***Theoretical Understanding of RNN, LSTM, & Encoder-DecodeR***

***GLA- 2370300004***

***TASK 1 Conceptual Questions***

**Q1. What is the difference between RNN and LSTM?**

Answer:  
RNN (Recurrent Neural Network) is used for sequence data like text, time series, etc. But it forgets long-term information easily.

LSTM (Long Short-Term Memory) is a type of RNN that remembers things for a longer time using gates (input, forget, output).

Example:  
Imagine you're reading a story.

* An RNN remembers only the last few lines.
* LSTM remembers important details from the beginning to the end of the story.

**Q2. What is the vanishing gradient problem, and how does LSTM solve it?**

Answer:  
When training deep models, the vanishing gradient problem happens when the model’s updates become very small. It stops learning long-term patterns.

LSTM solves this using gates that control which information to keep or forget. This helps the model learn better over time.

Example:  
Think of passing a message through 10 people.

* In RNN, the message gets lost.
* In LSTM, each person checks what’s important before passing it forward.

**Q3. Explain the purpose of the Encoder-Decoder architecture.**

Answer:  
The Encoder-Decoder model is used in tasks like language translation.

* The encoder reads the input (e.g., an English sentence) and summarizes it into a context vector.
* The decoder uses this context to generate output (e.g., a French sentence).

Example:  
Input: *“I love apples”* (English)  
→ Encoder creates context  
→ Decoder outputs: *“J’aime les pommes”* (French)

**Q4. In a sequence-to-sequence model, what are the roles of the encoder and decoder?**

Answer:

* The encoder understands and stores the input sequence into memory (context vector).
* The decoder uses this memory to generate the output sequence word by word.

Example:  
Encoder: reads *“What is your name?”*  
Decoder: writes *“¿Cómo te llamas?”*

**Q5. How is attention different from a basic encoder-decoder model?**

Answer:  
In a basic model, the decoder looks at only the final output from the encoder.  
But with attention, the decoder looks at all words in the input while generating each word of the output.

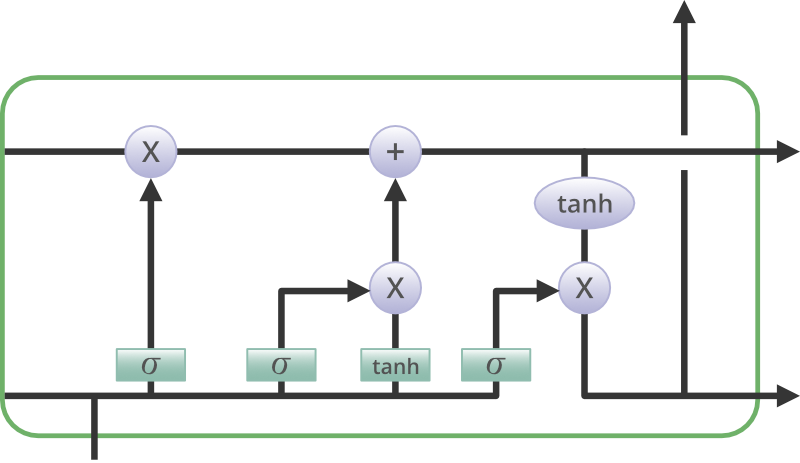
Example:  
Without attention: trying to answer a question by remembering only the last sentence.  
With attention: scanning the entire passage for the answer — more accurate!

***TASK 2 Sequence-to-Sequence Data Flow***

Draw or describe the data flow in an encoder-decoder model using RNN/LSTM.

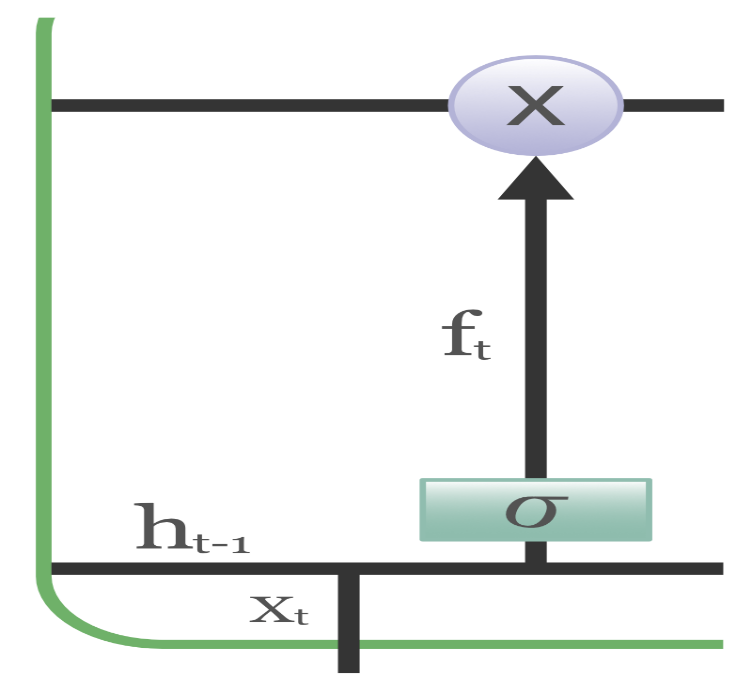
Clearly label:

* Input sequence
* Hidden states
* Context vector
* Output sequence



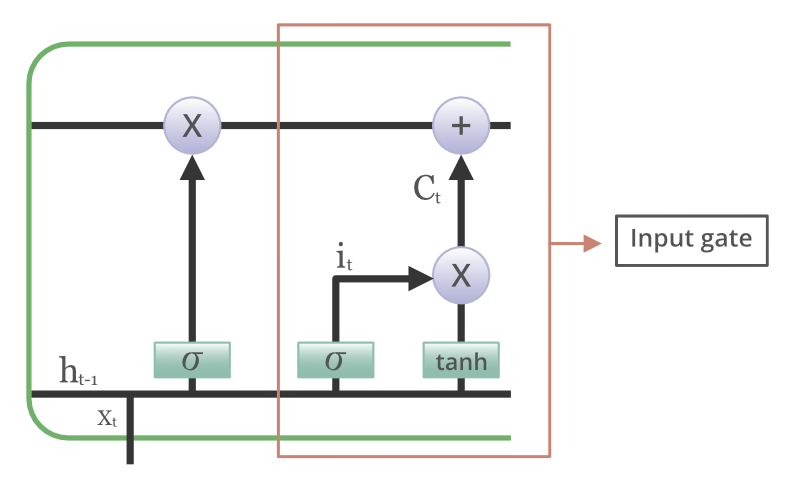
**Forget Gate**

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x\_t (input at the particular time) and h\_t-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state, the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.



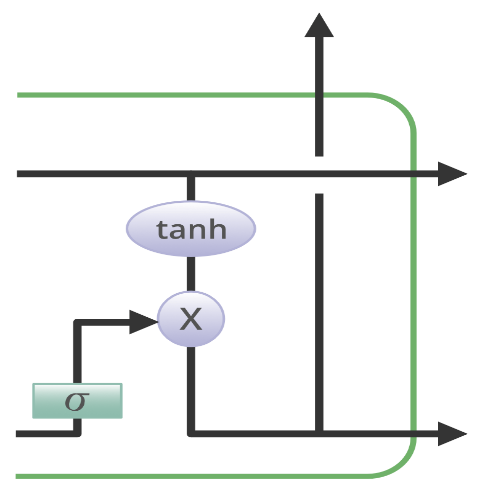
**Input gate**

The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h\_t-1 and x\_t. Then, a vector is created using the tanh function that gives an output from -1 to +1, which contains all the possible values from h\_t-1 and x\_t. At last, the values of the vector and the regulated values are multiplied to obtain useful information.

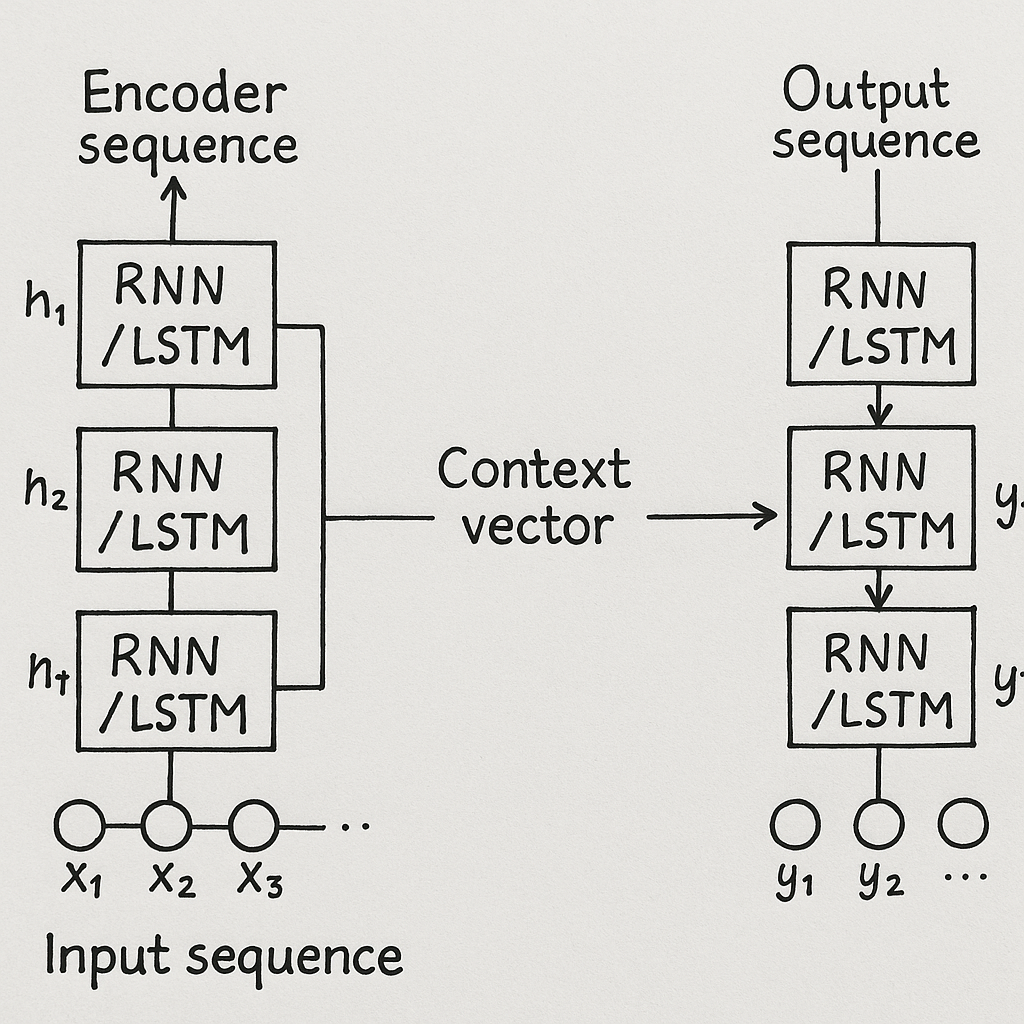


**Output gate**

The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying the tanh function on the cell. Then, the information is regulated using the [sigmoid function](https://www.geeksforgeeks.org/machine-learning/types-of-activation-function-in-ann/) and filtered by the values to be remembered using inputs h\_t-1 and x\_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.



|  |  |
| --- | --- |
| Label | Meaning |
| Input Sequence | The **input sentence** given to the model, usually in the source language (like English). It is first **converted into a sequence of tokens**, such as [x1, x2, ..., xn]. Each token represents a word or sub-word in numerical form for processing by the encoder. |
| Hidden States | These are the **memory units** (h1, h2, ..., hn) generated by the encoder at each step. Each hidden state carries forward the meaning of the sentence word by word. They help store **what the model has learned so far** from the input sequence. |
| Context Vector | The **final hidden state** (hn) of the encoder, also called the **context vector**, is a summary of the whole input. It contains the most important information about the input sentence and is passed to the decoder to help it generate the output. |
| Output Sequence | The **translated or predicted sentence**, created by the decoder. It is generated **one word at a time**, starting with a <start> token and using the context vector. Example: if the input is *“I love you”*, the output might be *“Je t’aime”*. |



\*\*\*\*