Introduction to RNN and LSTM

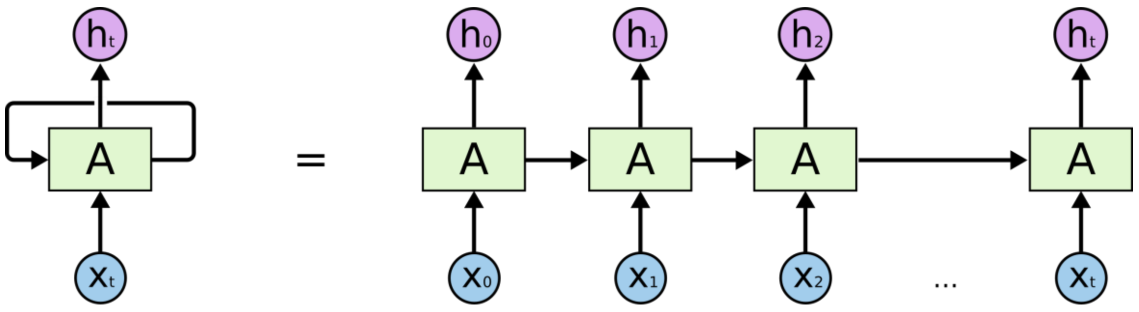
Research on language modeling has been an increasingly popular focus in recent years. Its ability to spontaneously recognize, summarize, translate, predict and generate text and other contents for an AI machine enables its broad application in various fields. However, text-based data, which we call sequential data, is difficult to model due to its variable length. Feed-forward neural networks work well with fixed-size input but do not take unfixed structure into account; convolutional neural networks, on the other hand, can deal with long sequences of data but are limited by the fact that they do not preserve the order of the sequence throughout the modeling process.

The Recurrent neural network model family generally performs better when modeling sequential data by using the output from the prior encounter as an input data source for the future. This blog post will walk you through three types of models in the recurrent neural network model family: RNN, LSTM, and Bidirectional LSTM and provide examples in the field of natural language processing.

What is a Recurrent Neural Network?

When humans read a block of text and go through every word, they don't try to understand the word starting from scratch every time, instead, they understand each word based on the understanding of previous words. In short, human thoughts have persistence over sequential data.

Traditional neural networks generally can't implement such persistence. On the other hand, recurrent neural network addresses this issue. By feeding the output of one layer to itself and thus looping through the exact same layer multiple times, RNNs allow information to persist through the entire model.



Unrolled Recurrent Neural Network Structure

We can see how past observations flow through the unfolded network. Each cell in an RNN has two inputs: the past and the present, where the past is the output of the previous RNN cell and represents short-term memory.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They perfectly represent the natural architecture of neural network to use for text-based data.

The Problem of Long-Term Dependencies

One of the features that distinguishes RNNs from other neural network structures is that in theory, they are able to connect previous information to the present task, in the case of text data, using previous words to reveal the information about the present word in one complete sentence. However, unfortunately in practice, RNNs do not always do a good job in connecting the information, especially as the gap grows.

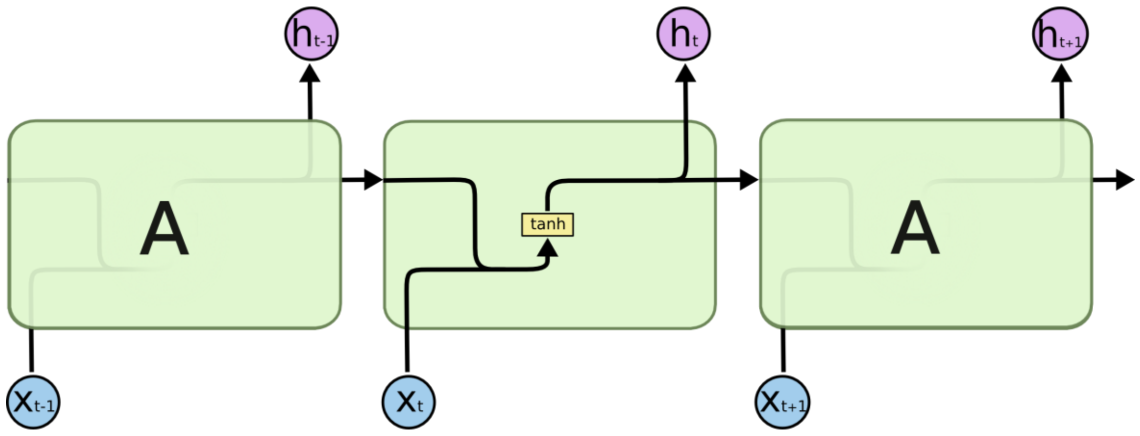
If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – we could derive from the previous five words that “sky” is the next word. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.

On the contrary, 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. However, RNNs generally do not do a good job of modeling within such an situation. Alternatively, we use LSTMs to solve this problem.

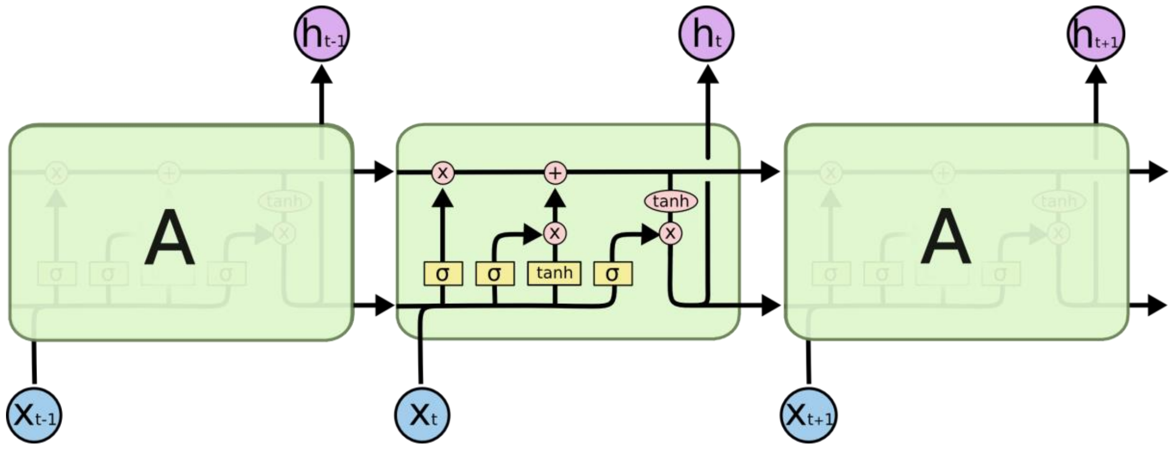
A Quick Peak into LSTMs

Long Short Term Memory networks (LSTMs) are a special kind of RNN, capable of learning long-term dependencies. 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. Connecting information among long periods of time is practically their default behavior.

LSTMs, like RNNs, also have a chain-like structure, but the repeating module has a different, much more sophisticated structure. Instead of having a single neural network layer, there are four interacting with each other.

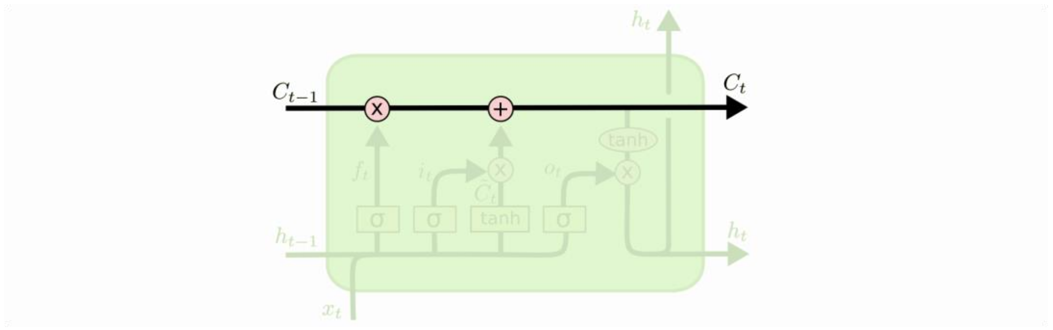


An RNN Module



An LSTM Module

Inside one LSTM module, the key component that allows information to transfer through the entire model is called the **cell state**.

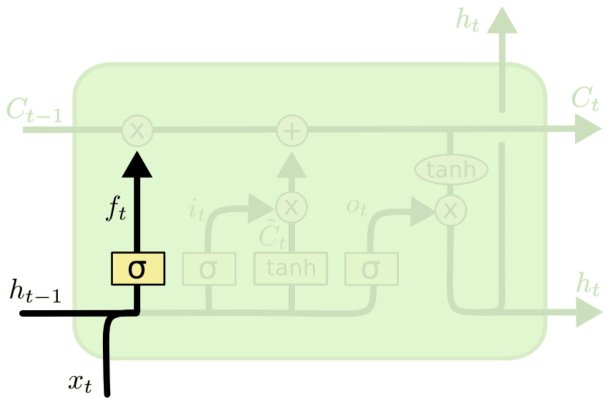


Cell State

In addition to transferring information, the module has the ability to add or remove information to the cell state, which is regulated by structures called gates. In a standard LSTM module, there are three gates in total.

NOTE: σ refers to the sigmoid function.

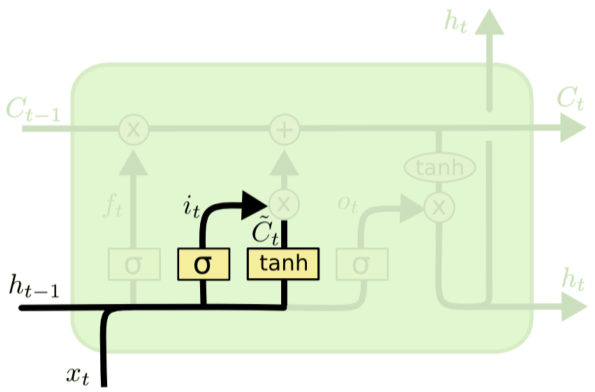
Forget gate:



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

In the case of a language model, 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 decide how much we want to forget about the gender of the old subject through the *forget gate*.

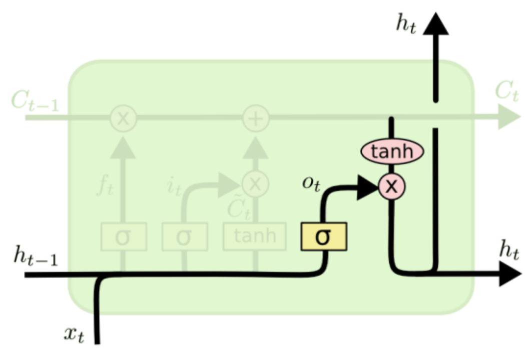
Input gate:



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Applying the above case, *input gate* decides how much we record the gender of the new subject to replace the old one (that we are forgetting in the *forget gate*).

Output gate:



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

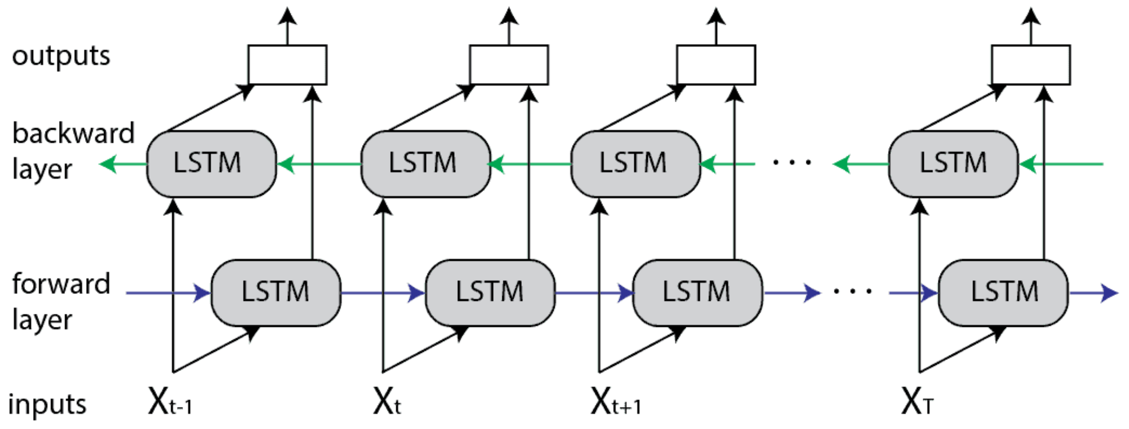
Applying the above case, this is where we'd actually drop the information about the old subject's gender and add the new subject's gender through the *output gate*.

In short, LSTM uses separate paths for long-term memory and short-term memory and uses the three gates: forget gate, input gate, and output gate to regulate cell state so that it tackles RNN's long-term dependency issues by filtering useless information out of the network.

LSTM vs. Bidirectional LSTM

A Bidirectional LSTM (BiLSTM) is a recurrent neural network used primarily on natural language processing. Unlike standard LSTM, the input flows in both directions, and it's capable of utilizing information from both sides, which makes it a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence.

BiLSTM adds one more LSTM layer, which reverses the direction of information flow. It means that the input sequence flows backward in the additional LSTM layer, followed by aggregating the outputs from both LSTM layers in several ways, such as average, sum, multiplication, or concatenation.



Unrolled Bidirectional LSTM Structure

This type of architecture has many advantages in real-world problems, especially in NLP. The main takeaway is that every component of an input sequence has information from both the past and present. With that being said, BiLSTM can produce a more meaningful output, especially in the case of building language models, since words in a text block are often connected in both ways - with previous words and future words.

For example, in the sentence "Apple is something that …", the word *Apple* might be about the apple as fruit or about the company Apple. The traditional LSTM won't be able to know what *Apple* means, since it doesn't know the context from the future.

In contrast, most likely in the following two sentences:

"Apple is something that competitors simply cannot reproduce."

and

"Apple is something that I like to eat." ,

Bidirectional LSTM can do a great job distinguishing apple the fruit from Apple the tech company, utilizing the information from its future context. So we can clearly see that the bidirectional LSTM model is beneficial in many NLP tasks, such as sentence classification, translation, and entity recognition. In addition, it finds its applications in speech recognition, protein structure prediction, handwritten recognition, and similar fields.

However, it's worth mentioning that bidirectional LSTM is a much slower model and requires more time for training compared to unidirectional LSTM. Therefore, for the sake of reducing computation burden, it is always a good practice to implement it only if there's a real necessity, for instance, in the case when a unidirectional LSTM model does not perform beyond expectation.