Working of LSTM

By Pritam Sutradhar

You might be thinking what is LSTM and why it was introduced if RNN was already available for sequential data.

The answer is because RNN suffers from two major problems

1. Problem of long term dependencies
2. Stagnetted Training

These problems cause two major issues in Deep learning world

1. Vanishing gradients
2. Exploding gradients

what are these?

Let's take an example…

"Bengali is spoken in West Bengal" here the word Bengal and Bengali are close so we can say that it's a short term dependency

Another example

"Jammu is a beautiful place. I went there last year, but I couldn't enjoy properly because I don't understand Kashmiri language" here the word Kashmiri and Jammu is related but due to the gap between these two words it became a long term dependency. Thus the gradients of "Jammu" words gets so small that it vanishes.

There are certain solutions to it

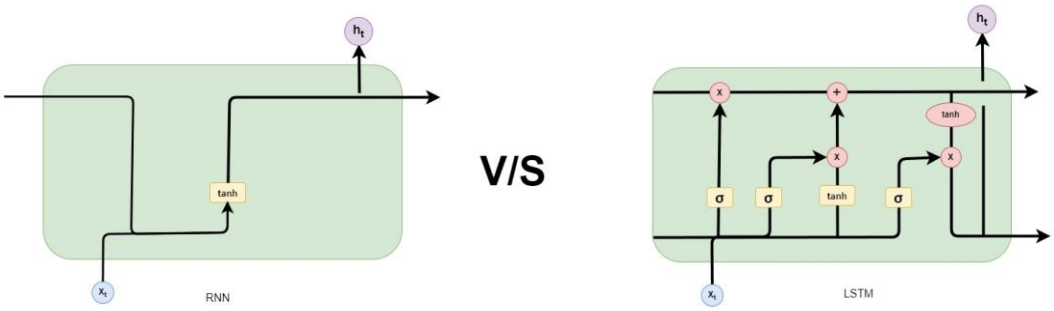
1. Using different activation function → relu or leaky relu
2. Better weights initialization
3. Skip RNN's
4. Or use LSTM

Using relu can also lead to another problem called exploding gradients because max(0, x) and produce very large numbers.

Some solutions are

1. Gradient clipping
2. Controlled learning rate
3. OR use LSTM

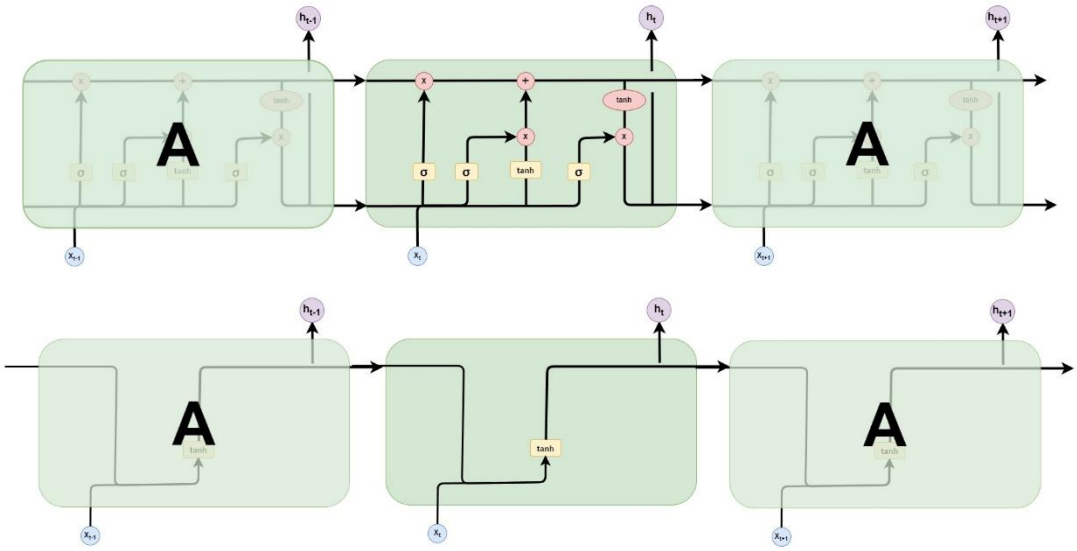
Now let’s explore architectural difference between RNN and LSTM



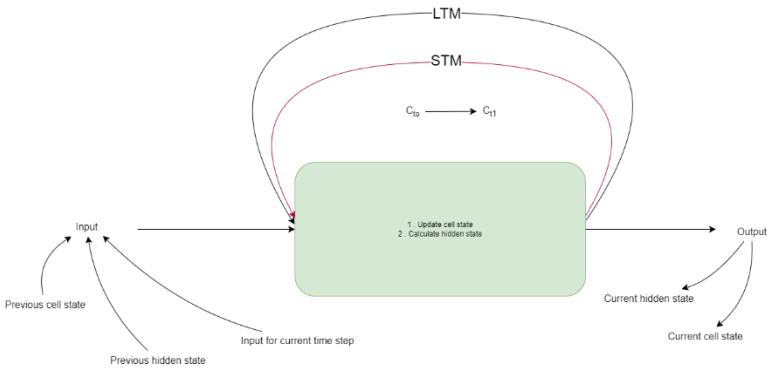
As you can see that there are more calculation happening in LSTM than RNN…….Why?

The answer is that LSTM use two memory architecture named STM(Short Term Memory) and LTM(Long Term Memory)….. Just like human brain.

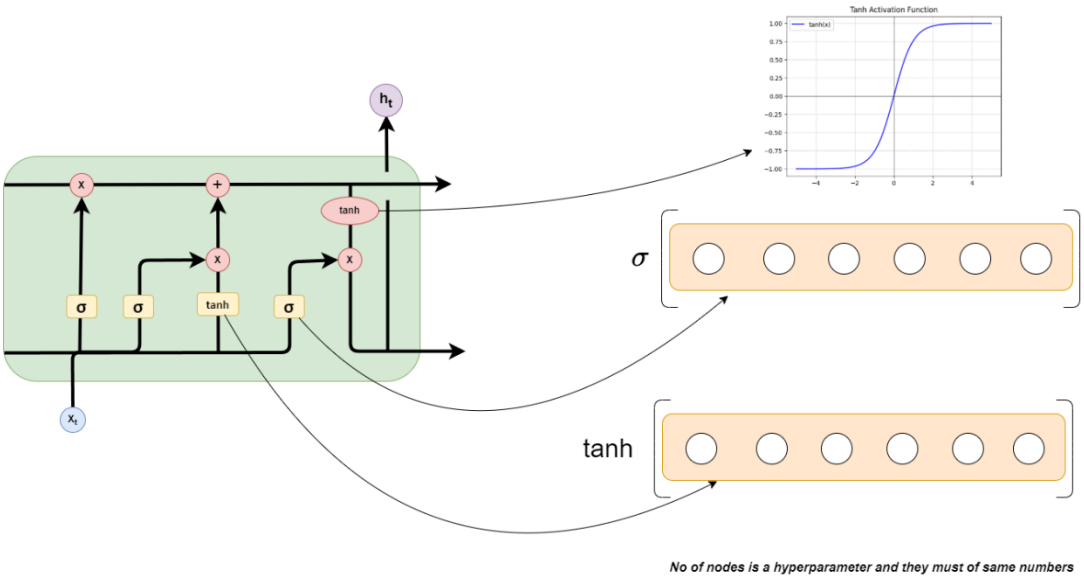
The STM store the recent context and LTM carry long term context unlike RNN which only use one state



All the scary calculation which is happening in LSTM is to make these two memory cells communicate with each other…

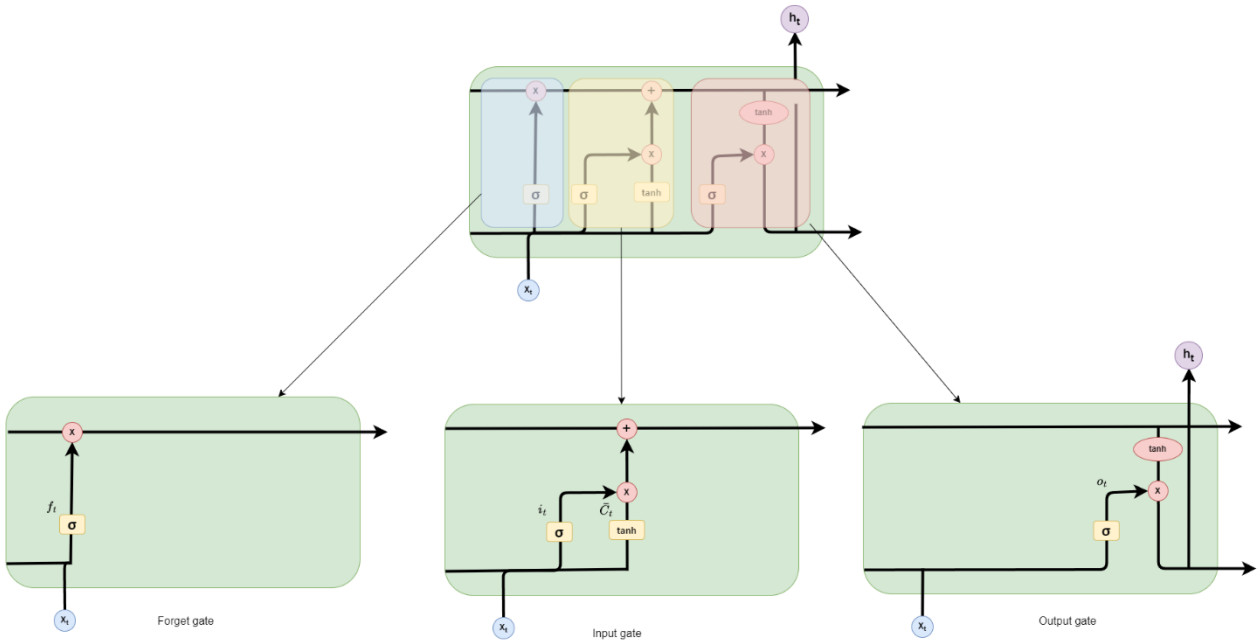


Let’s open this mystery box and see what’s inside



These yellow boxes are simple single layer neural network with sigmoid and tanh activation… keep that in mind that all these are separate NN with their own weights and biases… so not mix up thinking why not just use two NN of one with sigmoid and other with tanh and use their output…. 😝

This mystery box is made up of 3 gates…

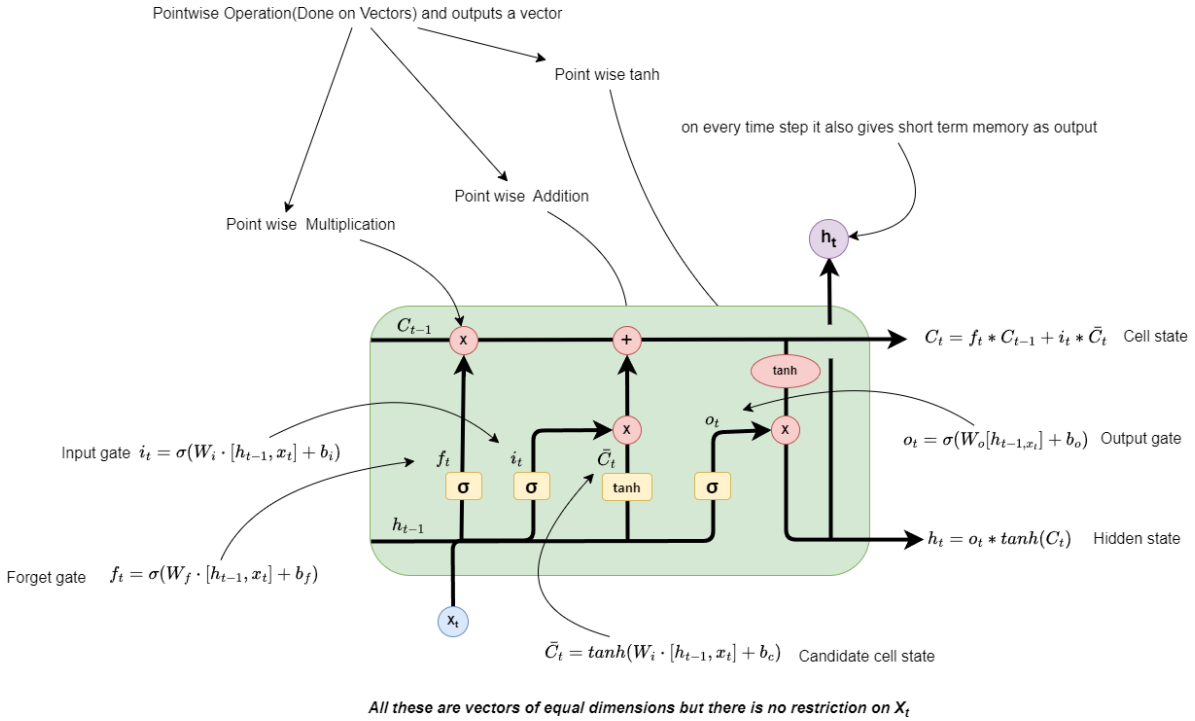


1. **Forget gate** – this part is responsible for removing unnessary context from LTM
2. **Input gate** – this part is responisible for filtering and then adding nessesary context to the cell state

If you look close enough then you might find similarity between RNN and this part (the input is passed through a NN with tanh acitvation)

3. **Output gate** – this part is responsible for generating the hidden state(STM) of that timestamp….

And all together this mystery box looks like this



All these are vectors of equal dimensions but there is no restriction on X_t

Don't be afraid by all these terms… we will be discussing each term later

C_{t-1} → is called the cell state …here it's the cell state of previous input….they carry LTM

X_t → these are input vectors of time step t

h_{t-1} → is called hidden state… here it's the hidden state of previous input…… they carry STM

f_t → it's called forget gate….used to remove irrelevant memory from LTM

i_t → it's a part of input gate….also used to filter out irrelevant info from input

$C^{\wedge}$ → is the candidate cell state

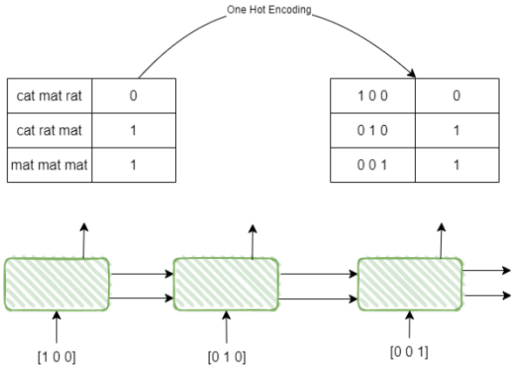
O_t → is the output state….later after doing some calculation with the cell state it becomes hidden state of this time step

and these small circles are pointwise operations

they are done on vectors and the output dimensions remain same after calculation

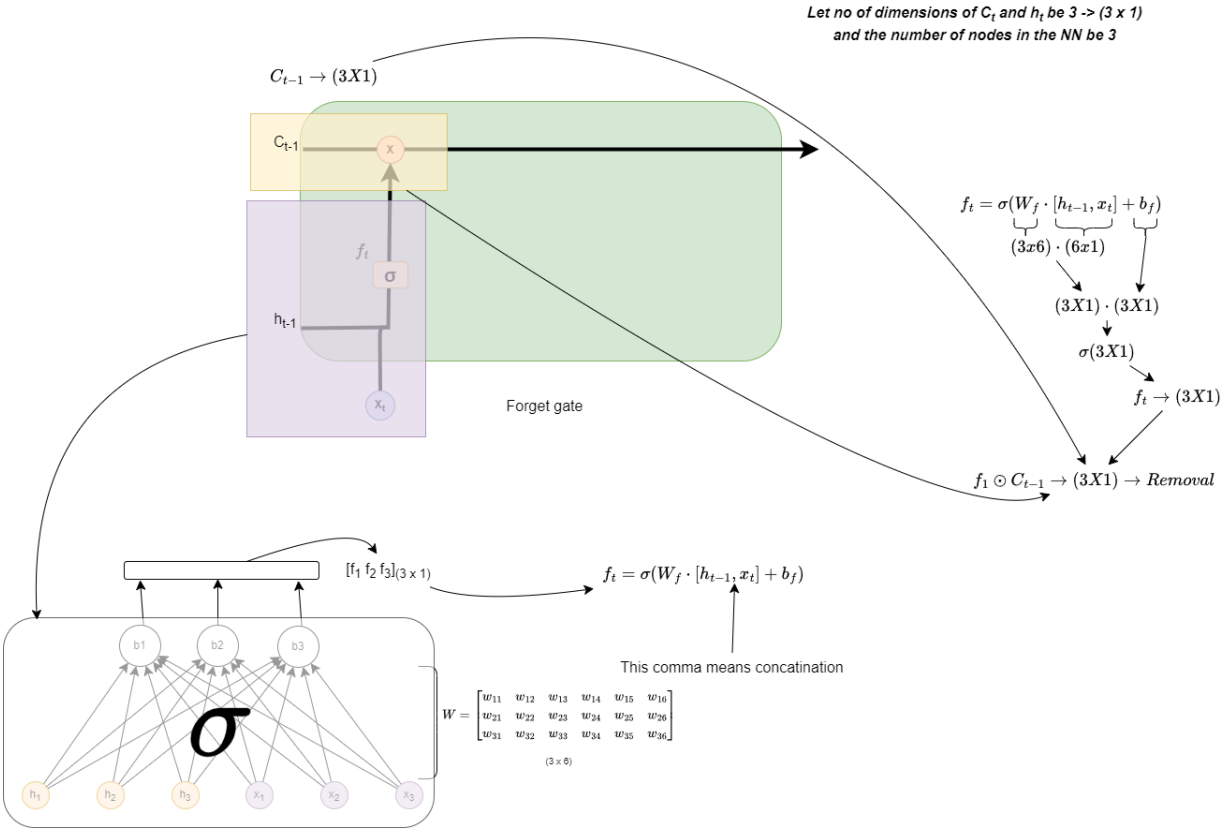
Now finally let's understand the working of LSTM

Let’s take this as an example



We one hot encode the input since NN doesn’t understands letters

Here’s how the forget gate will work



You might be thinking how pointwise multiplication can remove information from cell state…

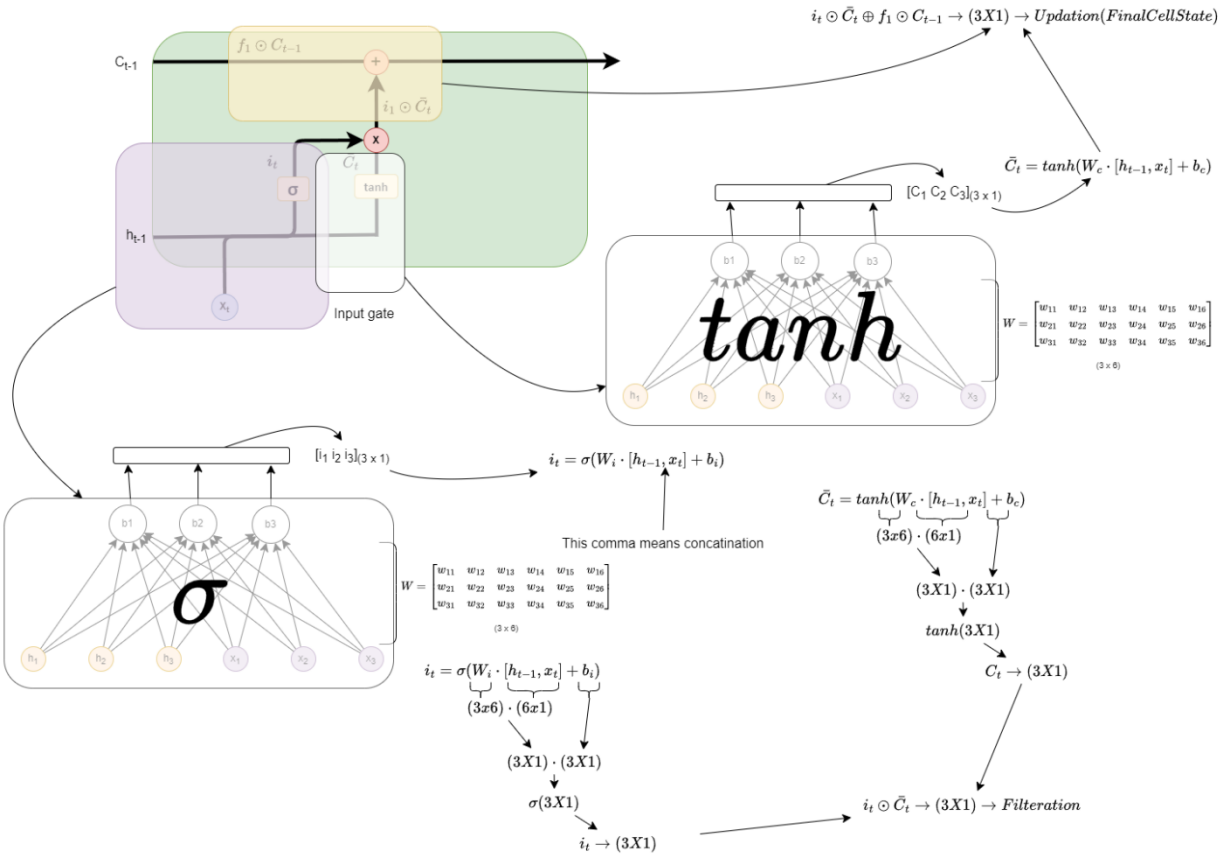
For that let’s take an example

We know that than sigmoid gives output in range [0 -1]

Now let C_{t-1} → [4 5 6] and f_t →[0.5 0.5 0.5] then the output of their pointwise multiplication gives C_{t-1} →[2, 2.5, 3]

As you can see that the information is halved meaning 50% of info is removed…. If f_t → [0, 0, 0] then C_{t-1}→ [0, 0, 0] 100% info is removed…..if f_t → [1, 1, 1] then C_{t-1} → [2, 2.5, 3] no data is removed 100% data is preserved

Now let's see how the input/update gate works



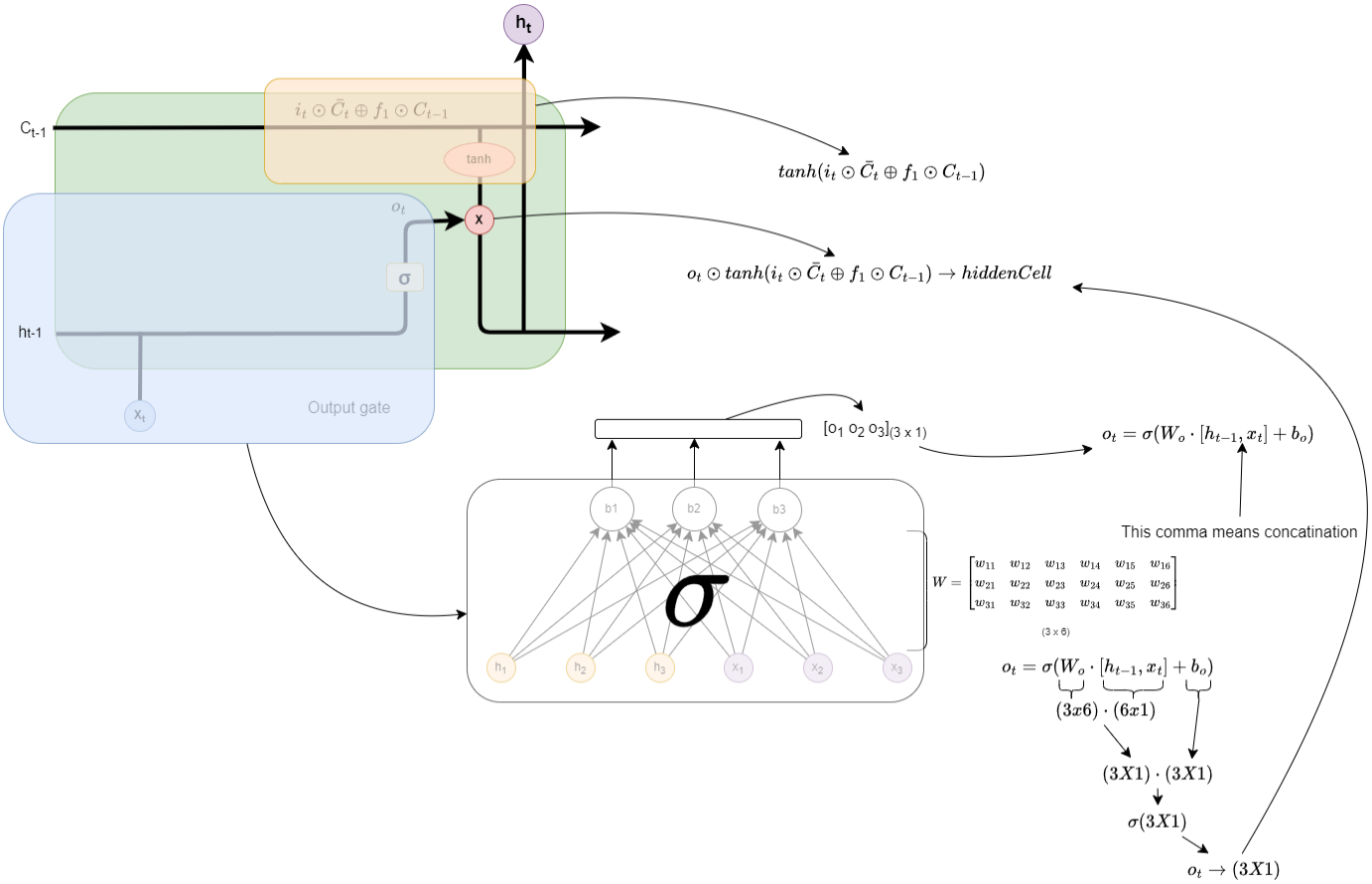
There's only two things happens inside this gate

1. Filteration of input data
2. Updation of cell state

Note that I have mentioned dimensions of output of every step… that's because it plays a very crucial role during pointwise operations…if dimensions don't match then operation can't be executed….

This gate gives the final cell state of this time stamp…meaning the LTM is updated here.

Now let's see how the output gate works



This gate produces the hidden state of this timestamp and also passes it onto next timestep.

Although this architecture solves the unstable gradient problem, but it's gets more complex as the sentence grow bigger and bigger… so solve that problem we will be discussing the infamous transformer model, which takes this NLP domain to whole different level….

To be Continued….