Long Short Term Memory (LSTM)

Recurrent Neural Network (RNN)

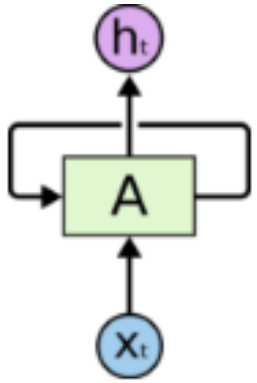
To understand RNNs, let's use a simple perceptron network with one hidden layer.

Loops ensure a consistent flow of information. A (chunk of neural network) produces an output h_t based on the input X_t .

A — Neural Network

X_t — Input

h_t — Output



Recurrent Neural Networks have loops.

Disadvantages of RNN

- RNNs have a major setback called vanishing gradient; that is, they have difficulties in learning long-range dependencies (relationship between entities that are several steps apart)

- In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them. The problem was explored in depth by Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.

Solution

- To solve this issue, a special kind of RNN called **Long Short-Term Memory cell (LSTM)** was developed.

Introduction to LSTM

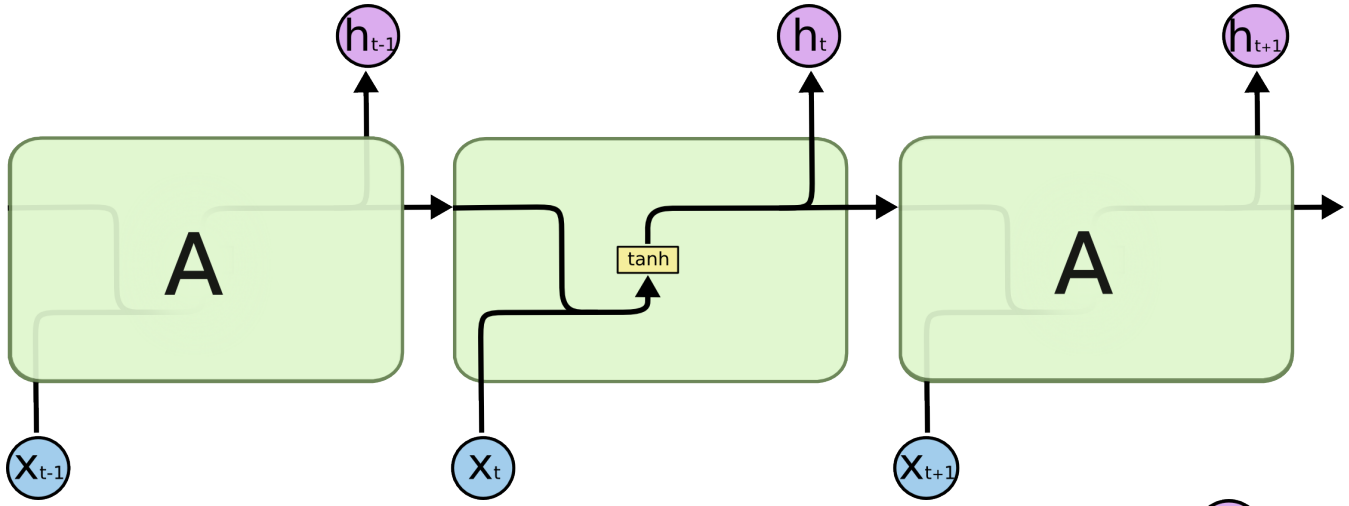
- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

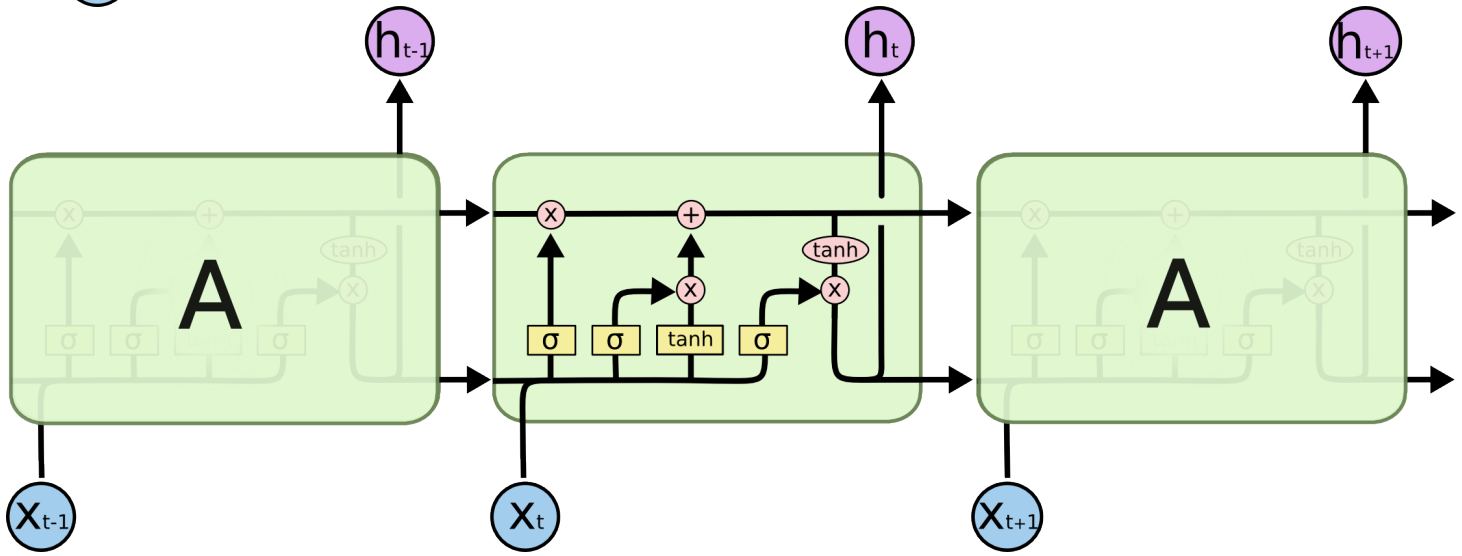
- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

- 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.

- 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.

Ordinary RNN vs LSTM

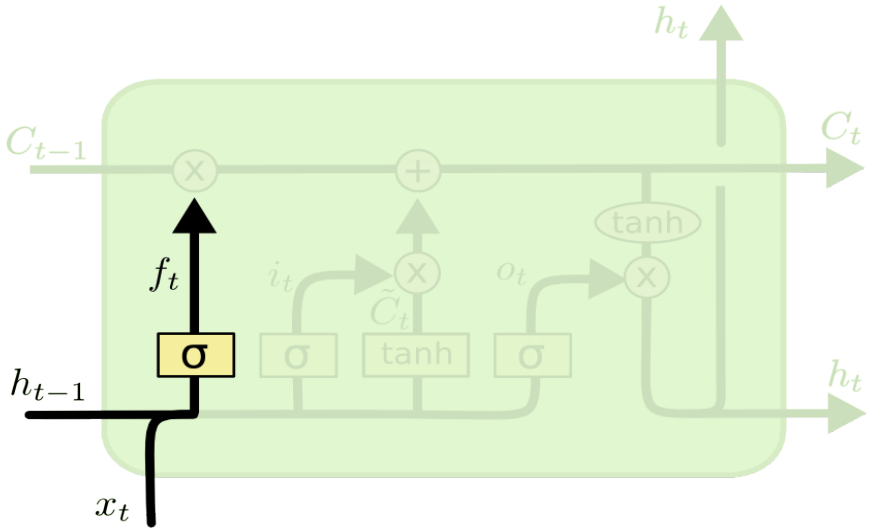




LSTM Walkthrough

- 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."

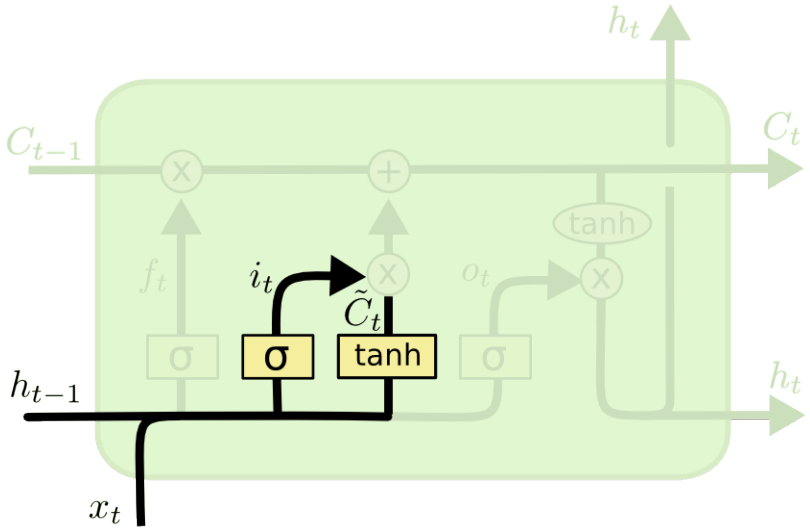
- It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

LSTM Walkthrough(contd.)

- 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.

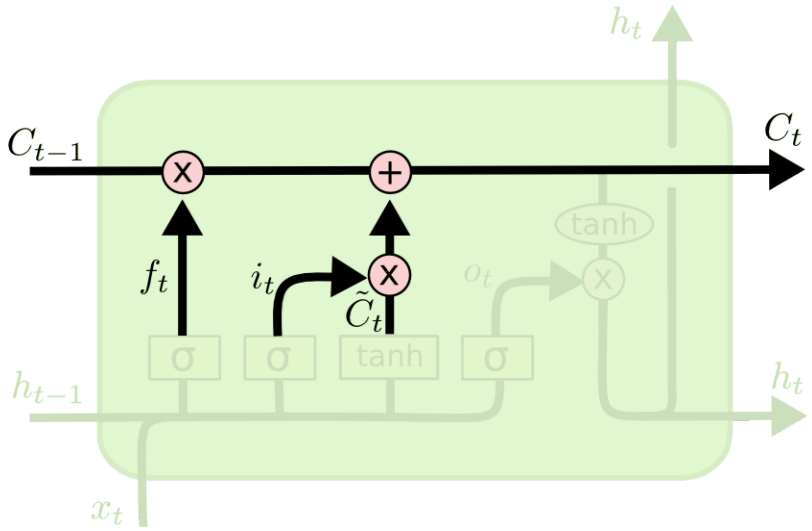


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTM Walkthrough(contd.)

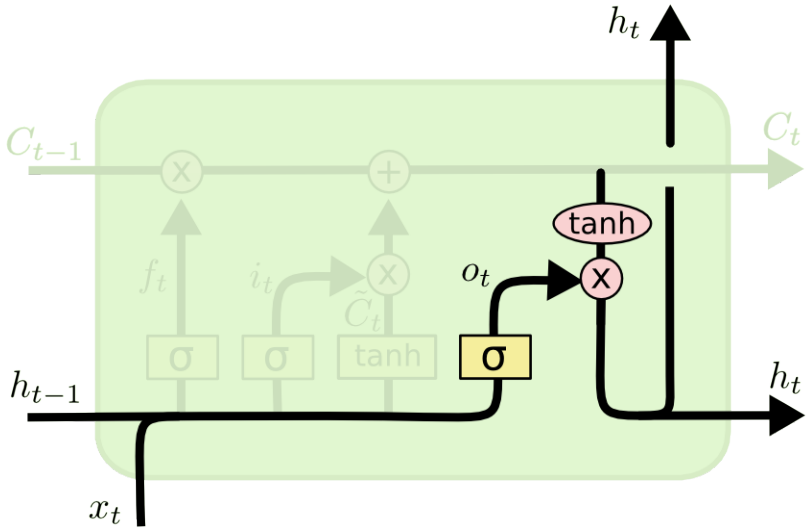
- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate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



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

LSTM Walkthrough(contd.)

- 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.



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

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

Acknowledgement

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://heartbeat.fritz.ai/a-beginners-guide-to-implementing-long-short-term-memory-networks-lstm-eb7a2ff09a27