Recurrent Neural Network (RNN)
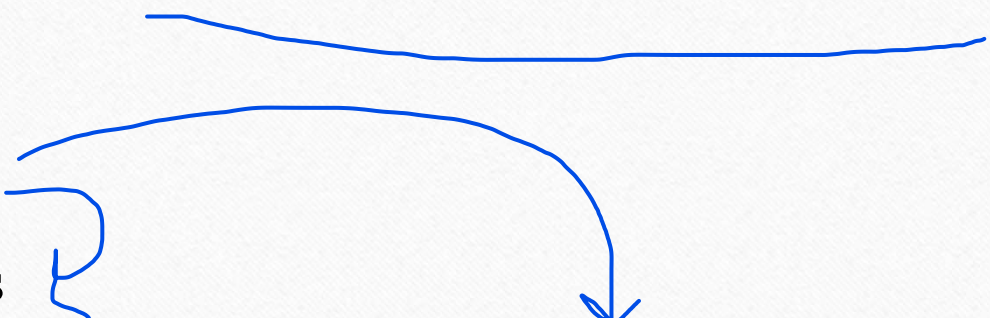
B. Keerthana

Assistant professor

CSE department

GVPCE(A)

Content

1. Introduction,

2. Simple Recurrent Neural Network,

3. LSTM Implementation,

4. Gated Recurrent Unit (GRU),

5. Applications of Recurrent Neural Network

1. Why RNN?

RNN were created because there were a few issues in the feed-forward neural network:

- Cannot handle sequential data
- Considers only the current input
- Cannot memorize previous inputs

The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

Feed-Forward Neural Networks vs RNN

|	MLP	RNN	CNN
Data	Tabular data	Sequence data (Time Series, Text, Audio)	Image data
Recurrent connections	No	Yes	No
Parameter sharing	No	Yes	Yes
Spatial relationship	No	No	Yes
Vanishing & Exploding Gradient	Yes	Yes	Yes

RNN vs CNN

	Convolutional neural network (CNN)	Recurrent neural network (RNN)
ARCHITECTURE	Feed-forward neural networks using filters and pooling	Recurring network that feeds the results back into the network
INPUT/OUTPUT	The size of the input and the resulting output are fixed (i.e., receives images of fixed size and outputs them to the appropriate category along with the confidence level of its prediction)	The size of the input and the resulting output may vary (i.e., receives different text and output translations—the resulting sentences can have more or fewer words)
IDEAL USAGE SCENARIO	Spatial data (such as images)	Temporal/sequential data (such as text or video)
USE CASES	Image recognition and classification, face detection, medical analysis, drug discovery and image analysis	Text translation, natural language processing, language translation, entity extraction, conversational intelligence, sentiment analysis, speech analysis

What is RNN?

- Recurrent neural networks (RNN) are a class of neural networks that are helpful in modeling sequence data.

- The most important component of RNN is the Hidden state, which remembers specific information about a sequence.

- RNNs have a Memory that stores all information about the calculations. It employs the same settings for each input since it produces the same outcome by performing the same task on all inputs or hidden layers.
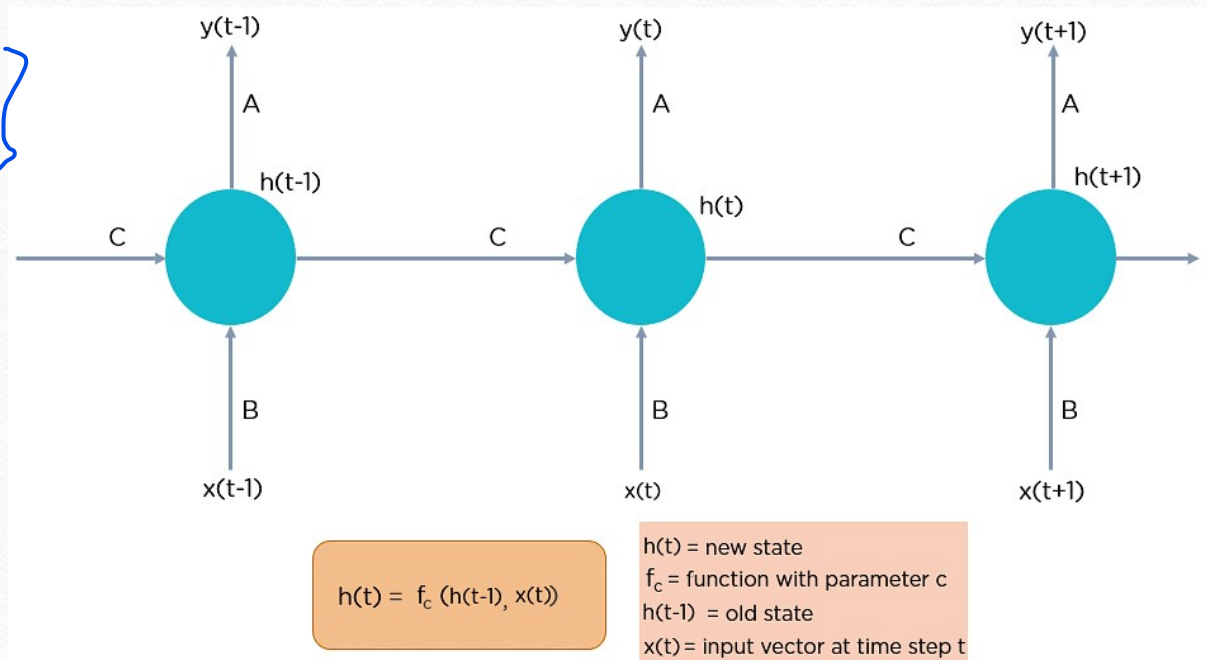
How Does Recurrent Neural Networks Work?

- In Recurrent Neural networks, the information cycles through a loop to the middle hidden layer.
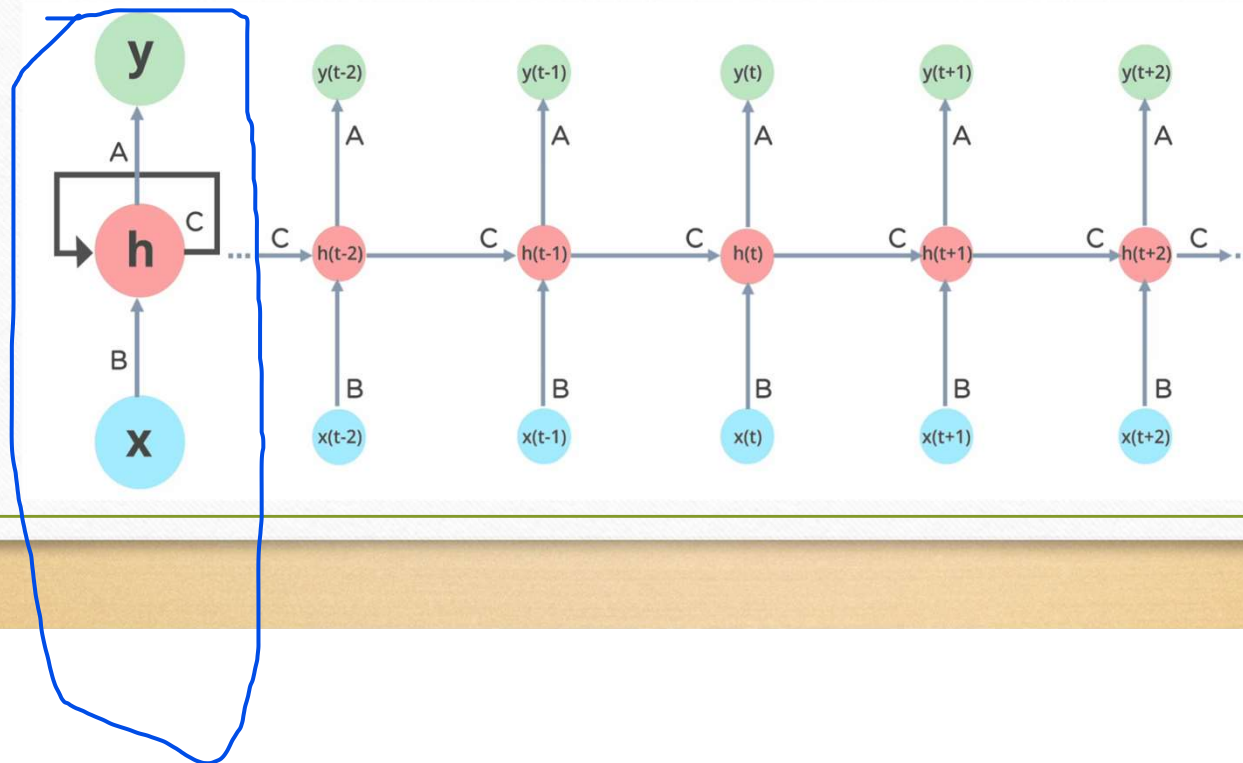
2. Fully connected RNN (simple RNN)

- Here, "x" is the input layer,
- "h" is the hidden layer, and
- "y" is the output layer.
- A, B, and C are parameters
- At time t, the current input
is a combination of input at
x(t) and x(t-1).

$$h(t) = f_c \ (h(t-1), \ x(t))$$

h(t) = new state
f_c = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

Single RNN cell pseudo code

- Compute the hidden state h_t with tanh activation function.
- Using your new hidden state h_t compute the prediction $\hat{y}_t$
- Store h_t , $h_{t-1,}$ x(t) and parameters
- Return h_t and $\hat{y}_t$

How to train RNN?

- To train the RNNs, BPTT is used. "Through time" is appended to the term "backpropagation" to specify that the algorithm is being applied to a temporal neural model (RNN).

- The task of BPTT is to find a local minimum, a point with the least error.

- By adjusting the values of weights, the network can reach minima. This process is called *gradient descent*.

- Gradients (steps) are computed by derivatives, partial derivatives, and chain rule.

BPTT (Back Propagation Through Time)

- Back Propagation through time usually abbreviated as BPTT is just a fancy name for back propagation, which itself is a fancy name for Gradient descent .

- This is called BPTT because we are figuratively going back in time to change the weights, hence we call it the Back propagation through time(BPTT).

- For example in order to determine the gradient at t=6, we would need to backpropagate five steps and sum up the gradients.

- In this we have few drawbacks 1. Vanishing gradient problem, 2. Exploding gradient problem.

  derivatives reaches to 0   derivatives becomes infinite

BPTT implementation

1. Generate input and output data.
2. Normalize the data with respect to maximum and minimum values.
3. Assume the number of hidden layers and number of neurons in the hidden layer.
4. Initialize weights between 0 and 1. let w be the weight connecting input neuron and hidden neuron and w1 be the weight connecting hidden and output neurons in the case of single hidden layer.
5. Compute the input and output at every neuron present in each layer and apply activation function.
6. Find the error and check the tolerance. If the error is above tolerance, update the weights.
7. Repeat step 4 till the error is within tolerance.

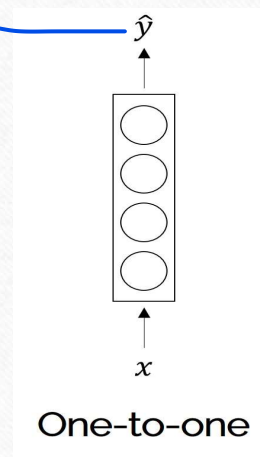finally minimum error point is reached
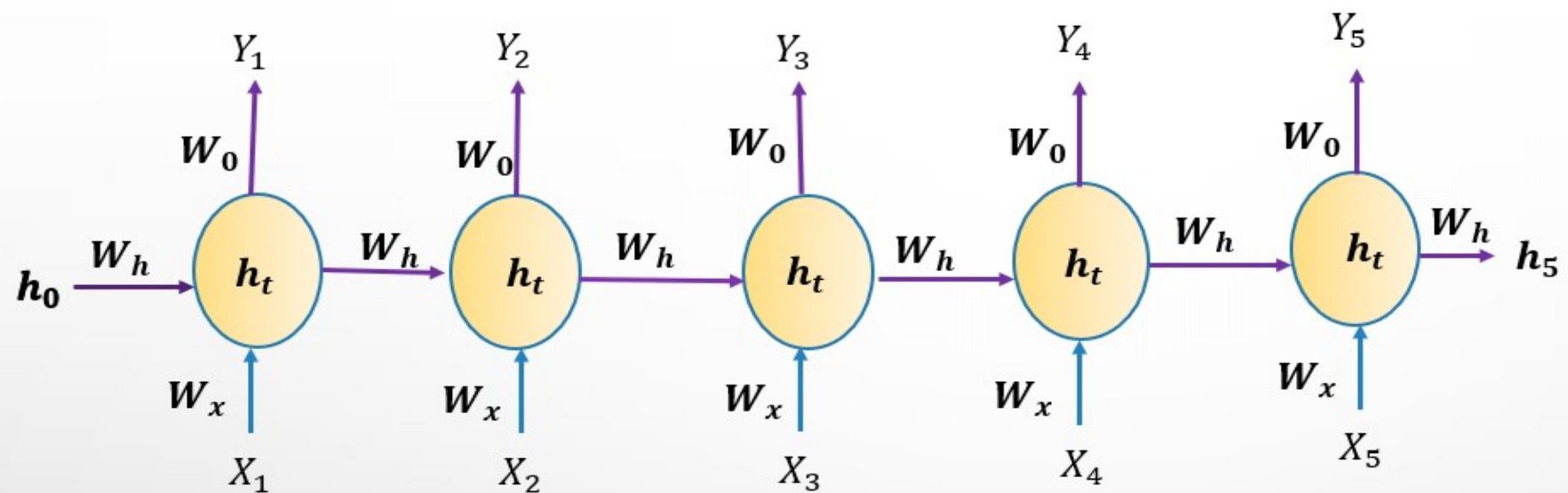
$$h_t = f\left(W_h^T h_{t-1} + W_x^T X_t \right) \quad ------(1)$$

previous word weightage   present input

$$Y_t = softmax\left(W_0^T h_t \right) \quad -----(2)$$

output

RNN topology

One to One: This model is similar to a single layer neural network as it only provides linear predictions. It is mostly used fixed-size input 'x' and fixed-size output 'y' (example: image classification)

One-to-one

RNN topology

- **One to Many**: represents a sequence output (Ex: image captioning acquires an image as input and outputs a sentence of words)
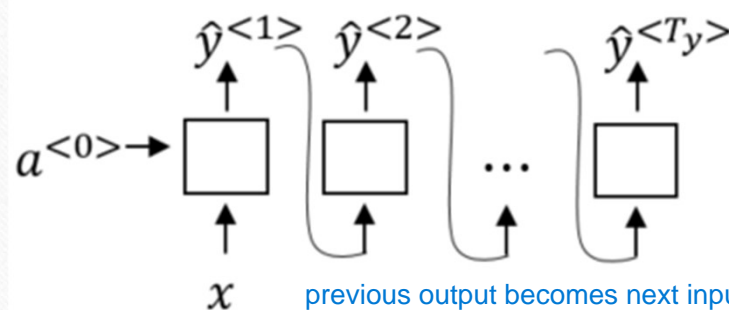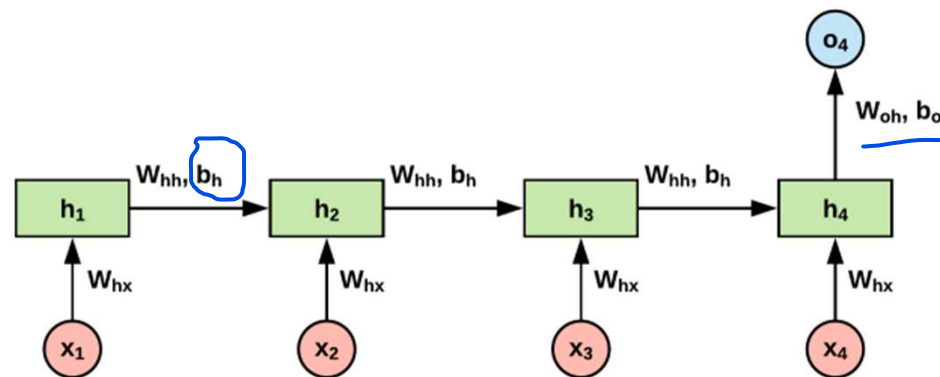

previous output becomes next input and it is a recurrence un till the o/p is found

One to many

RNN topology

- **Many to One** : represents a sequence input (Ex: sentiment analysis where a known sentence is classified as stating positive or negative)

RNN topology

- **Many to Many**:In this, a single frame is taken as input for each RNN unit. A-frame represents multiple inputs 'x', activations 'a' which are propagated through the network to produce output 'y' which are the classification result for each frame. It used mostly in video classification, where we try to classify each frame of the video.
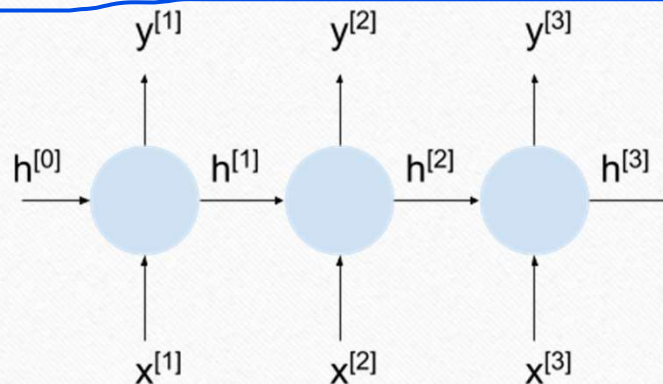
Bi- directional RNNs

- In this neural network, 2 hidden layers running in the opposite direction are connected to produce a single output

- These layers allow the neural network to received information from both past as well as a future state

- For example, given a word sequence: 'I like programming'. The forward layer will input the sequence as it is while the backward layer will feed the sequence in the reverse order 'programming like i'

- The output for this will be calculated by concatenating the word sequence at each time step and generating the weight.

Bi- directional RNNs
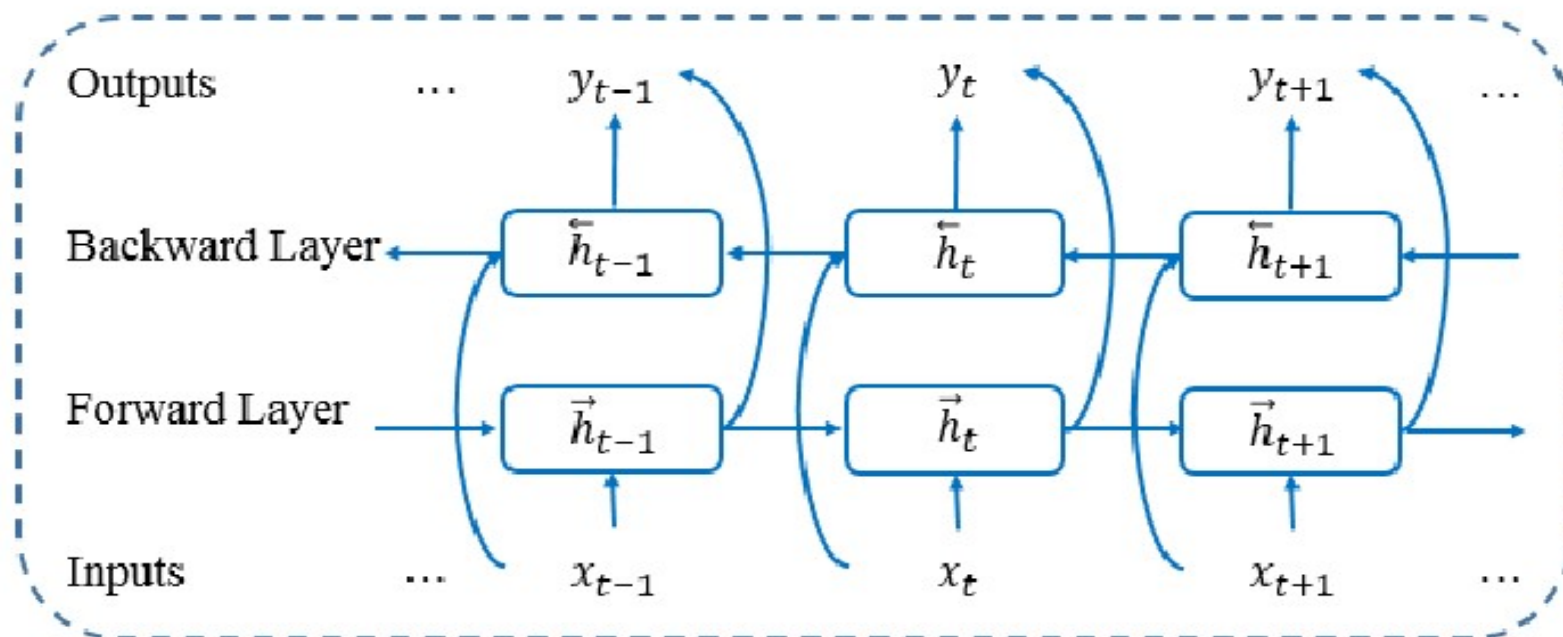
3. LSTM

- LSTM stands for long short-term memory networks, used in the field of deep learning.

- It tackled the problem of **long-term dependencies** of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information.

- LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

What is long term dependency?

- Sometimes we just need to look at recent information to perform the present task. For example, consider a language model trying to predict the last word in " **the clouds are in the sky**". Here it's easy to predict the next word as sky based on the previous words.

- But consider the sentence " **I grew up in France I speak fluent French.** " Here it is not easy to predict that the language is French directly.

- It depends on previous input also. In such sentences it's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

- In theory, RNN's are absolutely capable of handling such "long-term dependencies."

LSTM architecture

LSTM consists of 3 parts:

Forget gate: the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten.

Input gate: the cell tries to learn new information from the input to this cell.

Output gate: the cell passes the updated information from the current timestamp to the next timestamp.
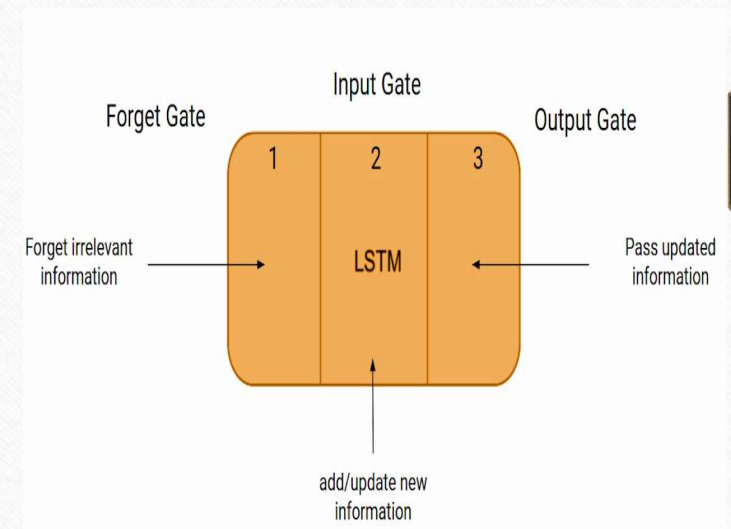
LSTM architecture (cont..)

- LSTM has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp.

- In addition to that LSTM also have a cell state represented by C(t-1) and C(t) for previous and current timestamp respectively.

- Here the hidden state is known as **Short term memory** and the cell state is known as **Long term memory**.
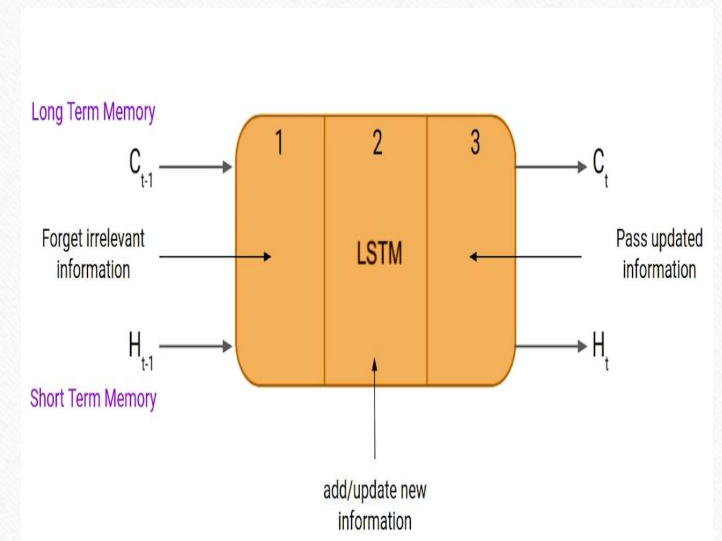
Example for Forget Gate

- In a cell of the LSTM network, the first step is to decide whether we should keep the information from the previous timestamp or forget it. Here is the equation for forget gate.

Forget Gate:

- $f_t = \sigma\left(x_t * U_f + H_{t-1} * W_f\right)$

Xt: input to the current timestamp.

Uf: weight associated with the input

Ht-1: The hidden state of the previous timestamp

short term

Wf: It is the weight matrix associated with hidden state

"Ex: Bob is nice person. Dan on the other hand is evil."

Example for Forget Gate

- Later a sigmoid function is applied over it. That will make f_t a number between 0 and 1. This f_t is later multiplied with the cell state of the previous timestamp as shown below.

$$C_{t-1} * f_t = 0 \quad ...if \ f_t = 0 \ \text{(forget everything)}$$

$$C_{t-1} * f_t = C_{t-1} \quad ...if \ f_t = 1 \ \text{(forget nothing)}$$

- If f_t is 0 then the network will forget everything and if the value of ft is 1 it will forget nothing.

Example for Input Gate

Ex: **"Bob knows swimming. He told me over the phone that he had served the navy for four long years."**

- First, he used the phone to tell or he served in the navy. In this context, it doesn't matter whether he used the phone or any other medium of communication to pass on the information.

- The fact that he was in the navy is important information and this is something we want our model to remember.

Example for Input Gate

- Input gate is used to quantify the importance of the new information carried by the input. Here is the equation of the input gate

Input Gate:

- $i_t = \sigma(x_t * U_i + H_{t-1} * W_i)$

Xt: Input at the current timestamp t

Ui: weight matrix of input

Ht-1: A hidden state at the previous timestamp

Wi: Weight matrix of input associated with hidden state

Example for Input Gate

New information

$$N_t = \tanh(x_t * U_c + H_{t-1} * W_c) \text{ (new information)}$$

- If the value is of N_t is negative the information is subtracted from the cell state and if the value is positive the information is added to the cell state at the current timestamp.

- However, the N_t won't be added directly to the cell state. Here comes the updated equation

$$C_t = f_t * C_{t-1} + i_t * N_t \text{ (updating cell state)}$$

Example for Output Gate

Example**: "Bob single-handedly fought the enemy and died for his country. For his contributions, brave_____ ."**

- In the sentence only Bob is brave, we can not say the enemy is brave or the country is brave. So based on the current expectation we have to give a relevant word to fill in the blank.

Output Gate:

- $o_t = \sigma(x_t * U_o + H_{t-1} * W_o)$

- Now to calculate the current hidden state we will use O_t and tanh of the updated cell state. (C_t is long term memory)

$$H_t = o_t * tanh(C_t)$$

Overall LSTM architecture
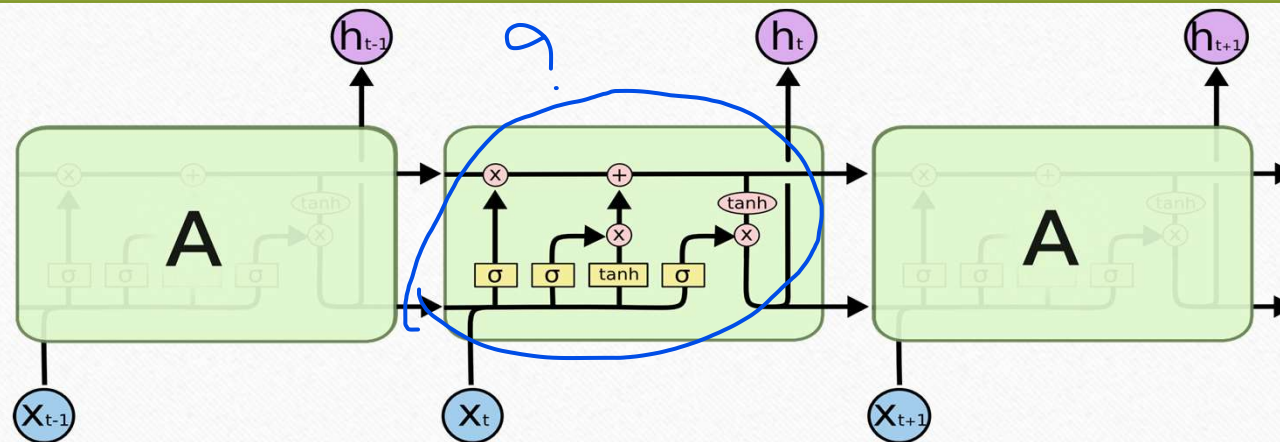
4. Gated Recurrent Unit (GRU)

- The Gated Recurrent Unit is a type of (RNN) that, in certain cases, has advantages over long short term memory (LSTM).

- GRU uses less memory, faster than LSTM and also solves the problem of vanishing gradient problem by using of two gates, the **update gate** and **reset gate**.

- GRU doesn't maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM is incorporated into the hidden state of the GRU.

- This collective information is passed onto the next Gated Recurrent Unit.
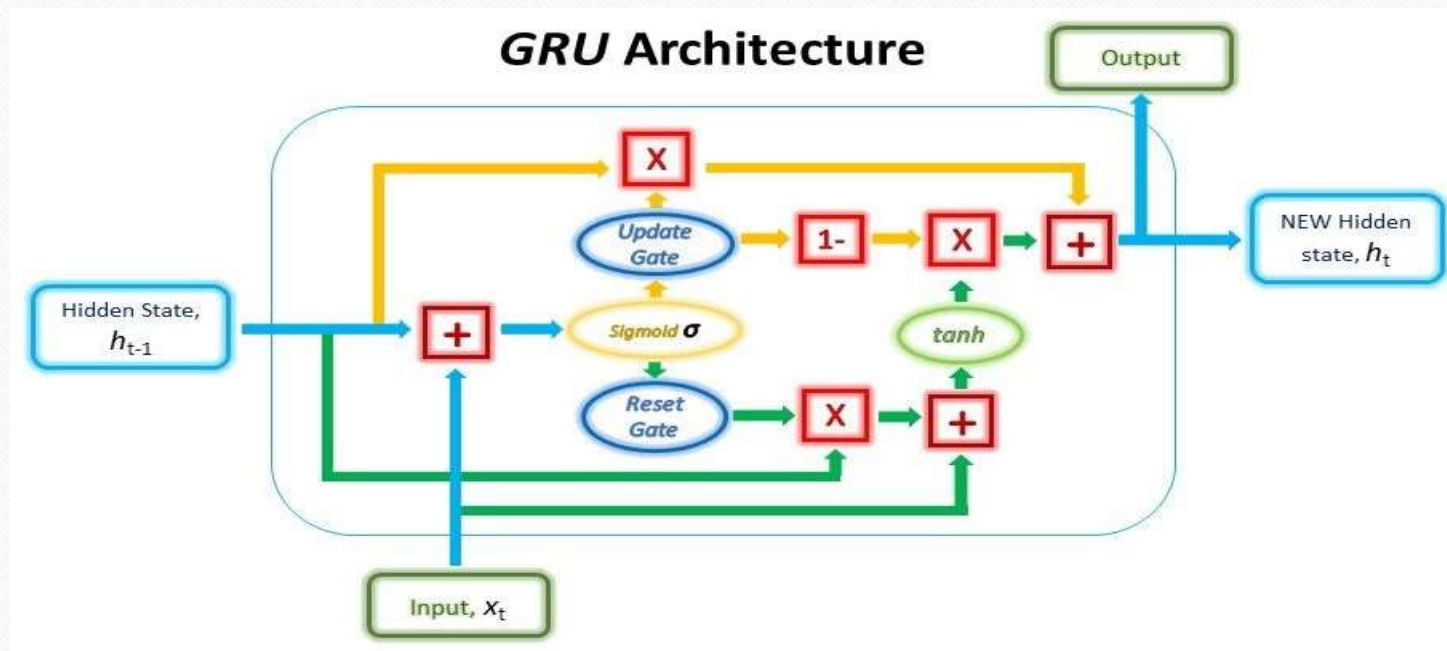
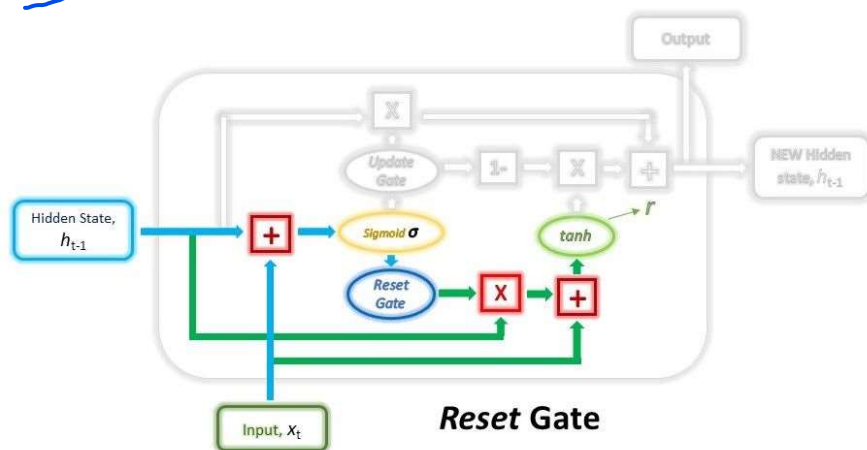# Gated Recurrent Unit (cont..)
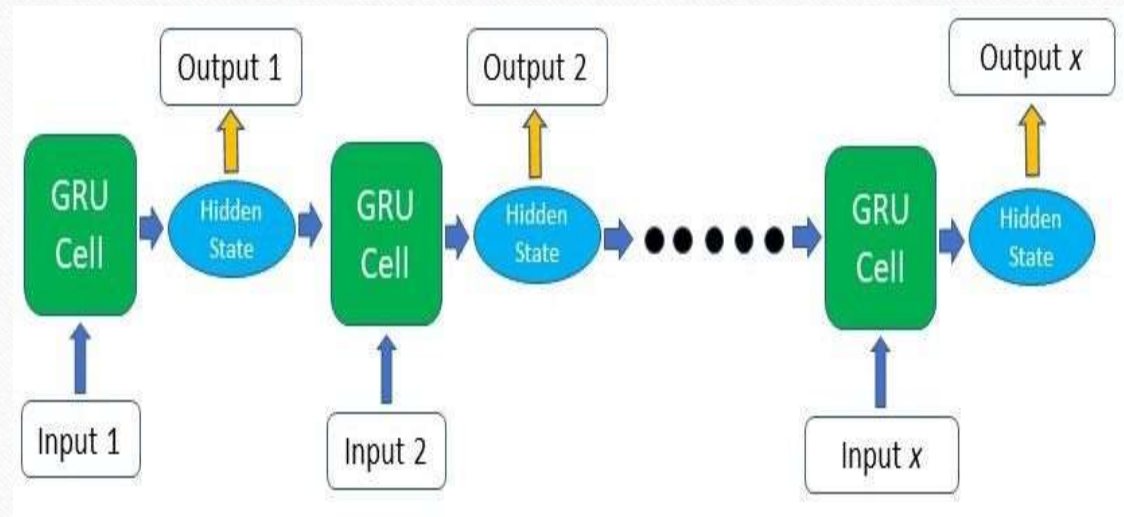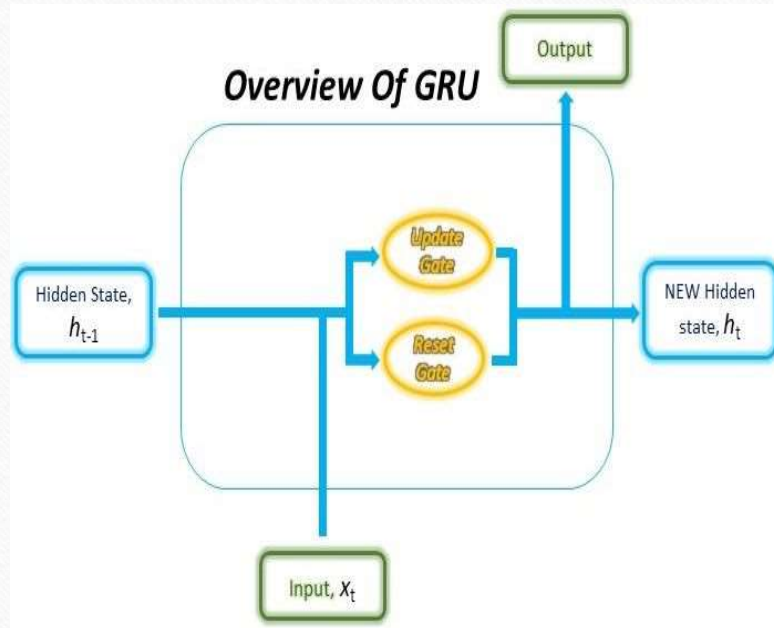
GRU vs LSTM
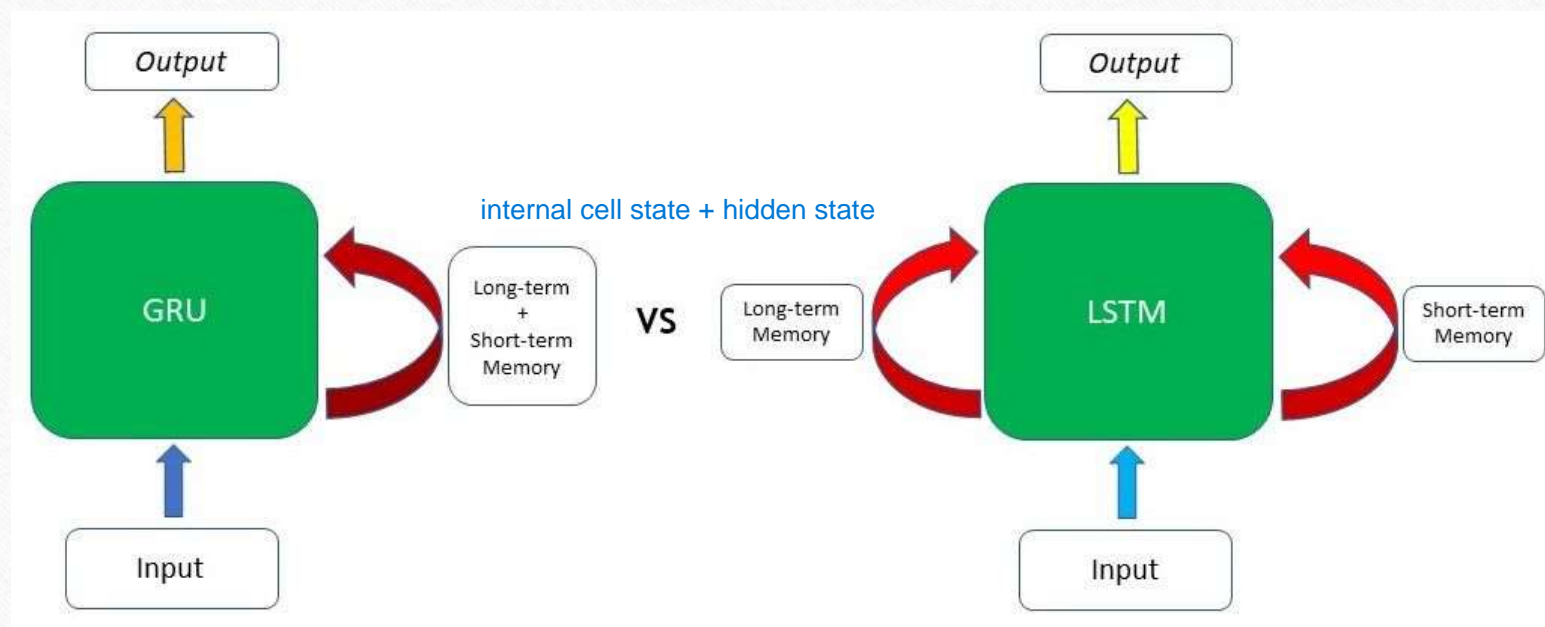
Internal working of GRU

Reset Gate

1. Reset Gate(r): It determines how much of the past knowledge to forget. This gate is derived and calculated using both the hidden state from the previous time step and the input data at the current time step.

Reset Gate

$$gate_{reset} = \sigma(W_{input_{reset}} \cdot x_t + W_{hidden_{reset}} \cdot h_{t-1})$$

$$r = tanh(gate_{reset} \odot (W_{h_1} \cdot h_{t-1}) + W_{x_1} \cdot x_t)$$
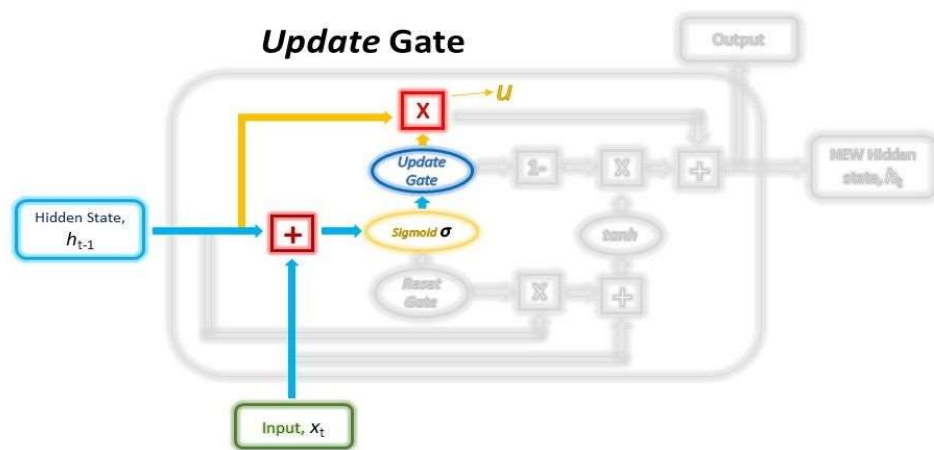
Reset Gate (cont..)

just explain what we are doing in above equation

- After doing the back propagation, **previous hidden state** will first be multiplied by a trainable weight and will then undergo an element-wise multiplication with the **reset vector**.

- This operation will decide which information is to be kept from the previous time steps together with the new inputs.

- At the same time, the **current input** will also be multiplied by a trainable weight before being summed with the product of the **reset vector** and **previous hidden state** above.

- Lastly, a non-linear activation *tanh* function will be applied to the final result to obtain *r* .

Update Gate

2. Update gate(z): It determines how much of the past knowledge needs to be passed along into the future. Just like the **_Reset_** gate, the gate is computed using the previous hidden state and current input data.

$$gate_{update} = \sigma(W_{input_{update}} \cdot x_t + W_{hidden_{update}} \cdot h_{t-1})$$

$$u = gate_{update} \odot h_{t-1}$$

Update Gate (cont..)

- Both the *Update* and *Reset* gate vectors are created using the same formula, but, the weights multiplied with the input and hidden state are unique to each gate.

- The *Update* vector will then undergo element-wise multiplication with the **previous hidden state** to obtain *u* , which will be used to compute our final output later.

- The purpose of the *Update* gate here is to help the model determine how much of the past information stored in the **previous hidden state** needs to be retained for the future.

Combining the outputs

- We will be taking the element-wise inverse version of the same **Update** vector (1 - **Update** gate) and doing an element-wise multiplication with our output from the **Reset** gate, *r*.
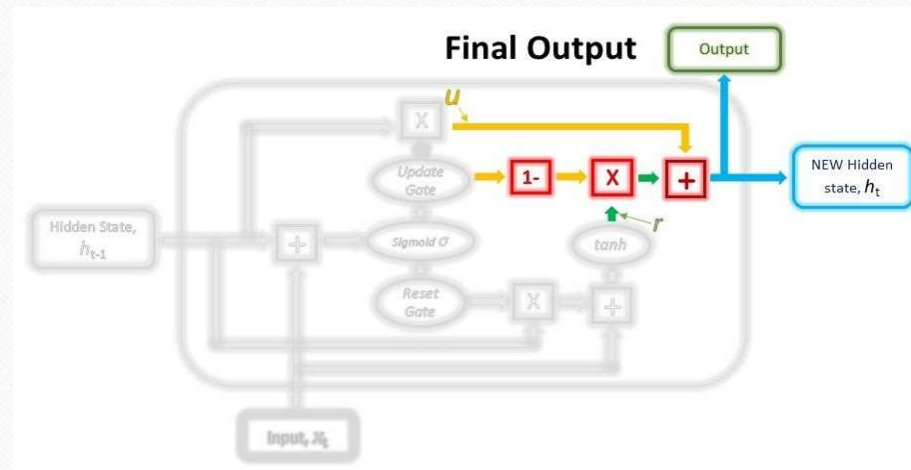
Combining the outputs

- The purpose of this operation is for the *Update* gate to determine what portion of the new information should be stored in the **hidden state**.

- Lastly, the result from the above operations will be summed with our output from the *Update* gate in the previous step, *u*. This will give us our **new** and **updated hidden state**.

$$h_t = r \odot (1 - gate_{update}) + u$$

Applications of RNN

- Speech Recognition

- Music Generation

- Machine Translation

- DNA Sequence Analysis

- Video Activity Recognition

- Sentiment Analysis