Recurrent Neural Networks:-

→ RNN is type of Sequential model to work on Sequential data.
↓
where sequence matters like Text.

→ Text, time series data, DNA are some eg's of sequential data.

→ RNN's are mostly used in NLP.

Why we don't use RANN for text:-

→ Biggest problem is textual data has different size.

→ We can use zero Padding for different size data but it is not efficie
↳ we'll be doing lots of computation.

→ If use ANN in text classification,

- need to do lot of unnecessary computation.
- there will be problem in prediction.
- we are totally disregarding sequence information.

RNN applications:-

(1) Sentiment Analysis
(2) sentence completion (smart reply) in gmails.
(3) Image caption generator.
(4) Google Translate.
(5) FAQ demo using BERT.

Forward propagation in RNN:-

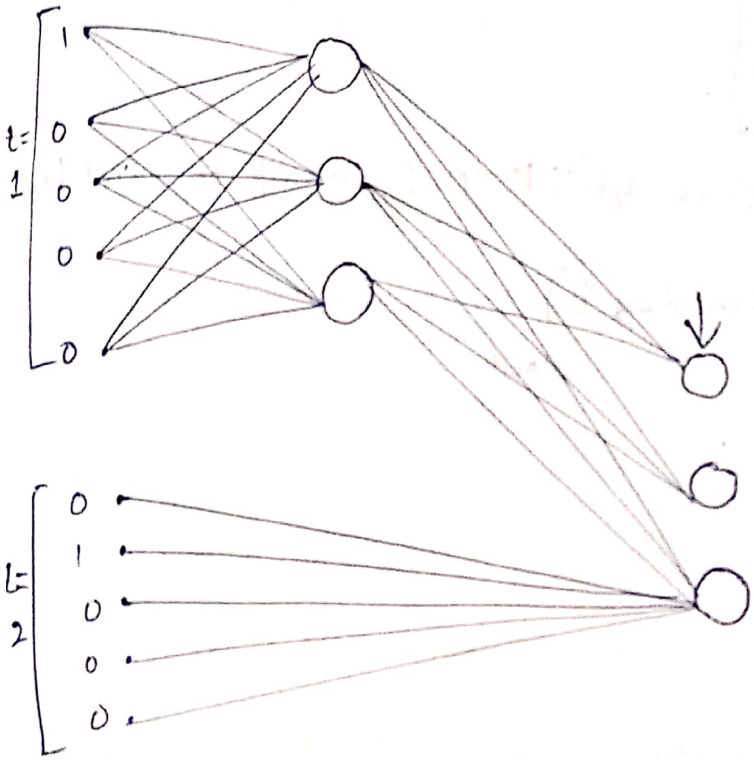
Review	Sentiment
Movie was good	1
Movie was bad	0
Movie was not good	0

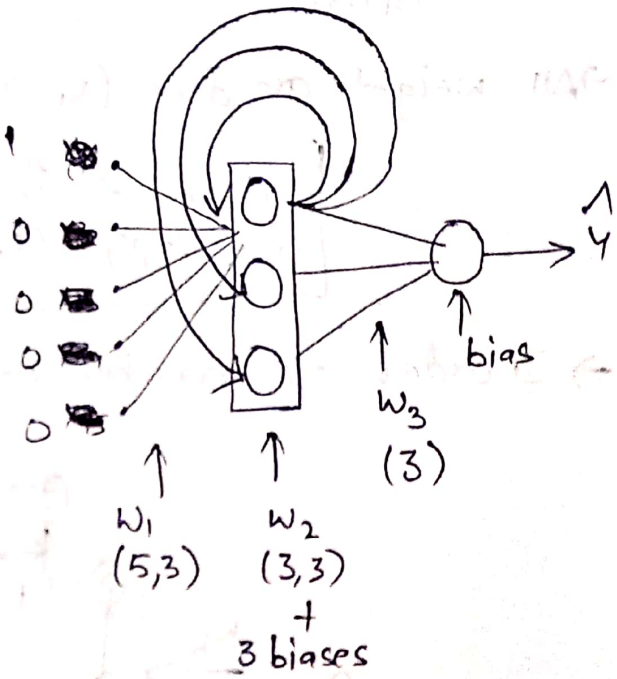
Movie → [10000]
was → [01000]
good → [00100]
bad → [00010]
not → [00001]
} Features

Review 1 [[10000][01000][00100]]

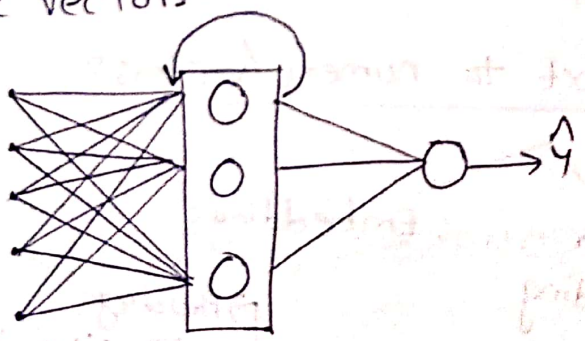
↳ shape is (3, 5)
→ # of features
→ No. of time steps

x_1	Movie	was	good	1
x_2	Movie (x_{21})	was (x_{22})	bad (x_{23})	0
x_3	Movie (x_{31})	was (x_{32})	not (x_{33}) good (x_{34})	0





→ x_{11} x_{12} x_{13} ... are vectors.



$x_{11} \rightarrow (1,5)$
$x_{12} \rightarrow (1,5)$
$w_i \rightarrow (5,3)$
$w_h \rightarrow (3,3)$
$w_0 \rightarrow (3,1)$

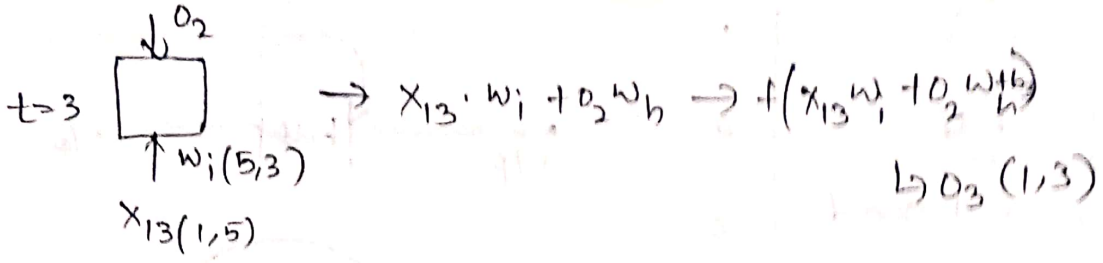
Tanh/ReLu

$t=1 \rightarrow x_{11} w_i \rightarrow f(x_{11} w_i + b_1)$

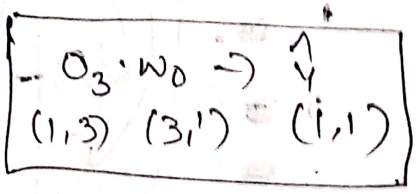
w_i (5,3), x_{11} (1,5), (1,3), O_1 (1,3)

$t=2 \rightarrow x_{12} w_i + O_1 w_h \rightarrow f(x_{12} w_i + O_1 w_h + b_2)$

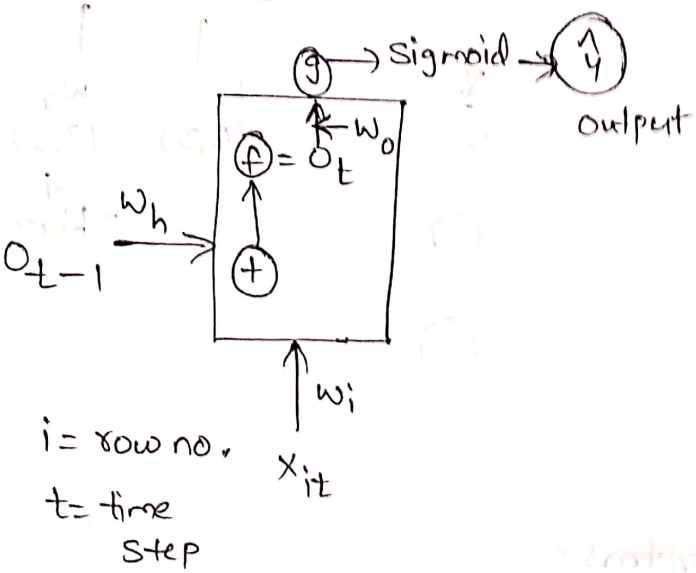
O_1 (1,3), w_i (5,3), x_{12} (1,5), (3,3), (1,3), (1,3), O_2 (1,3)



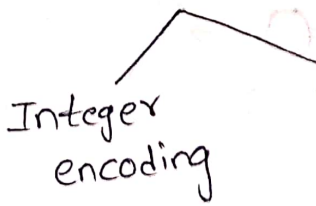
→ All weights are done (X_{11} X_{12} X_{13})



→ Inorder to maintain Consistency, We'll keep O_0 as o/p for first layer



How to convert text to numeric/vectors?



Padding

1 2 Hi there	1	→ [1 2 0]
How are you 3 4 5	0	→ [3 4 5]

} Likewise, we'll Convert text to vectors using integer encoding.

→ In NLP, word embedding is a term used for representation of words for text analysis, typically in the form of real valued vector that encodes meaning of word.

```
docs = ['India', 'Hi subbu', 'How is Rohit']

from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(oov_token='<nothing>')
tokenizer.fit_on_texts(docs)
tokenizer.word_index      # Index for all unique words
tokenizer.word_counts     # Freq. of each word.
tokenizer.document_count  # No. of sentences/rows
sequences = tokenizer.texts_to_sequences(docs)
>>sequences
 [3 5]
 [1 4 7]
from keras.utils import pad_sequences
sequences = pad_sequences(sequences, padding='post')
>>sequences
 [3 5 0]
 [1 4 7]
from keras.layers import Dense, SimpleRNN, Embedding, Flatten

model = Sequential()
model.add(simpleRNN(32, input_shape=(50,1), return_sequences=False))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(loss='bce', optimizer='adam')
model.fit(x_train, y_train, epochs=5)
```

$X_1 = [1\ 4\ 7\ 6\ 5\ 3\ 9]$ ⇒ Dense vector

$X_2 = [0\ 8\ 1\ 0\ 0\ 0\ 0]$ ⇒ Sparse vector.

→ word embedding captures semantic meaning.

↳ Means in which context word used.

word2vec, glove are techniques of word embedding in NLP.

→ In deep learning, we'll do embedding learn during training process

```
model = sequential()
model.add(Embedding(17, output_dim = 2, input_length = 5))
```

17 → Total (vocsize) unique words

2 → dimension of our dense vector, (each word represented in 2 vars)

```
model.predict(sequences)
```

Types of RNN:-

1) Many to one
2) one to Many
3) Many to Many
4) one to one

Many to one:-

I/p → Sequence (Sentences, chars...)

O/p → Non sequential (num, vectors)

↳ Eg is sentiment Analysis (Rating prediction)

O/p is num (1-5)

→ Multiple I/p's & one O/p.

one to Many:-

I/p → Non-sequential (Image etc)

o/p → sequential (words, Timeseries....)

↳ Eg is Image Caption

Many to Many :-

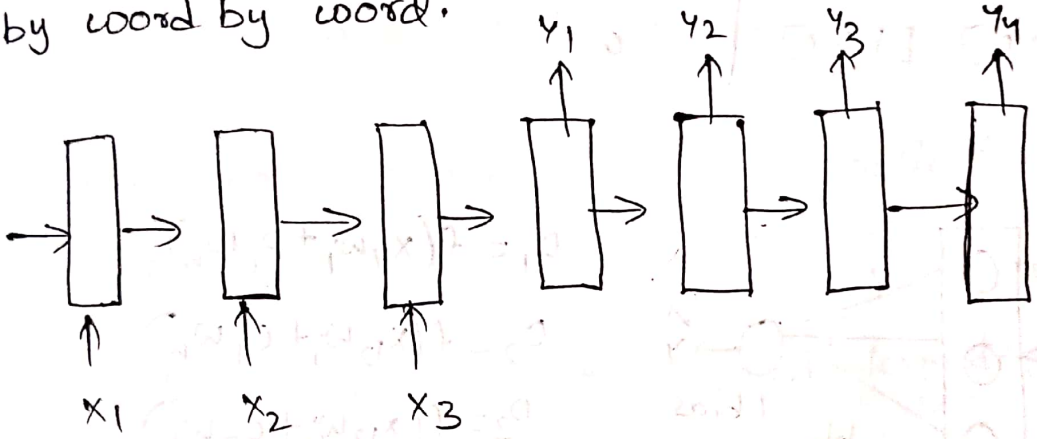
I/P → sequential
O/P → sequential

> seq to seq.

- Same Length.
- variable length.

↳ Eg is google translation.

→ Google translation is one of the apps which willnot translate by word by word.



One to one :-

I/P → Non sequential
O/P → Non sequential.

↳ Eg is Image classification.

→ Technically, one to one won't come under RNN.

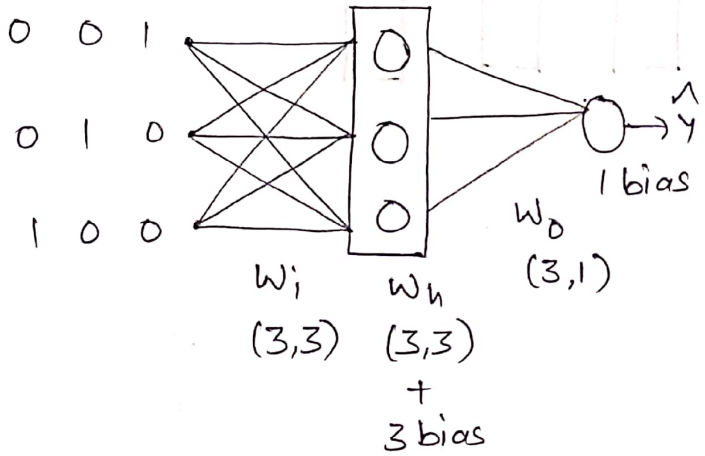
Back Propogation:-

Text	
Cat Mat Rat	1
Rat Rat Mat	1
Mat Mat cat	0

unique words = 3

Cat = [1 0 0] Mat = [0 1 0] Rat = [0 0 1]

⇓ x			y
x_1	[1 0 0]	[0 1 0] [0 0 1]	1
x_2	[0 0 1]	[0 0 1] [0 1 0]	1
x_3	[0 1 0]	[0 1 0] [1 0 0]	0



$$O_1 = f(x_{11}w_i + o_0 w_h)$$

$$O_2 = f(x_{12}w_i + O_1 w_h)$$

$$O_3 = f(x_{13}w_i + O_2 w_h)$$

$$\hat{y} = \text{Sigmoid}(O_3 w_0)$$

Using y and $\hat{y}$,

$$\text{Loss} = -y_i \log \hat{y}_i - (1-y_i)\log(1-\hat{y}_i)$$

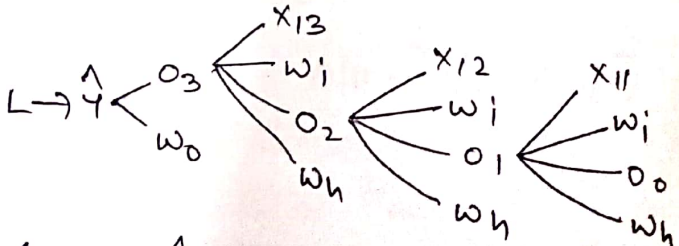
$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

$$w_h = w_h - \eta \frac{\partial L}{\partial w_h}$$

$$w_0 = w_0 - \eta \frac{\partial L}{\partial w_0}$$

} Such that loss should be minimum.

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_0}$$



$$\frac{\partial L}{\partial w_i} = \left(\frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial O_3} * \frac{\partial O_3}{\partial w_i}\right) + \left(\frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial O_3} * \frac{\partial O_3}{\partial O_2} * \frac{\partial O_2}{\partial w_i}\right)$$

↓ Short term dependency.

$$+ \left(\frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial O_3} * \frac{\partial O_3}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial w_i}\right) \rightarrow \text{long term dependency}$$

$\frac{\partial L}{\partial w_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial o_j} * \frac{\partial o_j}{\partial w_i}$; n = No. of time steps.

$\frac{\partial L}{\partial w_h} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial o_j} * \frac{\partial o_j}{\partial w_h}$; n = No. of time steps.

Problems with RNN:-

→ Text data / Time series.

→ RNN mainly works on sequential data. It has 2 main problems

(1) Problem of long term dependency.

(2) Unstable gradients. (or) stagnated training.

i) let's say I'm creating one keypad application using RNN where we predict next word.

→ Marathi is spoken in MH. - (Short term dependency)

→ MH is a beautiful place. I went there last year, but I could not enjoyed properly because I don't understand Marathi. → (long term dependency) **Here RNN fails.**

↓

Due to vanishing gradient

$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \frac{\partial o_{100}}{\partial o_{99}} \cdots \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_i} \Rightarrow \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \prod_{t=2}^{100} \left(\frac{\partial o_t}{\partial o_{t-1}}\right) \frac{\partial o_1}{\partial w}$

$o_1 = \tanh(x_{11} w_{in} + o_0 w_h)$

$o_t = \tanh(x_{it} w_{in} + o_{t-1} w_h)$

$\frac{\partial o_t}{\partial o_{t-1}} = \tanh'(x_{it} w_{in} + o_{t-1} w_h) w_h$

Lies b/w 0 to 1

$\Rightarrow \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} (0-1) \frac{\partial o_1}{\partial w}$

if we multiply 100 times it is ≈ 0

$\Rightarrow \approx 0$

(long term dependency is almost 0 & o/p depends on short term)

Solution :-

→ use diff. activation function i.e. ReLu/Leaky ReLu.

→ Better weight initialization.

→ Skip RNN and move to LSTM.

2) ⇒ If long term value increases exponentially and if it dominates short term value, there will be unstable training. This is due to exploding gradient prblm.

Why only sigmoid and Tanh in LSTM?

→ Sigmoid specifically, is used as gating func, for all three gates. Since it outputs a value b/w 0 and 1, it can either let no flow or complete flow of info, throughout the gates.

→ On the other hand, to overcome vanishing gradient problem, we need a function whose second derivative can sustain for a long range before going to zero. Tanh is good function with the above property.

Why only Tanh in RNN?

→ In RNN's, weights are shared across time steps. If you have ReLu activation and recurrence matrices, magnitude of state will increase exponentially with time steps during forward pass. In turn this will increase the magnitude of the gradients during backward pass, the two effects can easily feed into each other, leading to numeric overflow.

→ Properly placed Tanh can avoid this; at worst you'll get vanishing gradient, which is better than numeric overflow.

LSTM – The what?

→ Problem with RNN's is "Long-term dependancy".

→ Long term dependancy is because of "vanishing/Exploding" gradient Problem

→ RNN's are not able to handle long sequences.

A small story:-

→ Around 1000 yrs. back, there was a state in India called "Pratapgarh." In this state, there was a king called Vikram. Vikram was very powerful as well as kind. People in this kingdom are very happy & everything is going nice. Suddenly, neighbouring country king "khali" attacked pratapgarh. Vikram was killed by khali. After some years, vikram's son "Vikram Jr." became king. He is similar to his father, very powerful and kind. Vikram Jr. wants to take revenge and decided to kill khali. After 20 yrs, Vikram Jr. went for a war against khali and got killed. People were very sad at that moment. Vikram Jr's son Vikram Sub Jr. was not as powerful as vikram Jr, but is very smarter than his father & grand father. After he grew up, he also wants to take revenge. After 15 yrs. he attacked khali and he killed him smartly and took revenge.

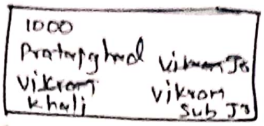
Now, classify whether story is "Good" or "not".

How we'll decide it is good or bad?

Whenever, any sequential data comes, our brain process it word by word basis.

→ our brain maintains 2 types of context. based on the current state.

Short-term context



Long-term context.

→ RNN only maintains one state. It is able to retain the info. about past data through single state. As time passes, STC dominates LTC and can't retain the context.

→ whenever RNN watches any series, it can remember only latest episodes and it doesn't have any mechanism to retain both the contexts.

→ This is the core idea of LSTM, it retains both long term and short term context.

RNN vs LSTM:-

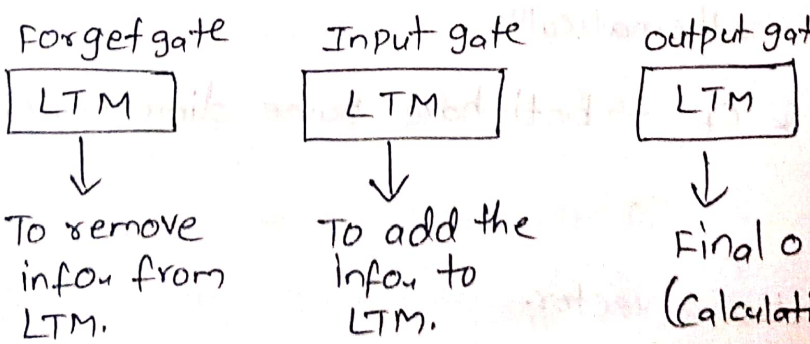
→ RNN has only one state, where as LSTM has 2 states.

→ RNN's architecture is simple, where as LSTM is complex.

LSTM - The How? The Architecture!-

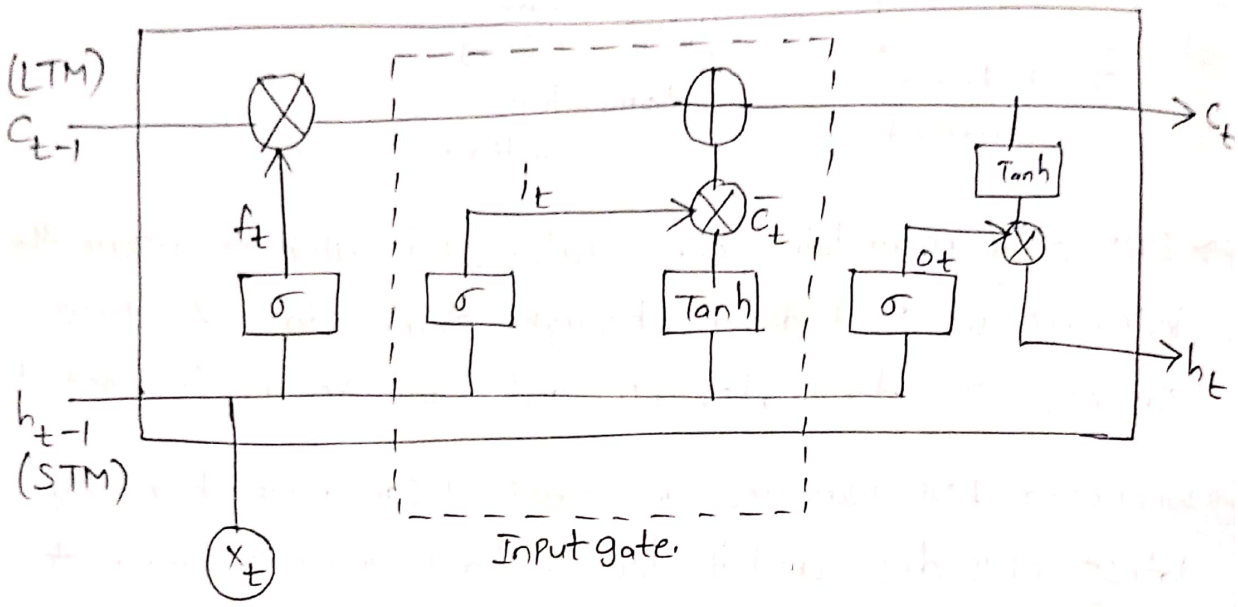
→ LSTM architecture has mainly 3 gates.

(i) Forget gate
(ii) Input gate
(iii) output gate.



LTM → Cell state.

STM → Hidden state.



Simple principle:-

(i) Update cell state.

(ii) Calculate hidden state.

3 I/P's:
- Prev. cell state (C_0)
- Prev. hidden state (h_{t-1})
- I/P for current (X_t) time stamp

2 O/P's:
- Current cell state (C_t)
- Current hidden state (h_t)

What is gate?

→ In LSTM, gate plays a crucial role in controlling the flow of info. through the N/w.

What are C_t & h_t ?

→ C_t and h_t are nothing but LTM & STM, which are vectors represented mathematically.

h_t & C_t → both have same dimensions.

What is X_t ?

→ X_t is I/P, also a vector.

cat	Mat	Rat	0	OHE ⇒	[1 0 0]	[0 1 0]	[0 0 1]	0
cat	Rat	Rat	0		[1 0 0]	[0 0 1]	[0 0 1]	0
Mat	Mat	cat	1		[0 1 0]	[0 1 0]	[1 0 0]	1

At $t=1 \Rightarrow [1\ 0\ 0]$
$t=2 \Rightarrow [0\ 1\ 0]$
$t=3 \Rightarrow [0\ 0\ 1]$

} X_t is just one word, converted into a vector.

⇒ X_t can be of any shape.

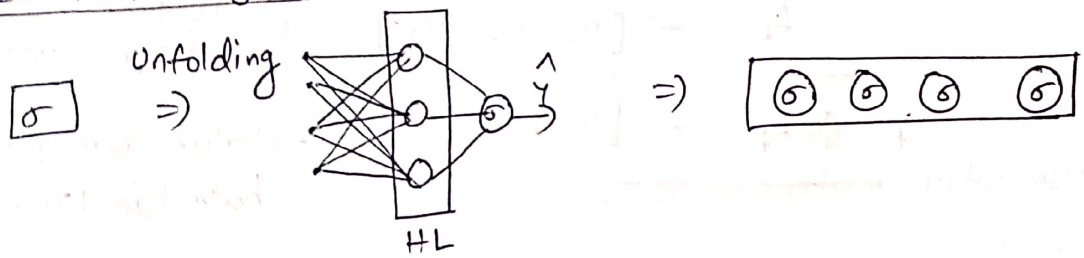
What are $f_t, i_t, \bar{C}_t, o_t$?

→ These vectors are used in gates.

$f_t \rightarrow$ forget gate
$i_t / \bar{C}_t \rightarrow$ I/p gate
$o_t \rightarrow$ o/p gate

} All four vectors should have same dimensions.

Neural N/w layers:-



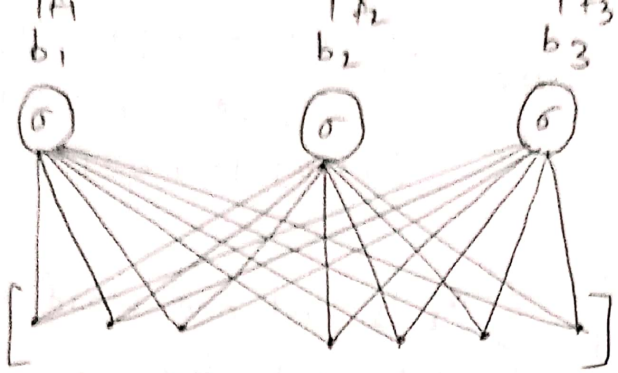
→ No. of units/nodes in hidden layer is hyperparameter. And these nodes should be same in all layers in cell.

(1) Forget gate:-

→ It decides what infor we are going to throw away from the cell state.

* "C_t" and "h_t" dimension should be as same as Neural N/w shape i.e, No. of nodes in HL.

(i) In first stage, we'll calculate 'f_t'.
(ii) In second stage, we'll do $f_t \otimes C_{t-1}$.



→ $[f_1\ f_2\ f_3] \Rightarrow f_t$

→ weights → $7 \times 3 = 21$
bias → 3.

→ 7 dim. input

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

→ Now, we'll do pointwise ⊗ operation with C_{t-1}.

Point-wise operation ↑ $f_t \otimes C_{t-1}$ → "vector," it represents that in C_{t-1}, removed useless info.

Let's say $C_{t-1} \rightarrow [4, 5, 6]$
$f_t \rightarrow [1/2, 1/2, 1/2]$

This is the Purpose of forget gate.

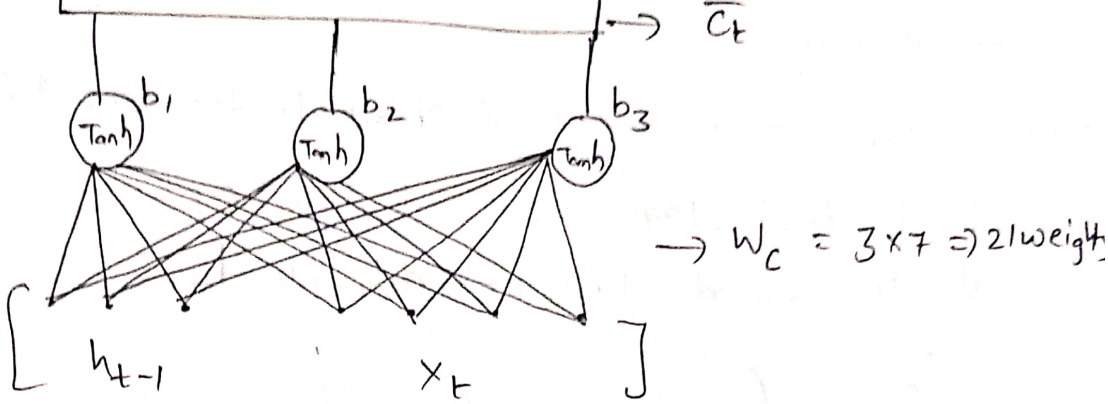
$C_{t-1} \otimes f_t \rightarrow [2, 2.5, 3]$ → Values are getting halved in LTM.

→ Because of 'f_t', you decide how much info. should flow in C_t.
How f_t decides, → based on current I/p & previous hidden.

(2) Input gate :-

→ This gate decides what info. we're going to store in the cell.

(i) In first stage, Calculate Candidate cell state ($\tilde{C}_t$).
(ii) In second stage, Calculate i_t.
(iii) In last Stage, Calculate Current cell state. (C_t).



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

(3×7) (7×1) → (3×1)

(3×1) → (3×1)

→ we'll calculate 'i_t' also similarly, instead Tanh we'll use 'σ'.

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

→ Candidate cell state is "Potential important information".

→ 'i_t' usually filter in $\bar{C}_t$.

→ i_t will decide what info, should go from $\bar{C}_t$ based on curr$_t$ input and prev. hidden state.

$i_t \otimes \bar{C}_t$ → Filtered candidate cell state.

we are adding this info, to the cell state using below operation.

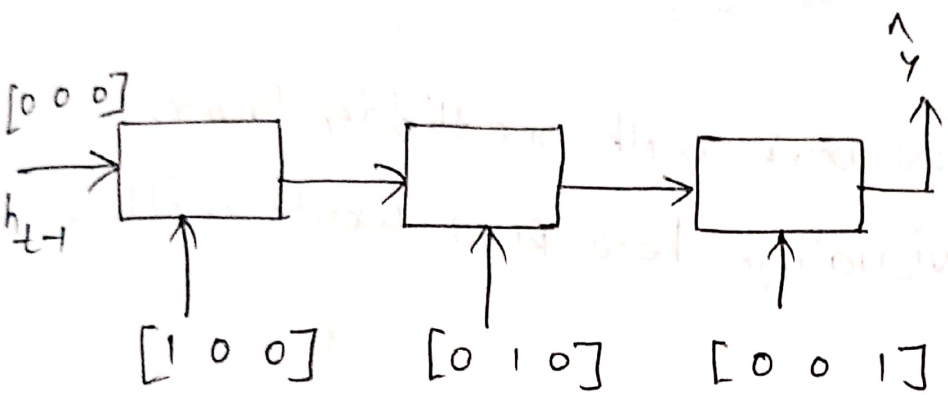
$$[f_t \otimes C_{t-1}] \oplus [i_t \otimes \bar{C}_t] \rightarrow C_t$$

h_{t-1}

Deep RNN's / Stacked RNN's / Stacked LSTM's:-

→ Deep RNN is nothing but, stacking one RNN cell on top of each other and unfolding all of them in time.

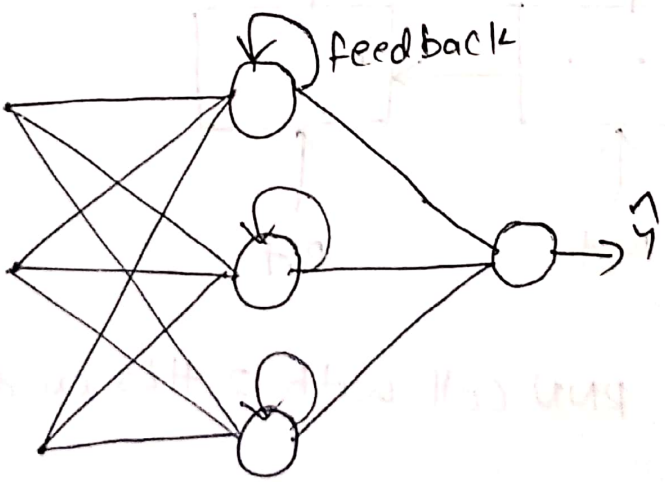
Unfolding Simple RNN with one layer:-



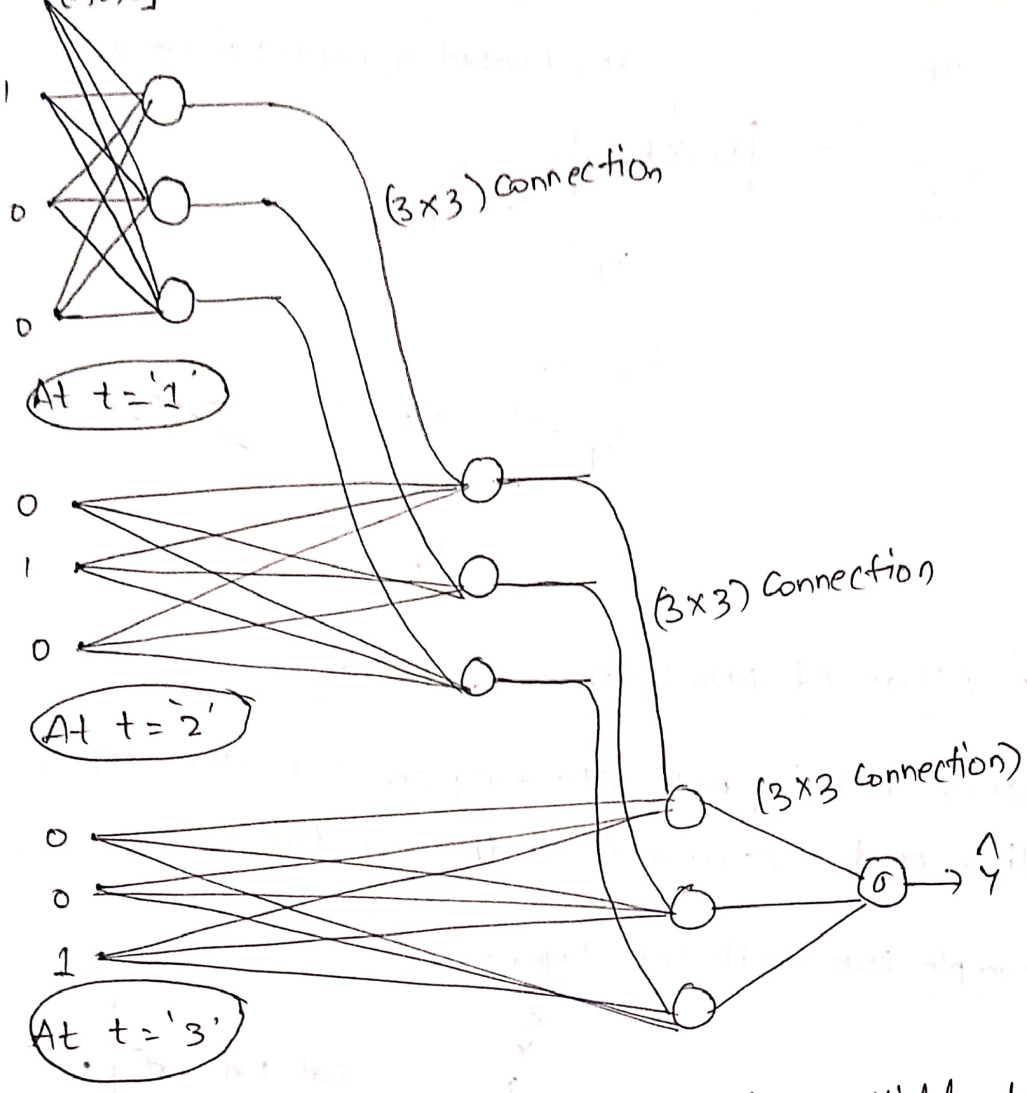
Cat Mat rat	1
rat rat Mat	0
Mat cat cat	1

[[1 0 0] [0 1 0] [0 0 1]]

→ Let's say we have 3 neurons inside RNN cell. We can visualize one RNN cell as below:-

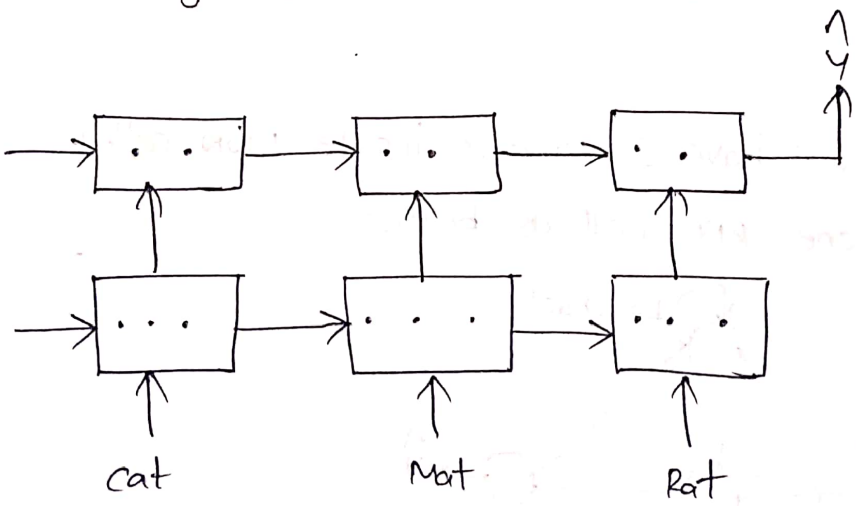


→ Let's unfold the above RNN how it works.



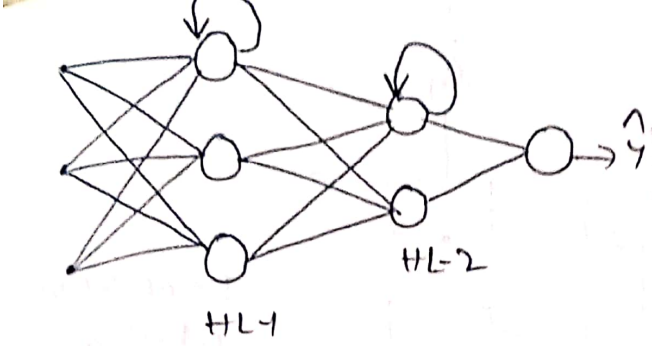
This is how simple RNN works with one hidden layer.

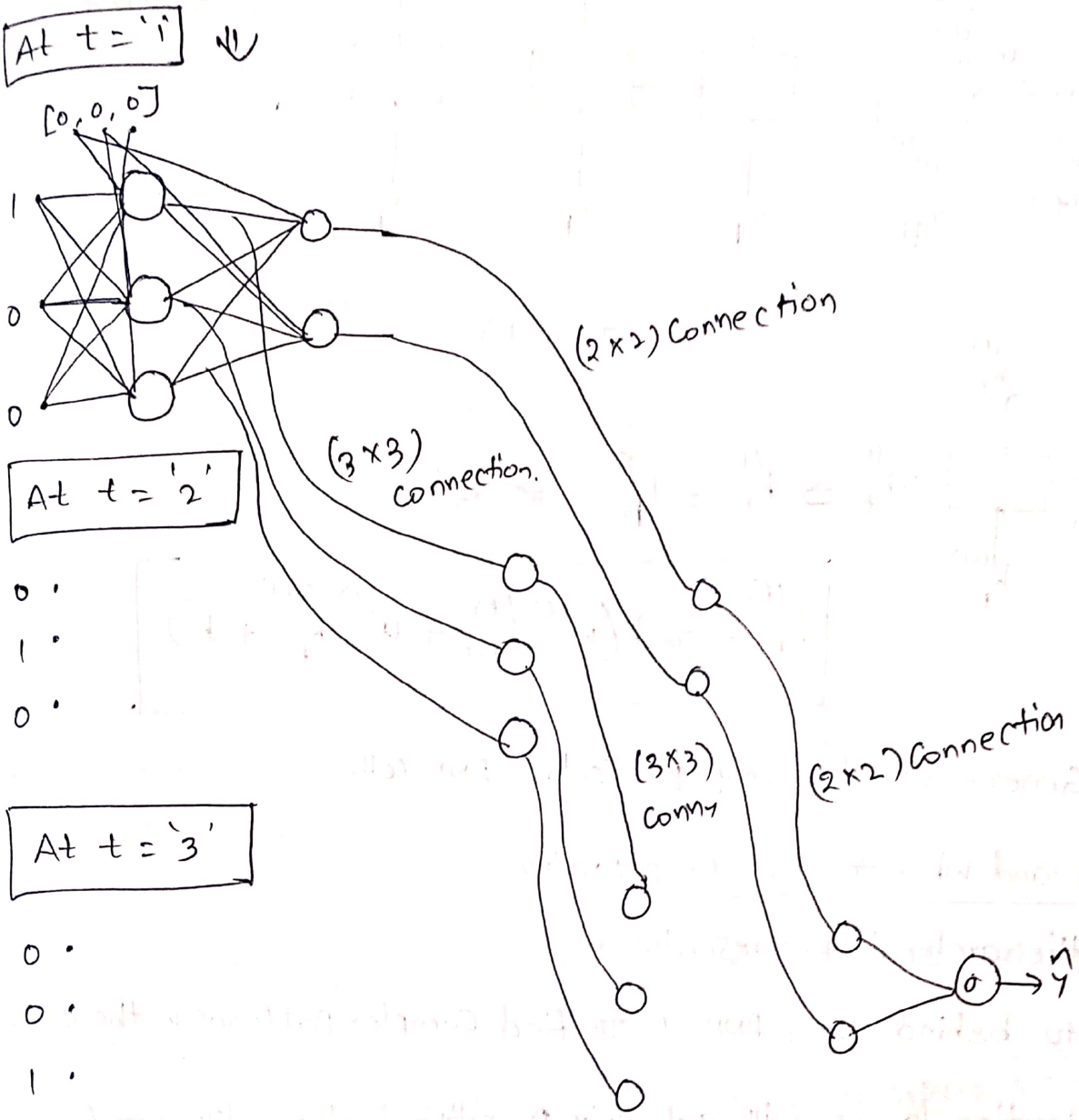
→ Now, let's see and visualize how RNN works with 2 hidden layers.



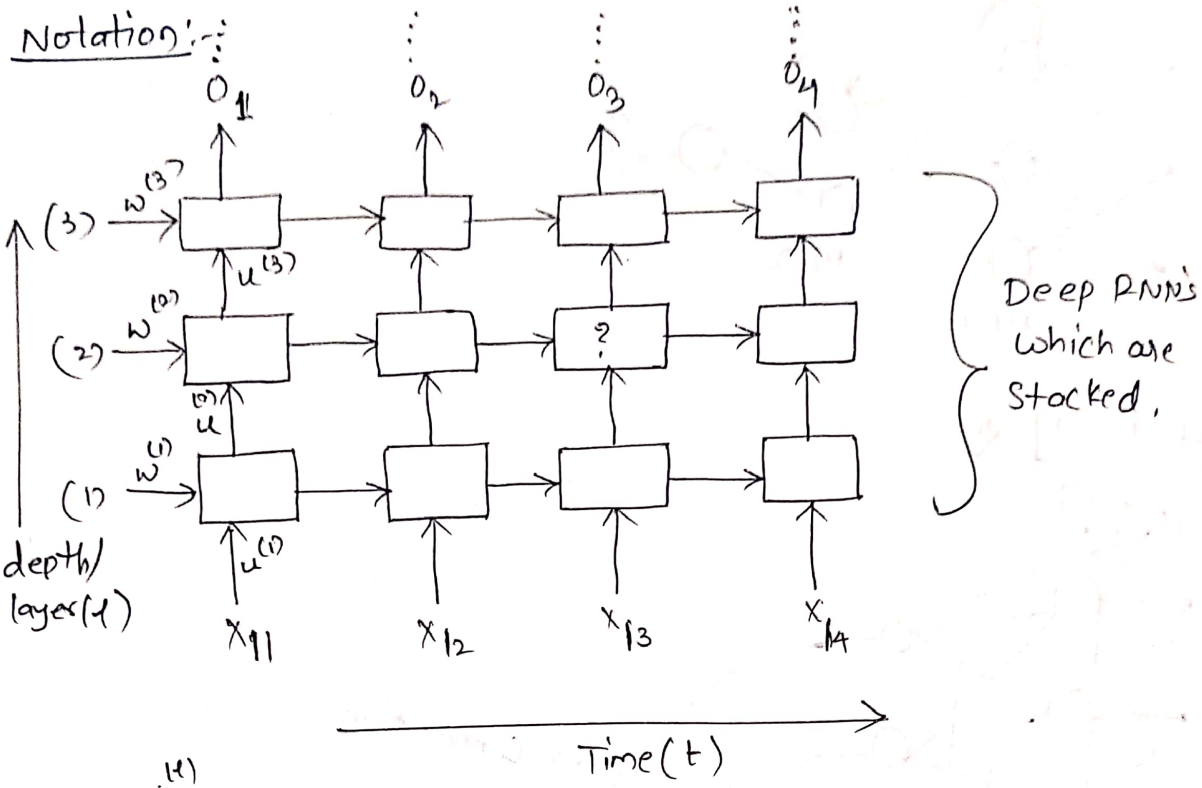
Let's unfold above RNN cell with 2 HL's and see how it actually looks.

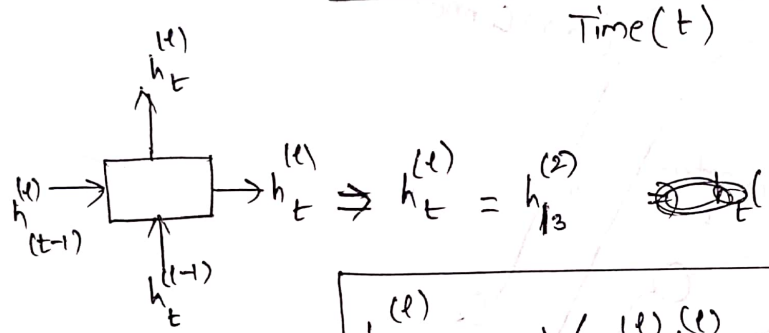
(First RNN cell has 3 neurons & 2nd one has 2 neurons).


ŷ
HL-2
HL-1


At t = '1'
[0, 0, 0]
1
0
0
At t = '2'
0 .
1 .
0 .
At t = '3'
0 .
0 .
1 .
(2x2) Connection
(3x3) connection.
(3x3) conny
(2x2) Connection
ŷ

Notation:-





$$h_t^{(l)} = \tanh\left(W^{(l)} h_{t-1}^{(l)} + u^{(l)} h_t^{(l-1)} + b\right)$$

→ General eq- for any particular RNN cell.

Why and when to use Deep RNN's?

(1) Hierarchical Representation.

Motto behind Deep RNN's is to find Complex Patterns in the data.

→ Starting layers will get only primitive features like words. (love, hate, sad, amazing).

→ Middle layers will find Patterns at Sentence level. (Audio is bad)

→ Deep layers will get you the patterns at document level (Audio is bad, Display is great. Overall I am happy).

(1) Complex tasks like Speech recognition, Machine Translation.
(2) When you have large datasets.
(3) When you have enough computational power.
(4) If accuracy is bad with simple RNN, try deep RNN's.

How to define Deep RNN model using keras:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

model = Sequential([
        Embedding(10000, 32, input_length=100),
                  # 10000 -> I/P of 10k words
                  # 32 -> Every word is 32 dimensional vector
        SimpleRNN(5, return_sequences=True),
        SimpleRNN(3),   # 3 units/nodes
        Dense(1, activation='sigmoid')])

modile.compile(optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=50, batch_size=8).
```

→ This stacking concept can be applied to LSTM's and GRU's also.

→ In practical, we'll use Deep LSTM's and Deep GRU's in most of the problems.

→ Disadvantages with Deep RNN's / Deep LSTM's is
(i) Overfitting may cause due to complexity.
(ii) Training time will be very high.

Bidirectional RNN/BiLSTM:

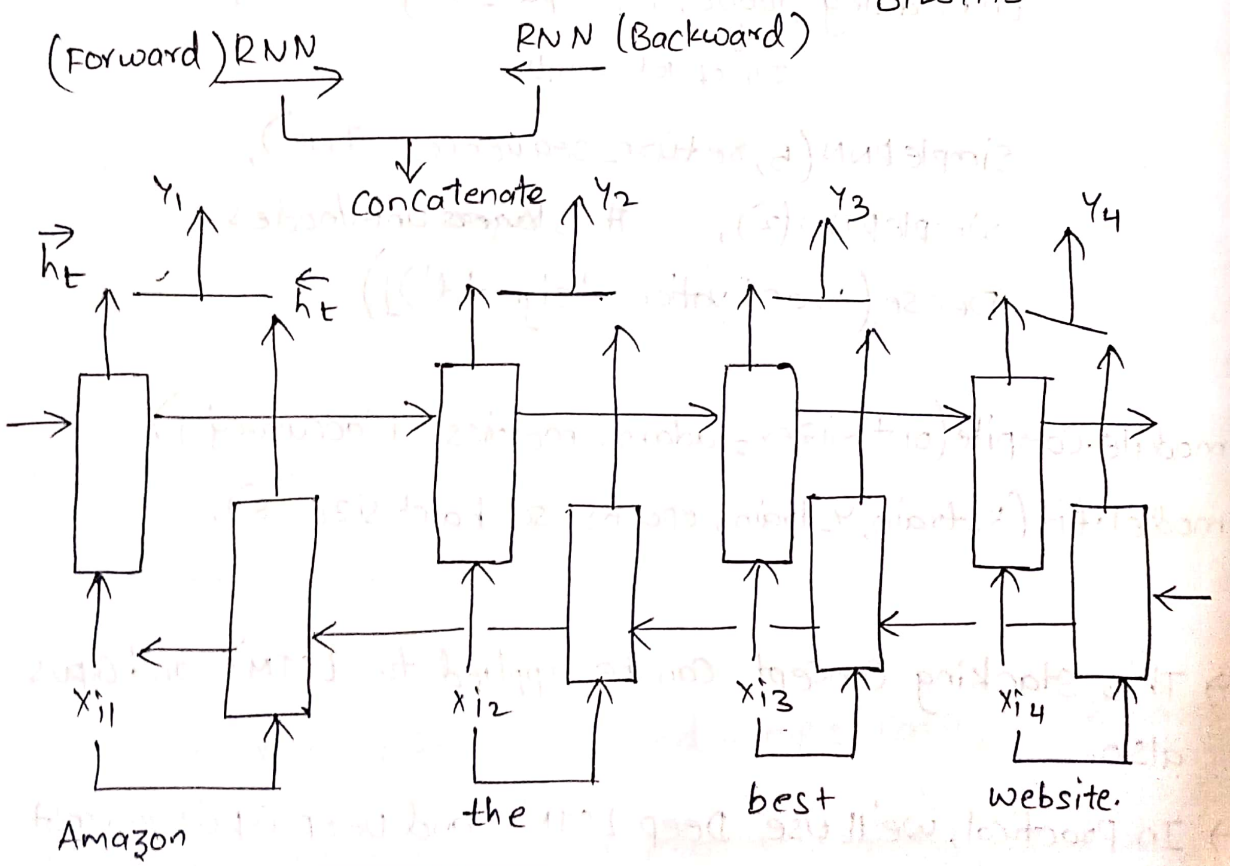
→ RNN's and LSTM's are uni-directional i.e. future inputs doesn't affect past outputs.

→ In tasks like "Named Entity Recognition", O/P will depend on future inputs too

Eg:- I love amazon, it's a beautiful company.
I love amazon, it's a great website.
I love amazon, it's a lovely river.

Bull is running high.

→ Tasks like NER, Machine Translation BiRNN's/are very useful.
BiLSTM's



$$\overrightarrow{h_t} = \tanh\left(\overrightarrow{w}\,\overrightarrow{h_{t-1}} + \overrightarrow{u}\,x_t + \overrightarrow{b}\right)$$

$$\overleftarrow{h_t} = \tanh\left(\overleftarrow{w}\,\overleftarrow{h_{t+1}} + \overleftarrow{u}\,x_t + \overleftarrow{b}\right)$$

$$Y_t = \sigma\left(V\left[\overrightarrow{h_t}, \overleftarrow{h_t}\right] + b\right)$$

Applications:-

(1) Named Entity Recognition.
(2) Pos Tagging
(3) Machine Translation.
(4) Sentiment Analysis
(5) Time series forecasting. (Stock & weather)

→ Main drawback is complexity.
→ one more drawback is we may not get data in both the directions everytime like Real-time speech recognition tasks.

... from LSTM's to ChatGPT:-

History of LLM's | From LSTM's to chatGPT:-

→ We have seen that there are three types of RNN's.

(1) Many to one → Sentiment Analysis

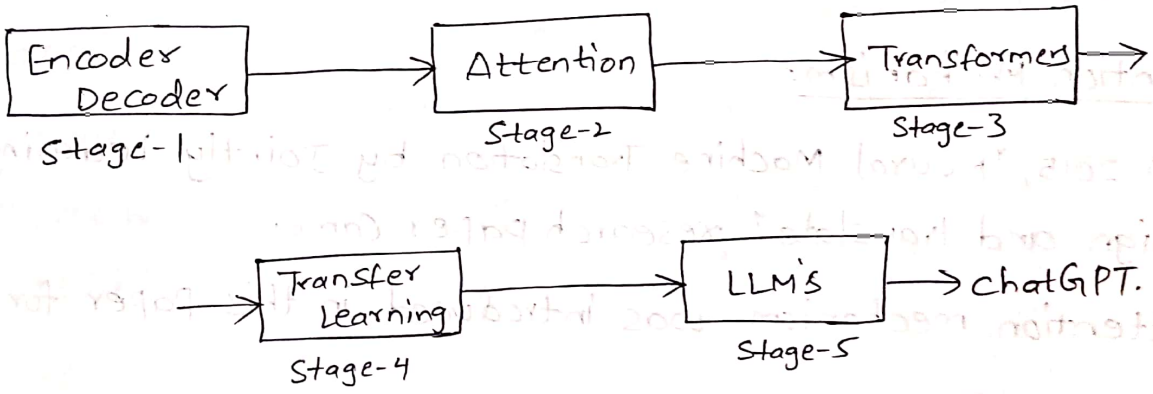
(2) one to Many. → Image Caption Generation

(3) Many to Many

- → Synchronous (fixed length) → Named Entity Recognition
- → Asynchronous (variable length) → Language Translator, Text Summarization, Chatbots.

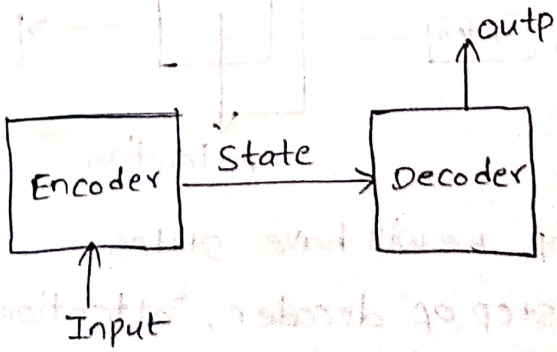
→ Seq2seq models mainly solves asynchronous many to many RNN problems.

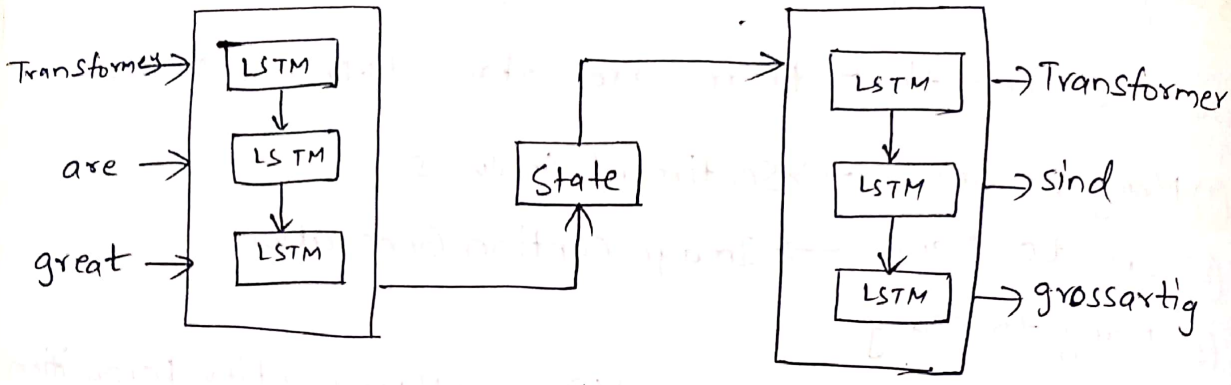
History of seq2seq models:-



→ In 2014, encoder-decoder architecture first came into picture.

→ "Sequence to sequence learning with NN", is the research paper they have published about encoder-decoder architecture.





→ Internal states(Ct&ht) inside LSTM update and summarize the information. This compressed representation known as "context vector" will be given to decoder.

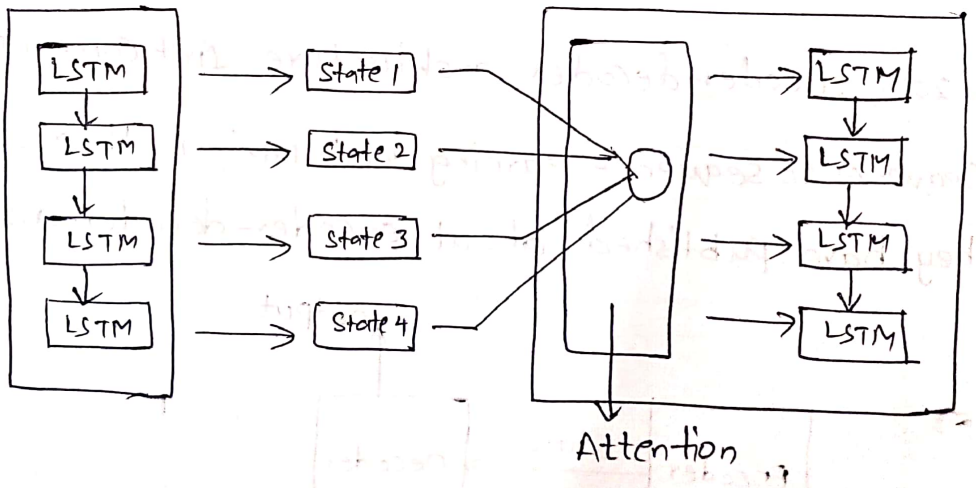
→ this architecture won't work efficiently if input size is ve large. Reason is context vector can't remember the word which came first. It will remember the words that cam at last.

→ To resolve this, "Attention Mechanism" came.

Attention Mechanism:-

→ In 2015, "Neural Machine Translation by Jointly Learning t align and translate" research paper came.

→ Attention mechanism was introduced in this paper for 1st t



→ Here, at each time stamp we will have states.

→ For the current timestep of decoder, "attention" tries to figu out which hidden state from encoder block will be useful for the output.

→ In encoder-decoder architecture we have only one context vector.

→ But in attention mechanism, at every timestamp, we'll have context vector.

→ Problem with this architecture is Training time is very high and computationally it is very complex.

→ Eventually researchers realized that, problem is not with attention mechanism, but with LSTM cells inside. so, they have removed this sequential nature and introduced parallel processing.

Transformers:-

→ In 2017, "Attention is all you need" research paper came.

→ This is the landmark research paper. Almost in every applications, we are using this architecture only.

→ Because of parallel processing, training is very fast.

→ Transformer model was easy to parallelize and it was possible to train it in fraction of time and cost.

Transfer Learning:-

→ If we train transformers properly, everytime we'll get state of Art results.

→ Training transformers from scratch is very difficult. We require lots of Hardware, data and time. we should require insane amount of data.

→ To solve this, we need "Transfer Learning".

Why Transfer Learning not used

→ Till 2018, Transfer Learning was not applied in NLP.

(1) Due to task specificity.

↳ Every task is independent with another task. There is no single model which learns one task and can be applied on diff. tasks. Model should learn language basics.

(2) Lack of data.

↳ should have lots of labelled data for Machine Translation

↦ Universal Language Model ~~for~~ fine-tuning.

→ In 2018, (ULMFIT) research paper came.

→ For pre-training they have used "Language Modelling." It is one of the tasks in NLP, where a Deep learning model will predict next word.

Language Modelling as a pre-trained task:-

(1) Rich feature learning.

↳ learns the features semantically & understands context.

↳ once we get the language understanding basically, that knowledge can be transferrable to diff. task.

(2) Huge availability of data.

↳ Language Modelling doesn't req. labelled data. we can use the data that is available in the internet

→ In 2018, we have two powerful technologies.

(1) In point of architecture → Transformers

(2) Training point of view → Transfer Learning.

LLM's:

→ In 2018, around October, two transformer based Language Models were released.

(1) Google BERT
(2) open AI GPT
} Both are very good at transfer learning.

→ We can download these trained transformers, fine-tune on our dataset and can achieve state of art results. After, these two models, NLP field got totally changed.

GPT → GPT 2.0 → GPT 3.0

Parameters in these models are very large and trained on huge datasets, people started calling these models as "Large language models (LLM's)". This is the starting point of LLM's.

Qualities of LLM's:

When a language model can be considered as LLM?

(1) Data (Billions of words) → GPT 3 trained on 45TB.
(2) Hardware → Clusters of GPU → " " 1000's of NVIDIA GPU
(3) Training time → days to weeks to train LLM.
(4) Cost → Hardware + Electricity + Infrastructure + Experts.
↓
Distributed computing / Super Computers.
(Millions of dollars).

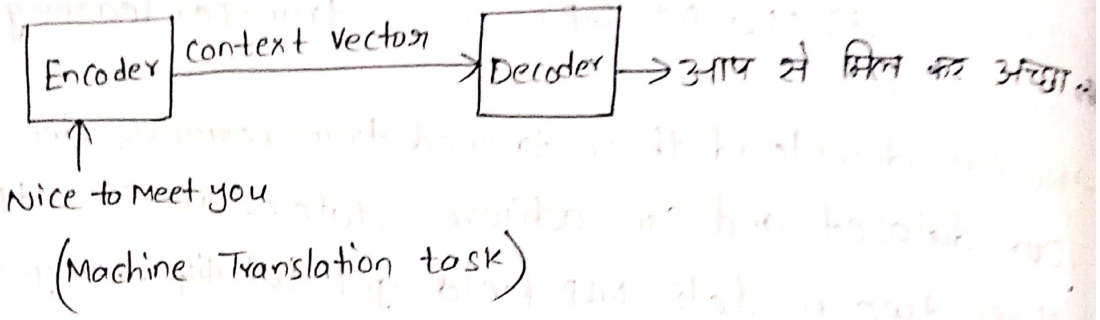
"Transfer Learning" is a technique in which knowledge learned from a task is re-used in order to boost performance on a related task.

(1) Pre-training
(2) Fine tuning.

(1) Encoder - Decoder Architecture

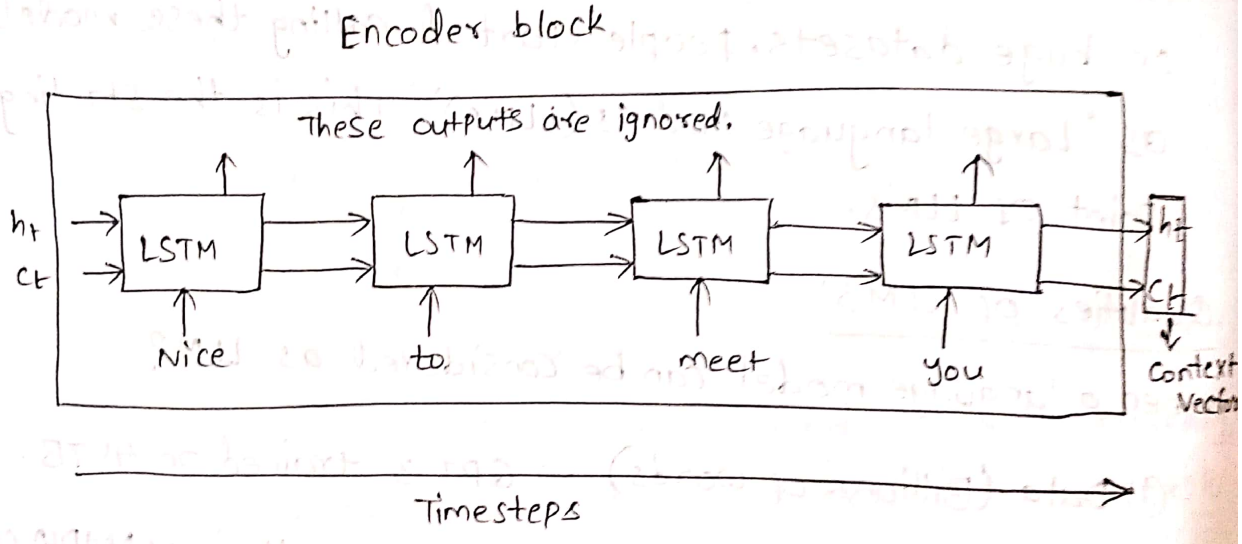
→ It mainly deals with Asynchronous RNN problems.

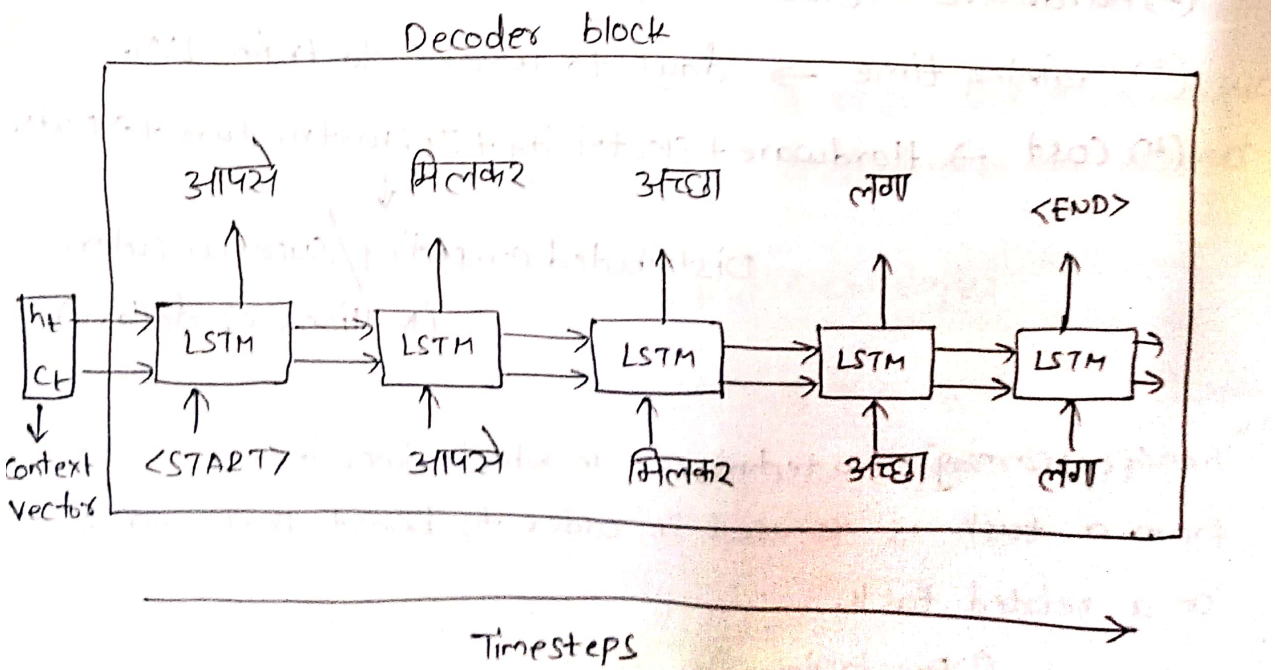
→ It is a Seq2seq model, where I/P & O/P both are sequences.



(Machine Translation task)

→ Context vector is nothing but the summary of input. Encoder summarizes the entire input and represent it in vector form.





Dataset will look like this.

Input	Output
Hello	नमस्ते
I am ok	मैं ठीक हूँ
Awesome	बहुत बढ़िया

→ It is a type of supervised learning.

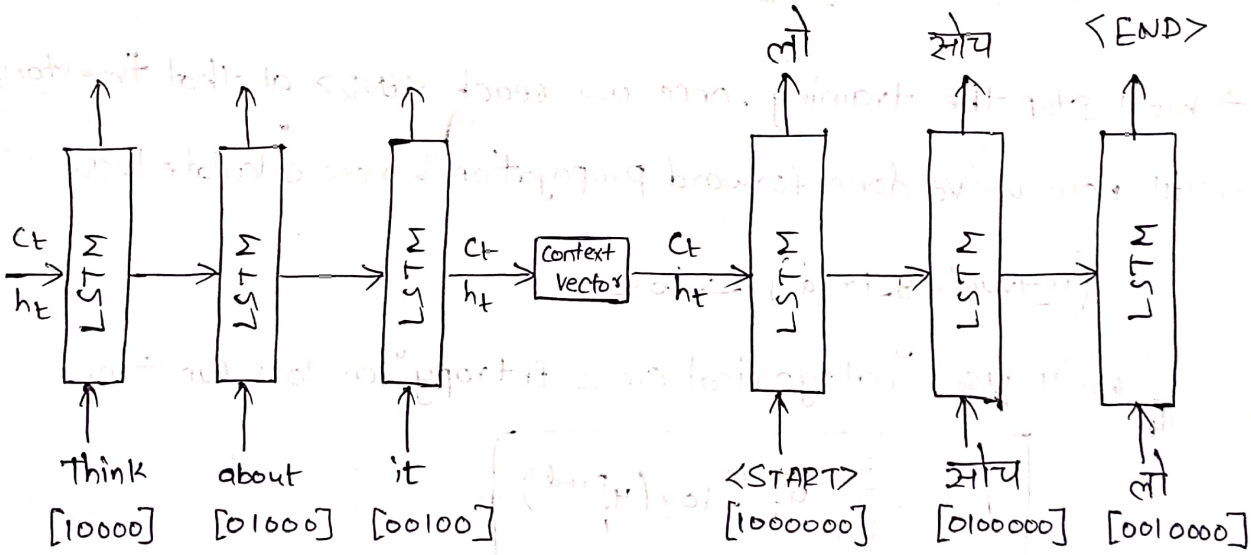
→ We will use this type of dataset, give this data as input and train the model.

Eng	Hin
Think about it	सोच लो
come in	अंदर आ जाओ
Turn off the light	लाइट बंद करो

⇒

think → [10000] About → [01000]
it → [00100] come → [00010]
in → [00001]

<start> → [1000000]
सोच → [0100000]
लो → [0010000]
अंदर → [0001000]
आ → [0000100]
जाओ → [0000010]
<END> → [0000001]



→ To fetch the o/p at decoder side, we'll add one softMax layer. No. of nodes in softmax layer will be equal to no. of words in vocabulary at output. (Here, we have total 7 words in o/p).

→ It gives probability at each node i.e, [0.2, 0.15, 0.7, 0.3, 0.07, 0.1, 0.04]

o/p ⇒ [0 0 1 0 0 0 0]

y_Pred = [0010000]

y_true = [0100000]

→ During training, irrespective of output, we'll give the true/actual input at decoder side. We'll do this because to improve the training period. This is called "Teacher forcing".

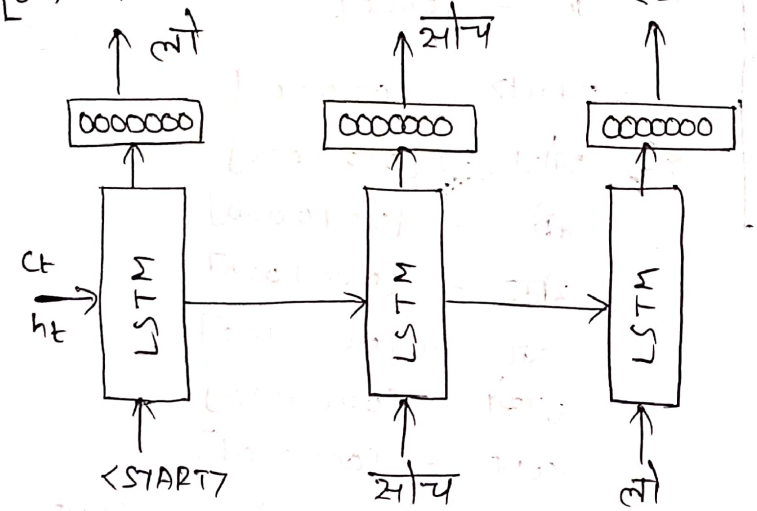
Decoder block :-

[0, 1, 0, 0, 0, 0, 0] → y_true

[0.2, 0.15, 0.7, 0.3, 0.07, 0.1, 0.04] → y_Pred

[0, 0, 0, 0, 0, 0, 1]

[0.1, 0.2, 0.2, 0.1, 0.3, 0.1, 0.6]



→ We'll stop the training, once we reach <END> at that timestamp.

→ Till now we've done forward propogation & now calculate loss.

(y_true - y_Pred) → loss

- We'll use "categorical cross entropy" as loss function.

$$L = -\sum_{i=1}^{7} y_i^{true} \log(y_i^{pred})$$

$L_{(t=1)} = -1 * \log(0.15) \Rightarrow 1$

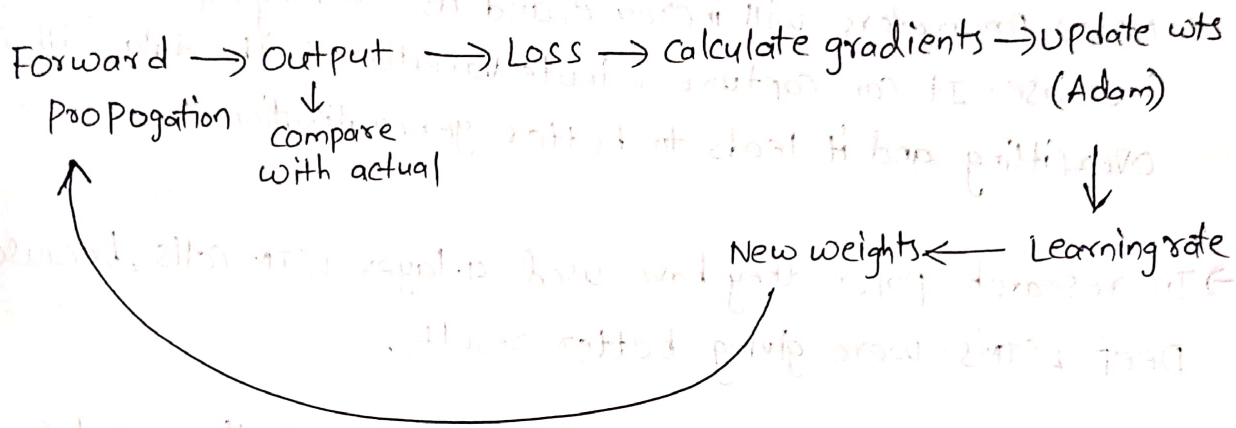
$L_{(t=2)} = -1 * \log(0.1) \Rightarrow 1$ } Incorrect Predictions

$L_{(t=3)} = -1 * \log(0.6) \Rightarrow 0.221$ } Correct Prediction

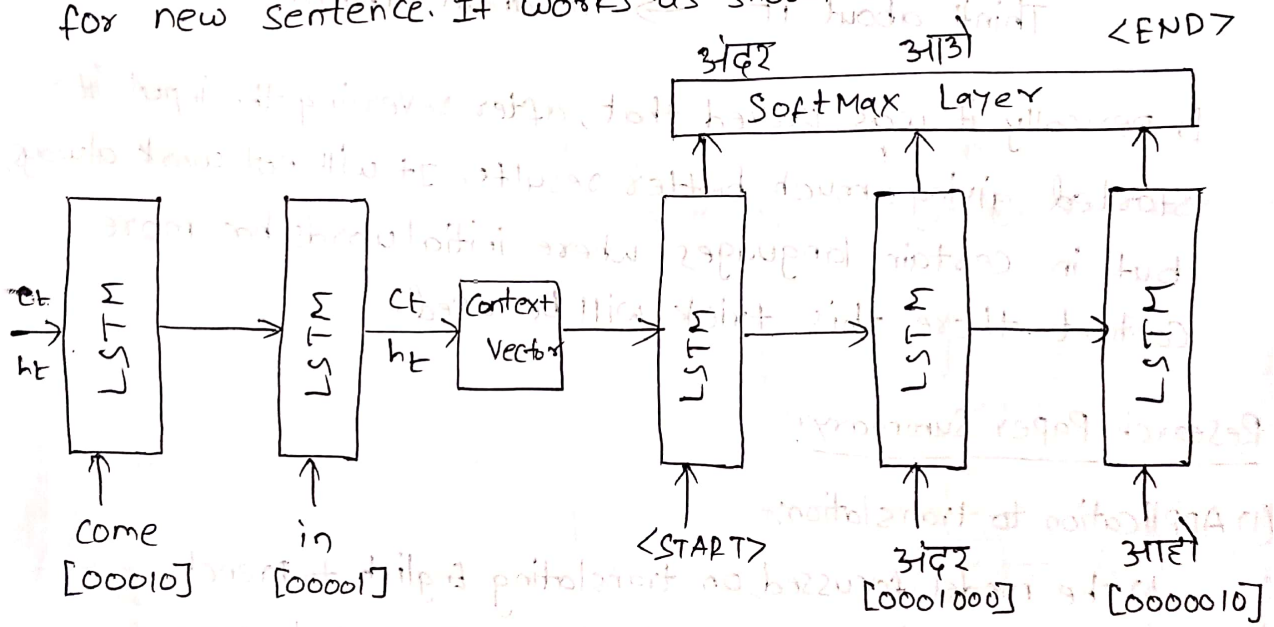
Total loss = 2.21

→ Now, we'll do Back Propogation, → Calculate gradients, → Update Parameters.

→ Calculate gradients for each of the trainable parameters. These gradients measures, how much a particular parameter is contributing to loss function and in what direction we need to adjust the parameter to reduce loss.



→ Our training process got completed and encoder-decoder architecture is now fully trained. Now, we need to predict for new sentence. It works as shown below.



→ This time, we'll not do teacher forcing. We'll give the output what we got at previous timestamp.

How to improve the above architecture?

(1) Instead of using OneHot Encoding, add one Embedding layer at input and convert it into vectors. Embeddings are low dimensional vectors and sparse. They will capture the context. For this, we can use Word2vec / Glove.

(2) Instead of using single layer

Why Deep LSTM's ?

→ It can handle long-term dependencies.

→ They can understand Hierarchical representation very well.

→ No. of Parameters will increase, and its learning capability will increase. It can capture minute variations in the data without overfitting and it leads to better generalization.

→ In research paper they have used 4-layer LSTM cells, because Deep LSTM's were giving better results.

(3) one more improvement is, while giving input to the encoder, give it in reverse order.

Think about it → it about Think.

Emperically it was proved that, after reversing the input it started giving much better results. It will not work always but in certain languages where initial words has more context, there this trick will be used.

Research Paper Summary :-

(1) Application to translation :-

↳ The model focussed on translating English to French, demonstrating the effectiveness of sequence-to-sequence Learning in Neural Machine Translation.

(2) Special End-of-Sentence Symbol :-

↳ Each sentence in the dataset was terminated with a unique symbol <Eos> to recognize the end.

(3) Dataset :-

↳ Model was trained on subset of 12 million sentences, comprising 348M French words and 304M English words.

(4) Vocabulary limitation:

↳ 160,000 most frequent english words and 80,000 french words. words not in these vocabularies were replaced with UNK token.

(5) Reversing input sequences.

(6) word Embeddings.

↳ Model used 1000-dimensional vector to represent I/p words.

(7) Architecture details.

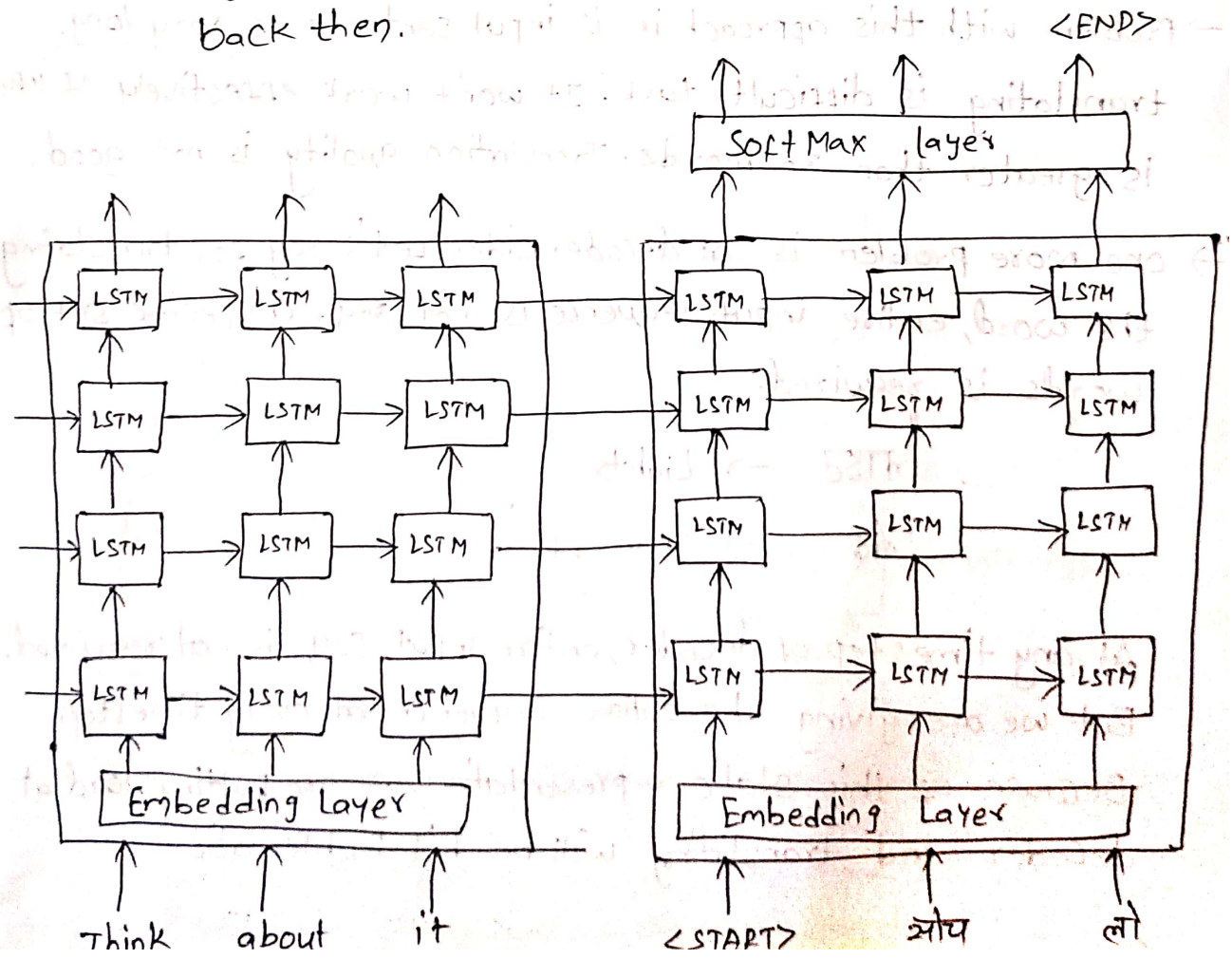
↳ Both Encoder and decoder had 4 layers, with each layer containing 1000 units.

(8) Output layer

↳ output layer employed a softMax func, to generate probabilities

(9) Performance - BLEU score:

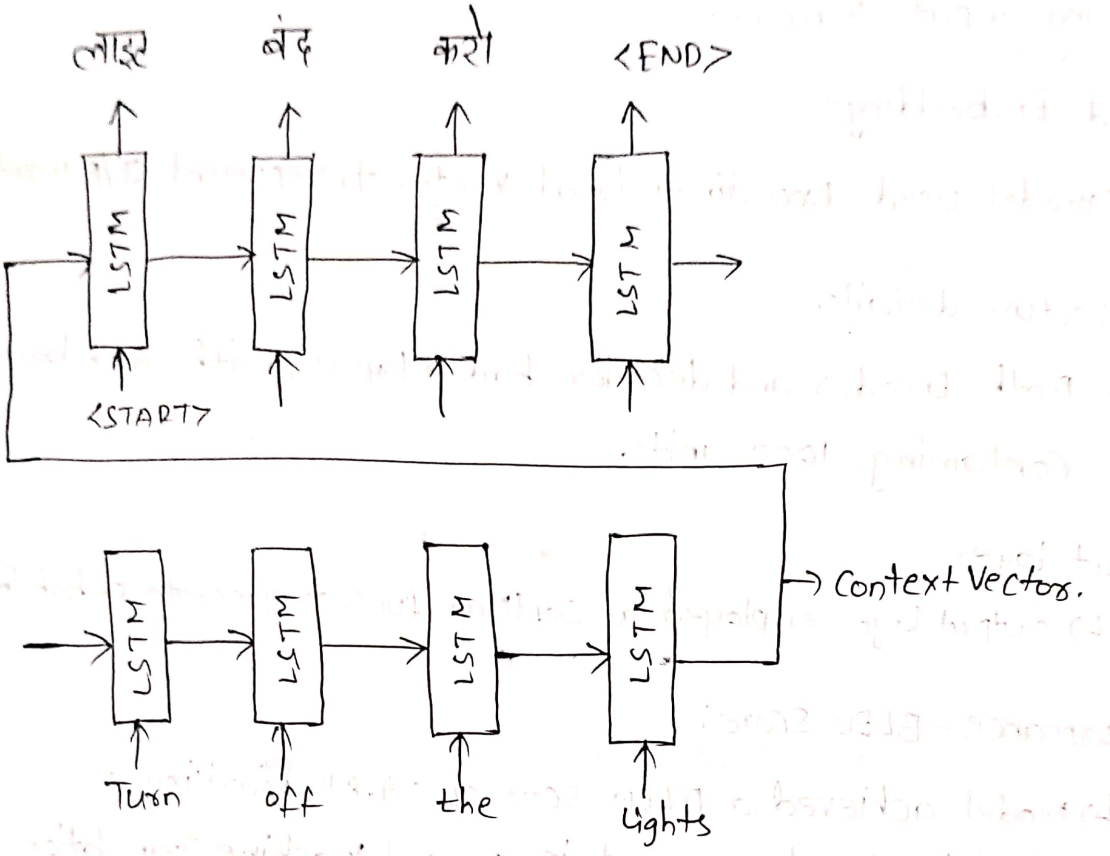
↳ Model achieved a BLEU score of 34.81, marking a significant advancement in Neural Machine Translation back then.



(2) Attention Mechanism

We have encoder-decoder arch. Again why Attention Mechanism?

The why?



→ Problem with this approach is if input sentence is very long, translating is difficult task. It won't work effectively if input is greater than 30 words. Translation quality is not good.

→ one more problem is at decoder side. Let's say for translating the word, entire input sequence is not req. a specific set of words is required.

लाइट → Lights

बंद → Turn off

At any timestep of decoder, entire input seq. is not required. But we are giving the entire sequence at every timestep. Because of this static representation, we are putting load at decoder and translating will be bit problematic.

→ Instead, we can keep attention at specific parts of words.

Solution:-

→ We can introduce one "attention mechanism" which focuses on specific part of input and it helps in translation for decoder

→ Attention tell us that which hidden state from encoder will be useful for translating at decoder at particular time stamp.

c_1 → Attention value at timestep 1

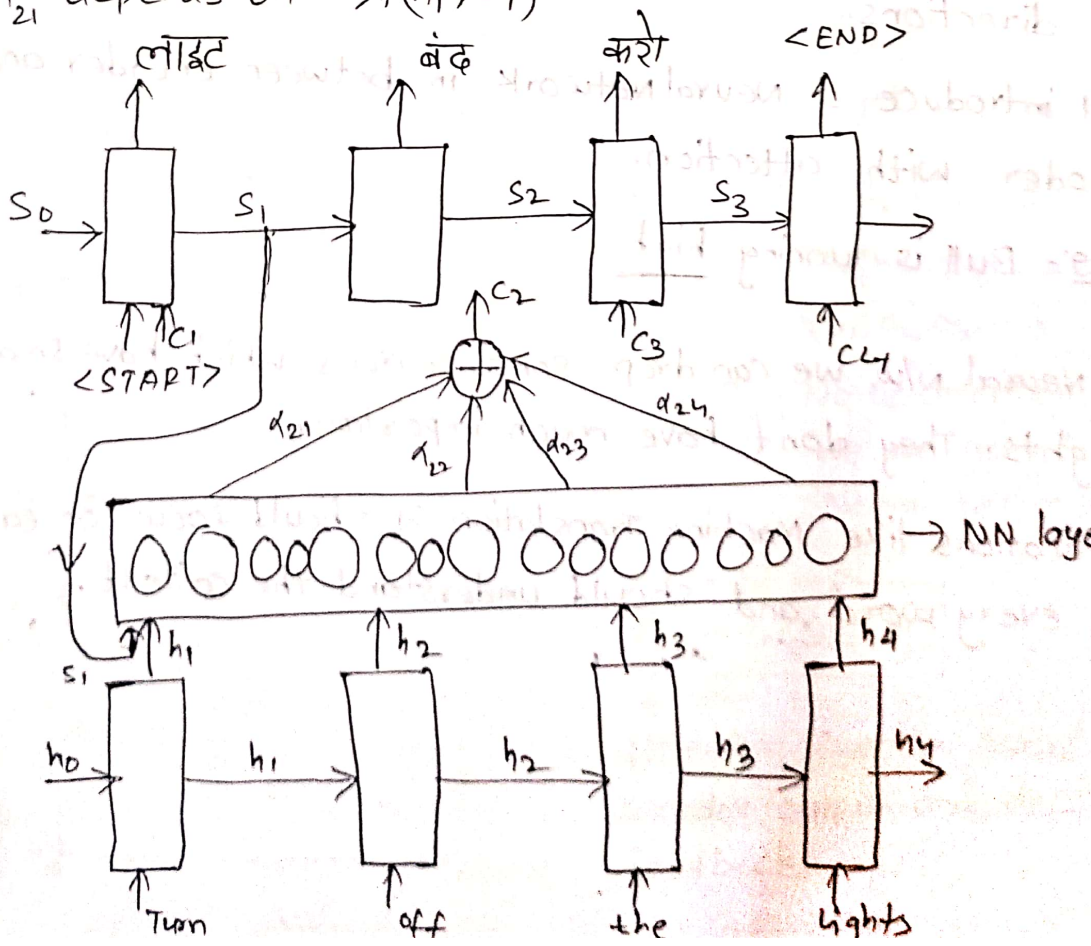
$$\boxed{c_i = \sum \alpha_{ij} h_j} \Rightarrow c_1 = \alpha_{11}h_1 + \alpha_{12}h_2 + \alpha_{13}h_3 + \alpha_{14}h_4$$

i → Timestep at decoder

j → Timestep at encoder.

α_{21} → (or) (alignment score) Nothing but a similarity scores which tell sme that, to print the output at decoderside at 2nd timestamp what is the role of encoder at 1st timestamp.

α_{21} depends on → $f(h_1, s_1)$



Neural Machine Translation using [illegible] paper:-

→ NMT aims at building a single Neural N/w that can be jointly tuned to maximize translation performance.

→ In basic-encoder, we have RNN/LSTM gates. In this model, we will use bidirectional LSTM.

→ Issue with vanilla/Basic encoder-decoder architecture is it will not depend on future words, it takes only Previous words

→ One more issue is that, it could be possible that all inputs are not concentrating in the encoder.

→ We need some kind of network which will be able to understand the context in the input.

→ Basic encoder-decoder attempts to encode whole input sequence into a single fixed-length vector. Instead, new model encodes the input sequence into sequence of vectors and chooses vectors while decoding.

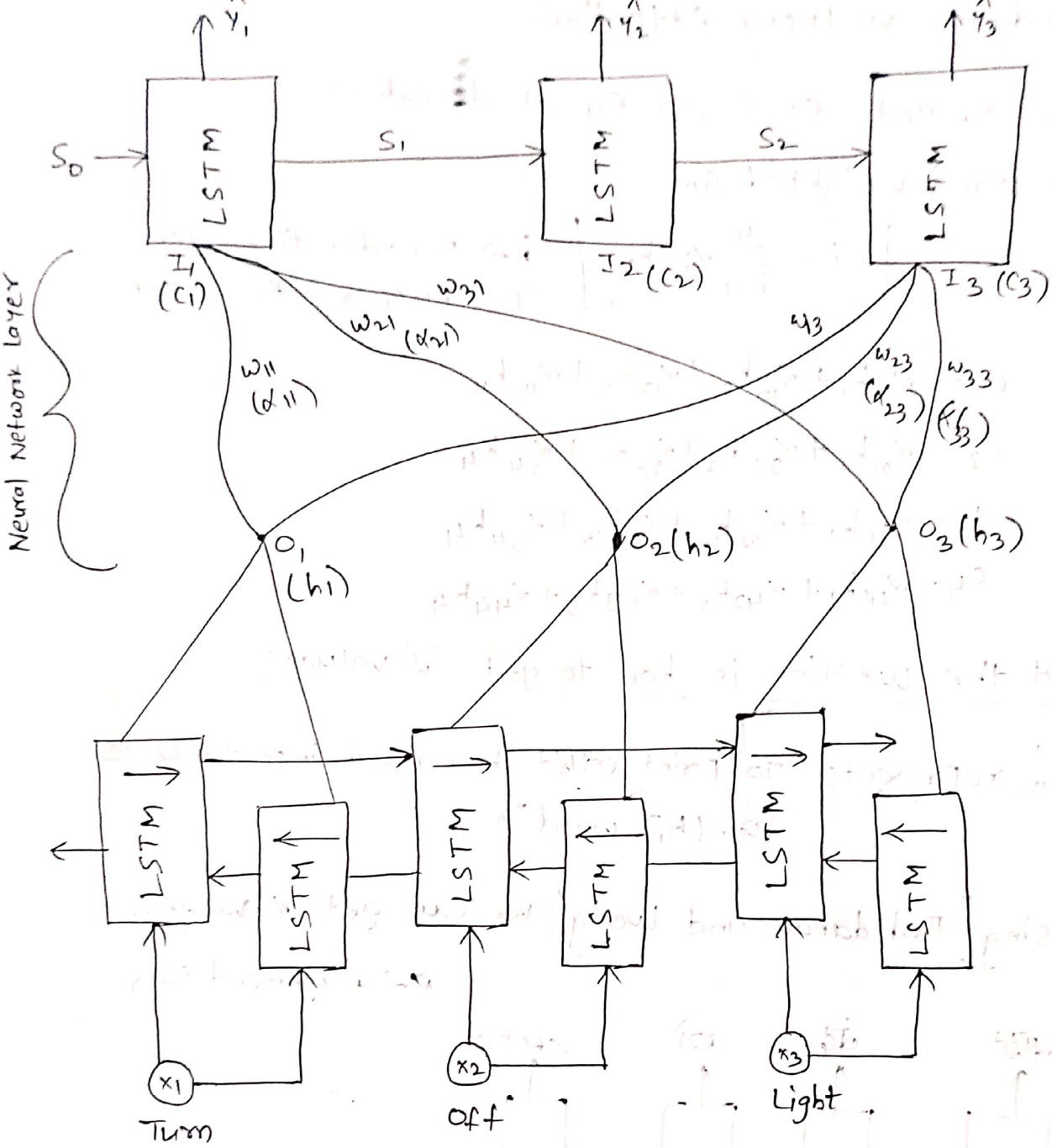
→ We need to train the model for better accuracy. in both the directions.

→ We'll introduce a Neural Network in between encoder and decoder with attention.

Eg:- Bull is running high

→ In Neural N/w, we can drop some neurons which have smaller weights. They don't have much importance.

→ Applications like Machine Translation, it should focus on each and every word and should understand the context.



Here, $c_i = \{I_1, I_2, I_3, \dots\}$

$\alpha_{ij} = \{w_{11}, w_{21}, w_{31}, \dots\}$

$h_j = \{o_1, o_2, o_3, \dots\}$

T_x = No. of attention inputs we are going to take in Neural Network.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Context Vector

$$c_1 = o_1 w_{11} + o_2 w_{21} + o_3 w_{31}$$

$$e_{ij} = a(s_{i-1}, h_j)$$

attention function based on encoder output and decoder feedback.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Bahdanau vs Luong Attention

How to get c_1, c_2, c_3, c_4 at decoder?

- using weighted sum.

$$c_i = \sum_{i=1}^{4} \alpha_{ij} h_j$$

$i \rightarrow$ Decoder time step
$j \rightarrow$ Encoder time step

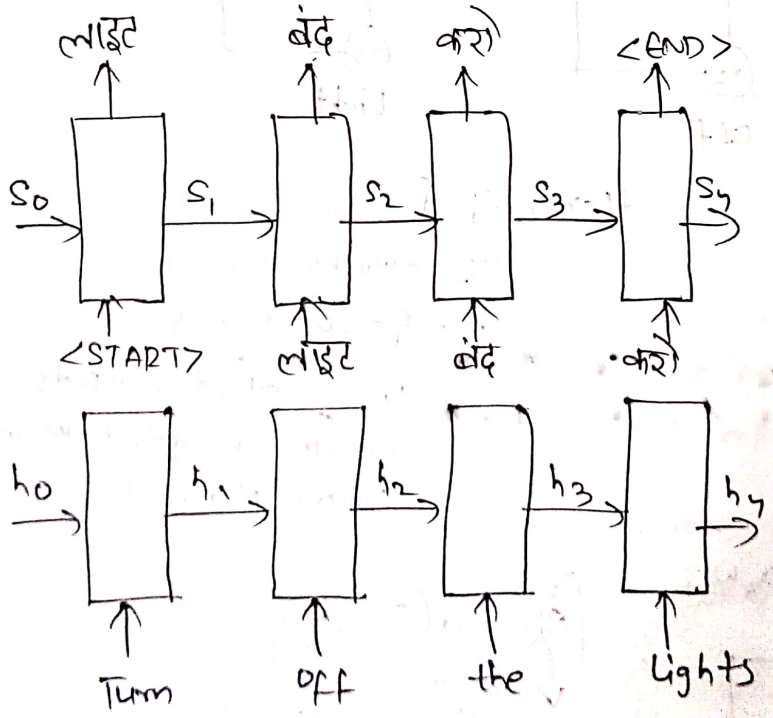
$$c_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3 + \alpha_{14} h_4$$
$$c_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3 + \alpha_{24} h_4$$
$$c_3 = \alpha_{31} h_1 + \alpha_{32} h_2 + \alpha_{33} h_3 + \alpha_{34} h_4$$
$$c_4 = \alpha_{41} h_1 + \alpha_{42} h_2 + \alpha_{43} h_3 + \alpha_{44} h_4$$

But the question is how to get 'α' values?

'α_{11}' represents "To print लाइट, how much importance is turn(h_1) word"?

→ Using "Bahdanau and Luong" we can get α-values.

α → Alignment score



$\alpha_{11} = f(\ )$

↓

we need to define one function inorder to get α-values

→ 'α' tells us that, "given translation till now, on this basis what is the role of current encoder hidden state, to print the o/p at decoder at current time step?"

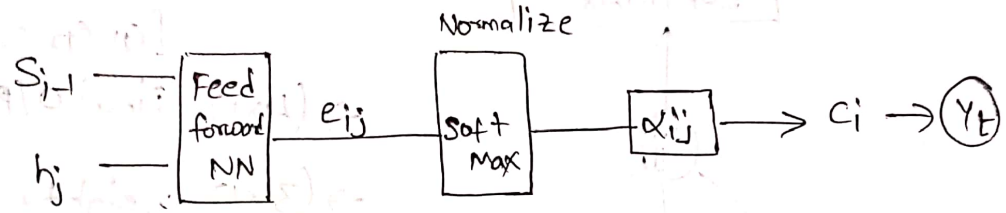
$\alpha_{11} = f(h_1, s_0)$; $\alpha_{21} = f(h_1, s_1)$

α_{11} → Depending on current encoder hidden state and Previous decoder hidden state.

$$\alpha_{ij} = f(h_j, s_{i-1})$$

→ What should be the Mathematical function we should use?

- You can use "Neural Networks" because feed forward NN's are universal function approximators.



Let's say $h_0\ h_1\ h_2\ h_3$ and $s_0\ s_1\ s_2\ s_3$ are 4-dim vectors.

$h_1 \rightarrow [h_{11}\ h_{12}\ h_{13}\ h_{14}]$

$h_2 \rightarrow [h_{21}\ h_{22}\ h_{23}\ h_{24}]$

$h_3 \rightarrow [h_{31}\ h_{32}\ h_{33}\ h_{34}]$

$h_4 \rightarrow [h_{41}\ h_{42}\ h_{43}\ h_{44}]$

$s_0 \rightarrow [s_{01}\ s_{02}\ s_{03}\ s_{04}]$

$s_1 \rightarrow [s_{11}\ s_{12}\ s_{13}\ s_{14}]$

$s_2 \rightarrow [s_{21}\ s_{22}\ s_{23}\ s_{24}]$

$s_3 \rightarrow [s_{31}\ s_{32}\ s_{33}\ s_{34}]$

$c_2 = \alpha_{21}h_1 + \alpha_{22}h_2 + \alpha_{23}h_3 + \alpha_{24}h_4$

$\alpha_{21} \rightarrow [s_{11}\ s_{12}\ s_{13}\ s_{14}\ h_{11}\ h_{12}\ h_{13}\ h_{14}]$

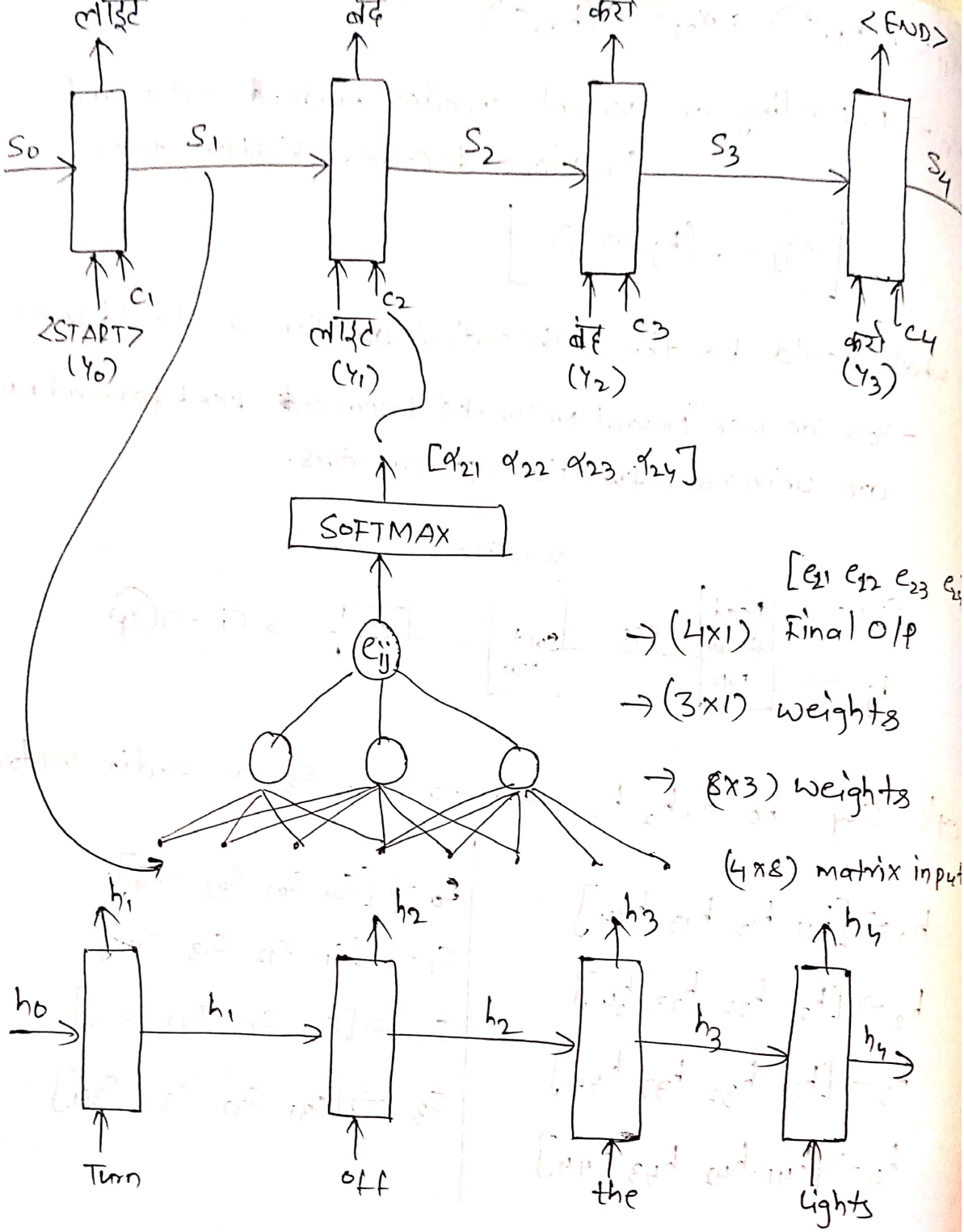
$\alpha_{22} \rightarrow [s_{11}\ s_{12}\ s_{13}\ s_{14}\ h_{21}\ h_{22}\ h_{23}\ h_{24}]$

$\alpha_{23} \rightarrow [s_{11}\ s_{12}\ s_{13}\ s_{14}\ h_{31}\ h_{32}\ h_{33}\ h_{34}]$

$\alpha_{24} \rightarrow [s_{11}\ s_{12}\ s_{13}\ s_{14}\ h_{41}\ h_{42}\ h_{43}\ h_{44}]$

4 rows and 8 columns.

→ We'll send this (4x8) matrix as input to NN.



→ Weights will be same at everytime-step. They will be change during back-propogation after calculating loss.

→ once we reach <END> at decoder, we'll calculate loss, compare with actual value and perform back propogati

$\alpha_{11} \rightarrow f(h_1, S_0)$ & $C_1 \rightarrow$ depends on α_{11}, h_1

$$\alpha_{ij} = \frac{\exp(e^{ij})}{\sum \exp(e_{ik})} \qquad e_{ij} = \tanh(w[S_{i-1} h_j] + b)$$

Luong Attention:-

→ Here also, we'll calculate α-values & c_i

Here, $\alpha_{ij} = f(s_i, h_j)$ → # 1st difference.

↓

Here, we'll not use Feed forward NN.

Instead, simply we can use $[s_i^T . h_j]$ Dot product

2nd difference.

$s_i \rightarrow [a\ b\ c]$

$h_j \rightarrow [e\ f\ g]$ ⇒ $\begin{bmatrix} a \\ b \\ c \end{bmatrix} [e\ f\ g]$ ⇒ $\underbrace{ae + bf + cg}_{\text{Scalar value.}}$

→ We are using current decoder hidden state just because of updated information. Because of this updated value, translations will be more accurate.

→ Neural Network, we used in Bahdanau Attention is also known as "Alignment Model."

→ Bahdanu Attention is also known as "Additive Attention".

→ Luong Attention is also known as "Multiplicative Attention"

Transformers Introduction:

What is Transformers?

→ Transformer is a NN architecture like ANN, CNN, RNN.

ANN → Tabular data

CNN → Image data

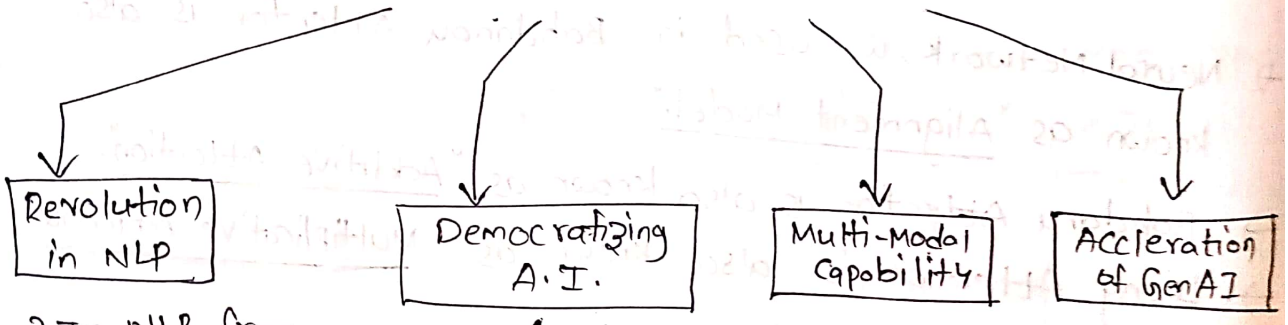
RNN → Textual data

Transformer → seq2 seq tasks.

→ Instead of LSTM, here we'll use self-Attention which offers Parallel processing.

→ In 2017, this architecture was introduced in research paper "Attention is all you need".

Impact of Transformers



→ In NLP, from last 5oyrs, research is going on. Due to Transformer arch, any NLP problem can be solved, with SOTA results.

→ using these pre-trained transformers, we can apply on our tasks and achieve SOTA results.

→ Transformers not only works on textual data. It is multi-modal and it will work on any type of data. Images/speech/video

Timeline:-

2000-2014 → RNN's / LSTM's
2015 → Attention
2017 → Transformers
2018 → BERT / GPT (Transfer learning era in NLP)
2018-2020 → Vision Transformers / Alphafold-2.0
2021 → Transformers in Gen AI
2022 → ChatGPT / Stable Diffusion.

Advantages:-

→ Scalability. (Because of parallel processing).
→ Transfer Learning. (Pre-training and fine tuning)
→ Multimodal input & output.
→ Flexible type of architecture.
→ Ecosystem. (Huggingface)
→ Easily integrated with other A.I. techniques.
  - GAN's + Transformers
  - RL + Transformers

Real-world Applications:-

(1) ChatGPT
(2) open AI Dalle 2
(3) Alphafold
(4) OpenAI Codex

Disadvantages:-

→ Requires high computation resources.
→ Huge amount of data.
→ Risk of overfitting.
→ Interpretability is poor.
  ↳ No one knows what is happening and why?

Future:-

→ Performance in efficiency. (Quantization)
→ Improving Multimodal capabilities. (Sensor/Bio-metric)
→ Responsible. (should overcome ethical concerns).
→ Domain specific. (Doctor GPT, legal GPT, Teacher GPT)
→ Multi-lingual. (Training trans. on regional langs)
→ Need to improve interpretability.

What is Self-Attention?

→ Almost in every NLP application, main aim is to convert words to vectors. This is the most important step.

Words → vectors } vectorization

→ "Word embeddings" is the most advanced technique and it captu semantic meaning.

→ Embeddings has the capability to convert the meaning of wor to a vector. But sometimes, this will fail.

Problem of Average Meaning:-

(1) An Apple a day keeps the doctor away.
(2) Apple is healthy.
(3) Apple is better than orange.
(4) Apple makes great phones.

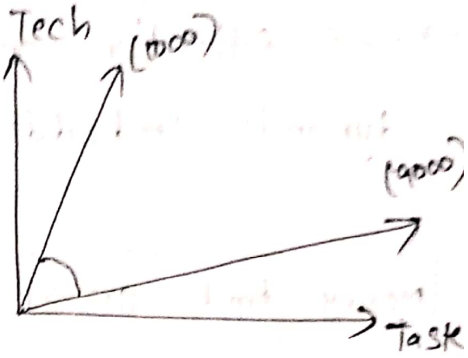
Let's convert Apple to a 2-dimensional vector.

[x, y]
↓ ↳ Technology
Taste

→ Basically word embeddings willnot capture the meaning. Infact, it captures average meaning.

→ It means that particular word, or an average, which meaning it is giving in our data.

Apple → ① [0.6, 0]
② [0.7, 0]
③ [0.8, 0]
④ [0.8, 0.2]



→ In our entire data, if there are 9000 fruit related words and 1000 tech related words, it will be biased towards fruit.

→ Here, word embeddings are static. once, they are created, we'll use the same embedding in all the sentences.

Eg:- Apple launched a new phone while I was eating orange.

→ Instead of using stactic embeddings, we should use "Contextual embeddings" to dynamically change the ~~embe~~ embeddings based on the context

→ "Self Attention" is the mechanism where we can generate contextual embeddings.