Chapter-07 LSTM Networks

7.1 Motivation

Standard RNN have poor memory:

Transition Matrix necessarily weakens signals.

- Need a structure that can leave some dimensions unchanged over many steps
- This is the problem addressed by so called. Long-short Term Memory RNNs (LSTM)

Make Remembering Easy:-

Define a more complicated update mechanism for the changing of the internal state

- By default, LSTMs remember the information from the last step
- Items are overwritten as an active choice

1.2 Long-Short Term Memory Networks (LSTM)

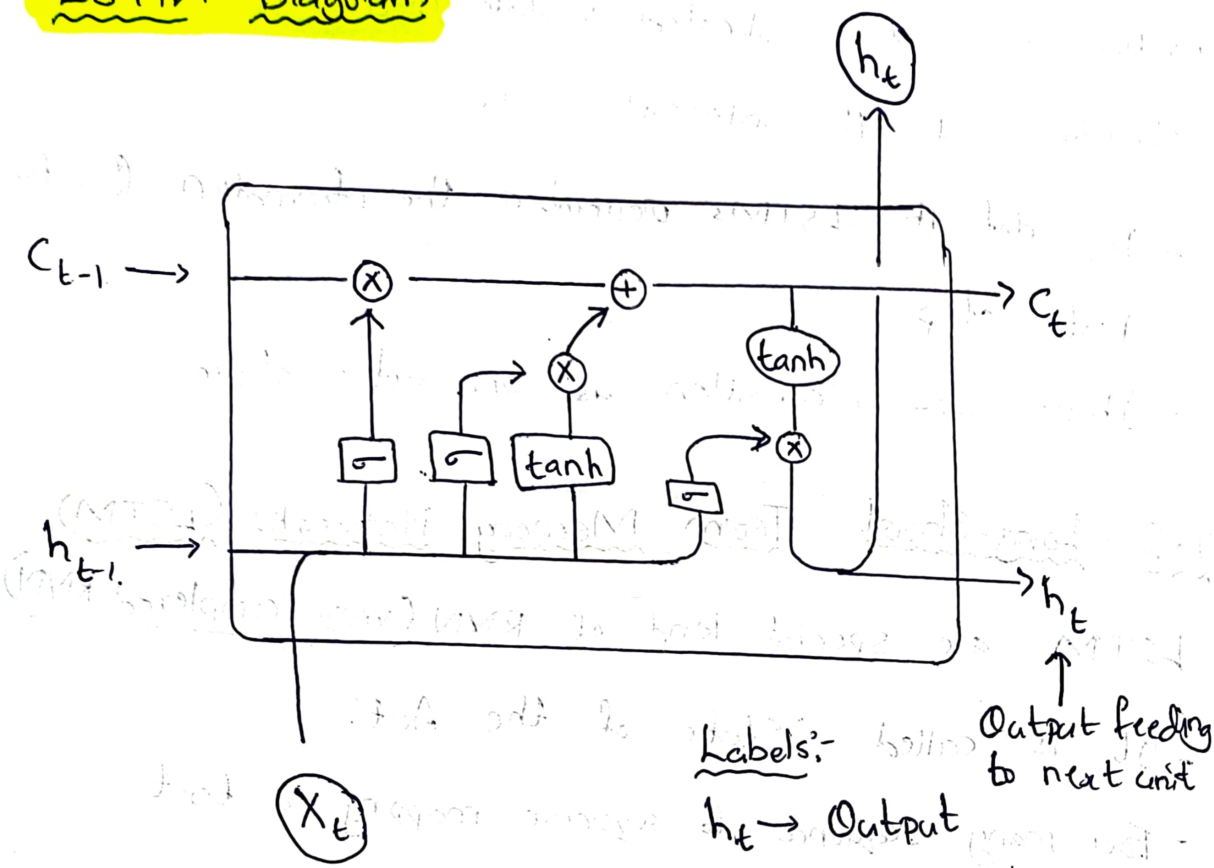
LSTM are special kind of RNN. (very complexed RNN)

- It is called "State of the Art".
- For many sequence to sequence mapping & text generation tasks.
- Adds an explict 'memory unit'

* Augment RNN with a few additional Gate Units:-
  - Gate units controls how long/if events will stay in memory
  - Input Gate:- If its value is such, it causes items to be stored in memory
  - Forget Gate:- If its value is such, it causes items to be removed from memory
  - Output Gate:- If its value is such, it causes the hidden unit to feed forward (output) in the network.

7.3 LSTM Explaination

LSTM Diagram



Labels:-

$h_t \rightarrow$ Output

$C_t \rightarrow$ Cell state

$X_t \rightarrow$ Input

$\otimes \rightarrow$ Forget gate

$\oplus \rightarrow$ Input gate

$W_f \rightarrow$ Transformation

① Cells gets updated in 2 stages, from C_{t-1} to the forget gate, decides what to "forgot", next to the "input gate", Add new info & passes the info which is updated to the next cell.

What to Forget:-

$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f)$ based on prev output & current input

This function is passed through the sigmoid function, to know ~~whether~~ what to forget ~~the~~ in the data ~~next~~.

Adding in new information:-

$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ → This at σ function

$C_t' = \tanh(W_c[h_{t-1}, x_t] + b_c)$ → This is at tanh function

i_t → what portion of that new info we would want to add on.

C_t' → The actual info you are deciding whether or not to add on

Cell State:- C_t

$C_t = f_i * C_{t-1} + i_t * C_t'$

Here * represents element wise multiplication

↑ Forget the old

↑ Add the new.

Final Stage:-

Output:-

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

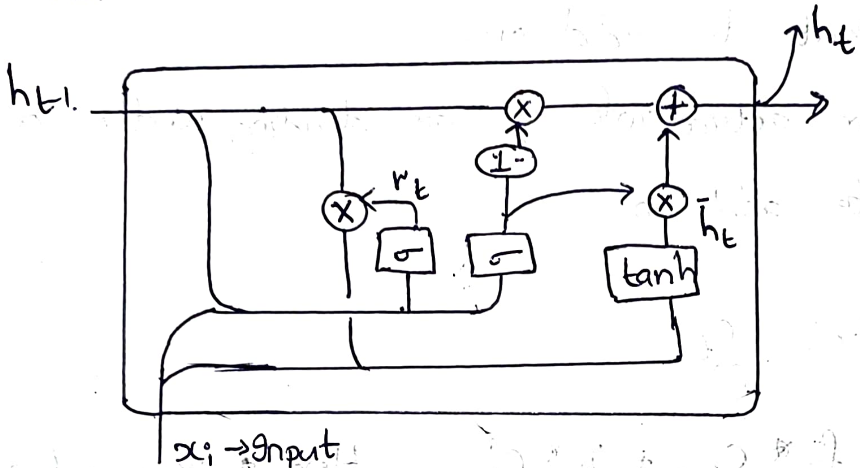
$h_t = O_t * \tanh(C_t)$

Note:- LSTM requires a good amount of memory & parameters, deciding what to keep & what to forget

7.4 Gated Recurrent Unit (GRU)

Removed Cell State:

- Past information is now used to transfer past info.
- Think of a simpler version & faster version of LSTM



Reset Gate:- Helps decide how much past info to forget [σ, r_t, ⊗] (left side of unit)

Update Gate:- Helps decide what information to throw away & what new info to keep.
[σ, 1-, ⊗] (middle of unit)

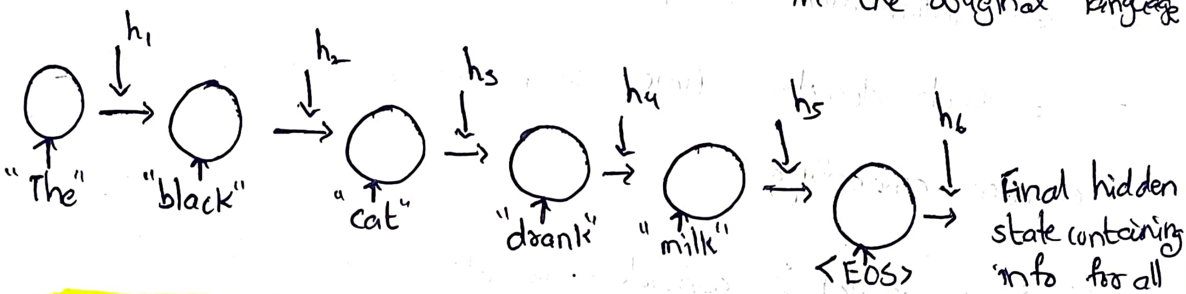
LSTM or GRU?

- LSTM are built more complex, & may therefore be able find more complicated patterns.
- Conversely GRU's are a bit simpler & therefore quicker to train
- GRUs will generally perform about as well as LSTM with shorter training time, especially for smaller datasets.
- Luckily in Keras, all we need to do is call the layer type & it would ~~be~~ not be too complicated to quickly write up changes b/w the two.

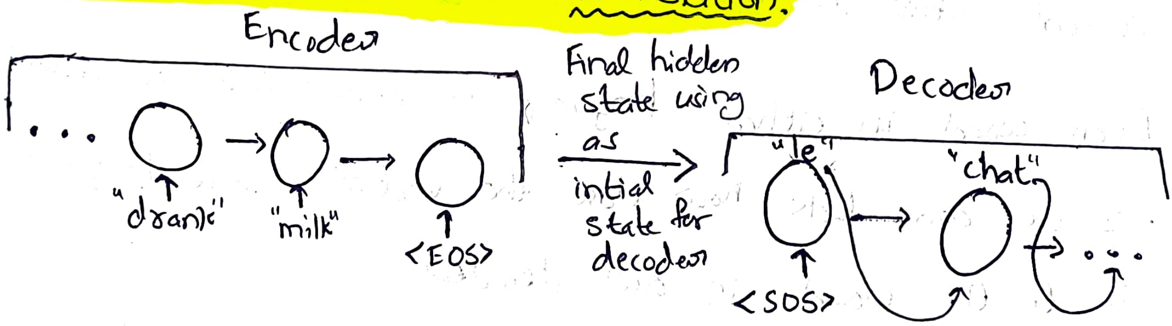
7.5 Sequence to Sequence (Seq2Seq):

Eg: English to French

- Thinking back to any type of RNN interprets, text, the model will have a new hidden state at each step of sequence containing information about all past words.
- At the end of sentence, the hidden state will have all info relating to past words
- The size of vector from the hidden state is the same no matter the size of sentence
- In machine translation, the 'encoder': corpus of sentence in the original language



Seq2Seq for Machine Translation:-



7.6 Gated Recurrent Unit Details

Note: Current model is producing one word at a time conditional on prior word.

- If it produces one wrong word, we may end up completely throwing off the sequence of words

Beam Search:- A solution to this is to produce multiple different hypothesis to produce words until <EOS> & then see which full sentence is most likely

Attention:-

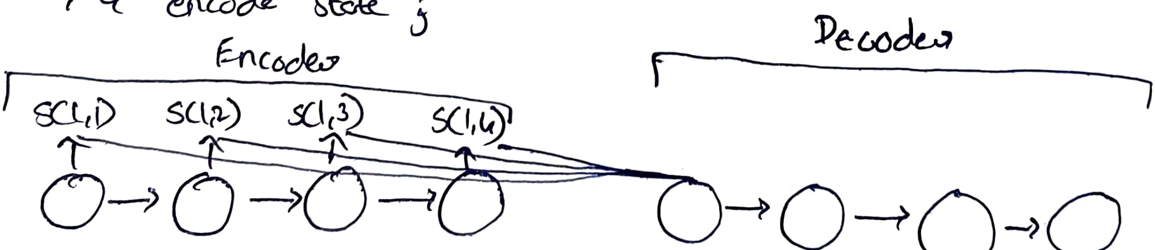
Current Framework : The generative process is all using all ~~with~~ information in the final hidden layer

- ie each decoder time step depends on the same encoder embeding

A stronger model :- consider words most similar to our current state in our sentence generation.

Each word in either language is represented as vector

- So we can look how close the vector in one language is to the word in our decoder
- s(i,j) will be similarity measure b/w the decoder state i & encode state j



The $s(i,j)$ function will weight the different embedding layers hidden states to give us a better embedding for predictions of next word

- This will allow you for better translation b/w different languages when the ordering of words may be different