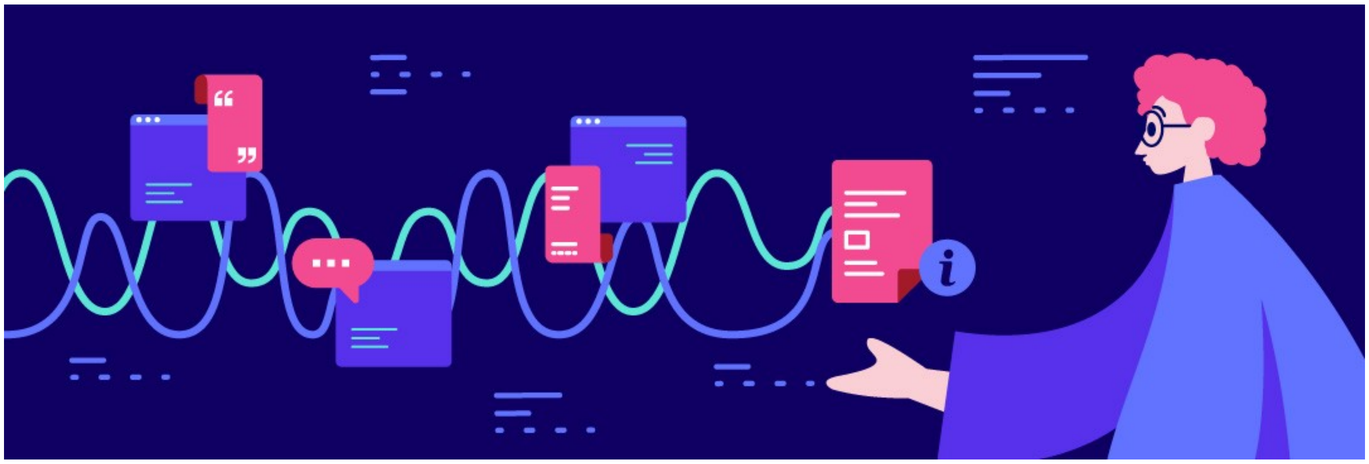


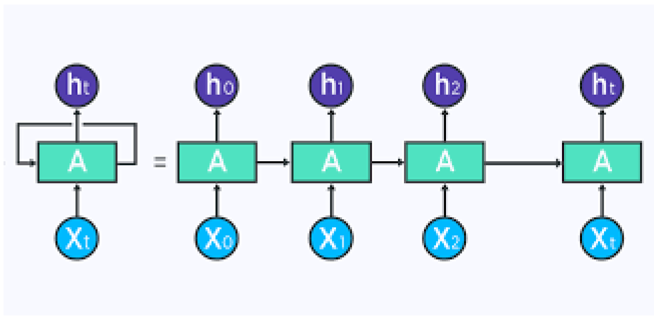
Long Short-Term Memory Networks

Vinithavn Follow
Nov 30 · 8 min read

Introduction

Neural networks are designed to mimic the behavior of human brains, to understand various relationships in the data. These networks have the power to understand complex non-linear relationships and can help us to make more intelligent decisions. Neural networks are used in fields like image processing, natural language processing, etc., and can outperform the traditional machine learning algorithms. But one basic drawback with the traditional neural network is that it cannot memorize things.

Let’s say, we are playing a game and we need a model to predict the next move of a player. This depends a lot on the previous moves and traditional neural networks will not perform well here. In this case, we need some model that can memorize the previous events and takes smart decisions accordingly. This is where Recurrent Neural Networks (RNN) came into the picture. RNNs can store the previous state information. It can then use it to predict the next event, which in this case, is the next move of the player.



On the left side of the above figure, you can see there is an input Xt and output ht. The loop indicates that this network is repeated for t times, as shown in the right figure. The neural network A takes the input X at a time t = 0 and produces the output h0. Now the information from A at t=0 and input X1 is passed to the next timestamp t=1 to generate output h1. Seems like a complete solution to the memorization problem.

RNNs have been used in a lot of sequence modeling tasks like image captioning, machine translation, speech recognition, etc.

Drawbacks of RNN

As we see, RNNs were gaining popularity and were used in most sequence-related tasks. But there were some disadvantages to this model.

Vanishing Gradients

RNNs are trained through Backpropagation through time (BPTT). Like Backpropagation, here also the weights of the neural network get updated through the gradients. But in RNN, we backpropagate through time and the layers. If the sequence is very large (many time steps) and if the neural network has more than one hidden layer, then there is a high chance that the gradients will become smaller and smaller as we backpropagate. This will eventually lead to the vanishing gradient problem. Once this problem occurs, the weight will be updated very slowly, preventing the network from learning. The vanishing gradient problem can occur in Deep neural networks also. But this effect is very common and much worse in RNN because here we need to backpropagate through time as well.

Handling long term dependencies

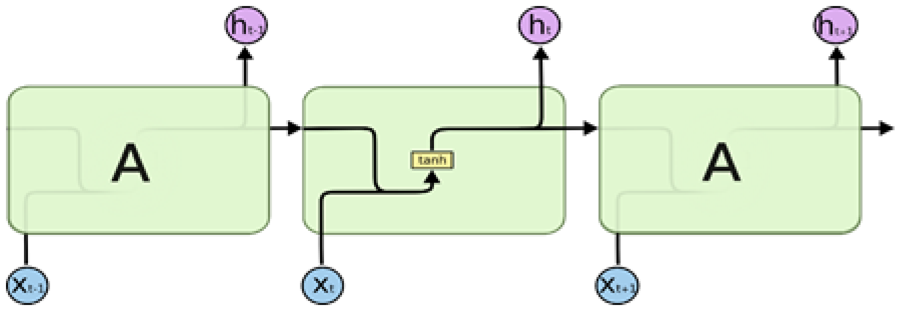
RNNs can make use of the previous states to predict the next event. This will be very useful in NLP, where the prediction of a particular word depends upon the context or the previous words. Consider the sentence "Apple is a fruit". Now let's say we are predicting each of the words through an RNN network. For predicting the word "fruit", we need the context word "Apple". Here since the context word is just before the event. So, in this case, RNNs will perform well.

But consider the sentence "I grew up in France and I speak fluent French". Here to predict the word "French", the context word is "France". These are called long-term dependencies and are very common in language models. It is found that RNNs poorly handle long-term dependencies. The performance of RNNs will drop as the gap between the event and context increases.

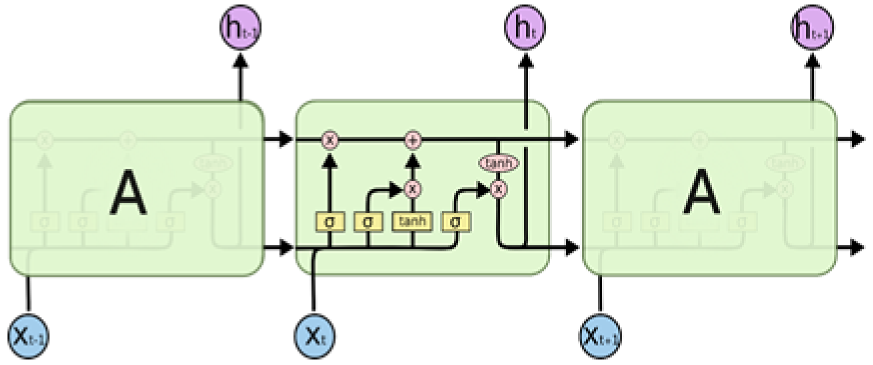
Introduction of LSTM

Long Short-Term Memory networks or LSTMs are specifically designed to overcome the disadvantages of RNN. LSTMs can preserve information for longer periods when compared to RNN. LSTMs can also solve the vanishing gradient problem. Let us see how LSTM can achieve these.

When we consider RNN, it has a neural network layer or a module that repeats for t time steps. Inside this module, RNN has a neural network with a simple activation layer. The activation layer can be tanh or sigmoid or any other function. This will be better understood from the below figure.



In LSTMS, however, the architecture is not that simple. Consider the figure below that shows the architecture of an LSTM network.



Don’t stress out. We will try to understand each of these units one by one in the next session.

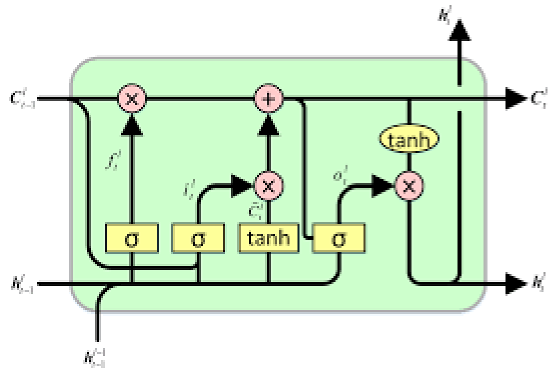
Now let us go back to our first example of predicting the next move in a game. The next move can be considered as an event that we need to predict. Obviously, there are various long-term and short-term dependencies associated with this event. You can consider these dependencies as context, or in this case, the tools, and accessories that we collected from the previous levels, or the previous incidents that occurred during the game, etc.

Let’s say we have a current event occurring and we need to predict the next event in the game. We also have the long-term and short-term information which we collected during the game. Long-term memories are the memory that is collected from a long time back and short-term memories are the information that is collected a few timestamps back. For predicting the next event obviously, we need the current event. But only some long-term and short-term memories will be useful. How do we get this?

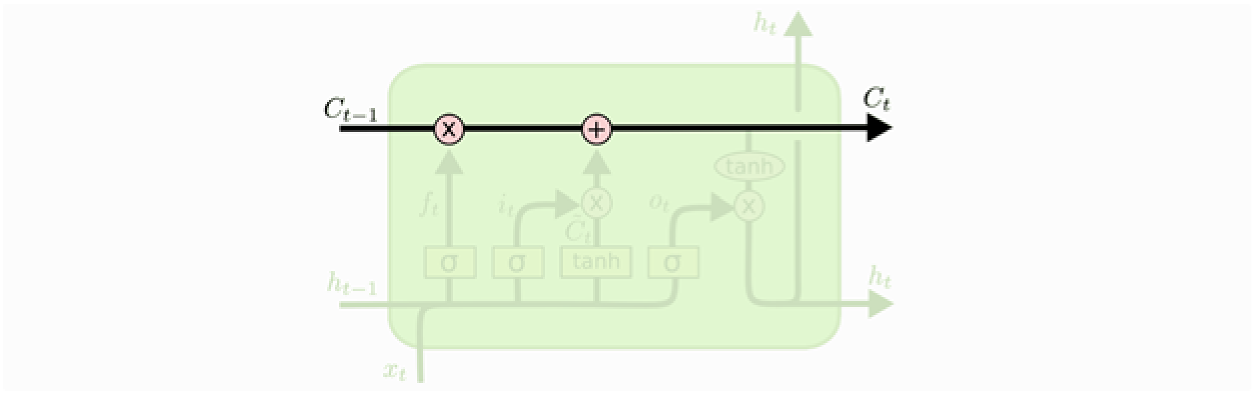
In LSTMs, we will use this long-term memory and short-term memory, and the current event, to generate a new modified long-term memory. While doing this, we will only remember those things which will be useful, and we will discard all the irrelevant information. Similarly, we will update the short-term memory by using some information and discarding others. In short, at each time step, we will filter the memory which needs to be passed to the next time step. This modified information is used to predict the next event. This will be clearer once you go through the next session.

Architecture of LSTM

Now let us dive deep into the LSTM model architecture and try to understand how it will handle the long-term and short-term dependencies. Consider the figure below.



You can see there are two outputs from one LSTM unit. Ct and ht. If you remember in RNN we only had one output, which is ht. The hidden state ht is the short-term memory that is obtained from the immediately previous steps. Now, what is this additional output in LSTM? The vector Ct is known as the cell state.

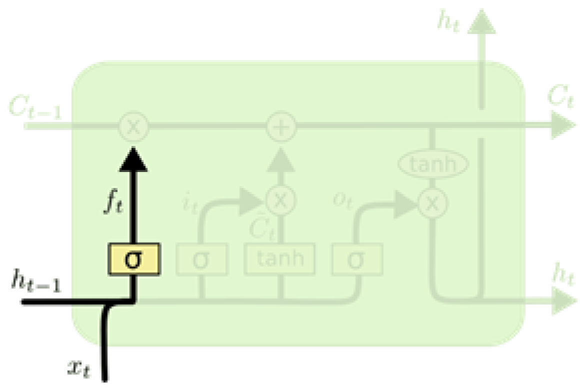


Now cell state in an LSTM is responsible for storing the long-term memory events. LSTMs will make use of a mechanism called gates to add and remove certain information into this cell state. Let us try to understand this in detail.

Forget Gate

As we discussed earlier, we can add and remove information to the cell state using gates. The below figure is nothing but the forget gate which filters only the required information and removes the rest. How is this achieved?

In the figure, the cell state or the long-term memory from the previous time step is multiplied with a function ft to get the new filtered memory where ft is the forget-factor. The forget -factor is calculated using the formula as shown below.

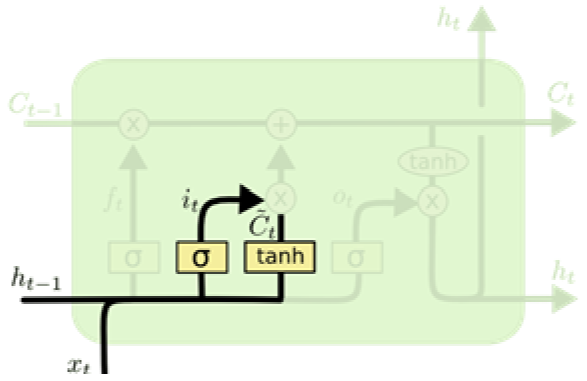


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The short-term memory or ht-1 from the previous timestamp and the current event is used to calculate the forget-factor. The short-term memory and the current event are concatenated, and a sigmoid layer is applied on top of that vector. The sigmoid function will produce an output ranging from 0 to 1 which will then get multiplied with each value in the previous cell state. A 0 value indicates that the information will be completely discarded and a value of 1 indicates that the information is kept as it is.

Learn gate

Now that we know what information to discard, our next goal is to find what new information we need to add. This is done through the learn gate. Now learn gate has two parts



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

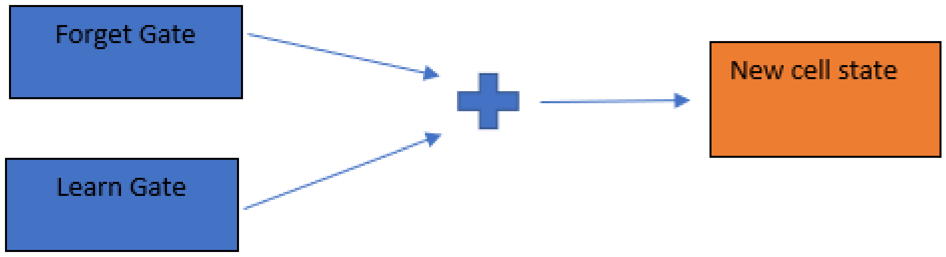
1. The previous short-term memory and the current event are concatenated and then passed through a tan h layer. This will generate new values c~t which is the new information. Now here also we don't require the whole new information. How do we ignore some part of this?

2. How much each of the new candidates gets updated is determined through another Forget gate. The output from forget gate will get multiplied with our new information and generate the final output

Now, why are we using tanh activation function in some places? Tanh activation function will output vectors (in range -1 to 1) which are centered around 0. This will distribute the gradients very well and allow the cell states to run longer. This eventually will solve the vanishing or exploding gradients problem.

Remember Gate

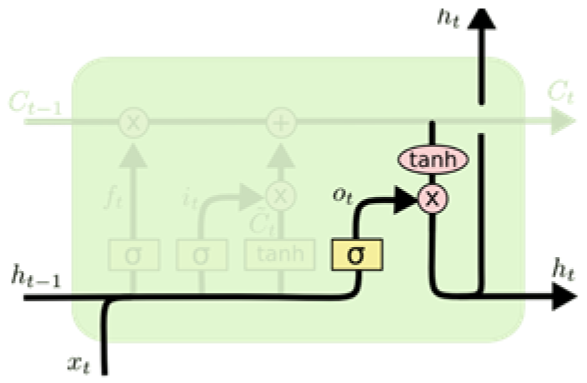
Now that we know what to keep and what to discard, it is time for us to update the new cell state or long-term memory. How do we do this? We just take the output from forget-gate and learn gate and we will add them.



We are almost there. Now the only step remaining is to calculate the output.

Output Gate

The output is a filtered version of the cell state. The cell state value from the remember gate is multiplied with a tanh activation function. The output from tanh will be between -1 and 1. We then multiply the output through a sigmoid function again, to forget some of the values.



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

That's it! We are done with the architecture.

Other variations of LSTM architecture

Apart from the above architecture, there are some other networks that will also work well in sequence data. Some of them are

· Gated Recurrent Unit (GRU)

· Peephole LSTMs

· Depth Gated RNNs

· Clockwork RNNs

I will not go into the details of these models. But you can always read about them in the above links.

Final thoughts

LSTM is a great milestone in the field of NLP and sequence models. But like all other models, LSTMs are also not perfect. The longer training times, large memory requirements, unable to parallel training, etc. are some of the drawbacks of LSTMs. New improved models and techniques were then developed, and one popular approach was Attention. Let us hope more and more interesting works will come around RNNs and sequence data.

I hope you enjoyed the blog. For any queries or clarifications, please feel free to connect through my LinkedIn

Machine Learning Data Science Rnn Lstm