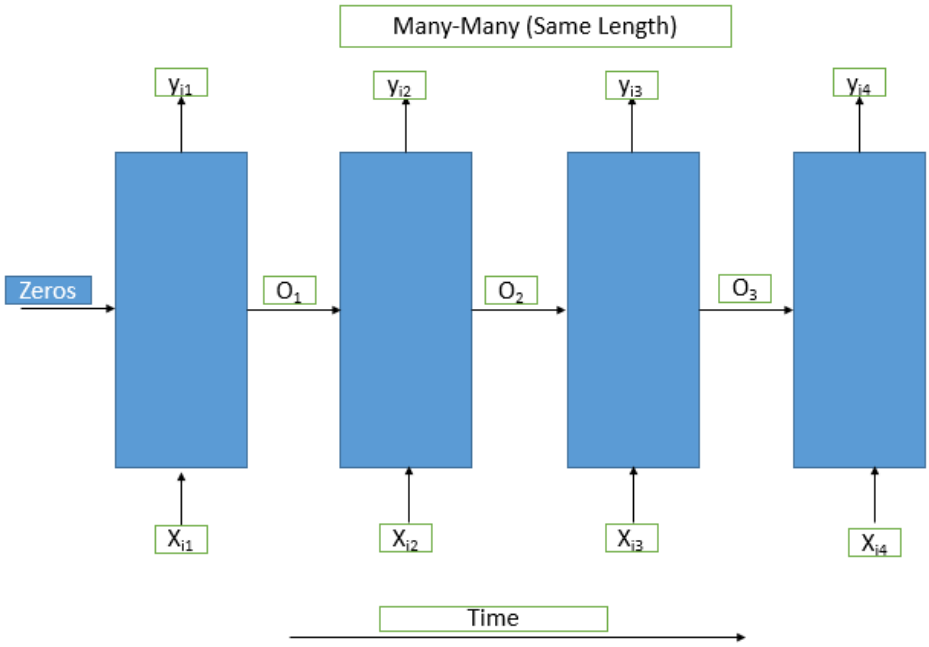
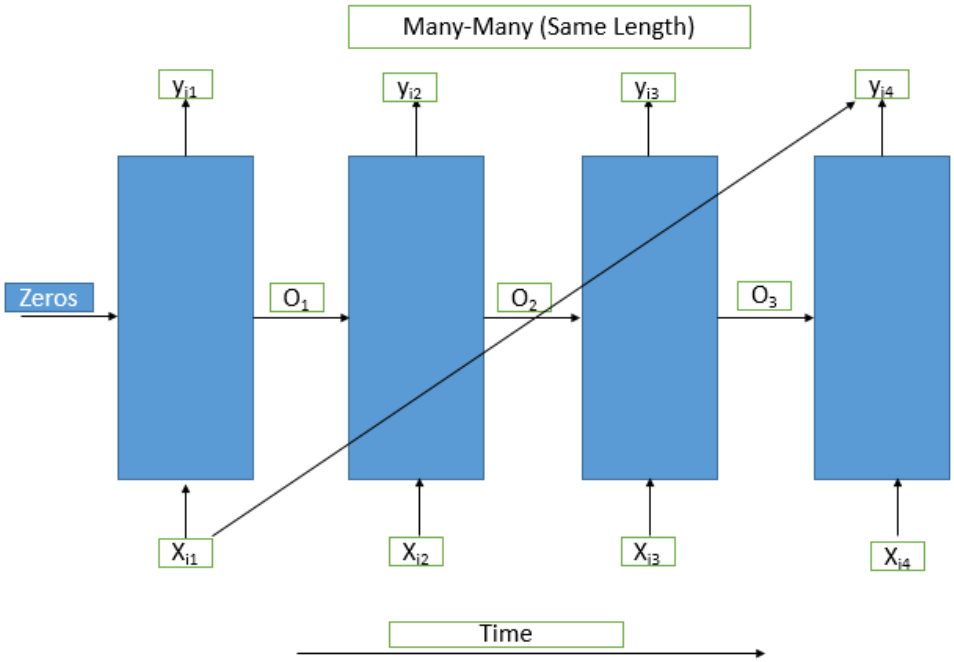
Q. What is the Problem With Simple RNN ?

- Ans:- Lets take Many-Many (Same Length)



- Here yi4 value depends a lot on xi4 and 03 and less depends on xi1 and 01 which is a limitation of this simple RNN because in real world application yi4 may depends more on xi1 and 01 and less xi4 and 03. this is called long term dependency.
- Simple RNN Cant handle long term dependency.
- long term dependency - Later output depends a lot on earlier input i.e. (yi4 -> xi1)



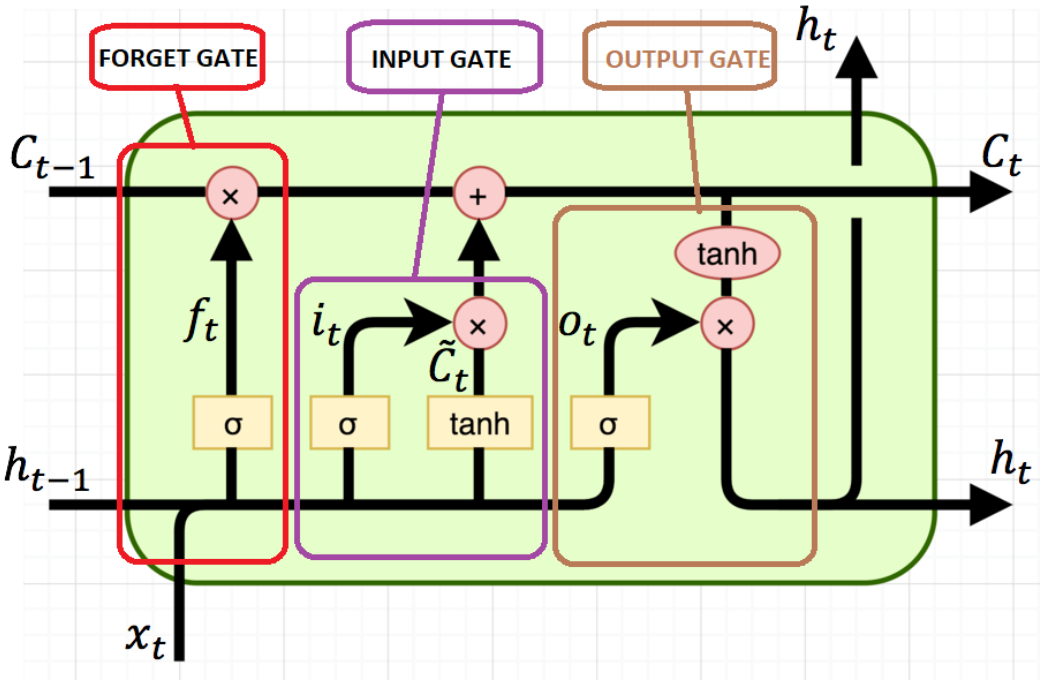
- So that small change in yi4 should create similar change in weight of xi1 result vanish the gradient. in simple RNN with Sigmoid and tanh later output nodes of network are less sensitive to the input this is happen due to vanishing gradient problem.

- If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.
- During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.
- So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory. If you want to know more about the mechanics of recurrent neural networks in general, you can read my previous post here.

- To overcome this we use LSTM (Long Short Term Memory) And GRU (Gated Recurrent Unit)

LSTM (Long Short Term Memory)

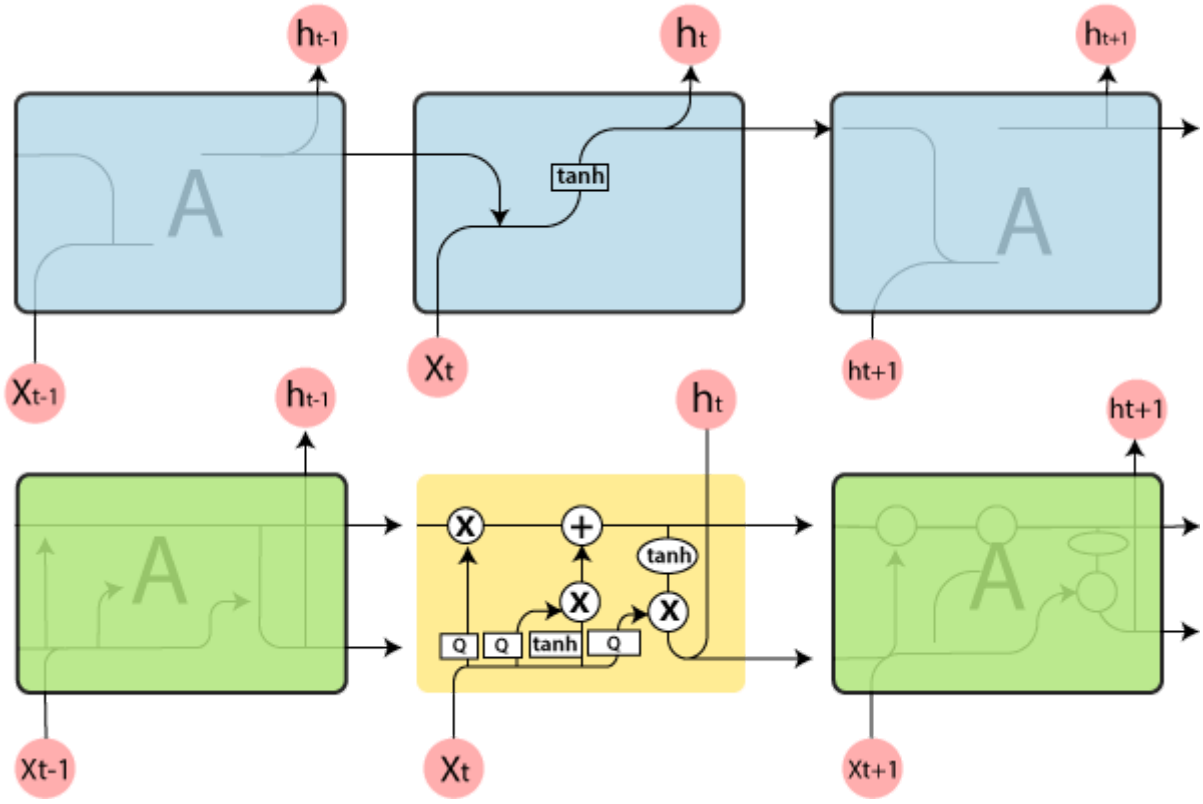
- It takes care both long and short term dependency.



1. Input gate- It discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0 or 1. And tanh function gives weightage to the values which are passed, deciding their level of importance ranging from -1 to 1.

2. Forget gate- It discover the details to be discarded from the block. A sigmoid function decides it. It looks at the previous state (ht-1) and the content input (Xt) and outputs a number between 0(omit this) and 1(keep this) for each number in the cell state Ct-1.

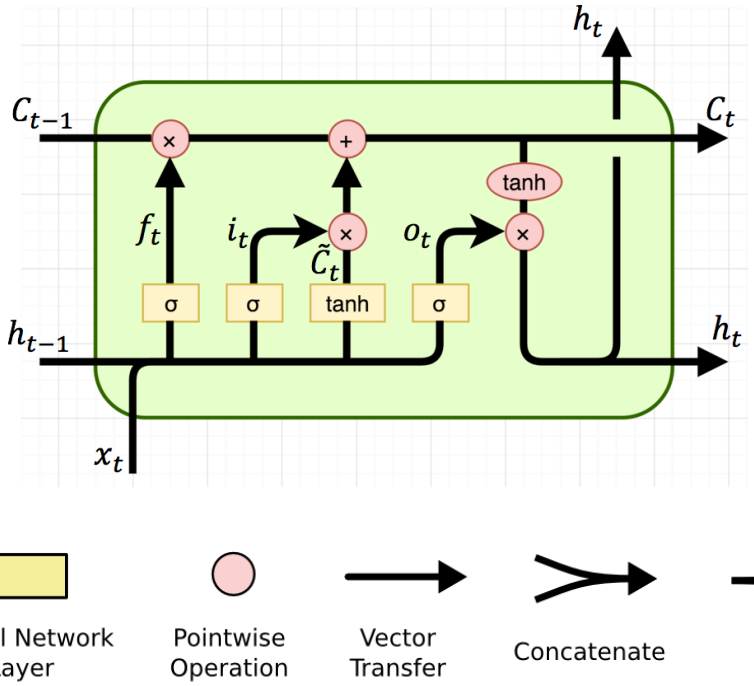
3. Output gate- The input and the memory of the block are used to decide the output. Sigmoid function decides which values to let through 0 or 1. And tanh function decides which values to let through 0, 1. And tanh function gives weightage to the values which are passed, deciding their level of importance ranging from -1 to 1 and multiplied with an output of sigmoid.



- It represents a full RNN cell that takes the current input of the sequence xi, and outputs the current hidden state, hi, passing this to the next RNN cell for our input sequence. The inside of an LSTM cell is a lot more complicated than a traditional RNN cell, while the conventional RNN cell has a single "internal layer" acting on the current state (ht-1) and input (xt).

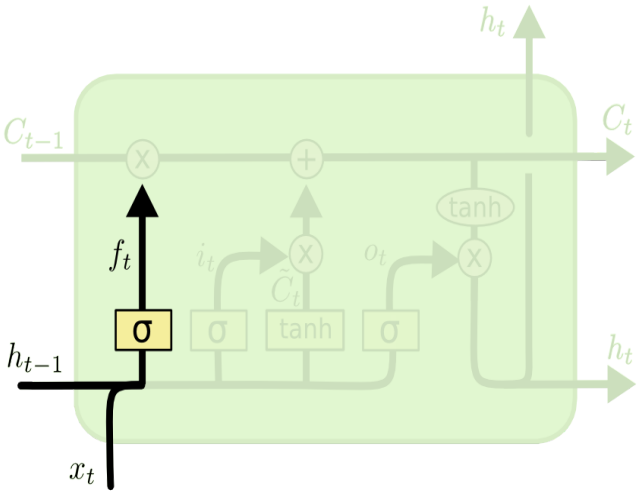
Step-by-Step LSTM Walk Through

- It takes care both long and short term dependency.



Step-1 (forget gate layer)

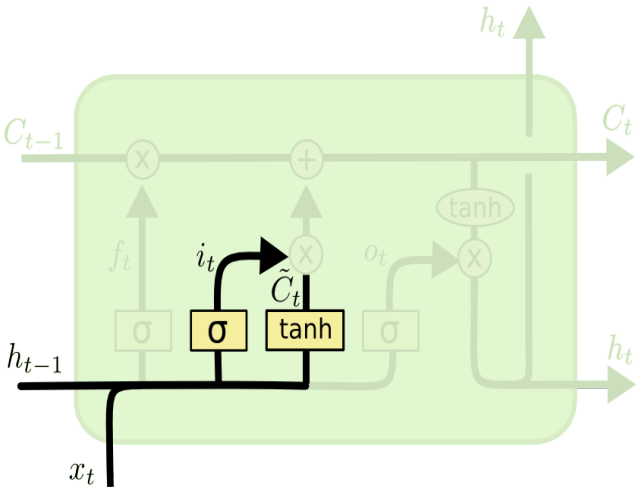
- The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."
- It looks at ht−1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ct−1. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

Step-2 (input gate layer)

- In this 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, C~t, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

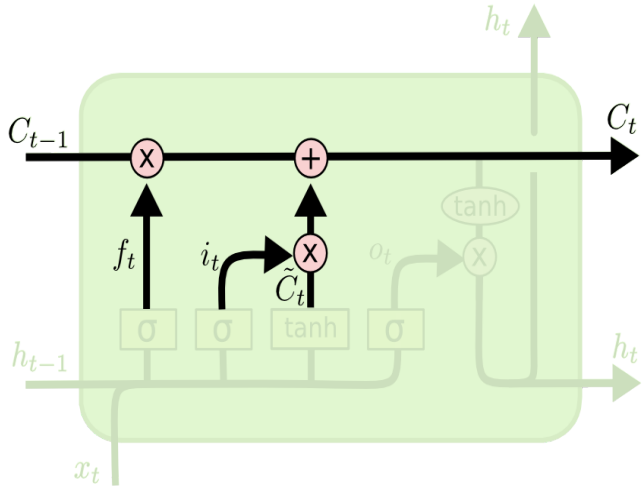


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Step-3 (input gate layer)

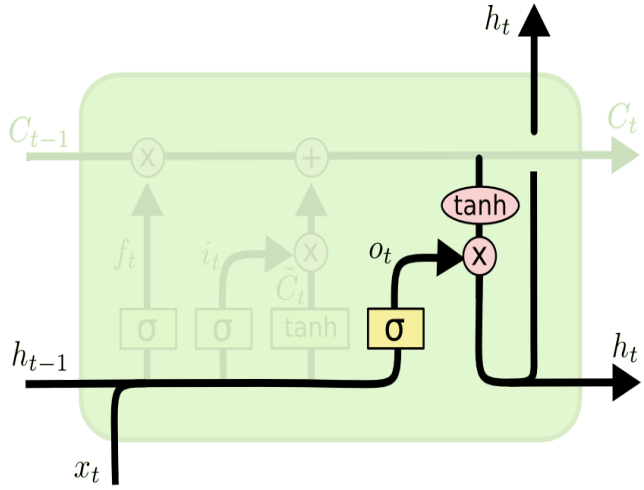
- It's now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗C~t. This is the new candidate values, scaled by how much we decided to update each state value.



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

Step-4 (output gate layer)

- 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.

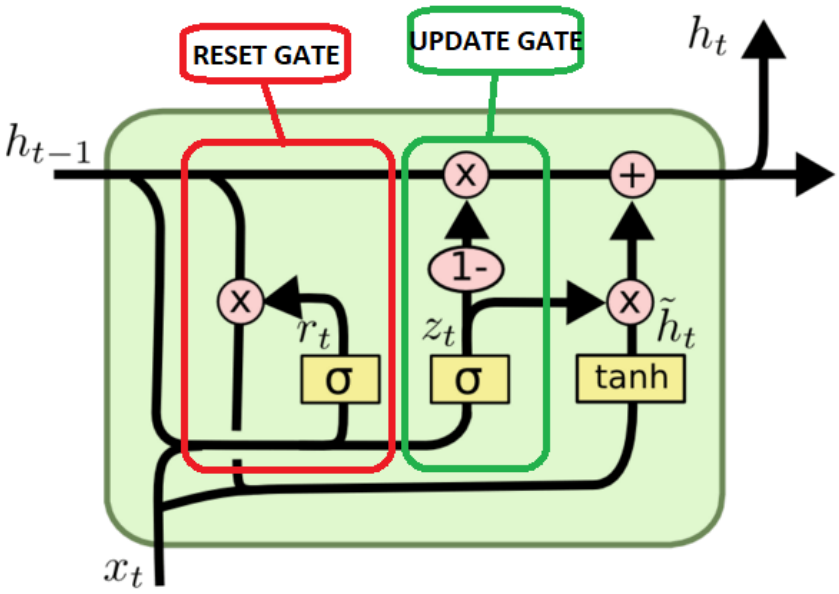


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

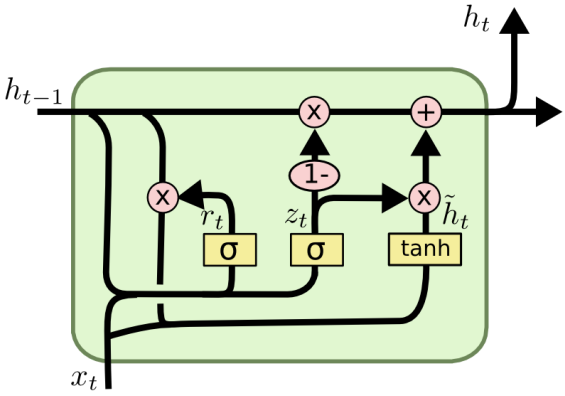
GRU (Gated Recurrent Units)

- So now we know how an LSTM work, let's briefly look at the GRU. The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. GRU's got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate.



1. Update Gate - The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.
2. Reset Gate- The reset gate is another gate is used to decide how much past information to forget.

- GRU's has fewer tensor operations; therefore, they are a little speedier to train then LSTM's. There isn't a clear winner which one is better. Researchers and engineers usually try both to determine which one works better for their use case.



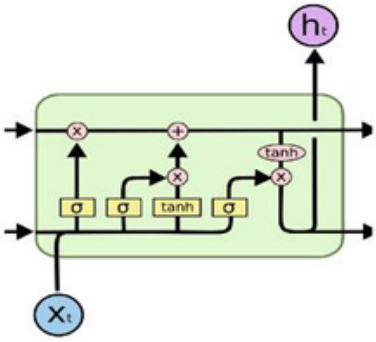
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

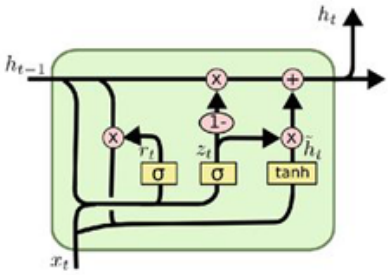
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM vs. GRU



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$