**LSTM**

A type of Recurrent Neural Network.

This is some thing that solves the long-term dependency problem. Let’s assume we have a long paragraph of information and we’re trying to predict the next word of the current sentence.

The problem here is that referring all these sentences and information is unnecessary and may even cause clashes. For example:

**Short Dependency:**

Sentence: "I grew up in France, and I can speak fluent \_\_\_".

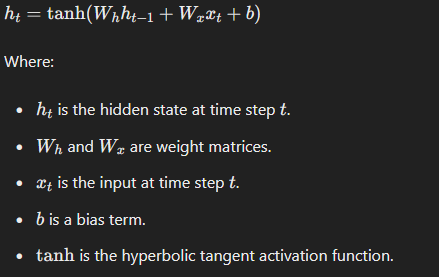
Here, the gap between "France" and "French" is small enough that an RNN might still retain some memory of "France".

**Long Dependency:**

Sentence: "I grew up in France. Since I was a child, I've travelled to many countries, experienced different cultures, learned new languages, but primarily, I can speak fluent \_\_\_"

In this case, the gap between "France" and "French" is much larger. A vanilla RNN would struggle to retain the memory of "France" due to the information being diluted over many steps.

A simple RNN updates its hidden state *ht* at time step *t* as follows:



During backpropagation, the gradients of the loss with respect to the weights​ are computed. If the gradients repeatedly shrink [due to the chain rule and the derivative of the tanh function being less than 1], they eventually become very small, making it difficult to learn dependencies from many steps ago. [Vanishing Gradient Problem]

The model does not understand exactly which words are important and how much so.

The LSTM provides a solution to this by adding an internal state to the RNN node. This state is called an LSTM cell and contains 3 values:

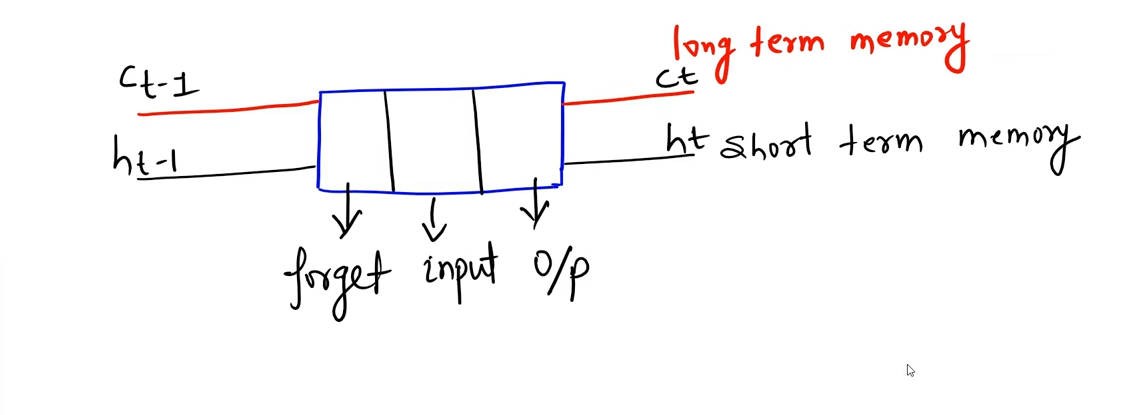
**Forget Gate**Tells what sort of state information can be forgotten for this state. Helps remove unnecessary information.

**Input Gate**Tells what new information should we add or update into this state Helps keep the necessary information.

**Output Gate**Tells out of all information stored in that state, which part should we output or send ahead. Gives the output basically.

Each range from a value 0 to 1 which is representative of their importance.

This is how an LSTM node may look like:



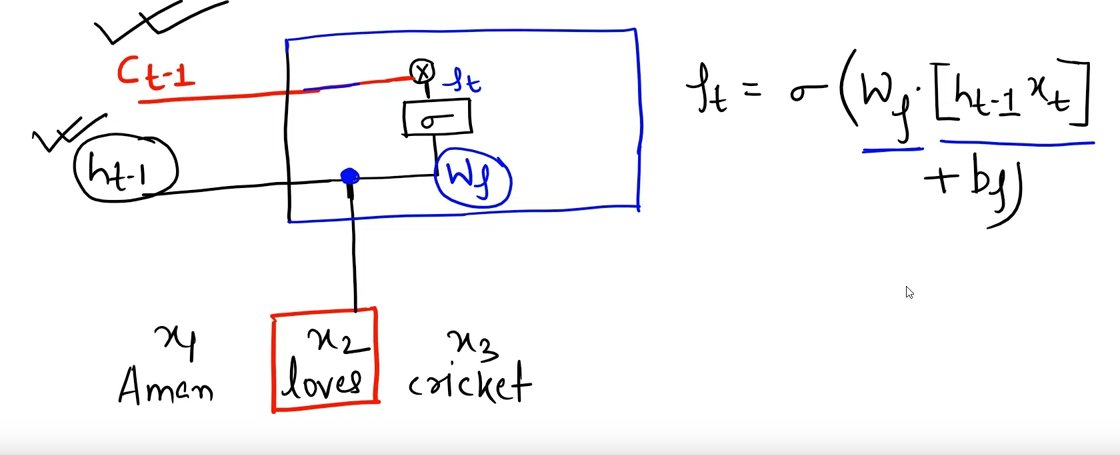
The node has 2 passes, one for long term memory [denoted by c], one for short term memory [denoted by h]. The time stamp is denoted in the subscript terms *t-1* and *t.*

**Important Applications:** Machine translation, Chatbot, etc.

**WORKING**

The cell state is kept track of in a conveyer belt fashion in the long-term memory term ct. All the outputs of the forget gate and input gate are fed into the ct term or belt.

**Forget Gate**



Here, x2 goes into the cell and first encounters the forget gate which is outlined in blue.

* The input word with the ht-1 term is assigned its respective weight, which then goes through the sigmoid function.
* Whatever is the output of the sigmoid activation function, is then multiplied with the previous long memory state or ct-1.
* The given formula is followed by the forget gate to assign the appropriate values to any and all terms.

In the formula:

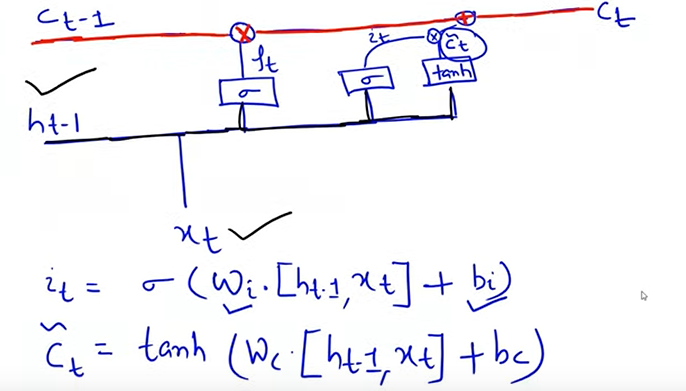
* Wf is the assigned weight
* ht-1 is the short-term memory pass
* xt is the current word
* bf is the bias term

All this goes into the sigmoid function.

The ultimate task here is to train our model such that we find the optimal values of the weights and biases.

**Input Gate**

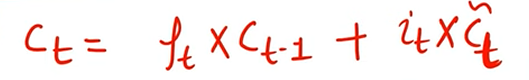
This extends the process of the forget gate.



* The ht-1 term passes once through sigmoid and once with tanh function.
* They follow these formulas and get to the it (from sigmoid function) and t term (from the tanh function) which use the same variables as that used previously but may have different values.
* Then it and are multiplied and added to the main ct-1 long term memory term.

The term is the candidate which is basically which word or combination of words do we give more importance to.

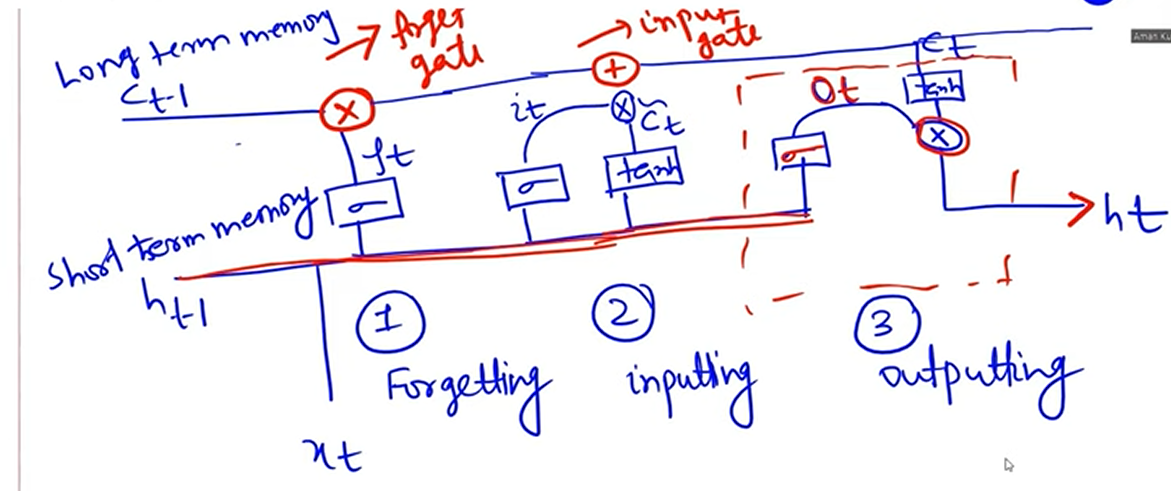
Now, we move onto calculating the ct term:



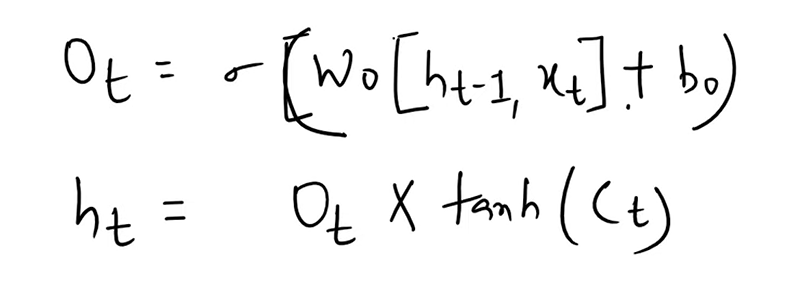
The new cell state ct is calculated from this formula, which would next become ct-1 for the coming input.

**Output Gate**

This further extends the input gate to give us our prediction and generating ot and ht.

****

* After getting the new ct from the input gate, we once again pass the ht-1 term through a sigmoid function and gives output ot.
* The ct is passed through a tanh function and finally multiplied with the ot term to give the new ht term which becomes ht-1 for the next iteration.



These are the mathematical formulae for ot and ht.