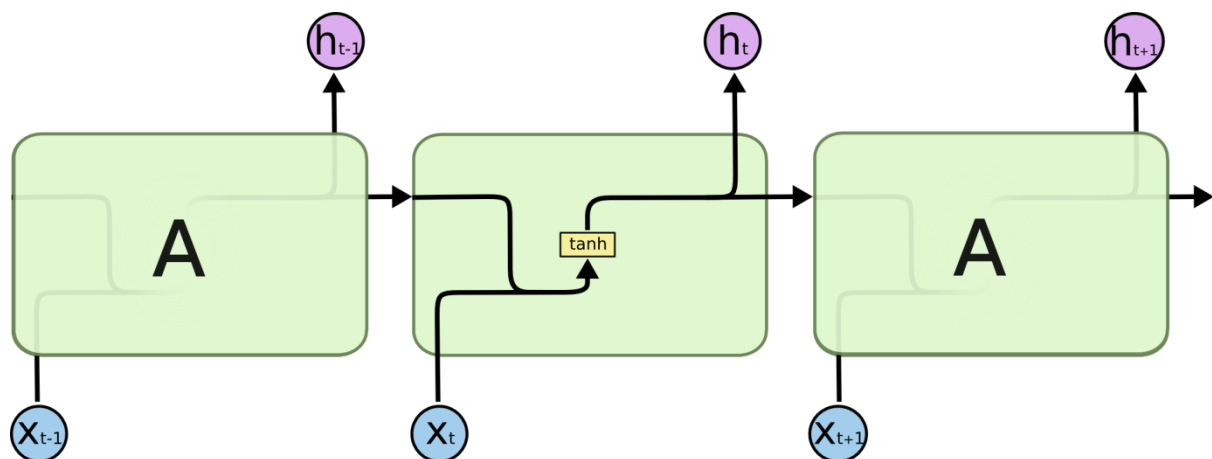


## LSTM Networks

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 & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn!

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.**

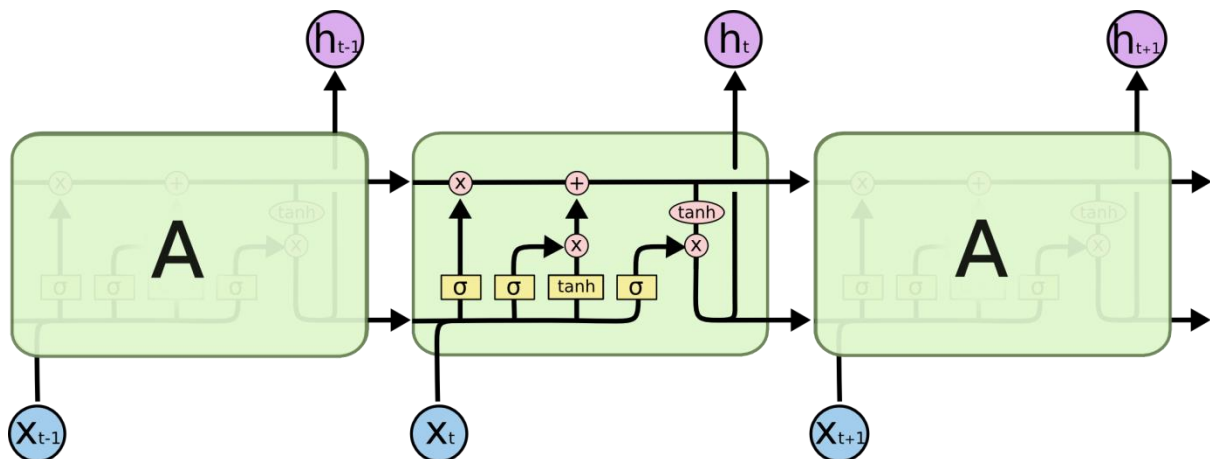
Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

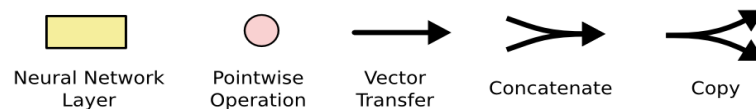
So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don't learn,

RNN's can forget what it seen in longer sequences, thus having a short-term memory.

Now, 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.**

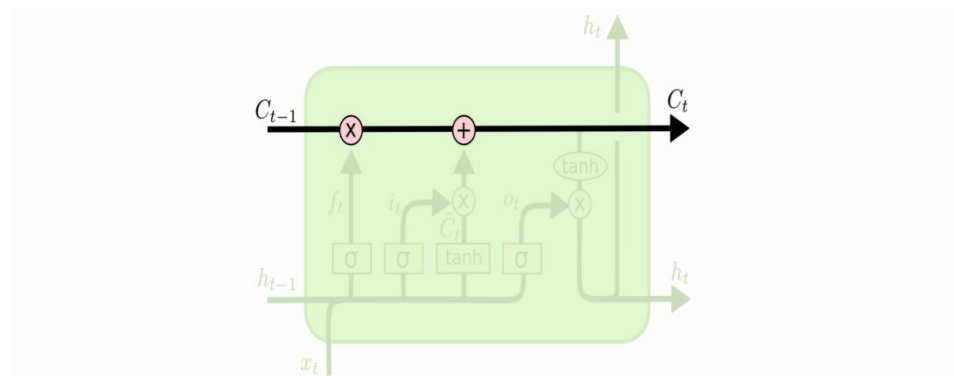


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent point wise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

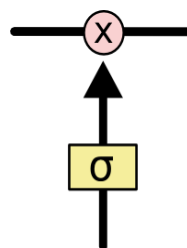
## The Core Idea behind LSTMs

The core concept of LSTM's are the cell state, and its various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the “memory” of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes

on its journey, information gets added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.



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



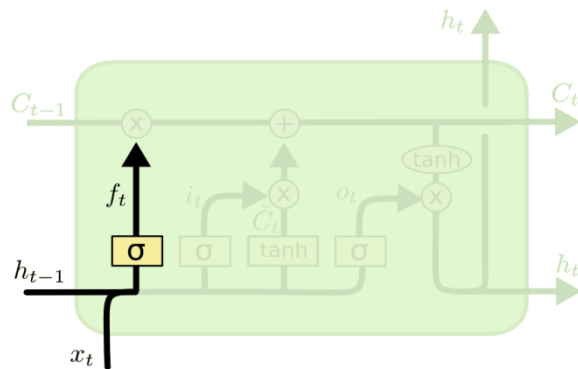
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

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  $C_{t-1}$ . A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”

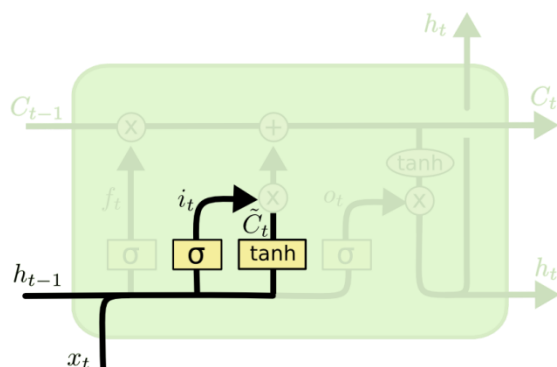
Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



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

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,  $\tilde{C}_t$ , which could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



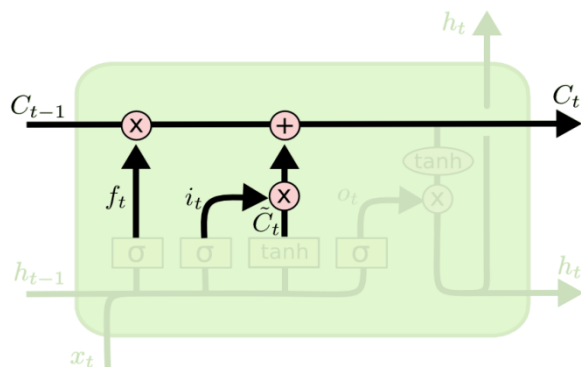
$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

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  $i_t \cdot \tilde{C}_t$ . This is the new candidate values, scaled by how much we decided to update each state value.

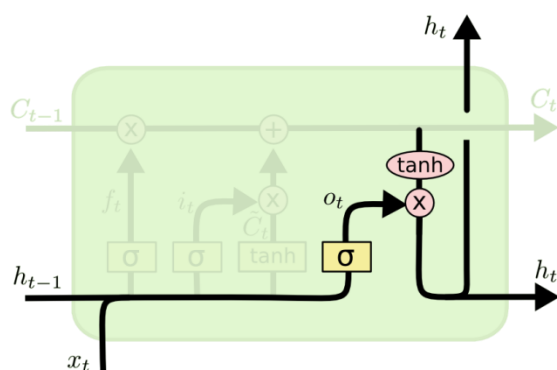
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



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

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.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



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

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

To review, the **Forget gate** decides what is relevant to keep from prior steps. The **input gate** decides what information is relevant to add from the current step. The **output gate** determines what the next hidden state should be.