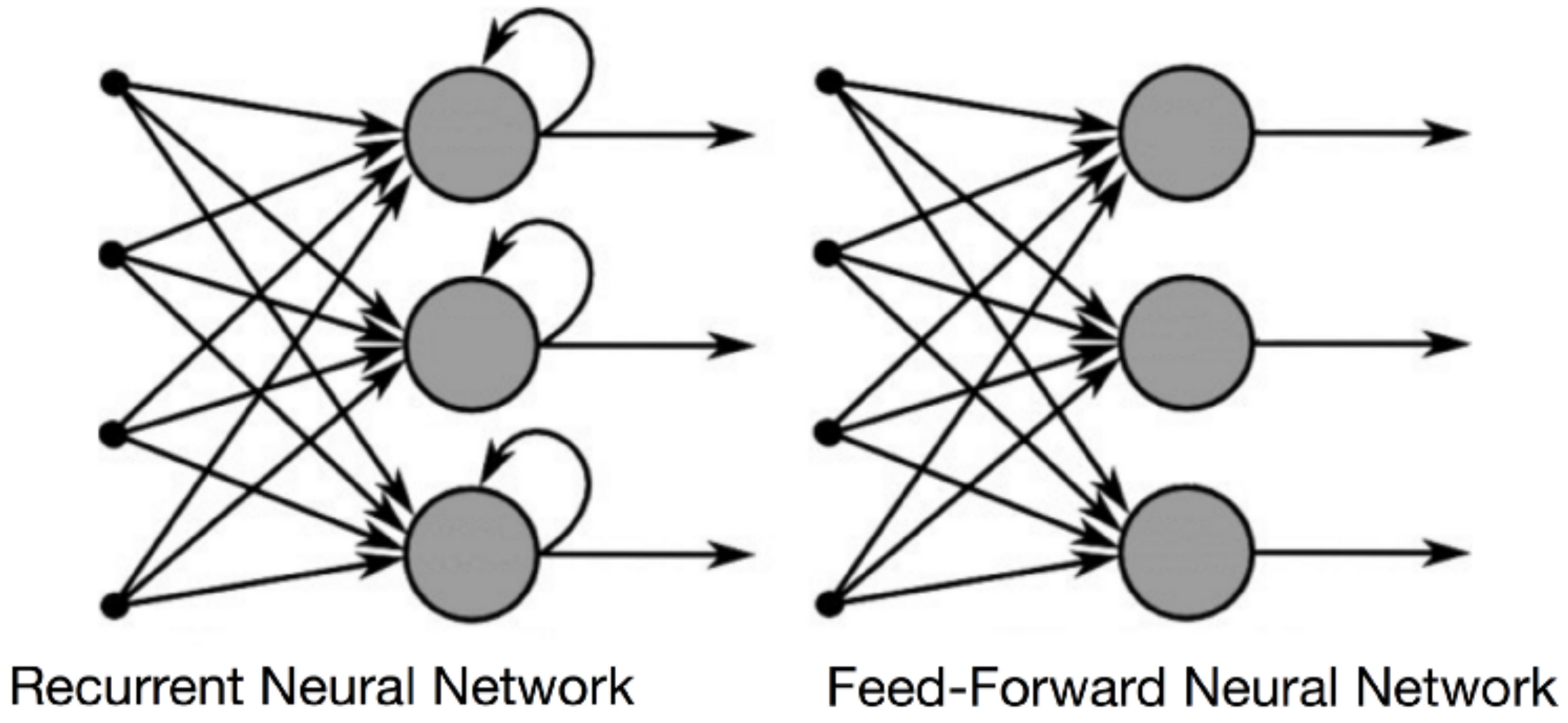
RNN

- The major shortcoming of traditional neural networks is their inability to model sequential data i.e remembering the past information.
- Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.
- This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.



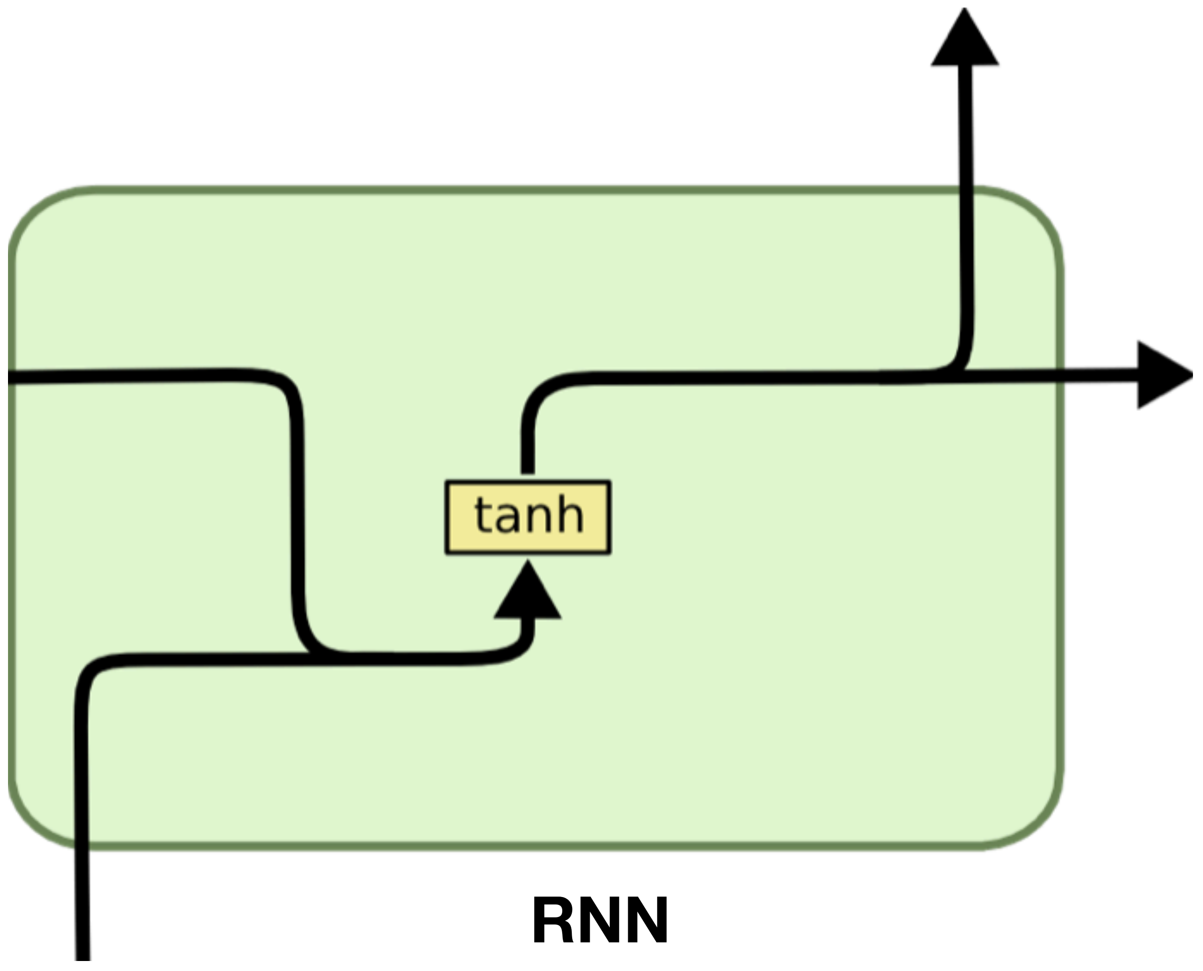
- RNNs are good in remembering short term information.
- For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “**the clouds are in the *sky*,**” we don’t need any further context – it’s pretty obvious the next 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.

- 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… 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.

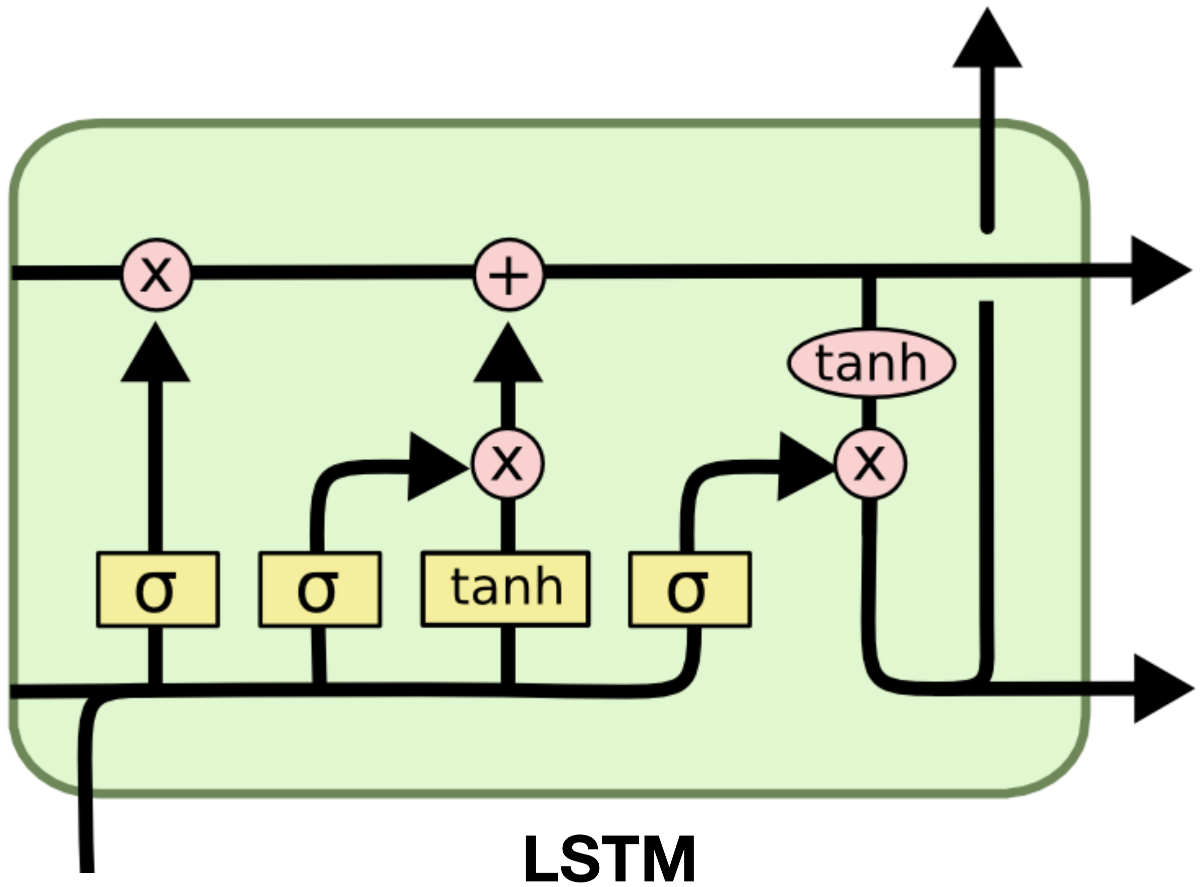
LSTM

Long Short-Term Memory

- LSTMs are explicitly designed to avoid the long-term dependency problem.
- They also have a chain-like structure like RNN, 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.

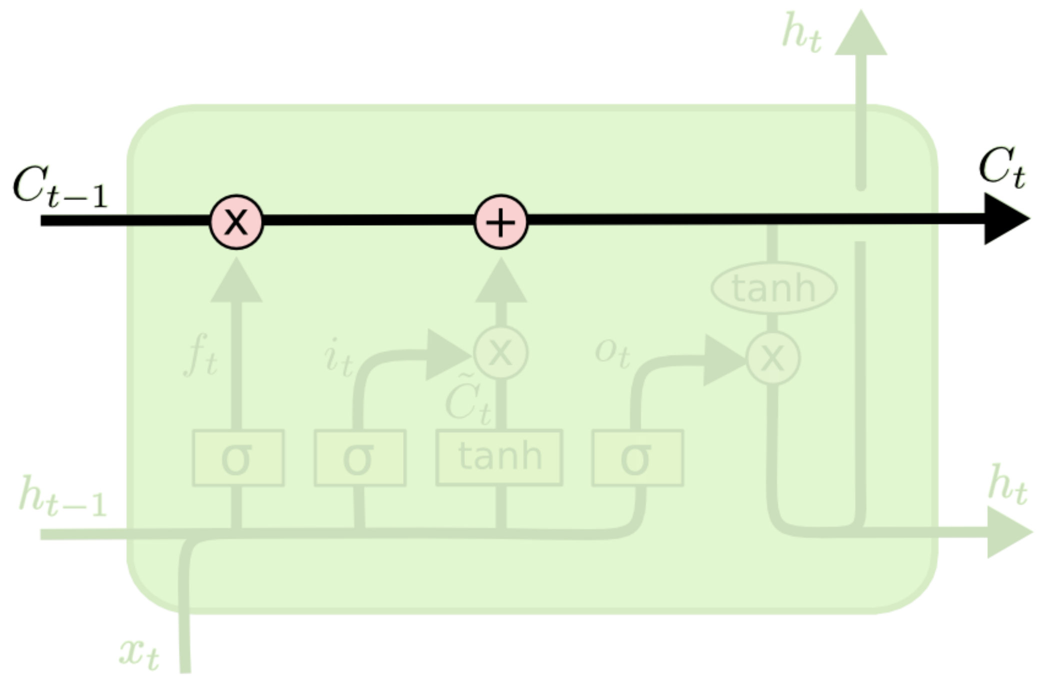


RNN

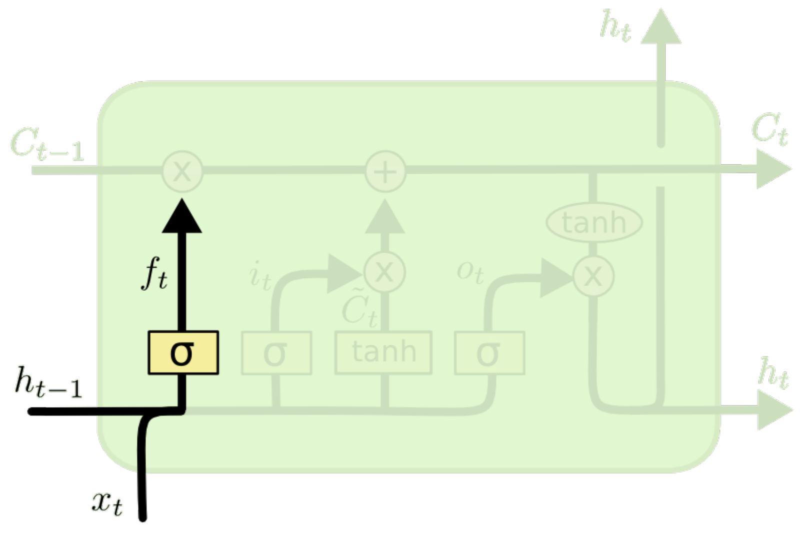


LSTM

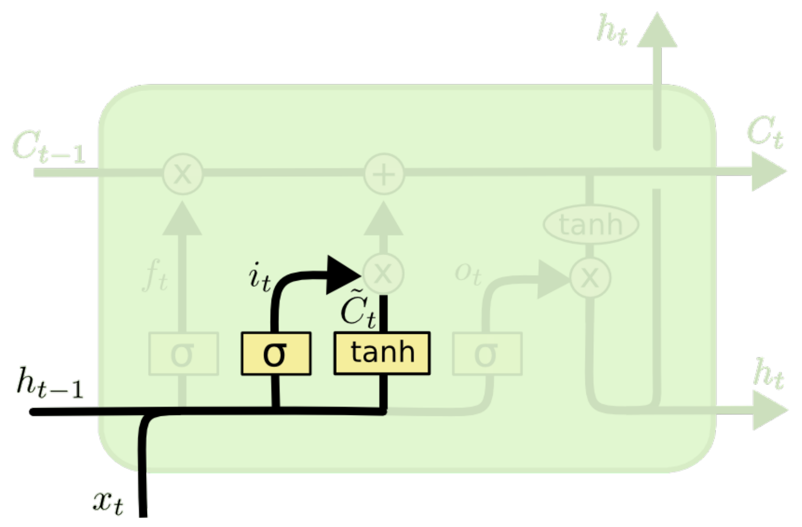
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. Information is added or removed from the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation.
- 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.



- The first step is to decide what information is needed to throw away from the cell state. This decision is made by a sigmoid layer called the "**forget gate layer.**"
- The next step is to decide what new information is going to be stored in the cell state.
- First, a sigmoid layer, "**input gate layer**" decides which values to update.
- Next, a tanh layer creates a vector of new candidate values, that could be added to the state.
- And then, these two are combined to create an update to the state.

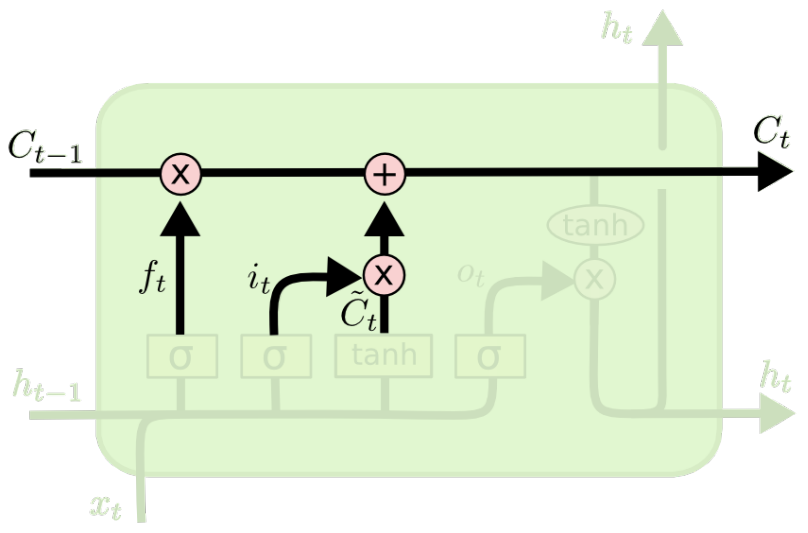


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



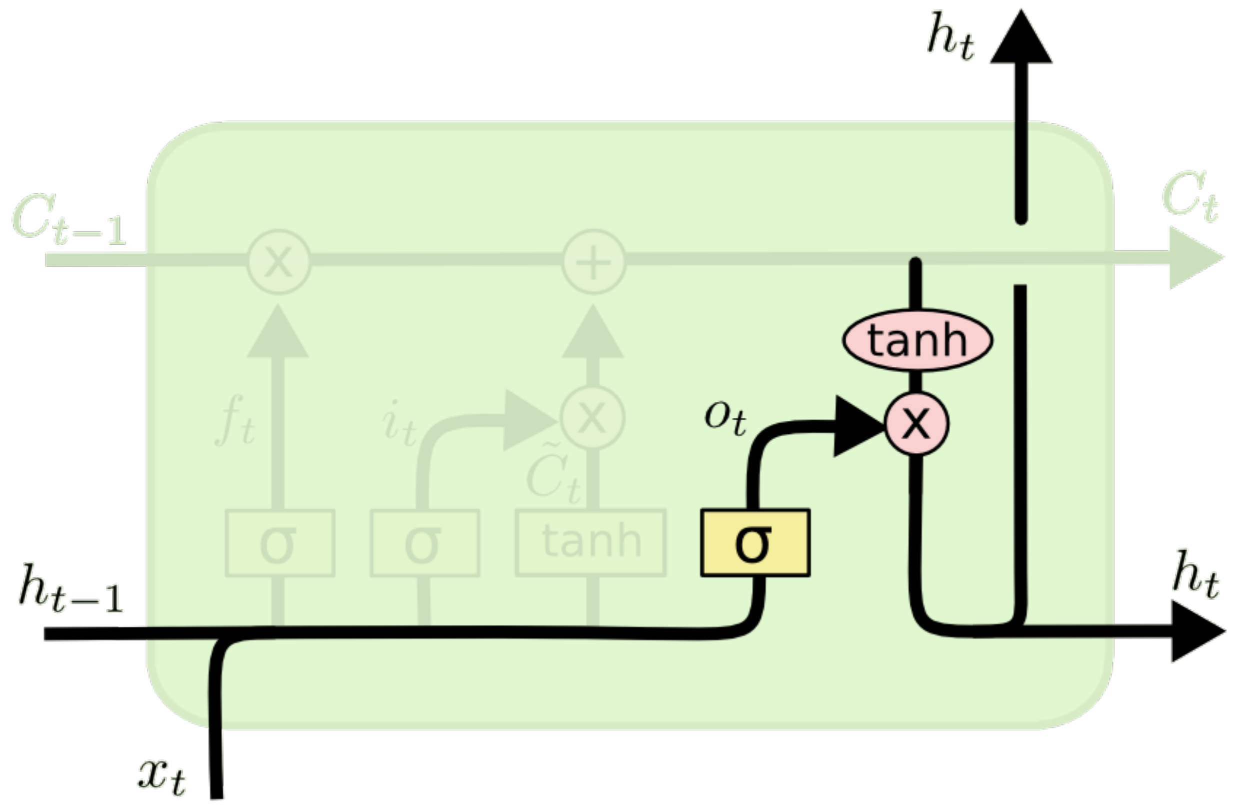
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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



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

- Finally, the output will be based on the cell state, but will be a filtered version.
- Filtered by a combination of input and hidden state.
- This job of selecting useful information from the current cell state and showing it out as an output is done via the “**output gate layer**”.



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

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

References

1. http://colah.github.io/posts/2015-08-Understanding-LSTMs/