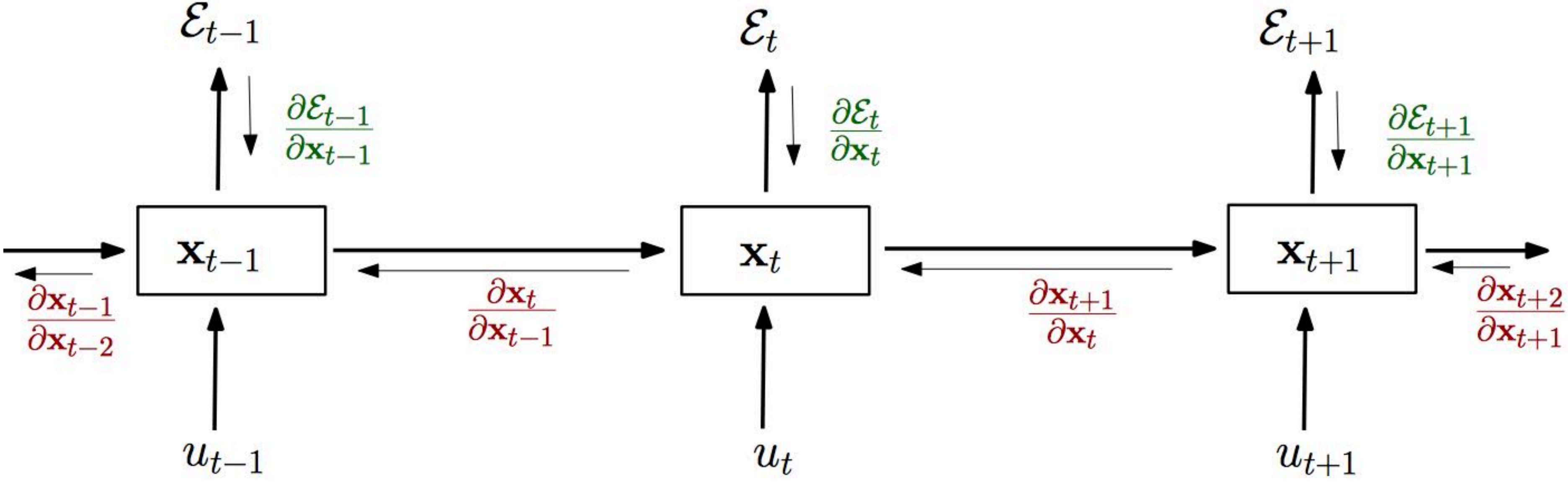
$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

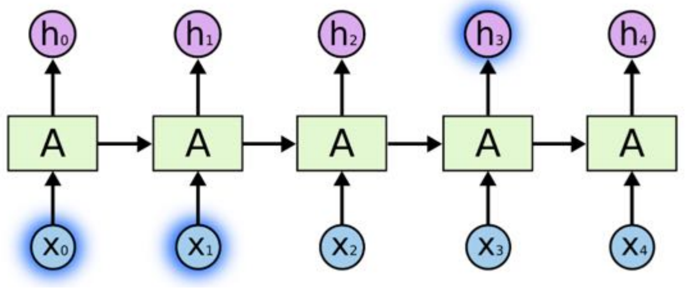
Backpropagation Through Time (BPTT)

- The recurrent model is represented as a multi-layer one (with an unbounded number of layers) and backpropagation is applied on the unrolled model

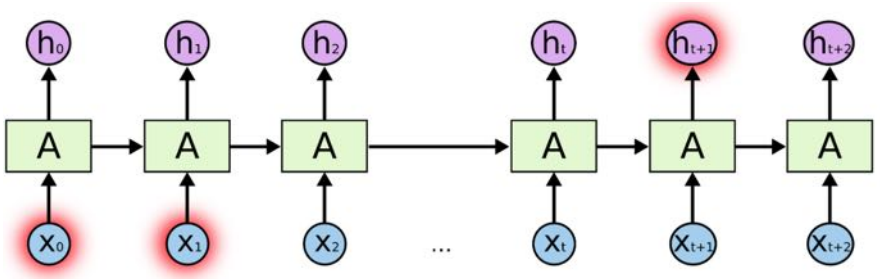


The problem of long-term dependencies

- (Vanilla) RNNs connect previous information to present task:
- \- enough for predicting the next word for "the clouds are in the *sky*"



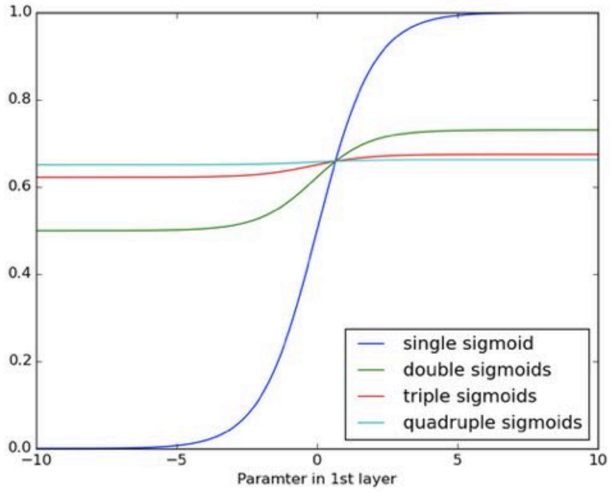
- \- may not be enough when more context is needed
- "I grew up in France… I speak fluent *French*."



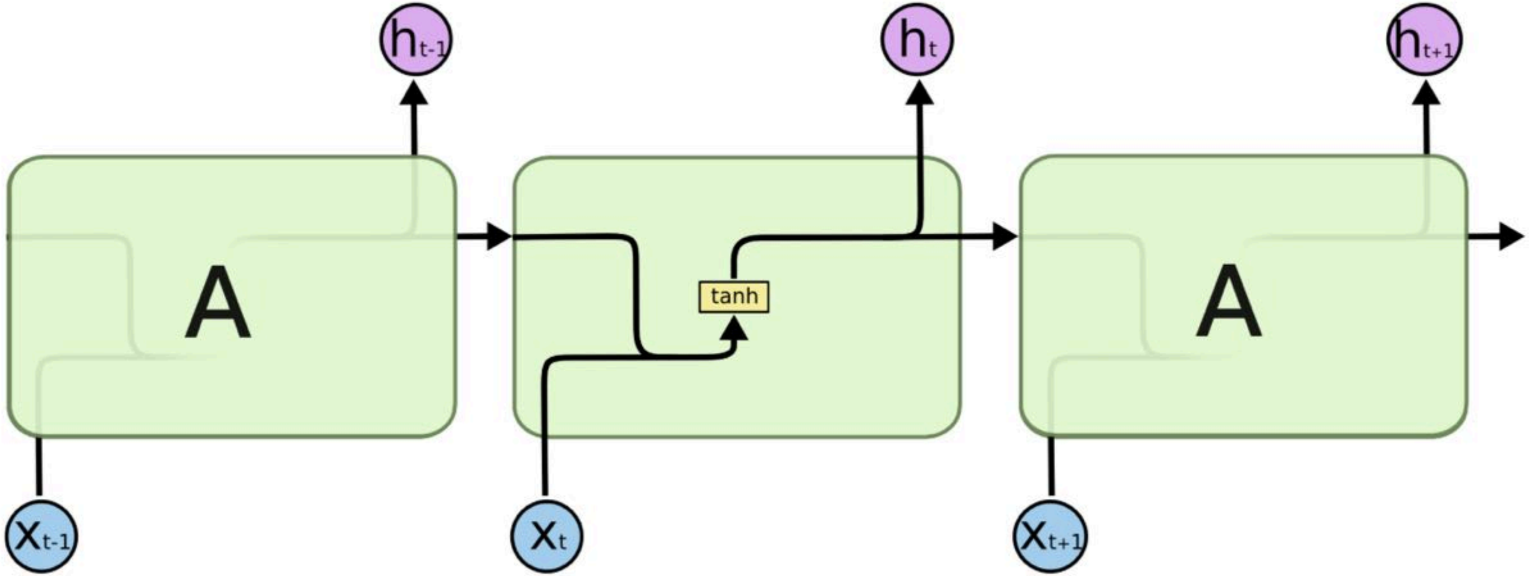
Adapted from: C. Olah

The problem of vanishing gradients

- In a traditional recurrent neural network, during the gradient backpropagation phase, the gradient signal can end up being multiplied a large number of times
- If the gradients are large
  - Exploding gradients, learning diverges
  - **Solution: Clip the gradients to a certain max value.**
- If the gradients are small
  - Vanishing gradients, learning very slow or stops
  - **Solution: introducing memory via LSTM, GRU, etc.**



All recurrent neural networks have the form of a chain of repeating modules of neural network

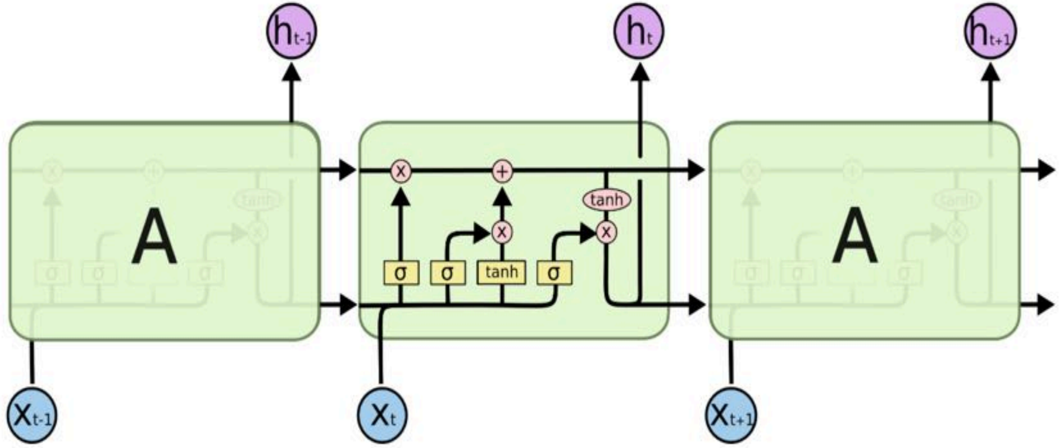


The repeating module in a standard RNN contains a single layer.

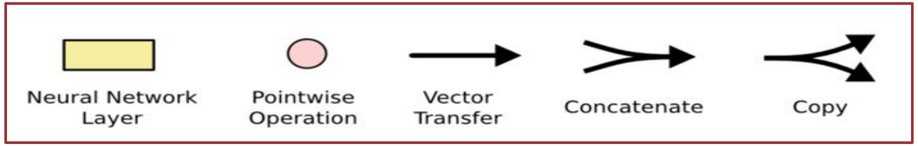
Adapted from: C. Olah

Long Short Term Memory (LSTM)[Hochreiter & Schmidhuber (1997)]

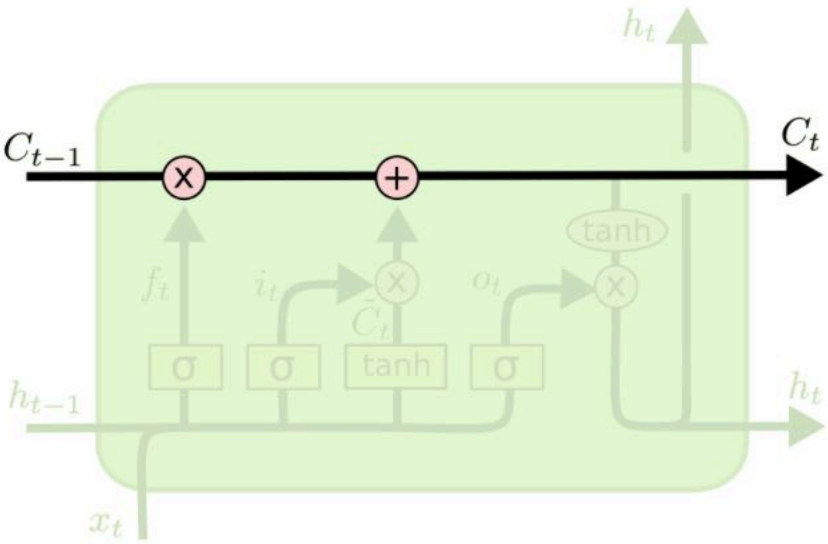
- A memory cell using logistic and linear units with multiplicative interactions:
- Information gets into the cell whenever its **input** gate is on.
- The information stays in the cell so long as its **forget** gate is on.
- Information can be read from the cell by turning on its **output** gate.



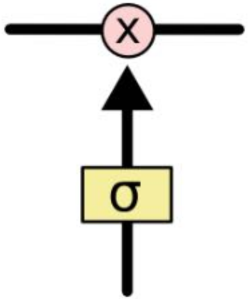
The repeating module in an LSTM contains four interacting layers.



The Core Idea Behind LSTMs : Cell State



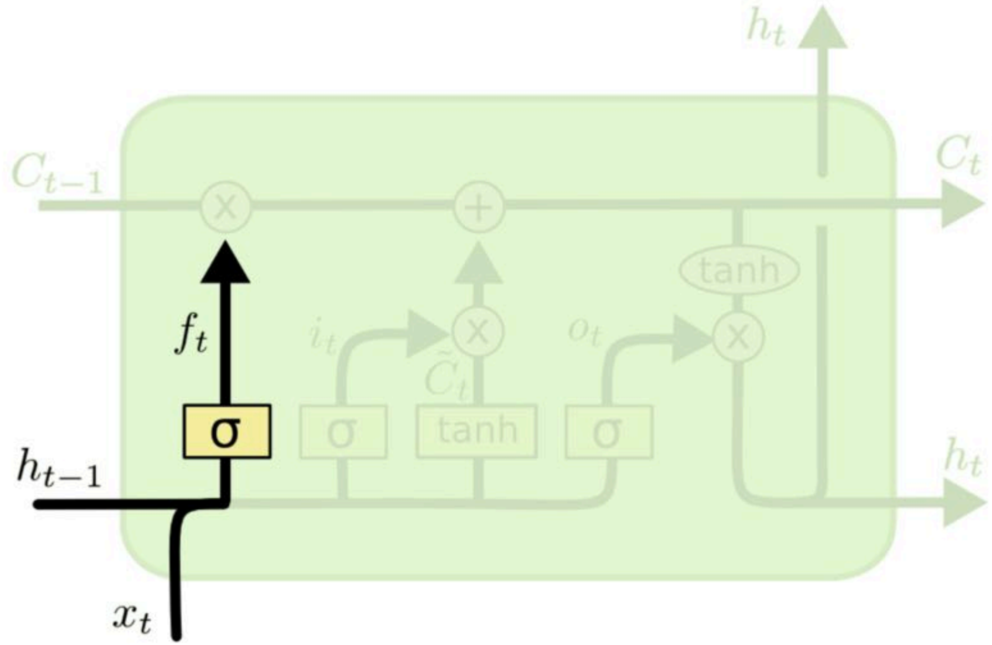
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



An LSTM has three of these gates, to protect and control the cell state.

Adapted from: C. Olah

LSTM : Forget gate



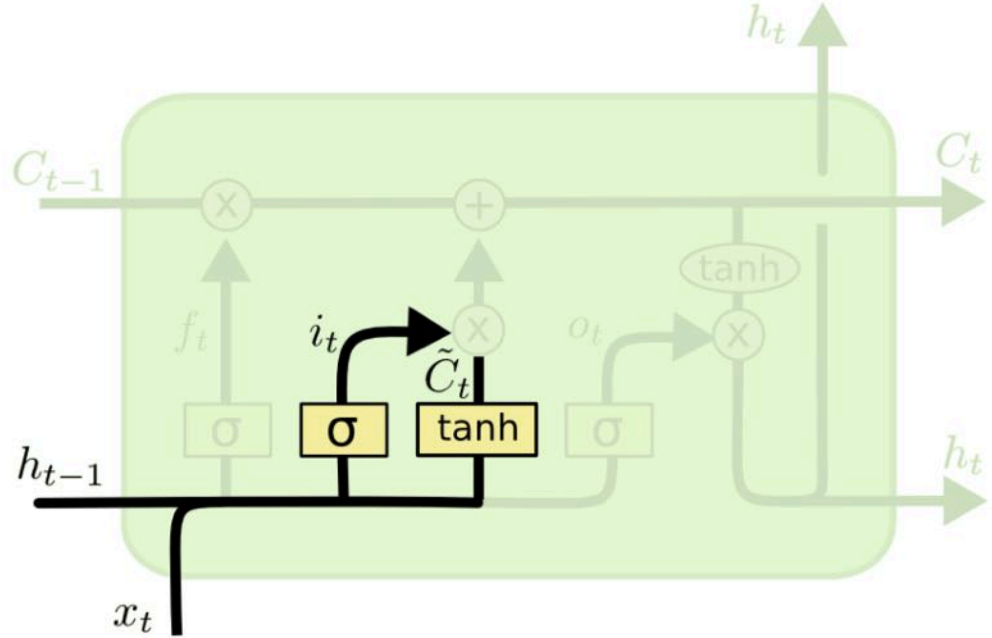
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

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.**

LSTM : Input gate and Cell State

The next step is to decide what new information we're going to store in the cell state.



a sigmoid layer called the **input gate layer** decides which values we'll update.

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

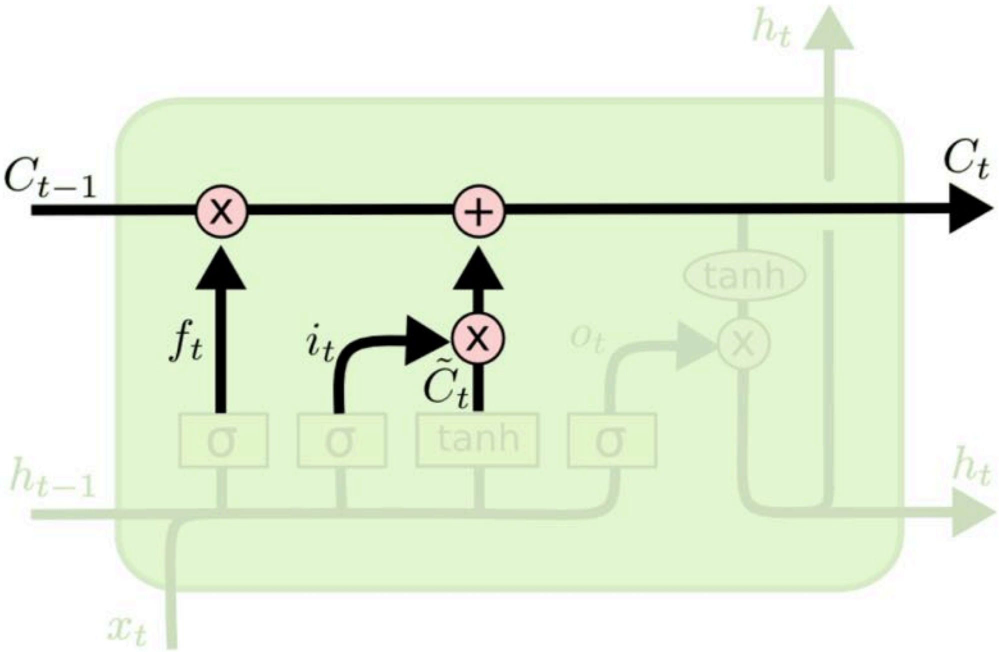
a tanh layer creates a vector of new candidate values, that could be added to the state.

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

Adapted from: C. Olah

LSTM : Input gate and Cell State

It's now time to update the old cell state into the new cell state.



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

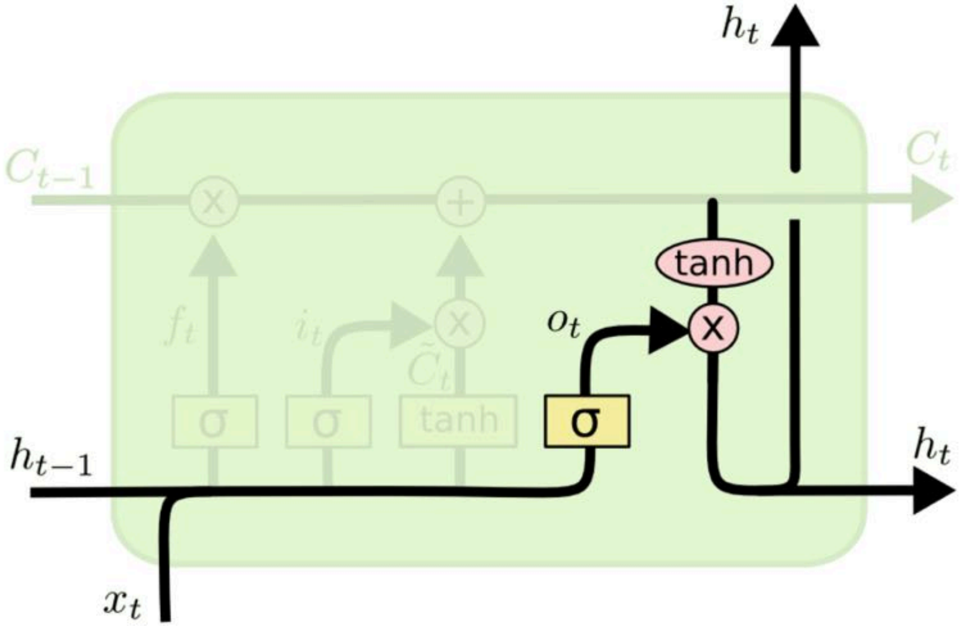
We multiply the old state by ft forgetting the things we decided to forget earlier.

Then, we add the new candidate values, scaled by how much we decided to update each state value.

Adapted from: C. Olah

LSTM : Output

Finally, we need to decide what we're going to output.



First, we run a sigmoid layer which decides what parts of the cell state we're going to output.

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

Then, we 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.

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

Adapted from: C. Olah