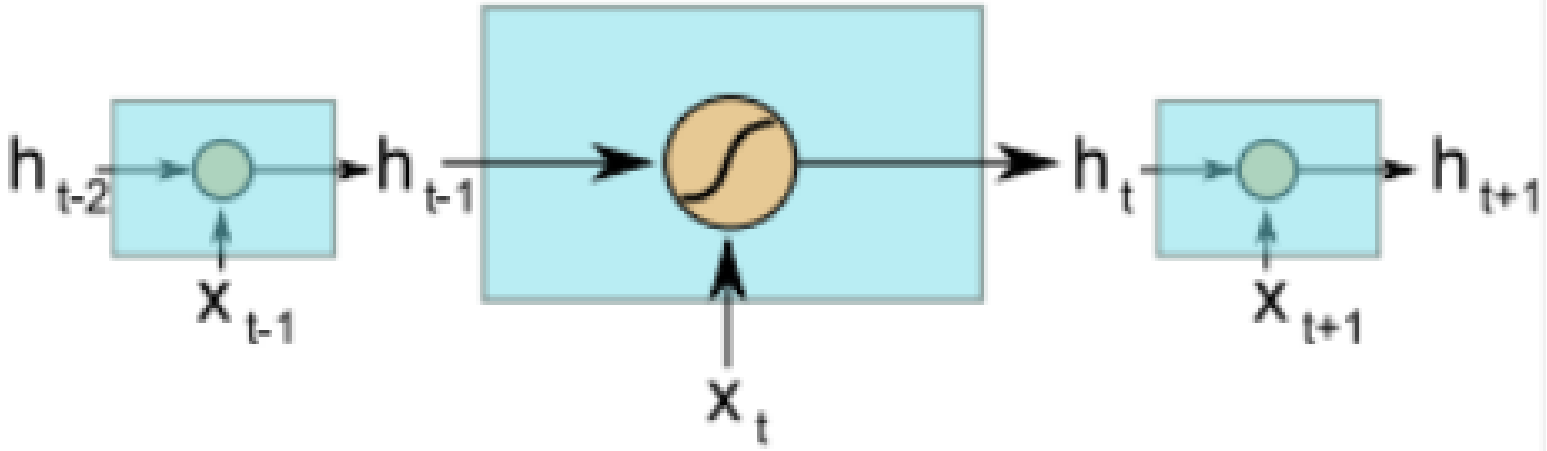
MODULE 3

RNN,LSTM,GRU

RNN

- Recurrent Neural Networks (RNNs) are a class of deep learning models that possess internal memory, enabling them to capture sequential dependencies.
- Unlike traditional neural networks that treat inputs as independent entities, RNNs consider the temporal order of inputs, making them suitable for tasks involving sequential information.
- RNNs apply the same operation to each element in a series, with the current computation depending on both the current input and the previous computations.
- A limitation of simple RNNs is their short-term memory, which restricts their ability to retain information over long sequences. To overcome this, more advanced RNN variants have been developed, including Long Short Term Memory (LSTM), bidirectional LSTM, Gated Recurrent Unit (GRU).



Simple RNN internal operation

depicts a simple recurrent neural network, where the internal memory (h_t) is computed using Equation

$$h_t = g(W x_t + U h_t + b)$$

In this equation, $g()$ represents the activation function (typically the hyperbolic tangent), U and W are adjustable weight matrices for the hidden state (h), b is the bias term, and x denotes the input vector.

PROBLEMS WITH RNN

- Exploding and vanishing gradient problems during backpropagation.
- Gradients are those values which to update neural networks weights. In other words, we can say that Gradient carries information.
- Vanishing gradient is a big problem in deep neural networks. it vanishes or explodes quickly in earlier layers and this makes RNN unable to hold information of longer sequence. and thus RNN becomes short-term memory.
- If we apply RNN for a paragraph RNN may leave out necessary information due to gradient problems and not be able to carry information from the initial time step to later time steps.

HOW TO SOLVE THESE PROBLEMS?

- The reason for exploding gradient was the capturing of relevant and irrelevant information. a model which can decide what information from a paragraph and relevant and remember only relevant information and throw all the irrelevant information
- This is achieved by using gates. the LSTM (Long -short-term memory) and GRU (Gated Recurrent Unit) have gates as an internal mechanism, which control what information to keep and what information to throw out. By doing this LSTM, GRU networks solve the exploding and vanishing gradient problem.
- Almost each and every SOTA (state of the art) model based on RNN follows LSTM or GRU networks for prediction.

LSTM

- **Long Short-Term Memory** is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber.
- A traditional_RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies.
- **LSTMs model** address this problem by introducing a memory cell, which is a container that can hold information for an extended period.
- LSTM architectures are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting

LSTM Architecture

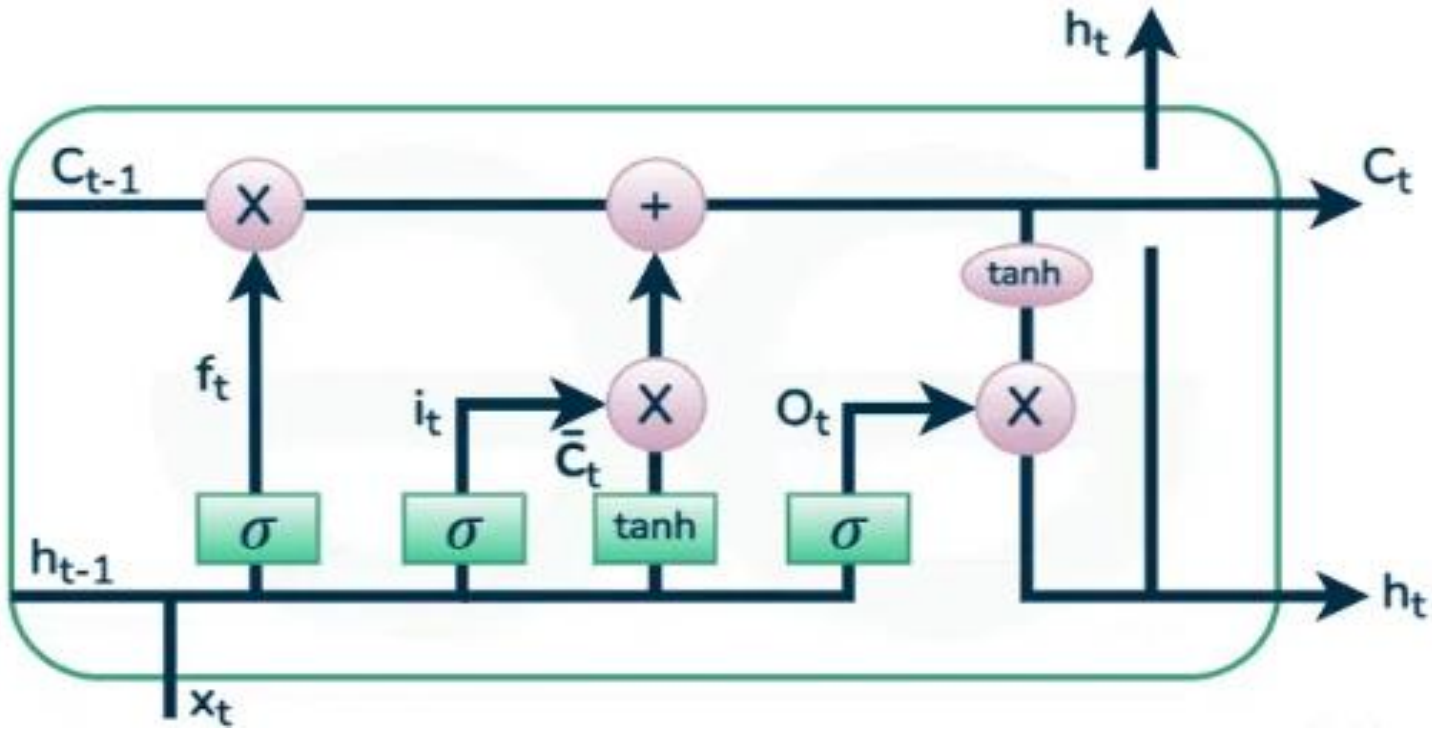
The LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

- The input gate controls what information is added to the memory cell.
- The forget gate controls what information is removed from the memory cell.
- The output gate controls what information is output from the memory cell.

This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.The LSTM maintains a hidden state, which acts as the short-term memory of the network. The hidden state is updated based on the input, the previous hidden state, and the memory cell’s current state.

LSTM Working

LSTM architecture has a chain structure that contains four neural networks and diff
memory blocks called **cells**.



Information is retained by the cells and the memory manipulations are done by the **gates.** There are three gates –

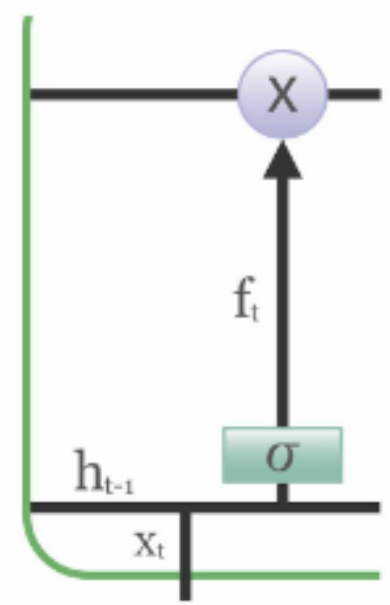
Forget Gate

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. The equation for the forget gate is:

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

where:

- W_f represents the weight matrix associated with the forget gate.
- [h_t-1, x_t] denotes the concatenation of the current input and the previous hidden state.
- b_f is the bias with the forget gate.
- σ is the sigmoid activation function.



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 *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 the useful information. The equation for the input gate is:

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

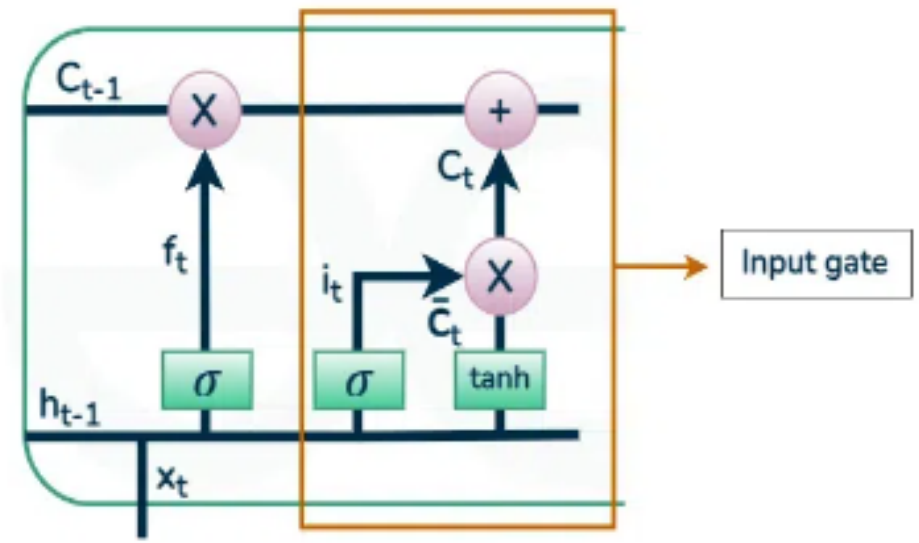
$$\hat{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

We multiply the previous state by f_t, disregarding the information we had previously chosen to ignore. Next, we include $i_t * C_t$. This represents the updated candidate values, adjusted for the amount that we chose to update each state value.

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

where

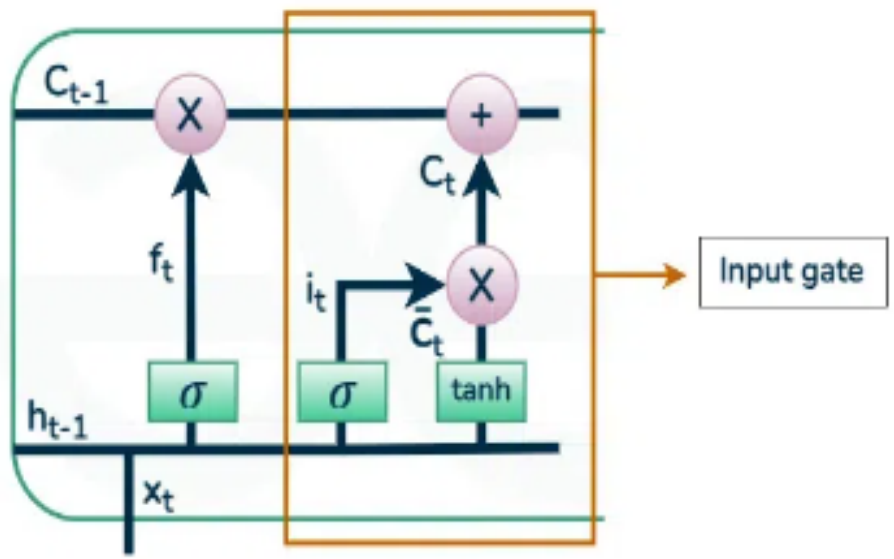
- $\odot$ denotes element-wise multiplication
- tanh is tanh activation function



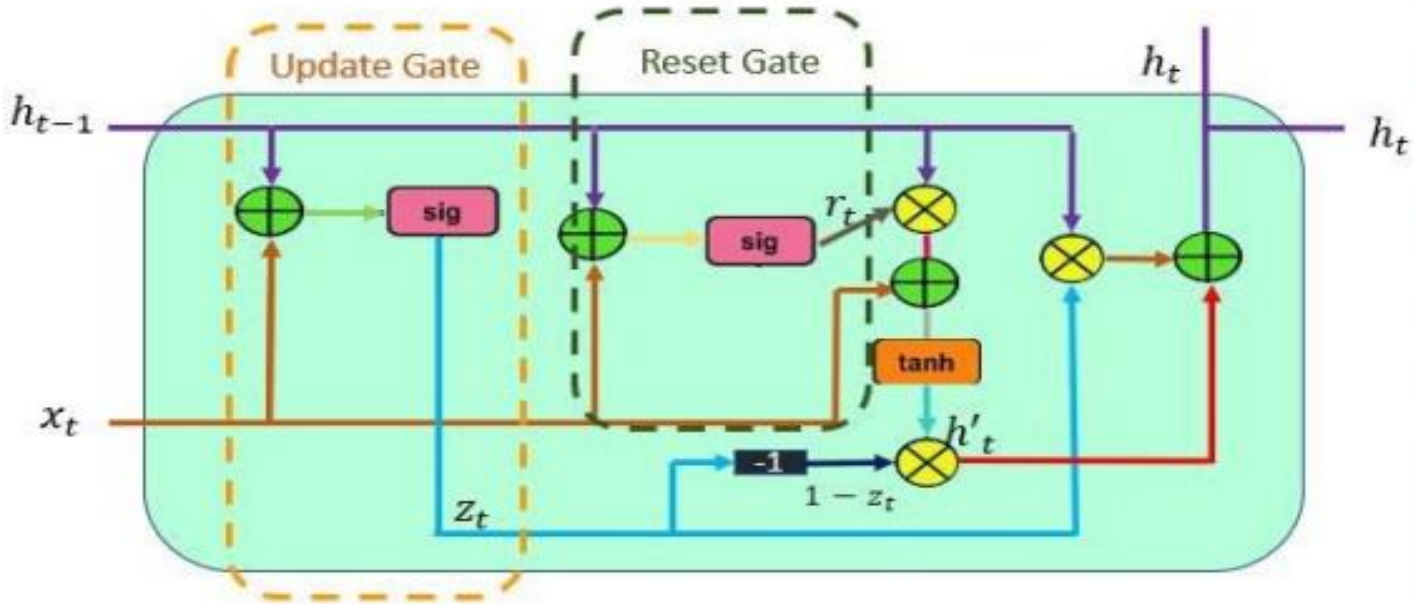
Output gate

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 tanh function on the cell. Then, the information is regulated using the sigmoid function and filter 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. The equation for the output gate is:

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



- GRU (Gated Recurrent Units) are similar to the LSTM networks. GRU is a kind of newer version of RNN. However, there are some differences between GRU and LSTM.
  - GRU doesn’t contain a cell state
  - GRU uses its hidden states to transport information
  - It Contains only 2 gates(Reset and Update Gate)
  - GRU is faster than LSTM
  - GRU has lesser tensor’s operation that makes it faster



Gated Recurrent Network (GRU)

sig = Sigmoid function

tanh = tanh function

= Hadamard Product operation

= addition operation

= vector connections

- 1. Update Gate
  - Update Gate is a combination of Forget Gate and Input Gate. Forget gate decides what information to ignore and what information to add in memory.
- 2. Reset Gate
  - This Gate Resets the past information in order to get rid of gradient explosion. Reset Gate determines how much past information should be forgotten.