UNIT- 4

Recurrent Neural Network (RNN)

Multilayered RNN	RNN
$x_i \rightarrow y_i$	$(x_i, x_{i-1}) \rightarrow y_i$
? are you how hello (no meaning)	Hello! How are you? (meaningful)

There are many tasks in everyday life which makes no sense if their order of sequence is changed.

Sequence Important

1. The sequence of words define their meaning
2. A time series of data where time defines the occurence of events

→ There are multiple such cases where in sequence of information determines the event itself.

→ If we are trying to use such data where sequence matters, we need

a network which has access to some prior knowledge about the data to completely understand it. Such network is called RNN.

Need of RNN

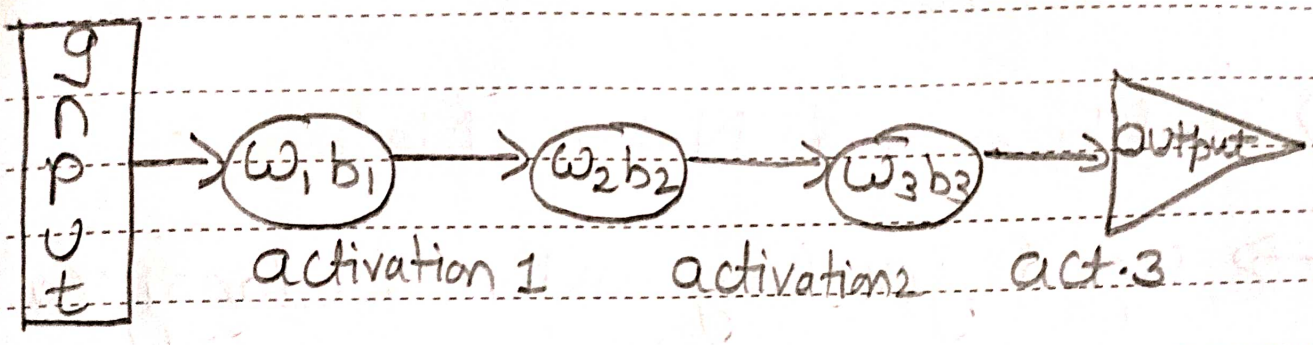
i) Prediction of next word in a sentence

If you want to predict the next word in a sentence, you better know which word came before it

In traditional Neural networks all the inputs & outputs are independent of each other

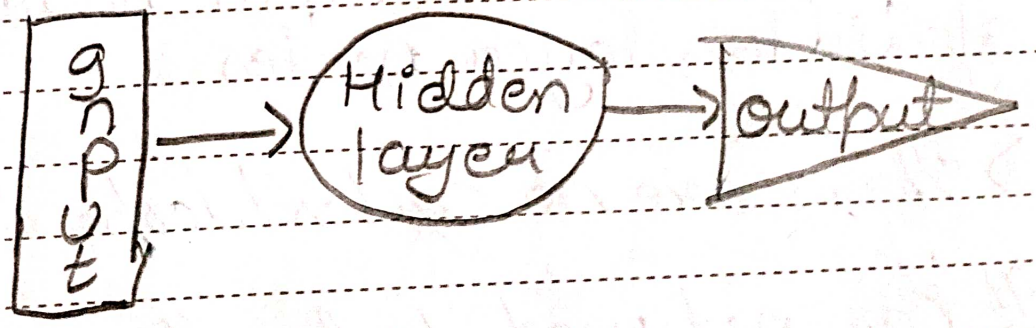
→ In deep network, multiple hidden layers are present. The input layer receives the input & the 1st hidden layer activations are applied, then activations are sent to the next hidden layer & successive activations through the layers to produce the output

→ Here each hidden layer is characterized by its own weights, biases & activations Therefore they behave independently



→ But in cases like when it is required to predict the next word of a sentence, the previous words are required & hence there is a need to remember the previous words.

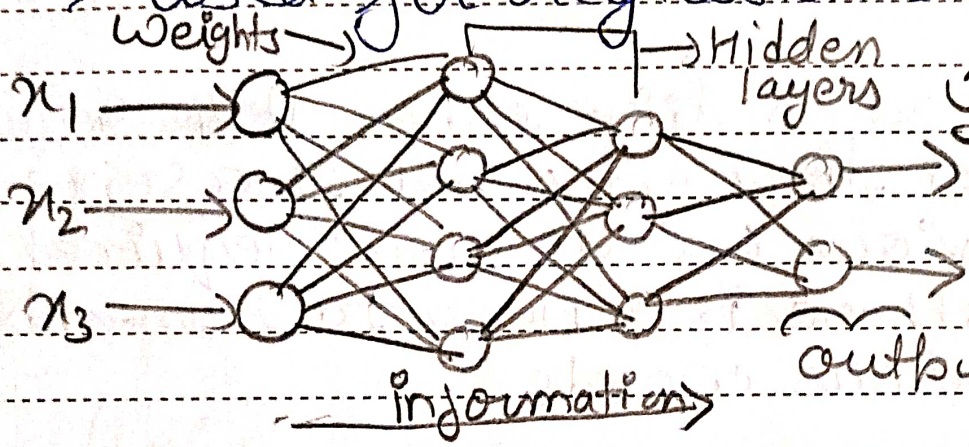
→ For this we use RNN, which ~~remb~~ remember some information about a sequence with the help of hidden laye



Difference Between Feed Forward & Recurrent Neural Networks

1) Feed Forward Neural Network

→ used for regression & classification



→ In a feed forward network, information flows from only in forward direction from input nodes to output nodes with help of hidden layer nodes

→ There are no cycles / loops in the network

→ It can not used to handle sequential data (like sentence)

→ It consider only the current state for prediction of output

→ It cannot ~~me~~ have memory of previous inputs that means it has no memory

→ It sends all inputs at the same time
→ Inputs are independent of each other
→ It use different (weight & bias) parameter at different layers
→ It is only used for Regression & classificatio

ii) Recurrent Neural Network

→ RNN are called Recurrent because they perform the same task for every element of a sequence with the output being depended on the previous computations.

→ RNNs have a memory which stores information about what has been calculated so far.

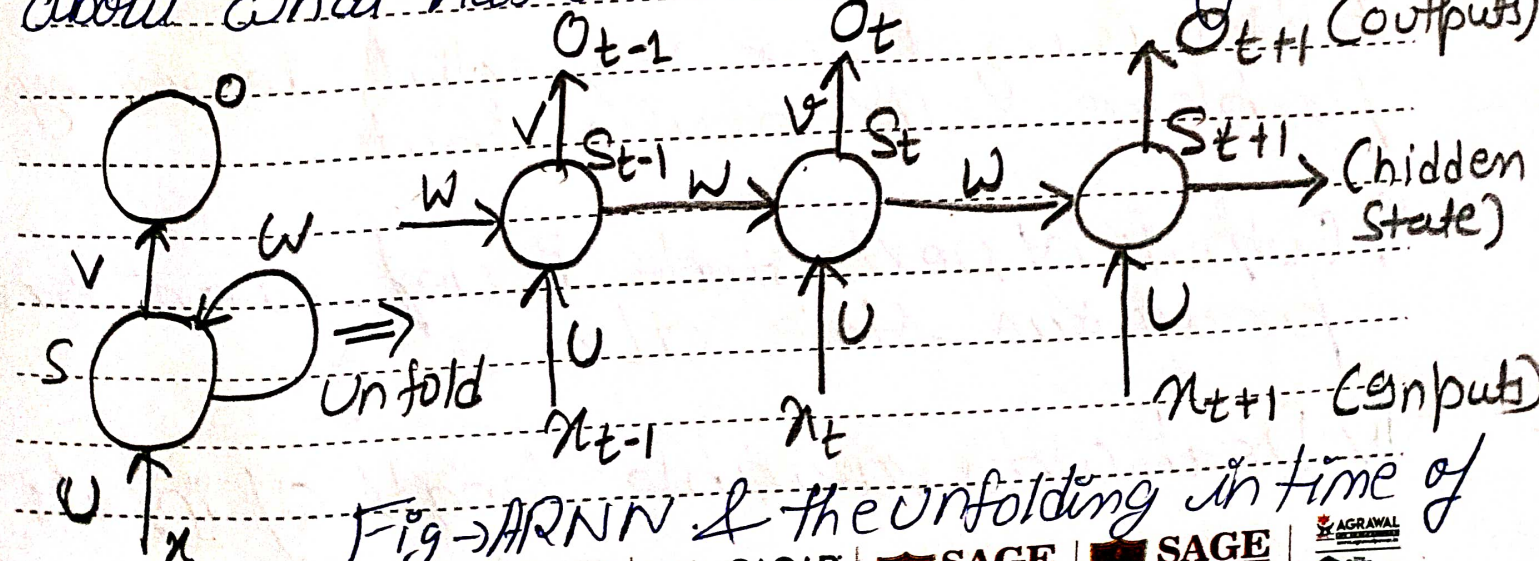


Fig → RNN & the unfolding in time of the computation involved in its

→ RNN uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output.

→ Hence, RNN reduces the complexity of parameters as compared to other neural networks

Nos. of layers in RNN = Nos. of ~~units~~ words in a sentence/sequence

→ If a sequence is a sentence of 3 words the RNN would be unrolled into a 3-layer NN.

→ Output from previous step are fed as input to the current step

* RNN is used for sequential learning problems & it ensures that

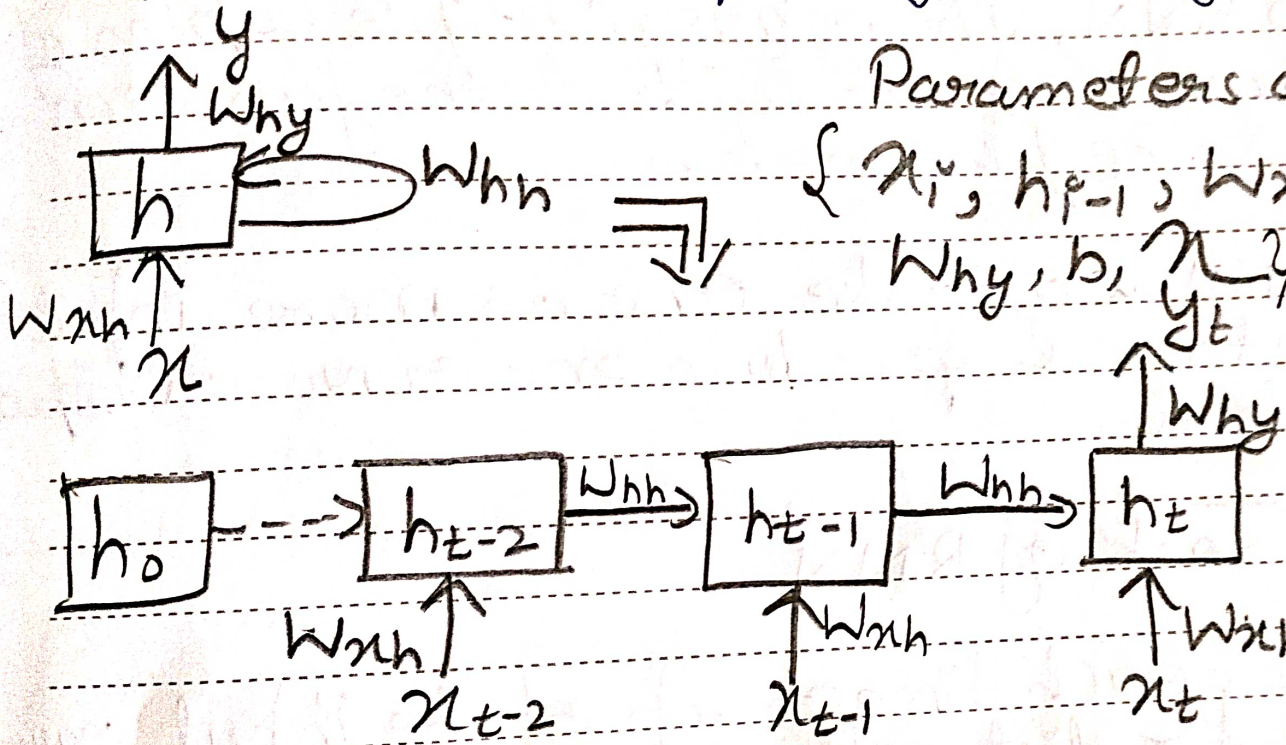
i) Output of next state is dependent on previous state also.

ii) Deal with variable length of inputs

iii) Function executed at each time step is the same

Working of RNN

→ RNN save the output of a layer & feed them back to the input in order to predict the output of the layer



→ Here, there is parameter sharing for

i) W_{hh} → Show relation b/w x_t & x_{t-1} is similar to relation b/w x_{t-1} & x_{t-2}

ii) W_{xh} → Identical feature extraction from input

→ In RNN, outputs are calculated not

only by weights ~~application~~ applied on inputs like a regular NN, but also by a "hidden state" vector representing the context based on prior input / output

→ In RNN, there are multiple copies of same network with same function & same parameters

→ RNN can take one or more input vectors & produce one or more output vectors

~~A~~ Task of RNN

1) For each timestamp of the input sequence 'x' predict output 'y' (synchronously)

2) For the input sequence x predict the ~~Sclalar~~ Scalar value of y (like a end of sequence)

3) For the input sequence x of length T_x generate output sequence y of different length T_y

∴ New / Current hidden state h_t can be calculated as

$$h_t = f(h_{t-1}, x_t)$$

OR $$h_t = f(W_{hh} h_{t-1} + W_{xh} x_t + b)$$

where

$h_t \rightarrow$ new state / hidden state vector at timestep 't'

$h_{t-1} \rightarrow$ previous state / —"— t-1"

$W_{xh} \rightarrow$ Weight matrix from input to hidden

$x_t \rightarrow$ current input / input at time step 't'

$W_{hh} \rightarrow$ Weight matrix from input to hidden

$W_{hy} \rightarrow$ Weight matrix from hidden to output

$b \rightarrow$ bias parameter vector

$f \rightarrow$ activation function (let tanh)

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b)$$

output vector at time step t can be calculated as

$$\boxed{y_t = W_{hy} \cdot h_t}$$

Training of RNN

1) Words are first transformed to machine readable vectors (x_t)

2) RNN process the sequence of vectors one by one that is single time step of the input is supplied to the RNN. (x_t is supplied to the RNN

3) Calculation of hidden state h_t (using above formula)

4) The current h_t becomes h_{t-1} for the next time step

5) We can go many time steps as the problem demands & combine the information from all the previous state.

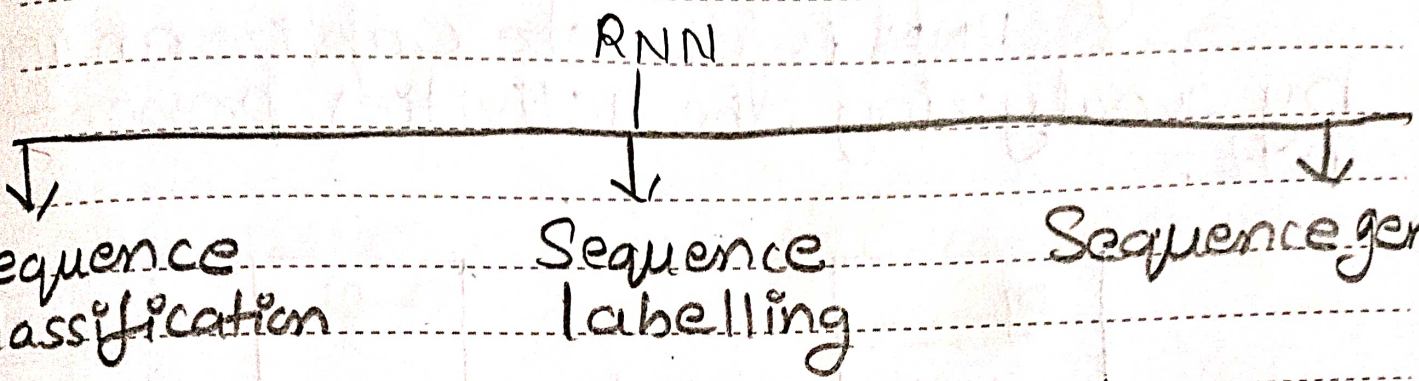
6) Finally RNN calculate output y_t (using above formula)

7) The output is then compared to the actual outputs & the error is generated.

8) The error is then back propagated to the network to update weights (using

Back propogation through time) & the network is trained

RNN Applications



1) Sequence Classification:-

Ex

A) Sentiment Classification

→ I ~~love~~ like my new watch ⇒ Positive Sentimen
→ I didn't like my new watch ⇒ Negative Sentiment

B) Video Classification :→ Many to one
I/P O/P

2) Sequence labelling:-

Ex

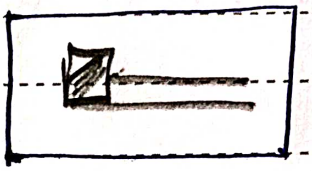
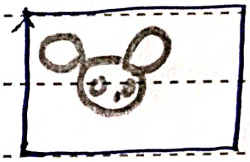
A) Part of speech tagging
(Noun, Verb, pronoun)

B) Named Entity recognition (M:M)

→ I/P: Rohan & Sohan are good friends
O/P: 1 0 1 0 0 0

Word prediction:-
I/P → Happy
O/P → Happy Birthday

c) Image Captioning : (1:M)
RNN is used to caption an image by analyzing the activities present in it.

O/P Image of playground Image of cat

→ Here, we have an image for which we need a textual description

Input → Single input (image)
Output → description of varying length

3) Sequence Generation
Ex
Language / Machine Translation
RNN can be used to translate the input to different language as output

English → Hindi
Hindi → French

Here, we have some text in a source language & we wish to translate it in another language.

m (Input) – Varying length
m (output) – Varying length

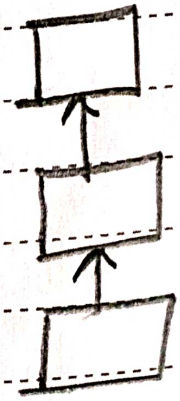
⇒ As each language has its own semantics and varying length for same sentance

Hence, RNN can be used for mapping inputs to outputs of varying type & lengths

Various Architecture of RNN

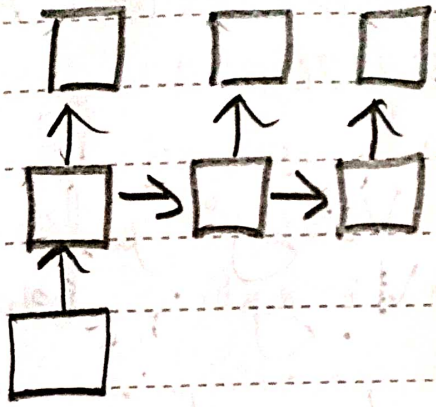
→ RNN has the flexibility for handling various types of data

A) One to One (Image classification) / Vanilla neural network



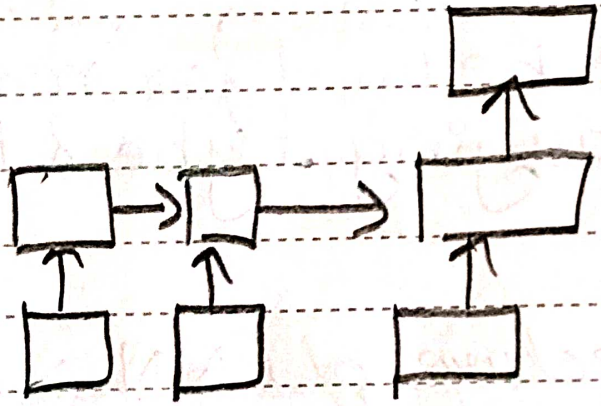
Input - 1
output - 1

B) One to Many (Image Captioning)



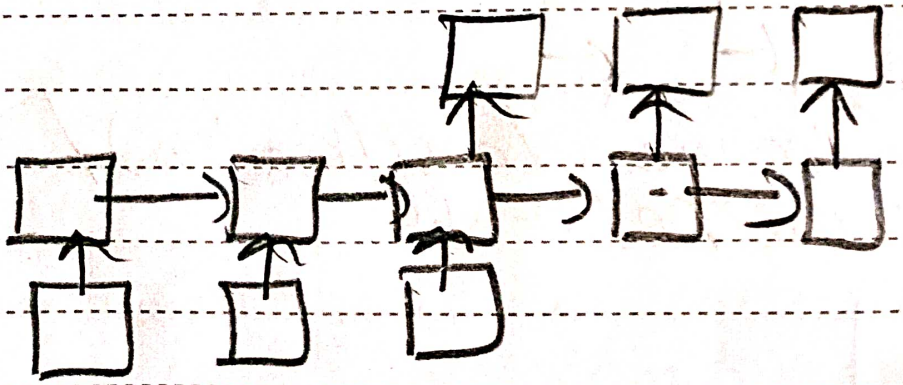
Input - 1
Output - Many

C) Many to ~~Many~~ one (Sentiment Analysis)



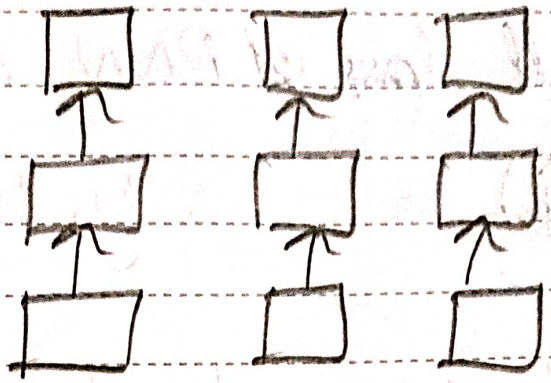
Input - Many
Output - 1

D) Many to Many (~~Language~~ Machine translation)



Input - Many
Output - Many

5) Many to Many (Labelling each frame of videos)



Backpropagation Through time Algorithm for RNN (BPTT)

→ It is the application of Backpropagation training algorithm to RNN applied to sequence data like a time series.

OR Backpropagation algorithm that works on sequence in time is called BPTT.

→ The main goal of using this algorithm is to obtain the parameters that optimize the cost function.

Here, formula used are-

$$h_t = \sigma(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b)$$

$$\hat{y}_t = \sigma(W_{hy} \cdot h_t)$$

→ Let error is calculated using ~~err~~ cross entropy loss function

So, overall loss of RNN

$$E(y, \hat{y}) = -\sum_t y_t \log \hat{y}_t$$

→ For each training epoch — start by training on shorter sequences & then train on progressively longer sequences until the length of max sequence (1, 2, - - - N-1, N)

→ For each length of sequence k unfold the network into a normal feedforward network that has k hidden layers

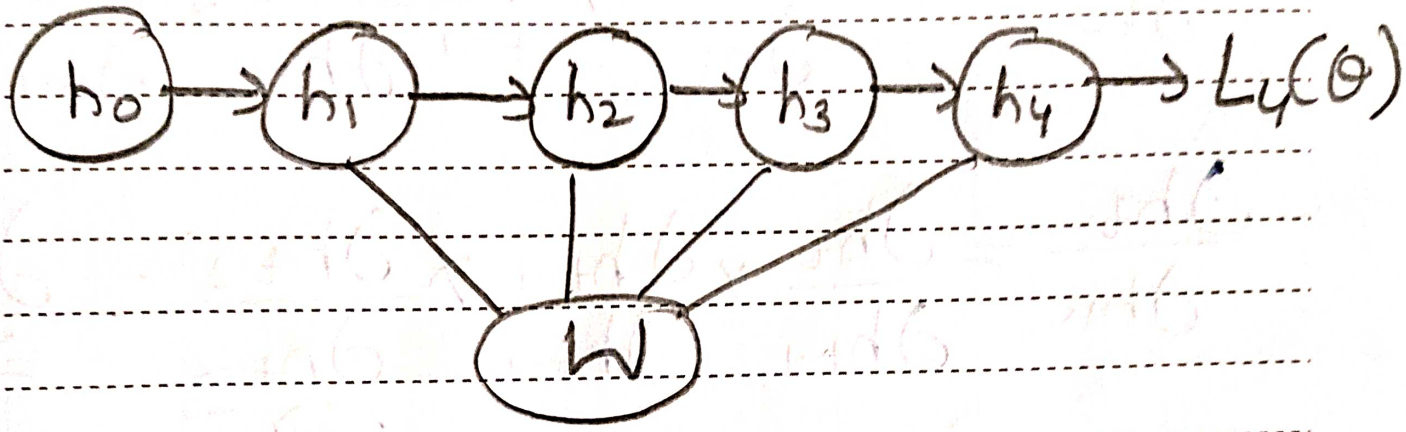
→ k words in a sentence — unfold to k hidden layers

→ BPTT works by unrolling all input time-stamps

→ Each timestamp has one input timestamp, one copy of the network & one output

→ Errors are then calculated & accumulated for each time stamp.

→ Backpropagate errors across the unfolded network & update the weights.



$$\frac{\partial L_4(\theta)}{\partial W} = \frac{\partial L_4(\theta)}{\partial h_4} \times \frac{\partial h_4}{\partial W}$$

$$\frac{\partial h_4}{\partial W} = \frac{\partial h_4}{\partial W} + \frac{\partial h_4}{\partial h_3} \times \frac{\partial h_3}{\partial W} + \frac{\partial h_4}{\partial h_3} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_2}{\partial W}$$

$$+ \frac{\partial h_4}{\partial h_3} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial W}$$

$$\frac{\partial h_4}{\partial W} = \sum_{K=1}^{4} \frac{\partial h_4}{\partial h_K} \times \frac{\partial h_K}{\partial W}$$

$$\frac{\partial L_4(\theta)}{\partial w} = \frac{\partial L_4(\theta)}{\partial h_4} \cdot \sum_{k=1}^{4} \frac{\partial h_4}{\partial h_k} \times \frac{\partial h_k}{\partial w}$$

In general at time stamp 't'

$$\frac{\partial L_t(\theta)}{\partial w} = \frac{\partial L_t(\theta)}{\partial h_t} \cdot \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \times \frac{\partial h_k}{\partial w}$$

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times \frac{\partial h_{t-2}}{\partial h_{t-3}} \cdots \frac{\partial h_{k+1}}{\partial h_k}$$

- high → Exploding gradient
- Low/less → Vanishing

If we have a long chain of gradient terms we can have two types of problems

1) Vanishing Gradients:

Gradient is going to be a product of many terms & if all these terms are very small then the gradient will vanish

ii) Exploding Gradient:-

It is going to be a product of many terms & if all these terms are very large then the gradients will explode.

Limitations RNN

1) For deep RNN, RNN suffers by vanishing / exploding gradients problem

2) During backpropagation, RNNs suffers from the vanishing gradient problem

3) Gradients are values used to update a Neural Network weights

→ The vanishing gradient problem is when the gradient shrinks as it back propagates through time.

→ If a gradient value becomes extremely small it doesn't contribute too much learning

→ It cannot process very long sequences if using tanh or ReLU as an activation function

→ Training an RNN is very difficult task

4) RNNs suffer from short-term memory. If a input sequence is long enough, RNNs may leave out important information from the beginning.

Ex If input is a long paragraph of text to do prediction.

RNNs can forget what it seen in longer sequences, thus having a short-term memory.

Long Short Term Memory (LSTM)

* RNN has two limitations

1) Due to the vanishing gradient problem, RNNs effectiveness is limited. When it needs to go back deep in to the context

2) There is no finer control over which part of the context needs to be carried forward & how much of the part needs to be forgotten

i) KNN → Overwrites its content each timestep unit

ii) LSTM:- It is able to decide whether to keep the existing memory via the introduced gates.

→ Gradient are values used to update a network weights.

→ In RNN, vanishing gradient problem occurs when the gradient vanish / reduce / shrinks as it backpropagates through time.

→ If a gradient value becomes extremely small it doesn't contribute too much learning.

$$W' = W - \eta \times \frac{\partial L}{\partial W}$$

$\eta \rightarrow 0.999$

$W = 1$

$W' = 0.001$ (small change in weight value)

→ So layers that get a small gradient update stops learning (usually early layers)

→ As earlier layers don't learn, RNNs can forget

What it seen in longer sequences thus having a short term memory

~~RNN~~ X LSTM

⇒ RNN suffers from short term memory if a sequence / sentence is long enough, it will not carry information from earlier time step to later ones

Forex

If we try to process a paragraph of text consists of many words to do predictions using RNN, it may leave out important information from the beginning

⇒ Like Consider a language model trying to predict the next word based on the previous one

Ex-1 The clouds are in the ___

Ans: Sky

In such cases where the gap between the relevant information & the place that its needed is small. RNNs can learn to

use the past information.

Ex2:- I grew up in France ------ I speak fluent ____

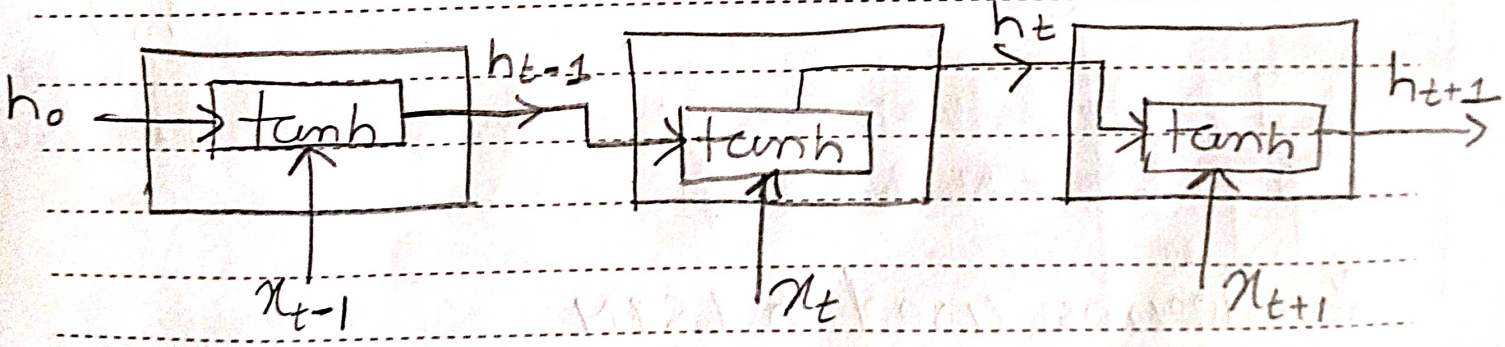
Ans → French

Here, two gap between the relevant information & the point where it is needed to become very large. RNN unable to predict.

⇒ As the gap grows, RNNs become unable to learn to connect the information {short term memory}

LSTM:-

Long short term memory networks are special kind of RNN, capable of learning long term dependencies.



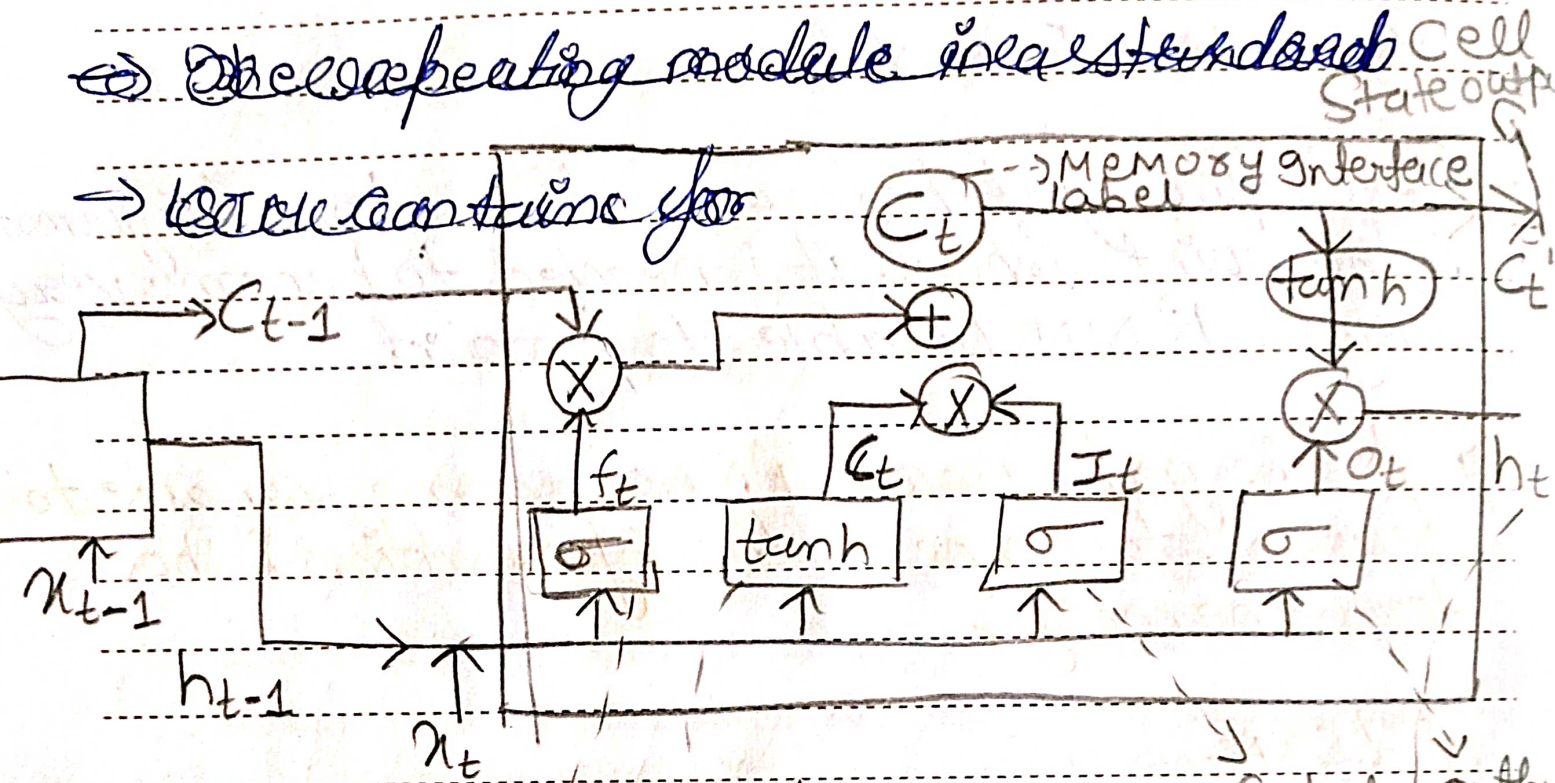
RNN

The repeating module in a standard RNN contains a single layer (tanh)

→ LSTM contains four interacting layers in a repeating module & different memory blocks called cells.

~~→ Repeating module in a standard~~

→ LSTM contains four



LSTM Unit

input gate | output gate

↳ forget gate

↳ Input Modulation gate

↳ hidden State output

□ → Neural Network layer

○ → Elementwise operation

Terms used in LSTM

1) Cell State — Memory of LSTM network

→ It can carry relevant information throughout the processing of the sequence

→ As the cell state goes on its learning information gets added / removed to the cell state via gates.

2) Gates
- → Input gate
- → forget gate
- → Output gate

→ The gates are different neural networks that decide which information is allowed on the cell state. ~~Gaegates~~

→ During training, the gates can learn what information is relevant to keep / forget

3) Sigmoid activation function range [0, 1]

→ It is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappear or be forgotten. & If any number by 1 is the same value therefore that values stay the same or is kept.

Element wise / pointwise product

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \odot \begin{bmatrix} 2 & 3 \end{bmatrix} = \begin{bmatrix} 1\times2 & 2\times3 \end{bmatrix} = \begin{bmatrix} 2 & 6 \end{bmatrix}$$

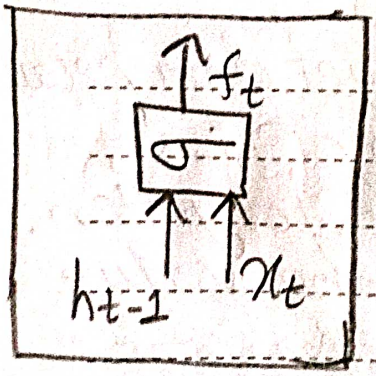
Step-1 Forget Gate Layer: {What to keep & What to from old information}

⇒ In first step, It is decided what layer useful information to throw away from the cell state. & this decision is made by Sigmoid layer (or forget gate layer)

→ Forget gate layer / Sigmoid layer takes h_{t-1} & x_t as inputs & outputs a number between 0 & 1 for each number in the cell state. C_{t-1}

Here 1 represents – completely keep this information

0 represents – Completely ignore / forget throw information from the cell state



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

Modulation

Step-2: Input Modulation Gate layer & Input gate layer {New information Collection}

→ In second step, it is decided what new information to store in the cell state.

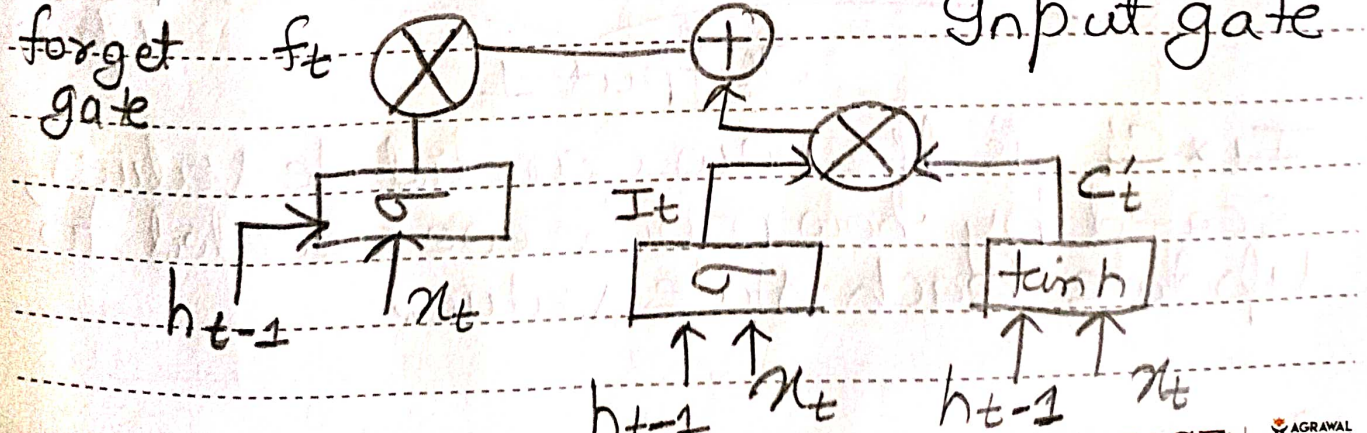
OR Addition of useful information to the cell state is done by input gate.

It has two parts:-

① A sigmoid layer (input gate layer) decides which values to update.

② Next, a tanh layer (input modulator gate layer) creates a vector of new candidate values, C_t' that could be added to the state

→ Now, Combine these two to create an update to the state.



→ Sigmoid function filter the values to be remembered using inputs h_{t-1} & x_t

→ tanh function creates a vector that gives output from -1 to +1 which contains all possible values from h_{t-1} & x_t

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

ii) $C_t' = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

→ Now, updation of old cell state C_{t-1} to the new cell state C_t

i) Multiply C_{t-1} by f_t → forgetting the things we decided to forget earlier
$(f_t \circledast C_{t-1})$

ii) $C_t = \underbrace{(f_t \circledast C_{t-1}) + (I_t \circledast C_t')}_{\text{Output-1}}$ {Memory Cell Update}

$I_t * C_t'$ is the new candidate values, scaled by how much we decided to update each state value.

Step-3: Calculation of output

In this step, we decide what will be the output of this hidden layer. This output will be based on cell state (filtered form).

Sigmoid layer (Output gate layer)

It decides what parts of the cell state go to output [The output gate decides what the next hidden state should be]

→ First cell state pass through the ~~tanh~~ tanh, It pushes the values between -1 & 1 & then multiply it by the output (O_t) of the sigmoid gate (output gate layer), so that we only output the parts we decided to

Input gate - Controls the degree to which the new memory content is added to the memory cell

C_t - Candidates for the state of memory cell (that could be filtered by I/P gate (later)

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

$$h_t = O_t * \tanh(C_t)$$ Hidden layer O/P

output-2

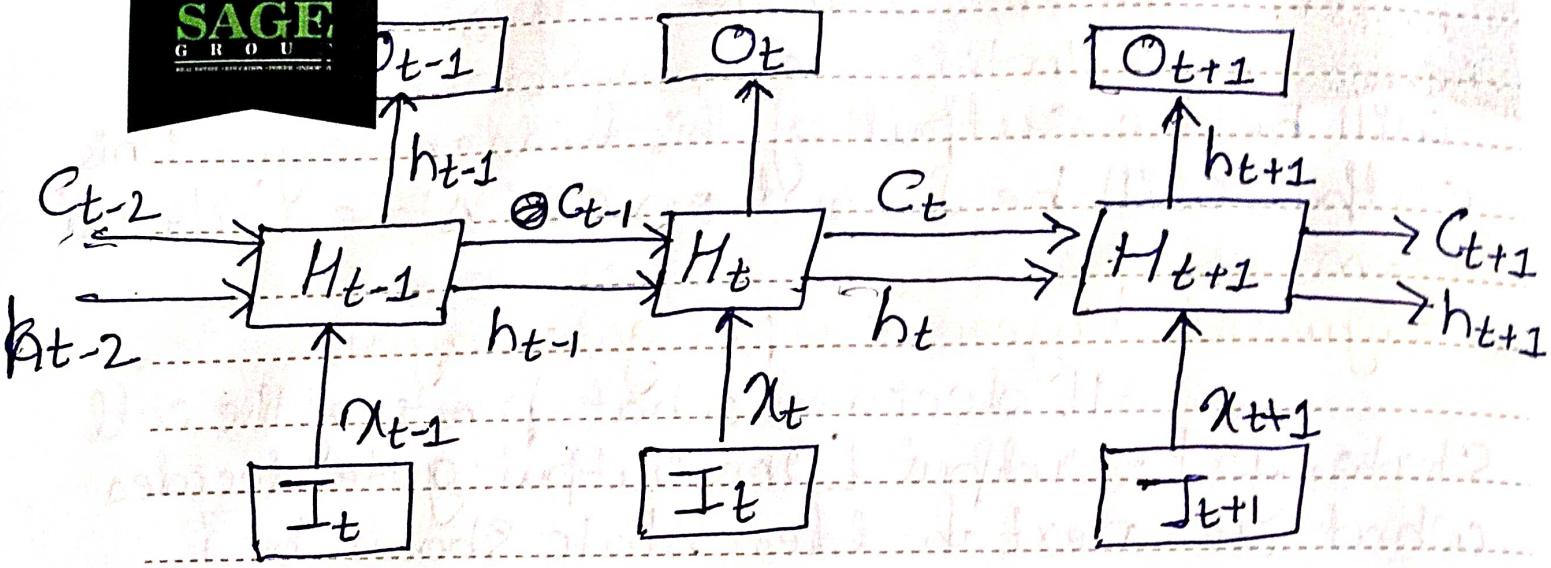
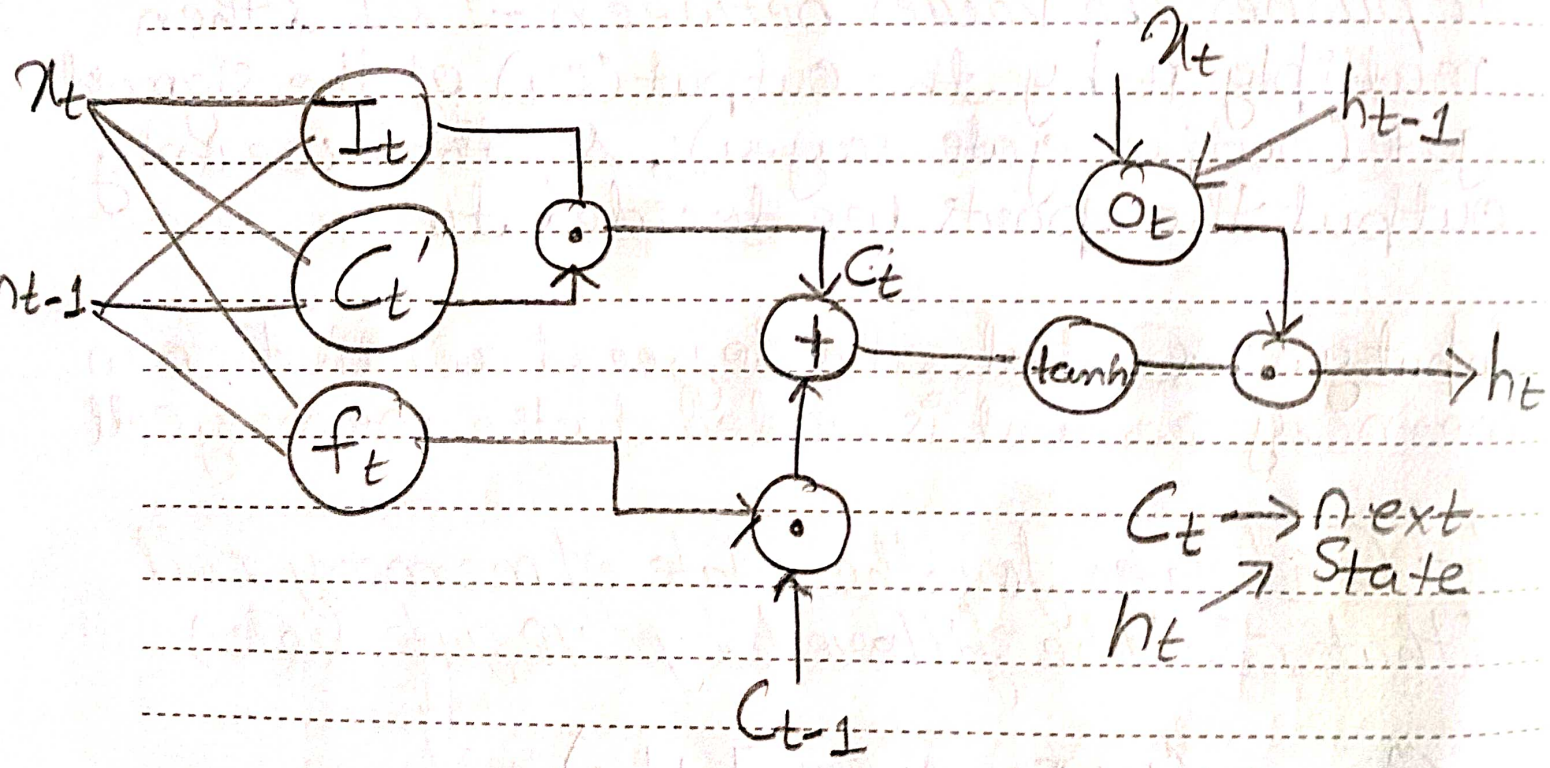
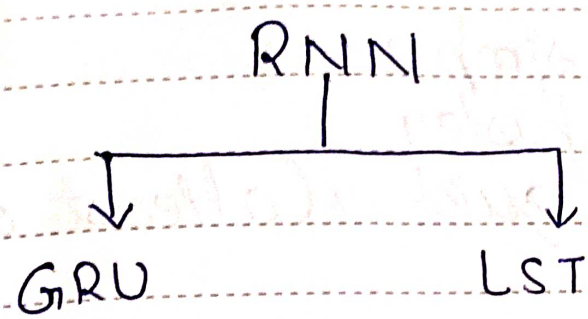


Fig : LSTM



Gated Recurrent Unit [GRU]



⇒ Both solve the vanishing gradient problem of Standard RNN.

→ Similar to LSTM, but there is no memory cell state, it uses hidden state to transfer information.

→ It only has two gates
- A reset gate &
- A update gate

i) Update Gate:-

→ The update gate gets similar to the forget & input gate of LSTM.

→ It decides what information to throw away & what new information to add.

ii) Reset Gate:-

It is used to decide how much past information to forget.

Advantages of GRU

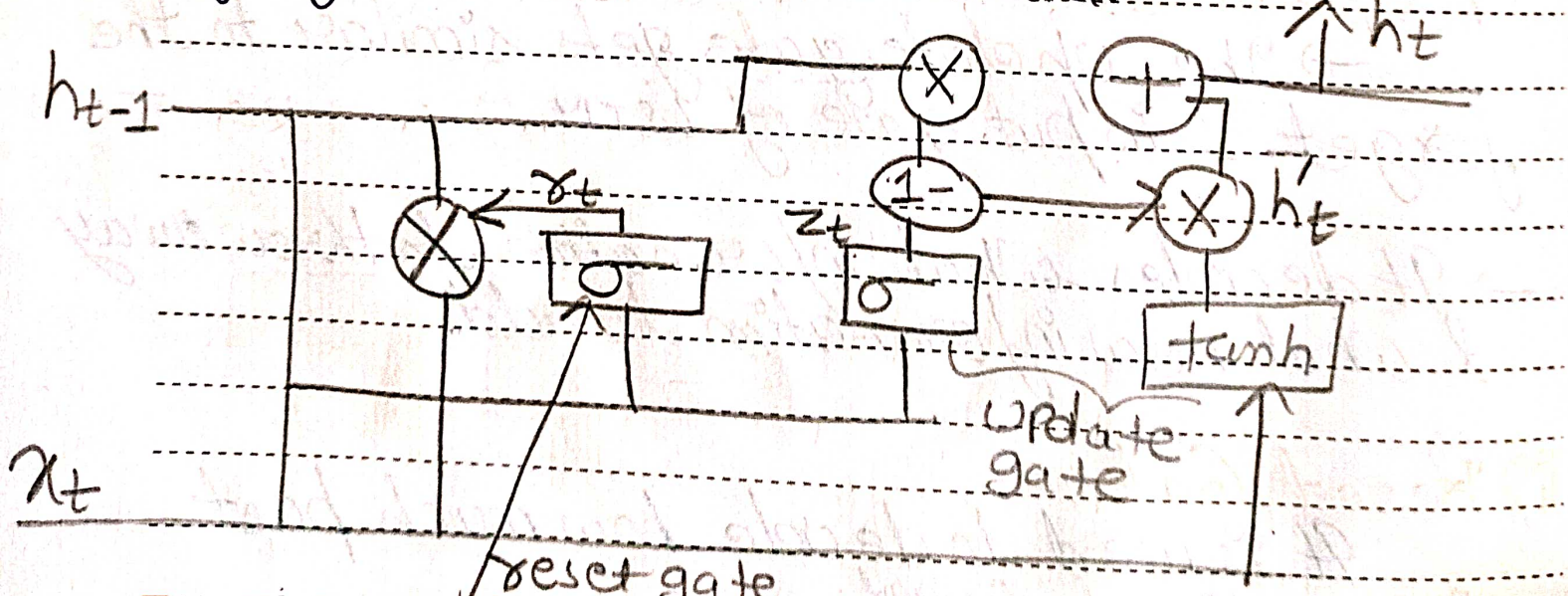
① It is simple
② It is faster
③ It optimizes quicker (atleast on sentiment

OR

→ It is similar to LSTM in using gating functi but differs from LSTM in that it doesn't have a memory cell / cell state & used the hidden state to transfer information.

→ Each GRU unit consist of 2 gates update gate & reset gate. While LSTM consist of 4 gates

⇒ LSTM or GRU, can learn to keep only relevant information to make predictions & forget non-relevant data.



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

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

$$h'_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_t + z_t * h'_t$$

Machine Translation

→ Machine Translation is an automated translation in which computer ~~software~~ softmax translates a text from one natural language to another. Ex → Google translator

Ex

English → [Trans lator] → Hindi

x_1 x_2 x_3 x_4

I have some money

$m = 4$ words

y_1 y_2 y_3 y_4 y_5

$n = 5$ words

$m \neq n$

Sequence to Sequence models are used for machine translation which are RNN based models.

→ Finds the most relevant words in sentence for a target language.

→ Two types of sequence to sequence models

A) Encoder-Decoder Model
B) Attention Model

A) Encoder-Decoder Model

Given,

(English) Input Sequence - $x_1, x_2, \dots x_m$

where x_i - words in input sequence

m = nos. of words in input sequence

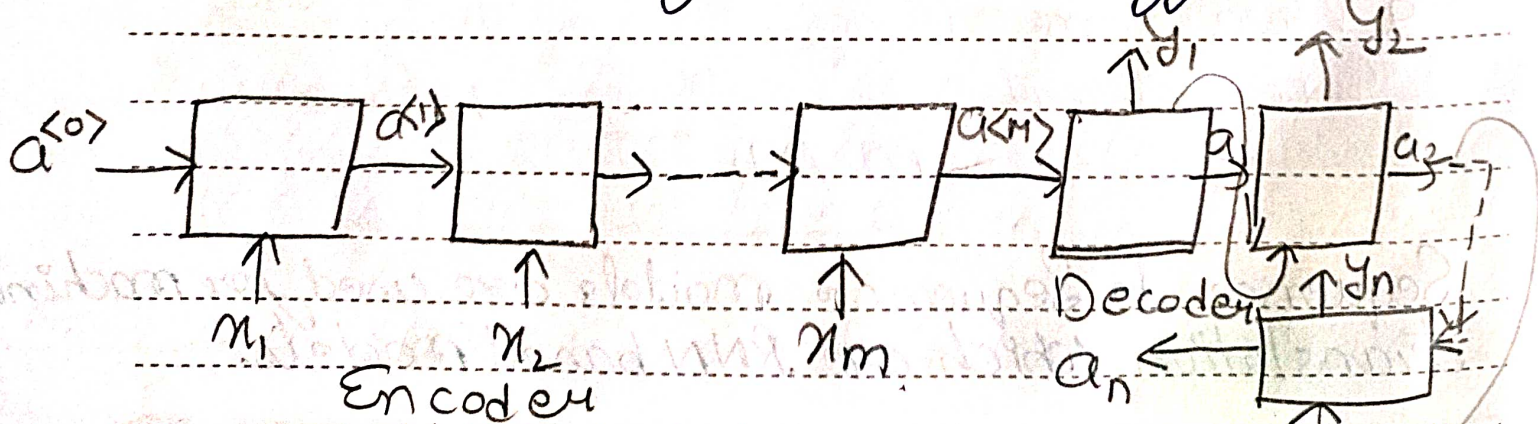
have to find out / predict

(Hindi): Output Sequence - $y_1, y_2 \dots y_n$

where y_i - words in input sequence

n = nos. of words in output sequence

where m & n may be equal / different



→ The encoder network is built as an RNN (GRU or LSTM) where we ~~feed~~ feed the english words, one word at a time

→ After processing the input sequence the output of the RNN is a vector that represents the input sequence

→ After encoder, there is a decoder network which takes as input the encoding output produced by encoder network.

→ Decoder is then trained to output the translation one word at a time until it outputs the whole output sequence

→ Decoder acts mainly as a language model (Probabilistic model) that are able to predict the next word in the sequence given the word that proceed it. Instead of having an input of all zeros, here the input of decoder is output vector of the encoder network

→ So Machine translation Model (like Encoder-Decoder) can be thought of as conditional language model for a system that translate English to Hindi that means instead of

modeling the probablity of any sentence, it is now shaping the likelihood of the output (Hindi translation) conditional on some input sequence (English text) Sentence

→ Due to the probabilistic behaviour of model, there can be more than one translation to input english sentence

Main aim → $\arg\max P(y^{<1>}, y^{<2>}, \ldots y^{<t_y>} | x)$

or

$\arg\max P(y_1, y_2, \ldots y_n | x)$ ↳ Most likely translation

Same input sequence

I/P Sequence
- Output Sequence 1 - P1
- —"— —"— 2 - P2 (max)
- —"— 3 - P3

→ Our Aim is to find the most suitable translation (Models)

→ So, model doesn't sample the outputs at random, but it finds the Hindi sentence that maximizes the conditional probablity

→ To maximize the conditional probablity

or to find a most likely translation for a given input sentence, the most common algorithm is used that is called Beam Search algorithm.

Beam Search

⇒ To explain Beam Search, let we are using a sequence to sequence model that uses an encoder & decoder framework with LSTM or GRU as the basic blocks.

→ Here encoder maps a source sequence encodes the source information (Let here english sentence) & the decoder takes the encoded data from encoder as input to produce an output sequence (Let here Hindi Sentence)

→ We always want the best & most likely words to be selected for the translation that matches the meaning of the english sentence.

→ There could be multiple likely translations of the source sentence (English) in the target language (Hindi)

→ So our goal is to pick the best & most likely translated word, so we choose the target word with the maximum probablity based on the Source Sentence.

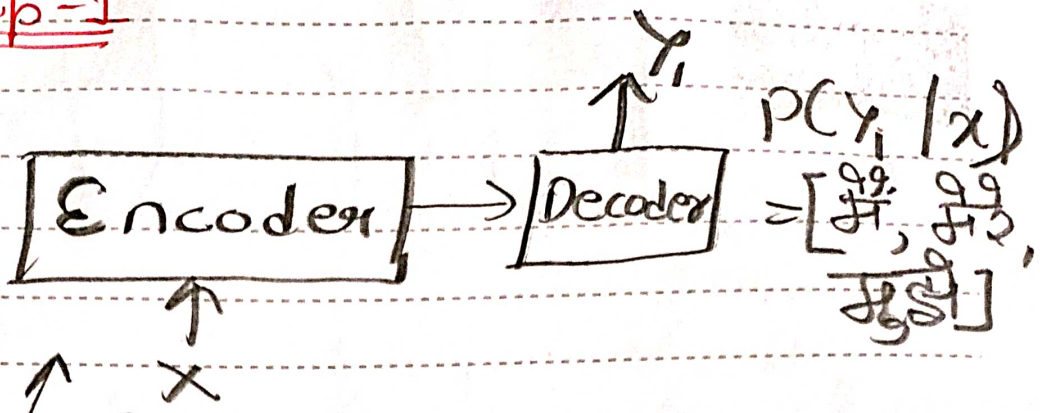
→ The Beam Search algorithm selects multiple alternatives for an input sequence at each timestamp based on Conditional probablity.

→ The number of multiple alternatives depends on a parameter called Beam width B.

→ So at each timestep, the beam search selects B number of best alternatives with the highest probablity as the most likely possible choices for the timestep.

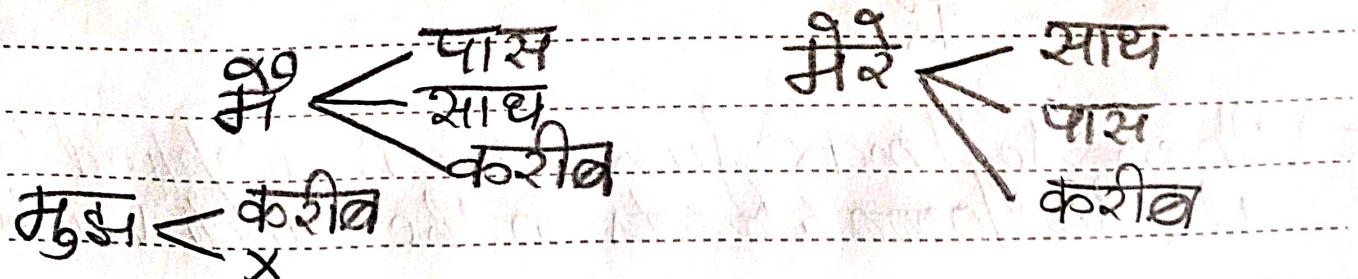
Ex Let Beam width = 3
Vocabulary of Hindi words = 1000 words

Step-1

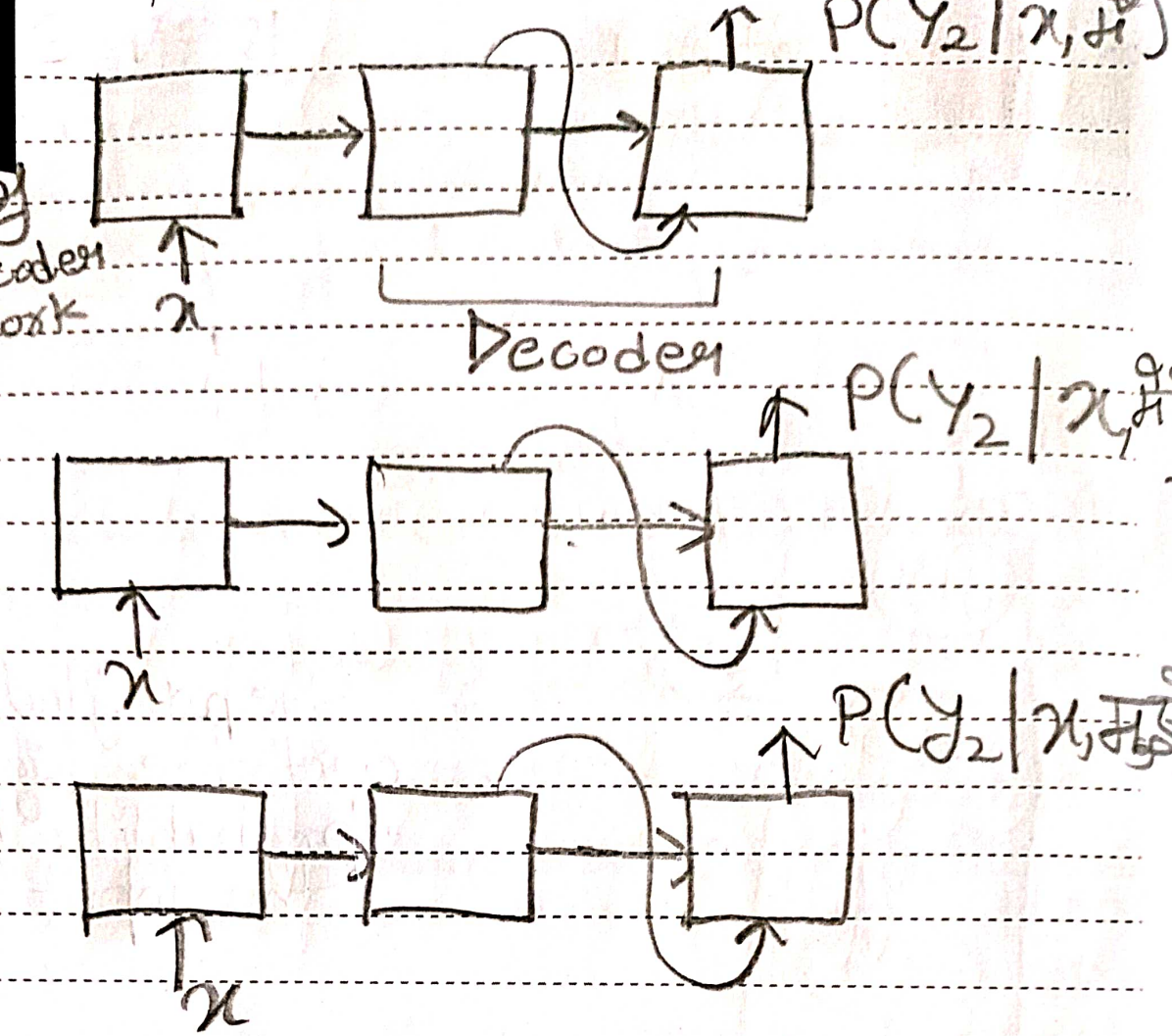


Step 1 finds 3 words with the highest probablity based on the input sequence

Step-2 This step finds top 3 pair of words for the first & second word of the translated sequence (Hindi).



As B=3



$P(y_1 | x) = [\text{मैं}, \text{मैं}]$

$P(y_1, y_2 | x) = [\text{मैं मैं}, \text{मैं मैं}, \text{मैं मैं}]$

→ The first 3 selected words (मैं, मैं, मैं) from step-1 is given as input to second step

→ Then softmax function is applied to all the 10000 words in the Hindi vocabulary

to find the 3 best alternatives for the second word, that are most likely to form a pair with first word using Conditional probablity

→ First we will take word मैं, and apply the softmax function to all 10000 words in the vocabulary which calculate the probablity w.r.t. मैं (10000)

→ Similarly probability for other two words, मैं और मुझे is calculated using softmax function (10000+10000 = 20000)

→ Run 30000 different combinations to choose the top 3 pairs of the first & the second words.

→ Let the top 3 first & second word pair Combination are

मैं पास, मैंने पास, मैंने साथ

→ Let we did not find any word pair with मुझे as the first word & second word having a Conditional probability some dropped it

→ Same process is repeated to find 3 best pair for the first, second & third word based on the input sequence & the chosen first & second word.

→ $P(Y_1|x) = [$ मैं, मेरे, मुझे $]$

$P(Y_1, Y_2|x) = \{$ मैं पास, मेरे पास, मेरे मुझे $]$

$P(Y_1, Y_2, Y_3|x) = [$ मैं पास कुछ, मेरे पास कुछ $]$

This process continues & finally we get two output sentences with highest conditional probability & different length

Input sentence	O/P
I have some money	मैं पास कुछ पैसे हैं (0.2)
	मेरे पास कुछ पैसे हैं (0.8)

→ We finally pick the output of the decoder as the sentence with the highest probability.

I have some money — मेरे पास कुछ पैसे हैं

→ So Beam search considers multiple best options based on beam width using Conditional probability

→ If we take higher beam width, it will give a better translation but would used a lot of memory & computational power

B = 3	10000 word vocabulary

For, Calculating Cond^n probability 30000 words at each time Step i.e. Memory ↑ Computation ↑

→ If we take lower beam width will Result in more low quality translation but will be fast & efficient in terms of Memory usages & Computational power