Department of Mechanical Engineering
Gitam School of Technology
GITAM (Deemed to be University)

Category of course Theory Semester II
19EOE746
OPERATIONS RESEARCH

Moodle - 3

①

RNN

- → Recurrent Neural N/W
- → Mainly used in speech recogni^{tn} & NLP
- → follows sequential approach
- → it trained the process and convert a sequential data i/p into a specific sequential data o/p.

$\underbrace{\text{sequential data}}$

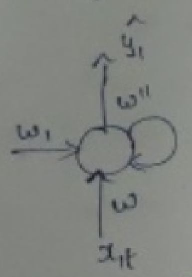
it is data,
such as words, sentences
or time series data.

remembers only previous data

Back propagat^{n} is used for weight adjustment

- → We use Back propagat^{n} algorithm.

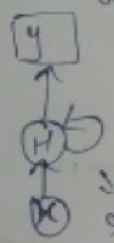
Why RNN



fold representat^{n}

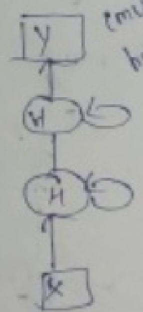
(or)

→ multiple hidden layers untaiyee

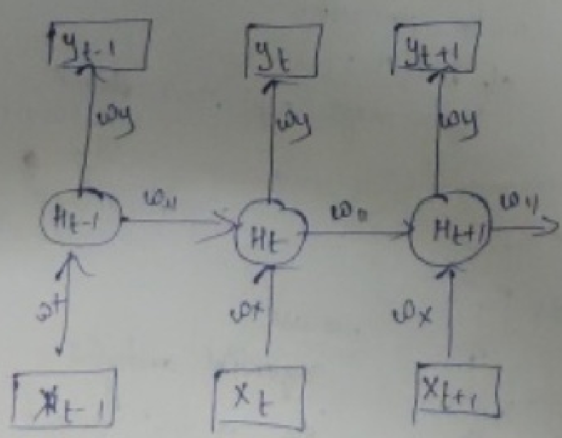


single layer RNN

Multiple RNN



(multiple hidden layers)



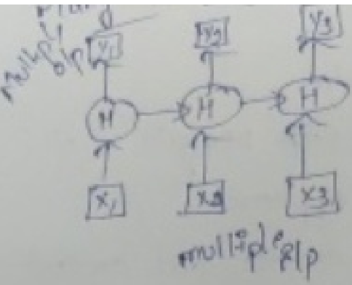
Types of RNN

i) one to one

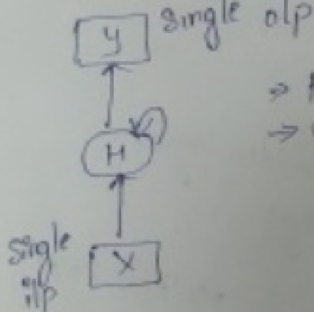
ii) one to many

iii) many to one

iv) Many to many



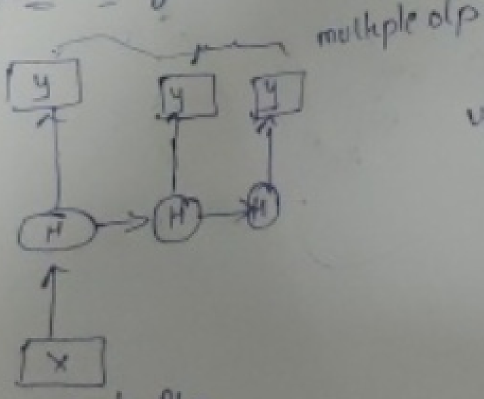
one to one :-



→ this is called vanila neural n/w

→ u~~sed for~~ ~~[illegible]~~ ~~data~~

one to many



usecase:-

image captioning

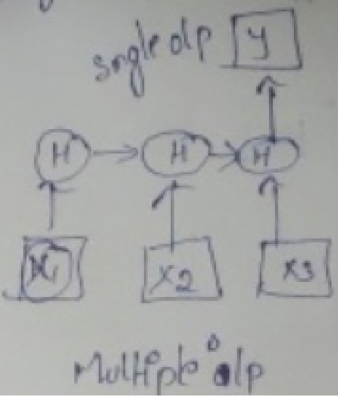
music generation

o/p is seqⁿ.

i/p – image o/p . Text descriptⁿ of i/p

it image

Many to one



usecase

sequential analysis.

i/p is seqⁿ output is fixed vector size

LSTM

(2)

Long short term memory

- It is an improved versn of rnn.
- Well-suited for seqn predictn tasks & excels in capturing long-term dependencies

* A traditional RNN has a single hidden state that passed through time which can ~~take~~ make it difficult for n/w to learn long-term dependencies

-> LSTMs address this problem by introducing a memory cell, which is a container that can hold informatn for an extended period.

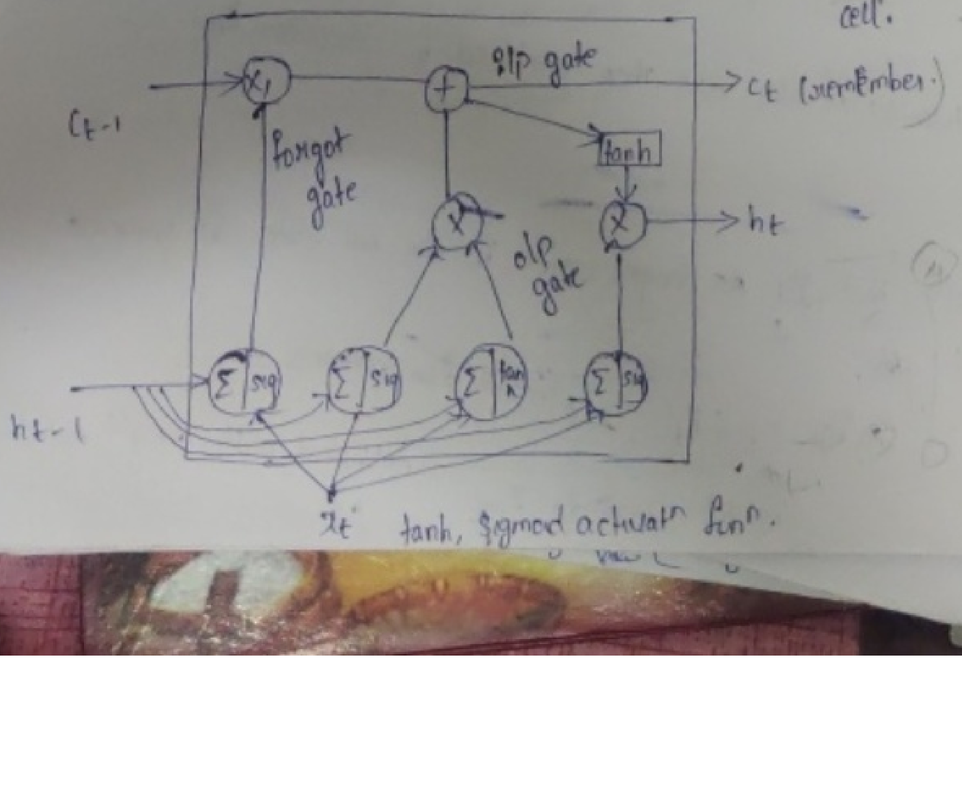
-> LSTM n/w are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translatn, speech recognitn & timeseries forecasting.

-> Memory cell is controlled by 3 gates :- i/p gate -> ctrls what informatn is added to memory cell

forget gate -> ctrl what informatn is removed from memory cell.

o/p gate -> ctrls what informatn is o/p from memory cell.

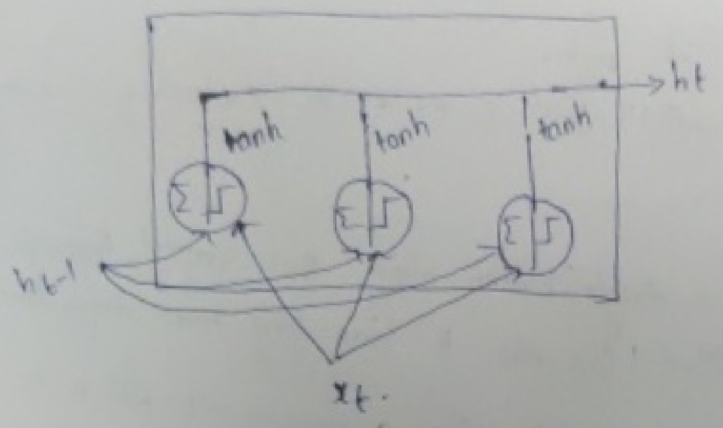
LSTM.



tanh, sigmoid activatn fn.

Types of RNN

Rnn form



Bidirectional LSTM

→ it is a RNN that is able to process sequential data in both forward & backward directⁿ.

→ allows Bidirectional LSTM to learn longer range dependencies in sequential data than traditional LSTMs which can only process sequential data in one directⁿ.

→ Bi LSTM made up of two LSTM n/ws.

i) i/p seqⁿ - in the forward directⁿ

ii) i/p seqⁿ - in the backward directⁿ

o/p of two LSTM n/w are then combined to produce final o/p.

forget Gates:-

informatⁿ that is no longer useful in the cell state is removed with forget gate.

2 i/p x_t & h_{t-1} ⇒ o/p cell from previous

i/p at particular time → multiplied with weight matrices followed by additⁿ of bias

Department of Mechanical Engineering
Gitam School of Technology

Semester II

Date 10.04.2024
90 Min

4

GRU

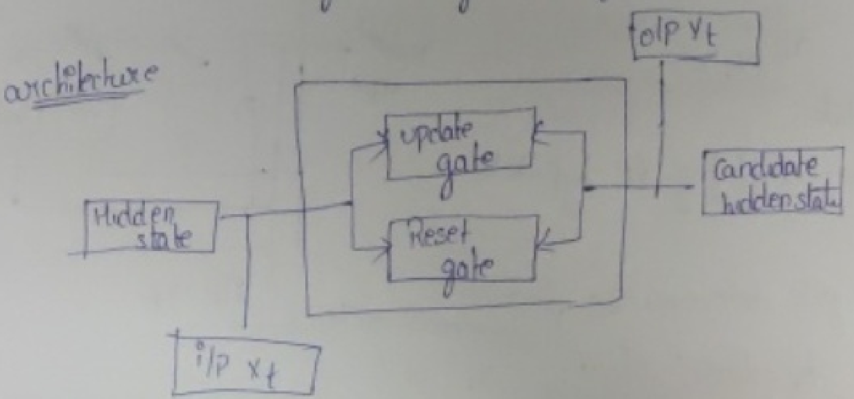
d Recurrent unit.
is a specialized variant of RNN
→ applications :- NLP
Speech recognitⁿ
time series predictⁿ

→ The GRU presents itself as an innovative solⁿ to the vanishing gradient problem in traditional RNN.

→ The architecture of GRU is designed with two specific gates
i) update gate → it recognizes long term connectⁿ
ii) Reset gate → identifies short term relatⁿships

The various components of architecture are :-
i) update gate (Z) → determines the degree of past informatⁿ forward to future
ii) Reset gate (R) → decides the amount of past informatⁿ to discard
iii) Candidate hidden state (H') → creates new representatⁿs considering both i/p & past hidden state.
iv) final hidden state (H) → A blend of new & old memories governed by update gate.

architecture



update gate:-

→ 1st step of GRU

→ It employs the current i/p & previous hidden state to decide how previous hidden state should be update

→ sigmoid funⁿ is used.

$$\underbrace{z_t}_{\text{update gate vector}} = \sigma(\underbrace{W_z}_{\text{weight matrix}} \cdot [\underbrace{h_{t-1}}_{\text{hidden state}}, \underbrace{x_t}_{\text{current i/p}}] + \underbrace{b_z}_{\text{bias}})$$

Reset gate:-

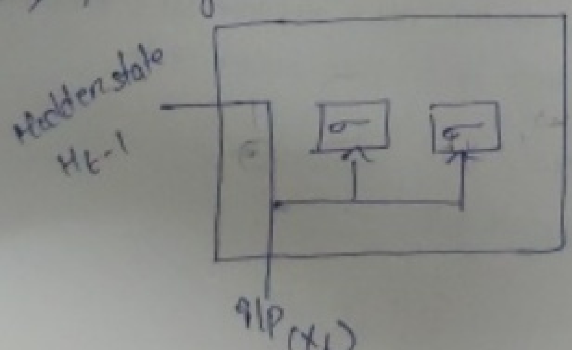
sigmoid funⁿ is used

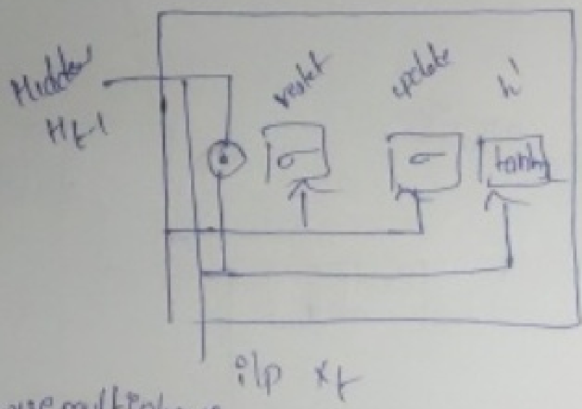
similar to update gat

it identifies the volume of past informatⁿ to be discarded.

$$\underbrace{r_t}_{\text{reset gate vector}} = \sigma(\underbrace{W_r}_{\text{weight matrix}} \cdot [h_{t-1}, x_t] + \underbrace{b_r}_{\text{bias}})$$

Reset & update gate follows





Candidate hidden state

it is computed employing the hyperbolic tanget funⁿ (tanh).

value of reset gate determines the previous hidden stat

$$h'_t = \tanh(W \cdot [\underbrace{r_t * h_{t-1}}_{\text{elementwise multiplicatⁿ}}, x_t] + b]$$

to decide how

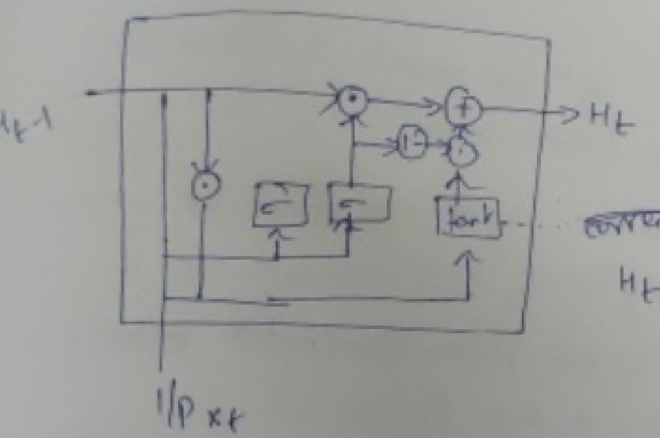
...den state:-

...known as current hidden state

...sequently defined through linear interpolatⁿ

it involves the previous hidden state & prospective hidden state.

$$h_t = (1 - Z_t) * h_{t-1} + Z_t * h_t'$$



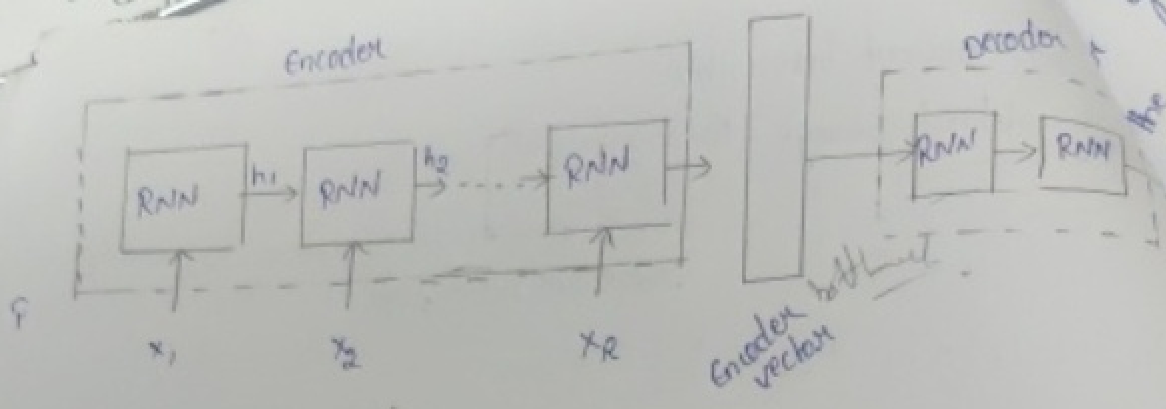
Encoder-Decoder architecture:-

→ The encoder and decoder architecture is a powerful framework used for various seqⁿ to seqⁿ tasks in d.L

→ useful for tasks like m/c translatⁿ, where i/p & o/p seqⁿ have different lengths

→ architecture consists of two components:- i) encoder

ii) Decoder.



Encoder:-

-> The encoder processes the i/p seq^n and converts it into a single dimensional vector (also called as hidden vector)

-> Multiple RNN cells can be stacked together to form the encoder.

-> RNN reads each i/p sequencially.

-> for everytime step (each i/p) t, the hidden state (hidden vector) h is updated according to the i/p at that timestep x[i].

-> After all the i/ps are read by encoder model, the final hidden state of model represents the contxt/sumary of whole i/p seq^n

Encoder vector:-

-> This is the final hidden state produced from the encoder part of model.

-> The vector aims to encapsulate the information for all i/p elements in order to help the decoder make accurate predict^n

-> it acts as the initial hidden state of the decoder part of model

Decoder:- (take encoder vector as i/p)

-> The decoder takes the hidden vector produced by the encoder & generates the o/p seq^n (translate the sentence in another language) generates o/p seq^n by predicting the next o/p y_t (6)

the hidden state h_t

for the decoder is the final vector obtained at the end of

coder model.

Each layer will have three i/ps hidden vector from previous layer h_{t-1} and previous layer o/p y_{t-1}, original hidden vector h.

o/p layer

use softmax activat^n fun^n

used to produce the probability distribut^n from a vector of values with the target class of high probability

o/p y_t at time step t is computed using

$$y_t = softmax(W^S h_t)$$

Applicat^n

text summarizat^n

speech recognition

Time series Applicat^n

Google's m/c translat^n