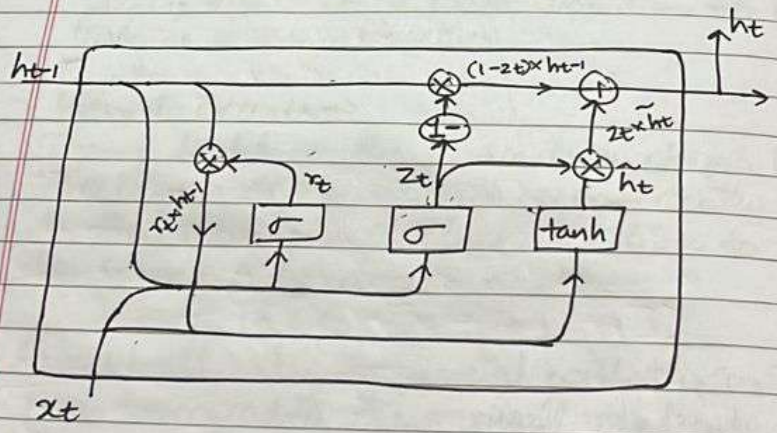
r_t → reset gate
$\tilde{h}_t$ → candidate hidden state
z_t → update gate

GRU (Gated Recurrent Unit) -

- Type of RNN architecture similar to LSTM.
- Designed to model sequential data by allowing info to be selectively remembered or forgotten over time.
- Simpler architecture than LSTM with fewer params, which can make it easier to train & more computationally efficient.

Architecture -



① **Input layer**: Takes in sequential data, such as seq. of words or a time series of values & feeds it ~~to GRU~~ into GRU.

② **Hidden layer**: This is where the recurrent computation occurs. At each time step, the hidden state is updated based on current i/p & prev. hidden state. The hidden state is a vector of nos. the represents NW's "memory" of prev. inputs.

③ Reset gate: Determines how much of prev. hidden state to forget. It takes as input prev. hidden state & current ~~off~~ i/p & produces a vector of nos. btw 0 & 1 that controls the degree to which the prev. hidden state is "reset" at the current time step.

④ Update gate: Determines how much of the candidate activatn vector to incorporate into the new hidden state. Takes as input prev. hidden state & the current input & produces a vector of nos. btw 0 & 1 that controls degree to which the candidate activation vector is incorporated into the new hidden state.

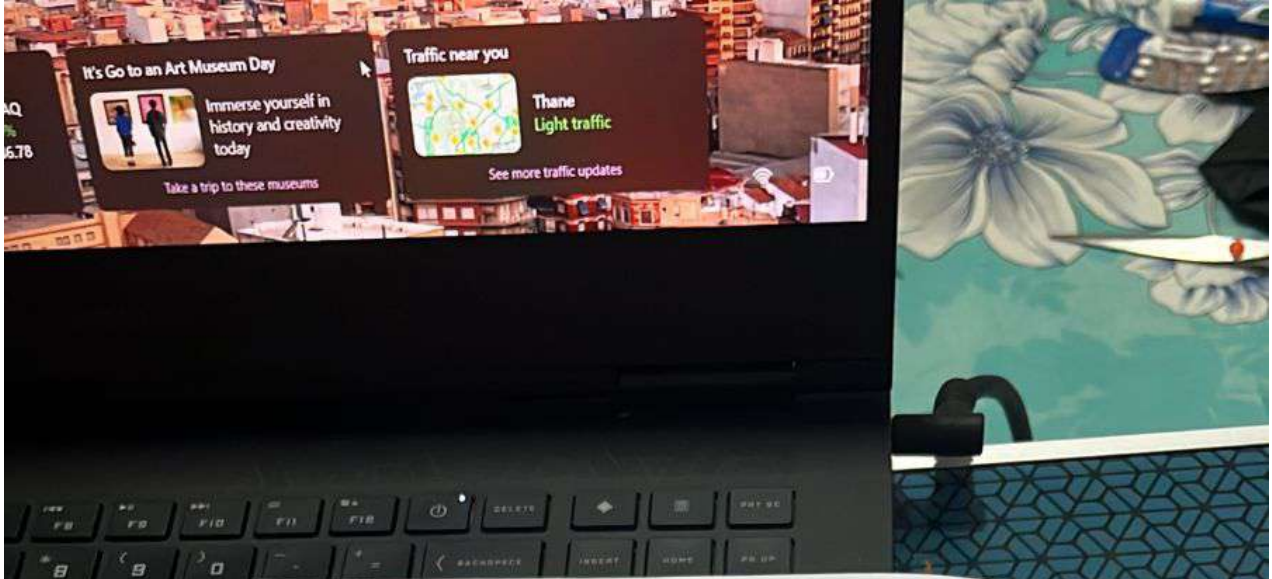
⑤ Candidate activatn vector: Modified version of prev. hidden state that is 'reset' by the reset gate & combined with the current input. Computed using tanh activatn that squashes its O/P btw -1 & 1.

⑥ Output Layer: Takes final hidden state as i/p & produces the network's output. This could be a single no., a seq. of nos. or a probability distributn over classes depending on task at hand.

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

No bias

- Reset gate determines how much of the prev. hidden state to remember or forget, while update gate determines how much of the candidate vector to incorporate into the new hidden state.



- Result is a compact arch. that's able to selectively update its hidden state based on the i/p & prev. hidden state, without the need for a separate memory cell state like the LSTM.

Also add after 6th pt →

- The reset gate r & update gate z are computed using the current input x & ~~prev~~ to prev. hidden state h_{t-1}.

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

where W_r & W_z are wt. matrices that are learned during training.

- The candidate activation vector $\tilde{h}_t$ is computed using current input x & a modified version of the prev. hidden state that's "reset" by reset gate.

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

- The new hidden state h_t is computed by combining the candidate activation vector with prev. hidden state, weighted by the update gate:

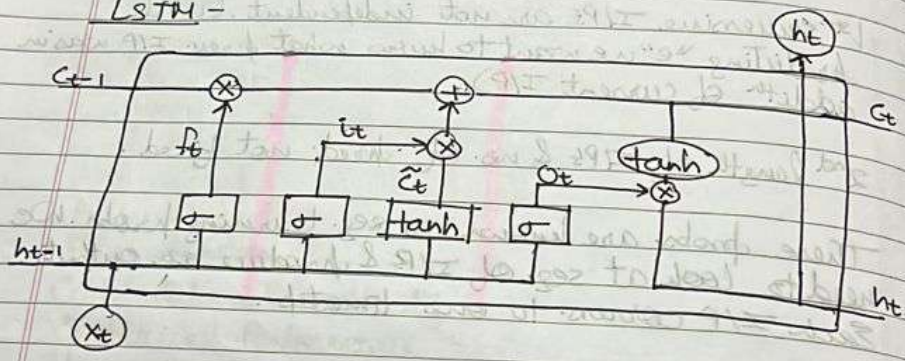
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM -

Cell State -

Key to LSTMS is the cell state, the horizontal line running through top of the diagram

LSTM -



Cell State -

- Horizontal line running through top of the diag.
- Runs straight down the entire chain with only some minor linear interations. (Very easy for info. to just flow along with it unchanged.)

LSTM does not have ability to remove or add info. to the cell state, carefully regulated by str. called gates.

Gates -

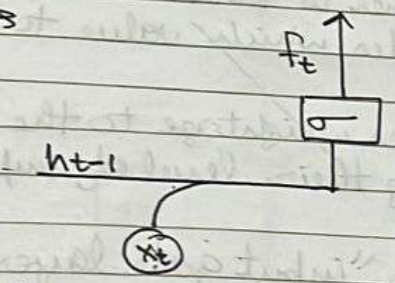
They are a way to let info. through. They are composed of σ NN layer & a pt wise multiplicatn operation.

σ layer O/P nos. b/w 0 & 1 →
0 → " let nothing through
1 → " let everything through

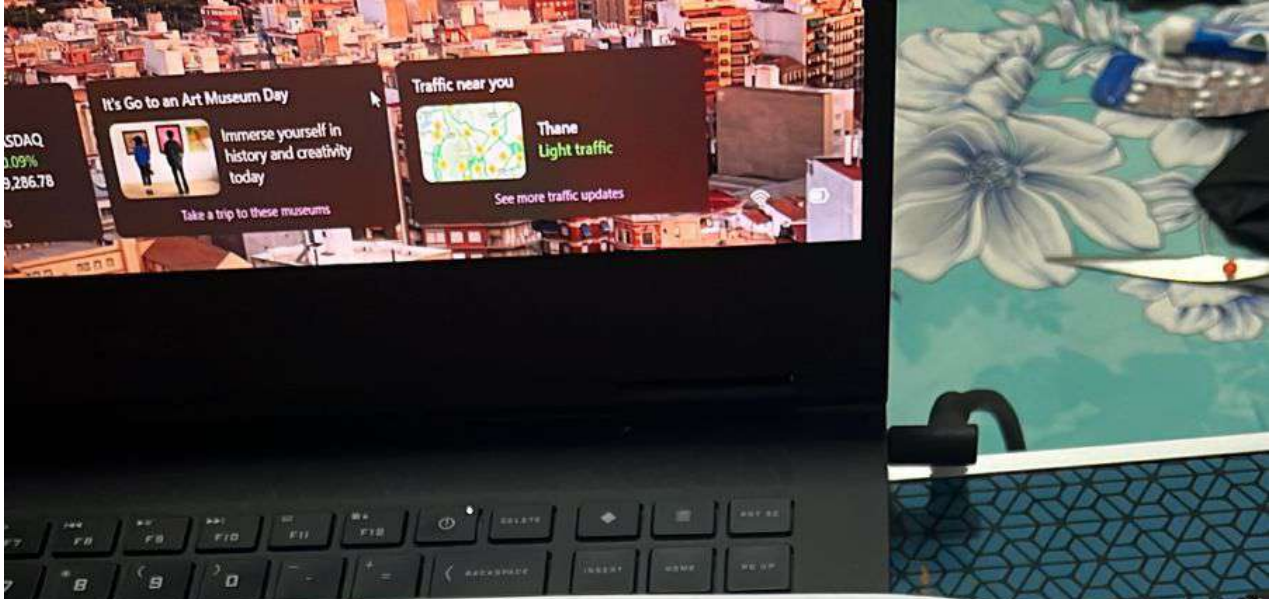
LSTM has 3 of these gates to protect & control C_t.

Working - 4 steps →

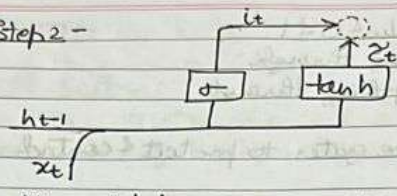
Step 1 →



→ 1st step is to decide ~~who~~ which info. should be omitted from the cell state in that particular timestamp.
→ This decision is made by a σ layer called "Forget Gate Layer".
→ It looks at h_{t-1} & x_t & outputs a no. b/w 0 & 1 for each no. in cell state C_{t-1}.
→ 1 represents "completely keep this" while 0 rep. "completely get rid of this".
→ $f_t = \sigma[\sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
W_f = wt. ; h_{t-1} = O/P from prev. timestamp
x_t = new i/p , b_f = bias.



Step 2 -



→ In the 2nd layer, ~~there~~ there are 2 parts → one is σ fn & other is tanh fn.

→ In σ fn it decides which value to let through (0 or 1)

→ In tanh fn it gives weightage to the value which is passed, deciding their level of importance (-1 to 1).

→ 1st σ layer called "input gate layer" decides which values we will update.

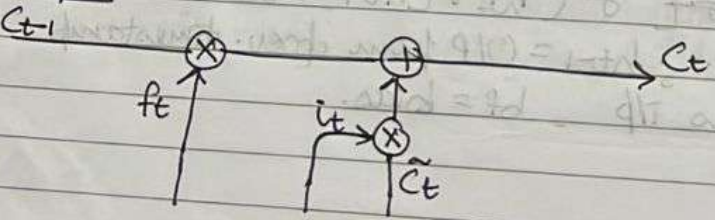
→ Next a tanh layer creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state.

→ In next step we combine these 2 to create an update to the state.

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

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

Step 3 -

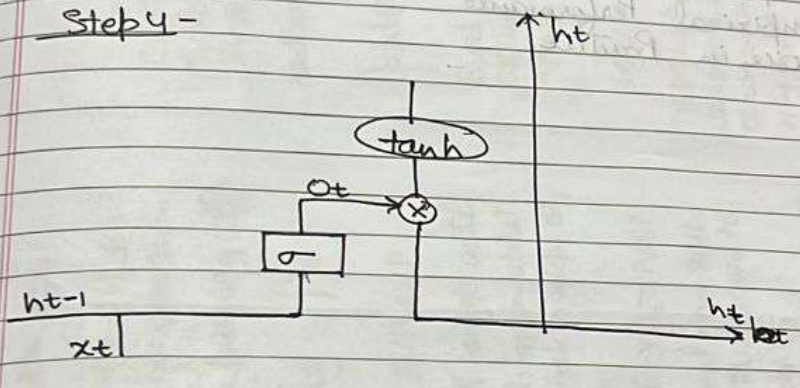


→ Now we will update the old cell state C_{t-1} into new cell state C_t. 1st we mul. old cell state C_{t-1} by f_t forgetting the things we decided to forget earlier.

→ Then we add $i_t * \tilde{C}_t$. These are 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 -



→ Finally we need to decide what we are going to O/P. This O/P will be based on our ~~cell~~ cell state, which will be the filtered version.

→ 1st we run a σ layer which decides what parts of the cell state we are going to O/P. Then we put the cell state through tanh (push the values to be b/w -1 & 1) & mul. it by O/P of σ gate, so that we only O/P the parts we decide to.

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