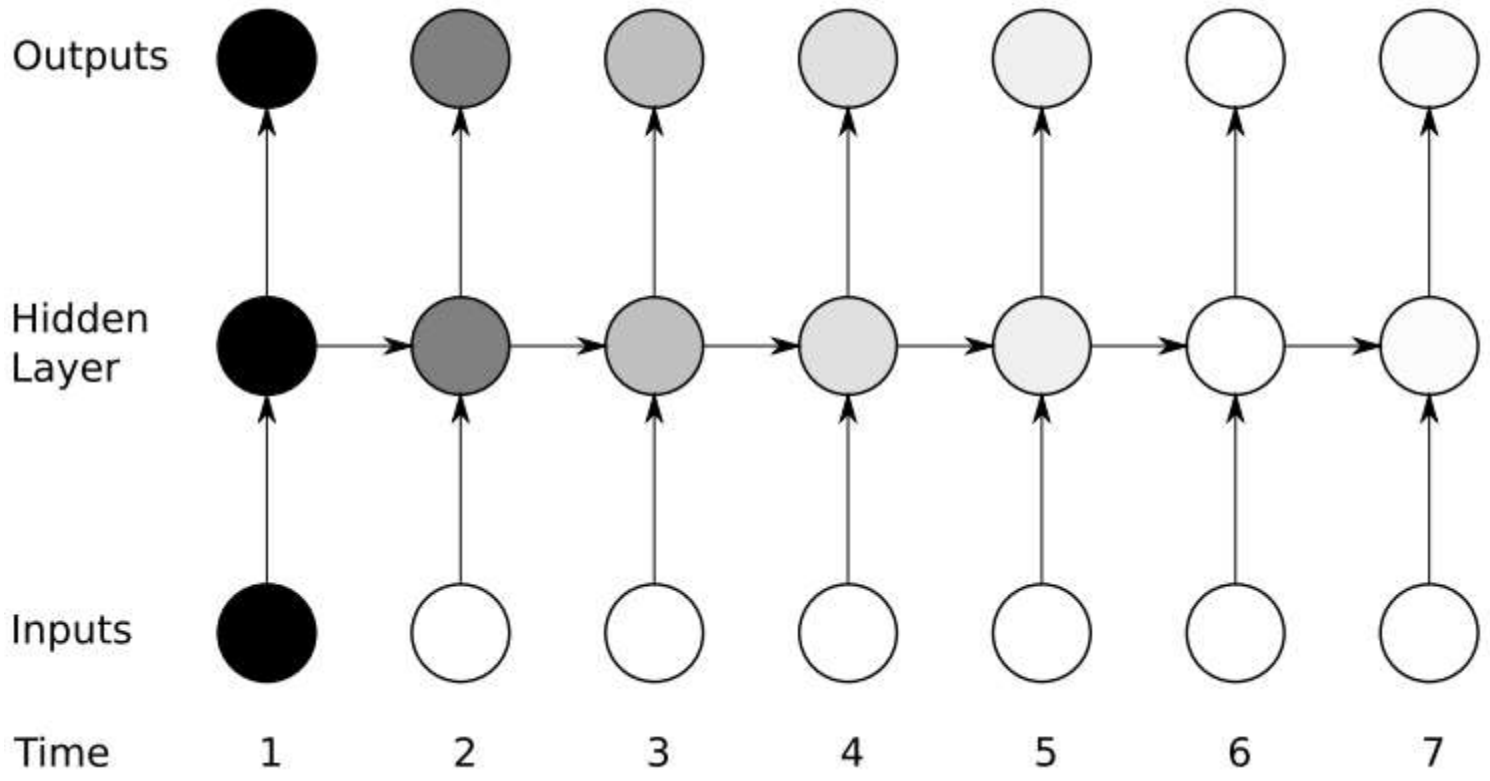
LSTM and GRU

What's wrong with Naïve RNN?

- When dealing with a time series, it tends to forget old information. When there is a distant relationship of unknown length, we wish to have a “memory” to it.
- Limitations of Backprop Through Time
  - Vanishing Gradients
  - Exploding Gradients

Vanishing Gradients

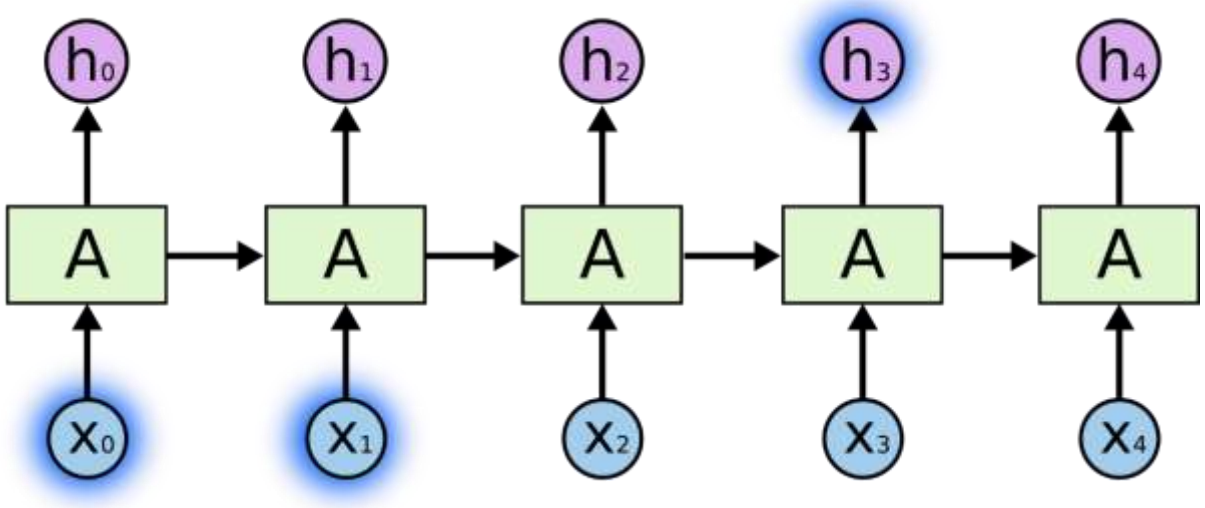


Exploding Gradients

- In the same way, gradients may be exploding for each time step if gradient computed at each step is increasing
- One solution is to clip the gradient to a standard value
- i.e. Gradients larger than certain value can be changed into the maximum gradient value

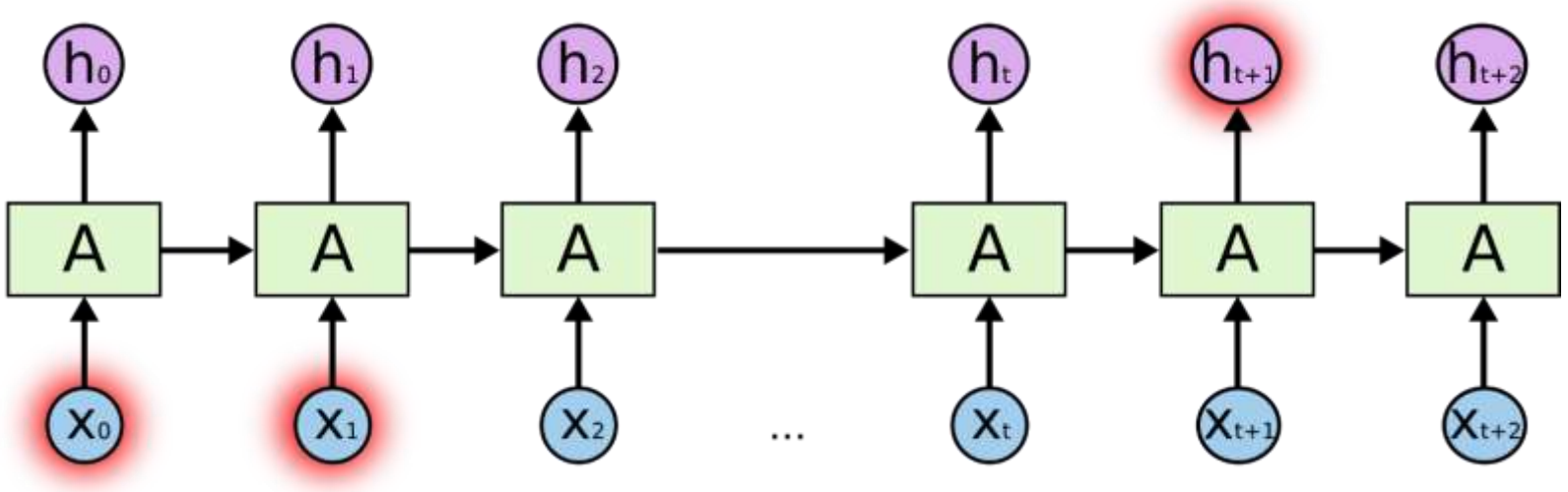
The Problem of Long-Term Dependencies

- 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.



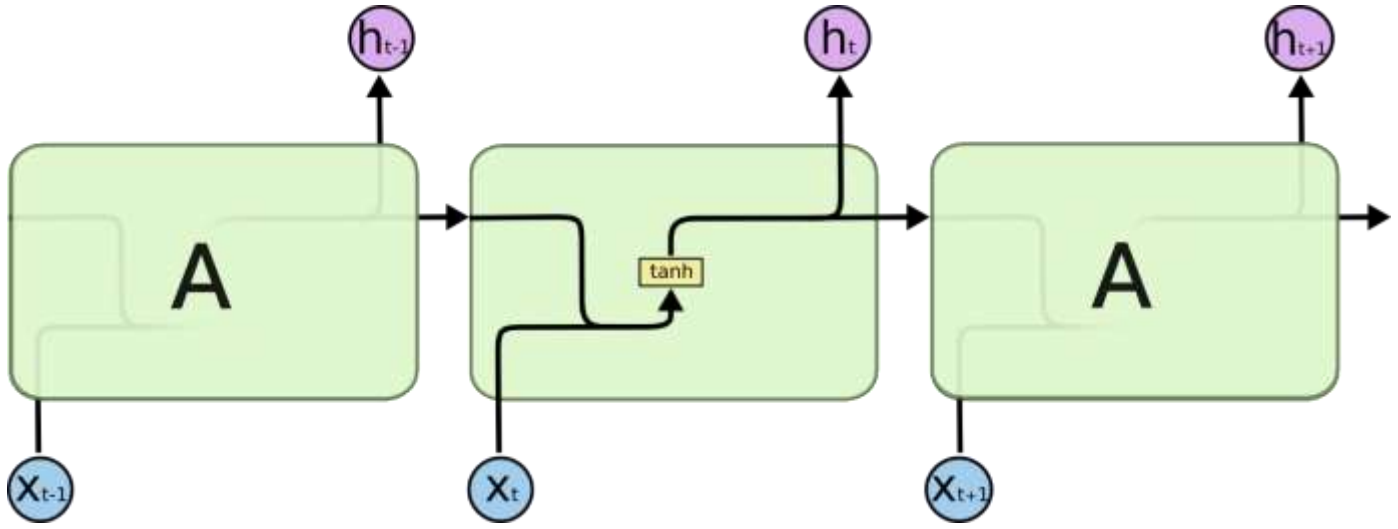
The Problem of Long-Term Dependencies

- Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, 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.



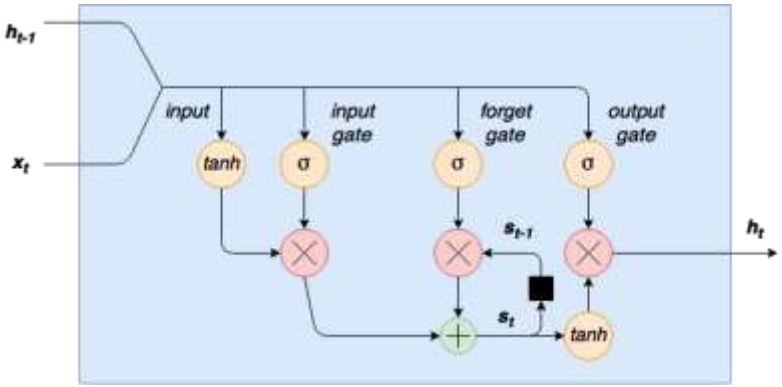
Moving from RNN to LSTM

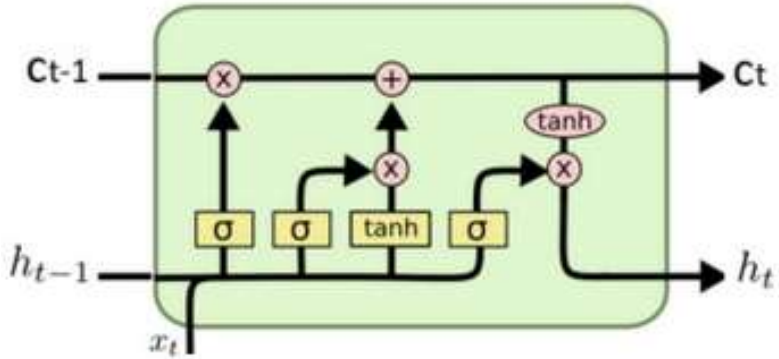
- All recurrent neural networks have the form of a chain of repeating modules of neural network.
- In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

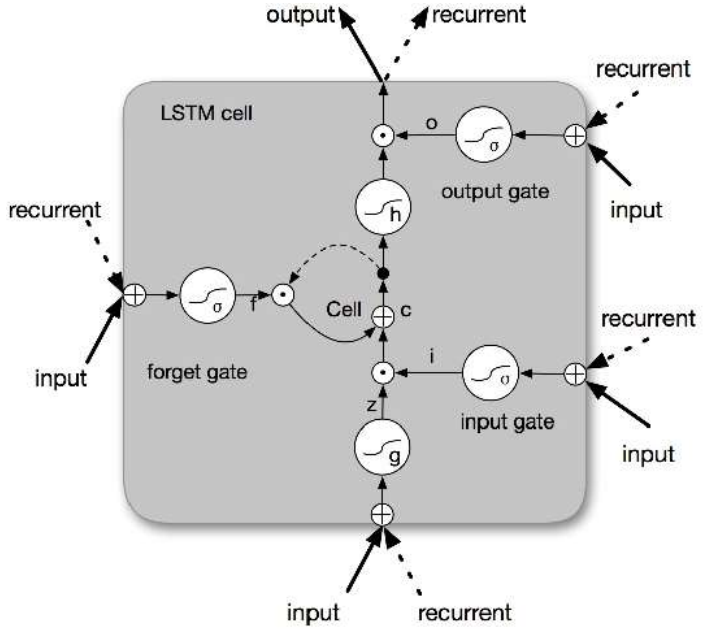


The repeating module in a standard RNN contains a single layer.

Same LSTM but….

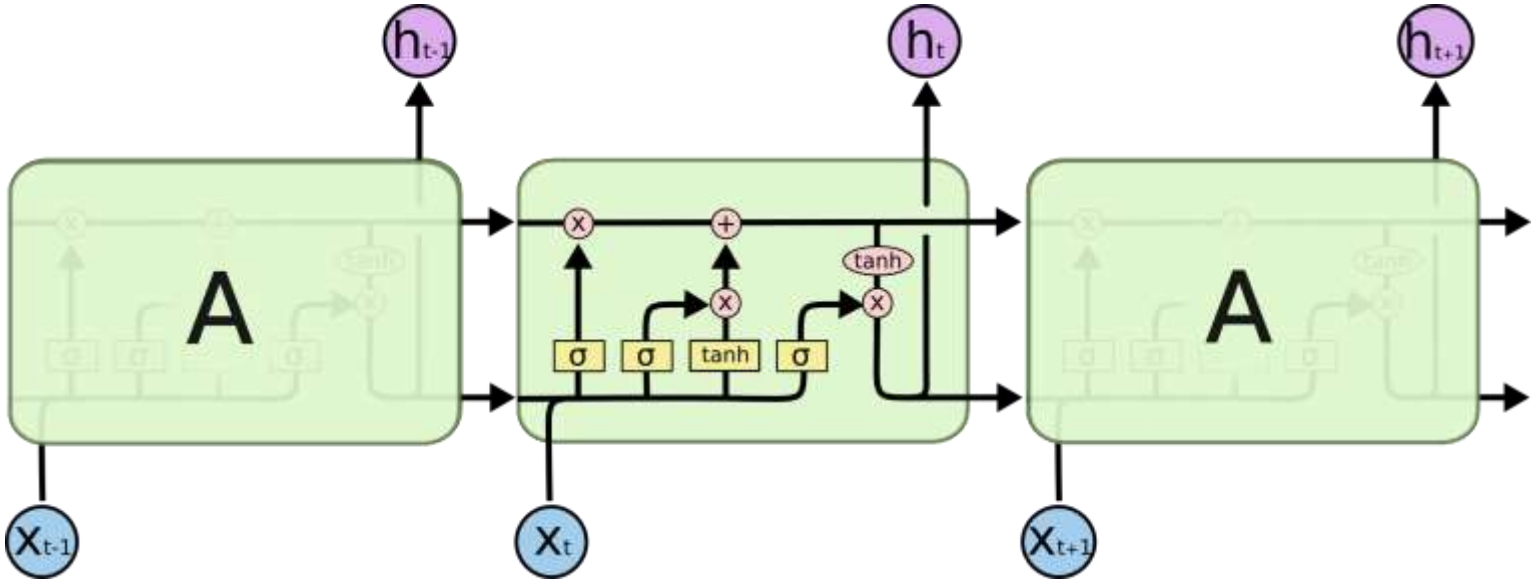






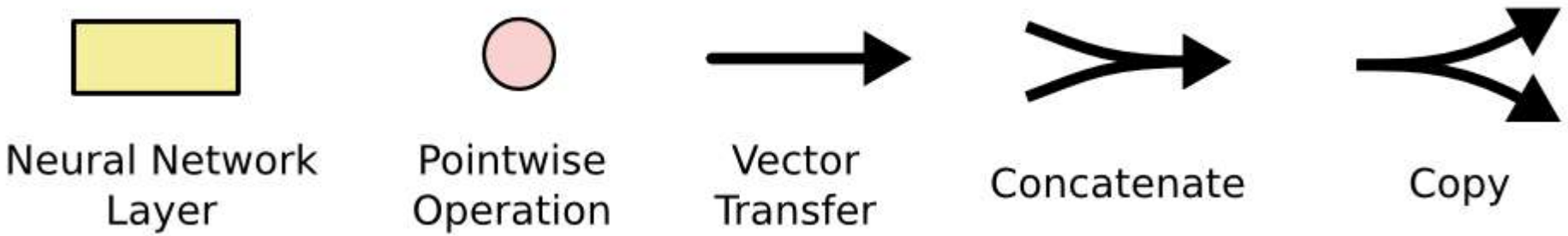
Long Short Term Memory (LSTM)

- LSTMs also have this chain like structure, but the repeating module has a different structure
- Instead of having a single neural network layer, there are four, interacting in a very special way



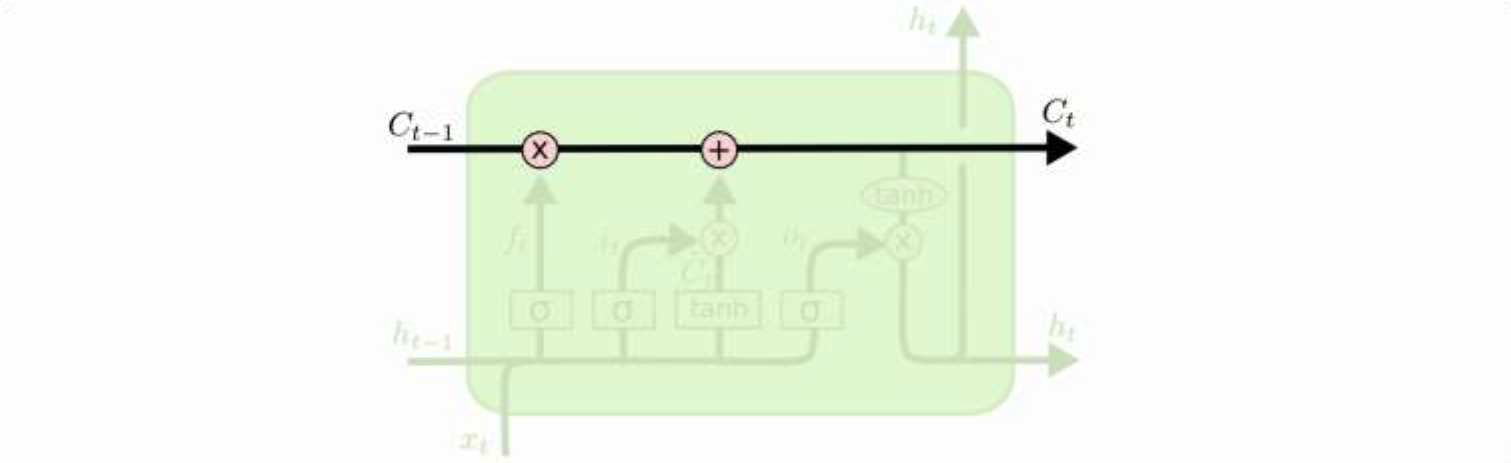
The repeating module in an LSTM contains four interacting layers.

Notations


Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

Cell State

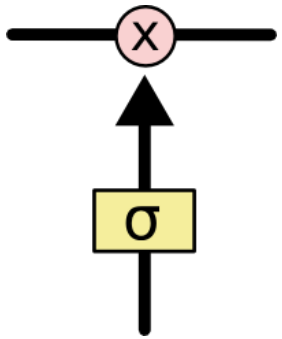
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.



- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Long Short Term Memory (LSTM)

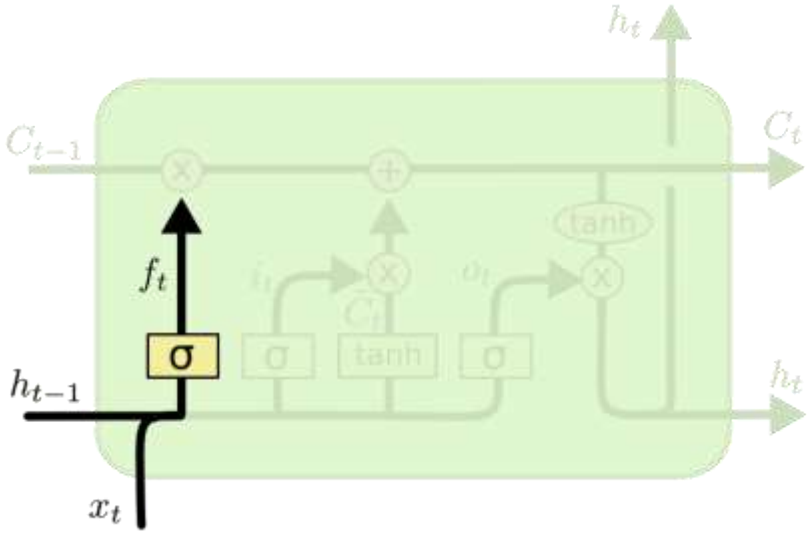
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”
- An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

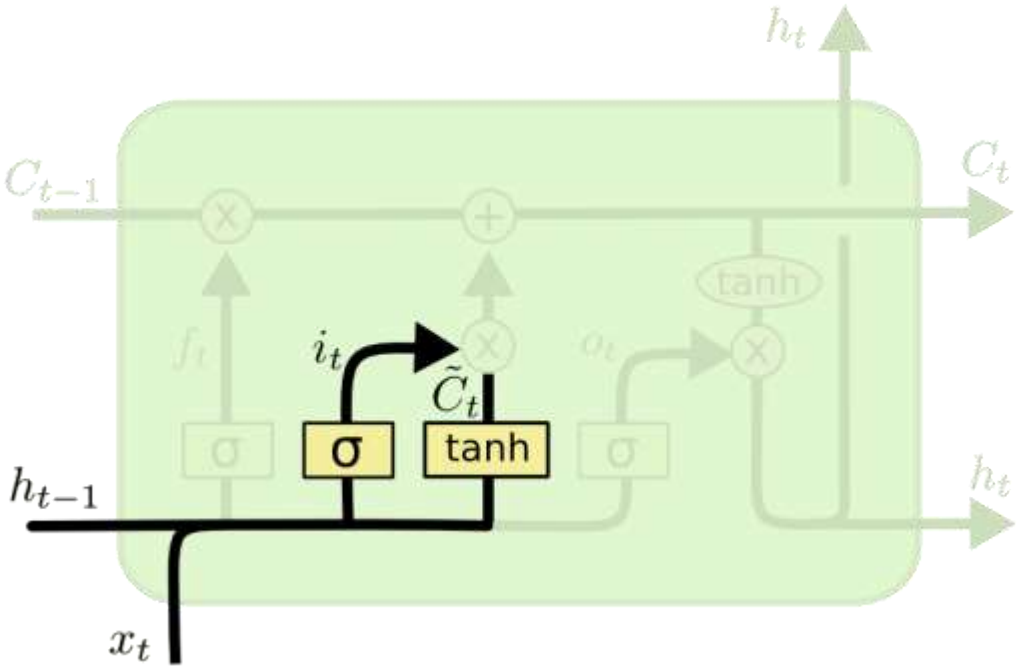
- The first step in our LSTM is to decide what information we're going to throw away from the cell state.
- This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
- 1 represents "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

- 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.

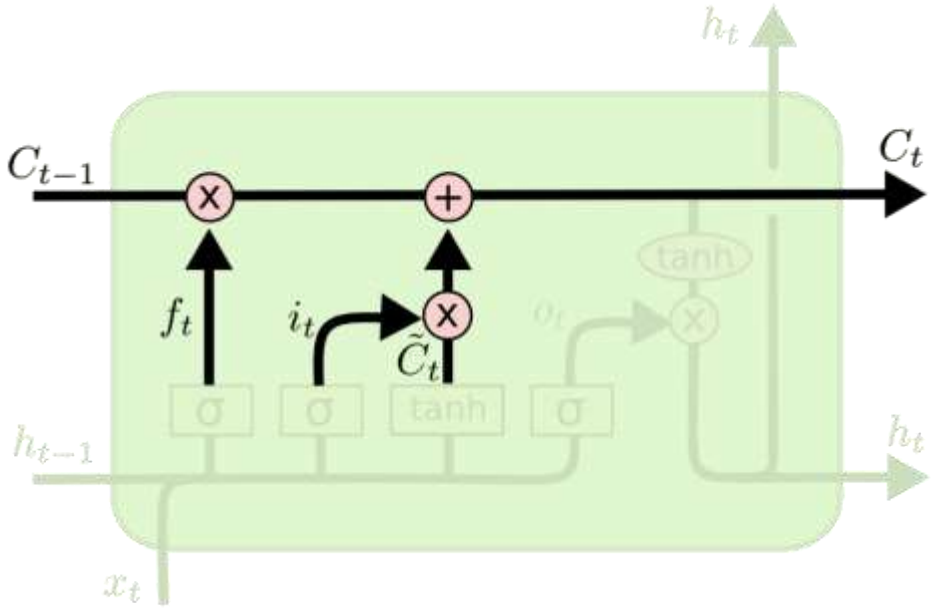


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Step-by-Step LSTM Walk Through

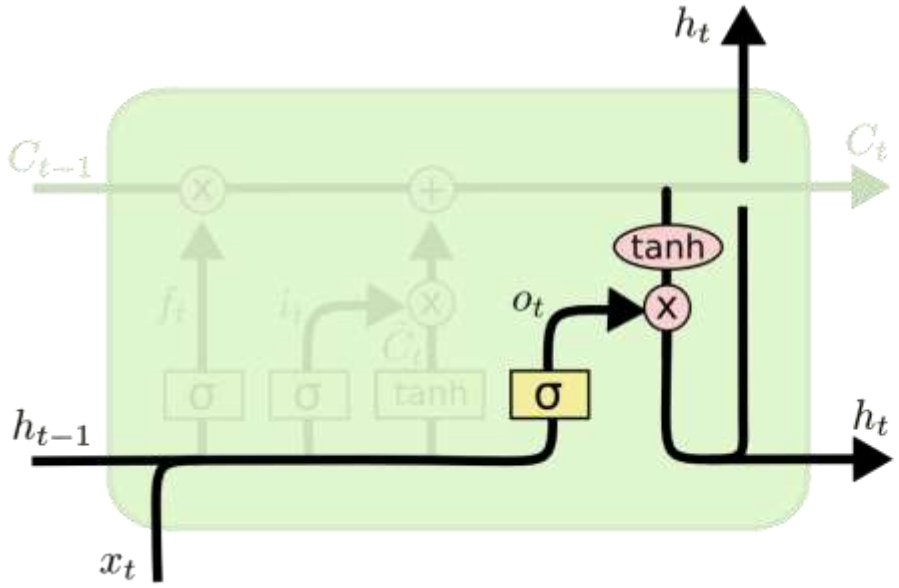
- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



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

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** (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.



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

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

LSTM using Keras

- **</>**

- keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, dropout=0.0, recurrent_dropout=0.0)

Advantages of LSTM

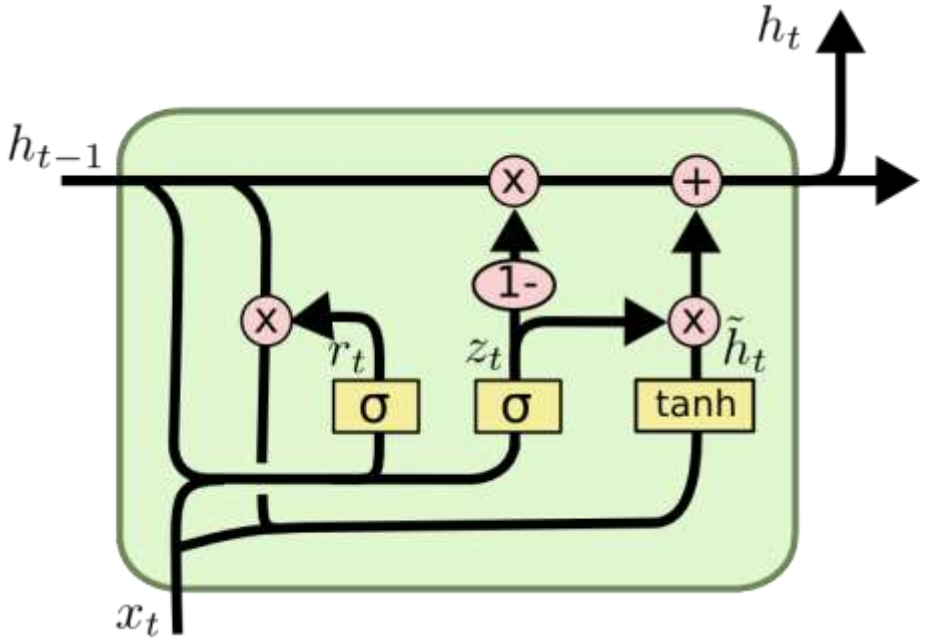
- Non-decaying error backpropagation.
- For long time lag problems, LSTM can handle noise and continuous values.
- No parameter fine tuning.
- Memory for long time periods

LSTM Conclusions

- RNNs - self connected networks
- Vanishing gradients and long memory problems
- LSTM - solves the vanishing gradient and the long memory limitation problem
- LSTM can learn sequences with more than 1000 time steps.

Gated Recurrent Units (GRUs)

- A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate."
- It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU using Keras

- **</>**

- keras.layers.GRU(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, dropout=0.0, recurrent_dropout=0.0)

LSTM vs GRU

- A GRU has two gates, an LSTM has three gates. What does this tell you?
- In GRUs
  - No internal memory (c_t) different from the exposed hidden state.
  - No output gate as in LSTMs.
- The input and forget gates of LSTMs are coupled by an update gate in GRUs, and the reset gate (GRUs) is applied directly to the previous hidden state.
- GRUs: No nonlinearity when computing the output.