Transfer learning

Why:-

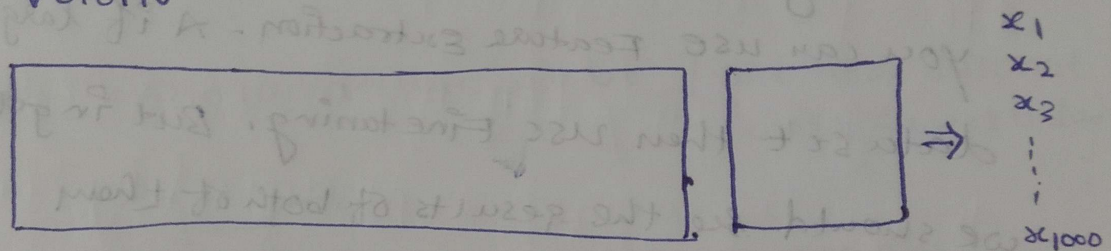
What if your are doing a project, where the pre trained models are not trained on that dataset.

Defn:- Transfer learning is a research problem in ML that focuses on storing knowledge gained while solving one problem and applying it to different but related problem.

*** How it works:-**

We use the pretrained models ⇒

Eg:- VGG16



Step-1:- we freeze the conv-base or conv-layers then we remove the FC layers.

Step-2:- With the dataset u have we train another Fully connected layer.

Step3:- Now we attach the traind dense Fully connected model with conv-layers that we freezed before.

Inlays of Doing Transfer learning :-

i) Feature extraction (discused before)

ii) Fine tuning.

Fine tuning :-

In fine tuning we also train the last convolutional layers with Fully connected dense layers.

When you should use fine Tuning :-

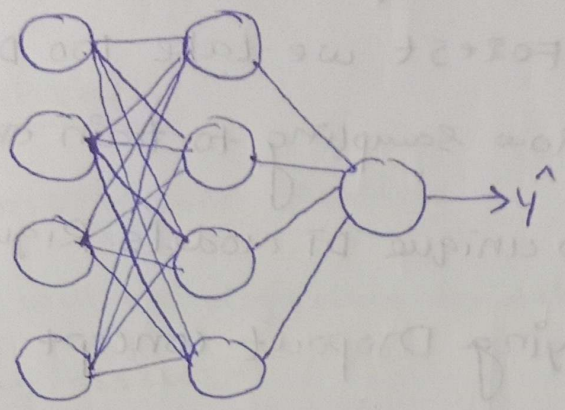
* Generally if you have a small dataset then you can use Feature extraction. & if large dataset then use Fine tuning. But in general we should see the results of both of them

Inlays to Decrease the overfitting in Neural Networks.

1) Add more data (using data augumentation)
2) Reduce the complexity of Neural Network)
3) Early stoping
4) Regularization $\begin{cases} L1 \\ L2 \end{cases}$
5) Dropout
6) Batch Normalization

1) DropOuts:- It is technique where we turn off the nodes randomly from hidden & input layer in every epoch. (p)

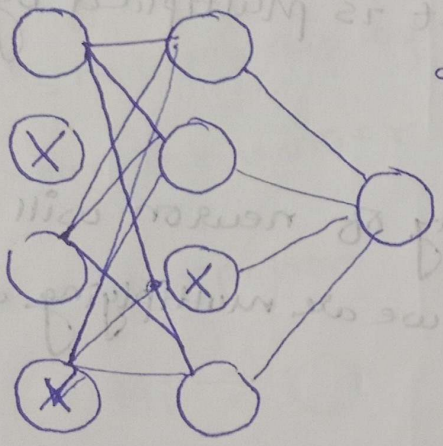
Eg:-



intial ANN Model with } Then

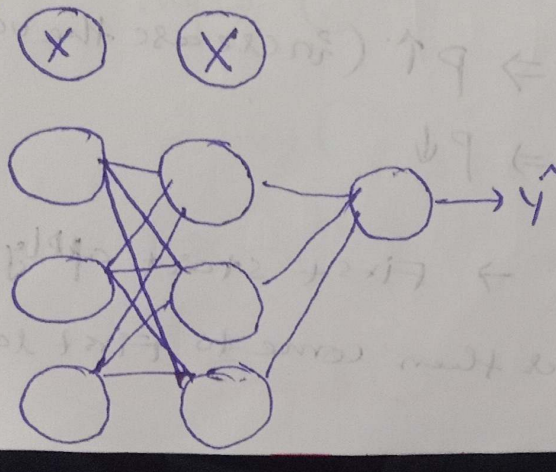
p=0.25 p=0.25

1-epoch: The architecture will be: (one node will not work in each layer)



{ we can see the weights are reduced now there is less probability for overfitting }

2-epoch



* Therefore we can say that there will be different architecture

* Random Forest Analogy:-

As in random Forest we take 100 DT (suppose) and will do col / row sampling to train every DT. Now we have 100 unique DT models Right!!.

Similarly on Applying Dropout concept there will unique architecture in every epoch.

How prediction works:-

During testing All the neurons will work.

And every weight is multiplied by "(1-p)" value. Bec:-

(1-p) = probability of neuron will work right. so we are multiplying with it

Pratical Tips:-

1) overfitting ⇒ p↑ (increase the value of p)

underfitting ⇒ p↓

2) last layer → First start applying Dropout for last layer then come to First layers.

3) In case of AxIxI : p range ⇒ 0·1 to 0·5

CAIAI :- p → 0·4 - 0·5

RAIN :- p → 0·2 - 0·3

Drawbracks

i) convergence → delay

ii) It will be difficult to debug back when had any issuese.

Why RNN's

Eg:- Hi my name is Jayath (5) 1

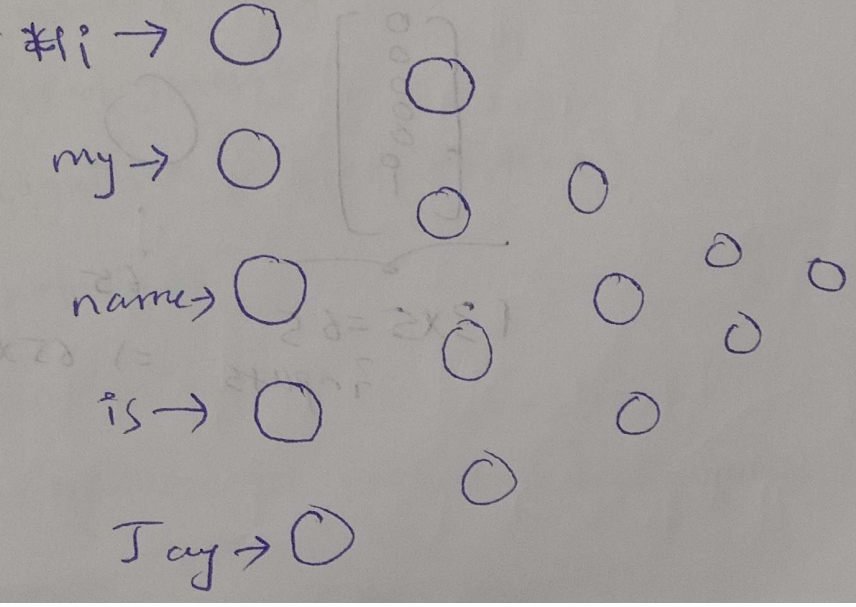
It is not my cup of tea (5) 0

your the winner (3) 1

13

unique words (+ve/-ve) output

Let's take a prepctron

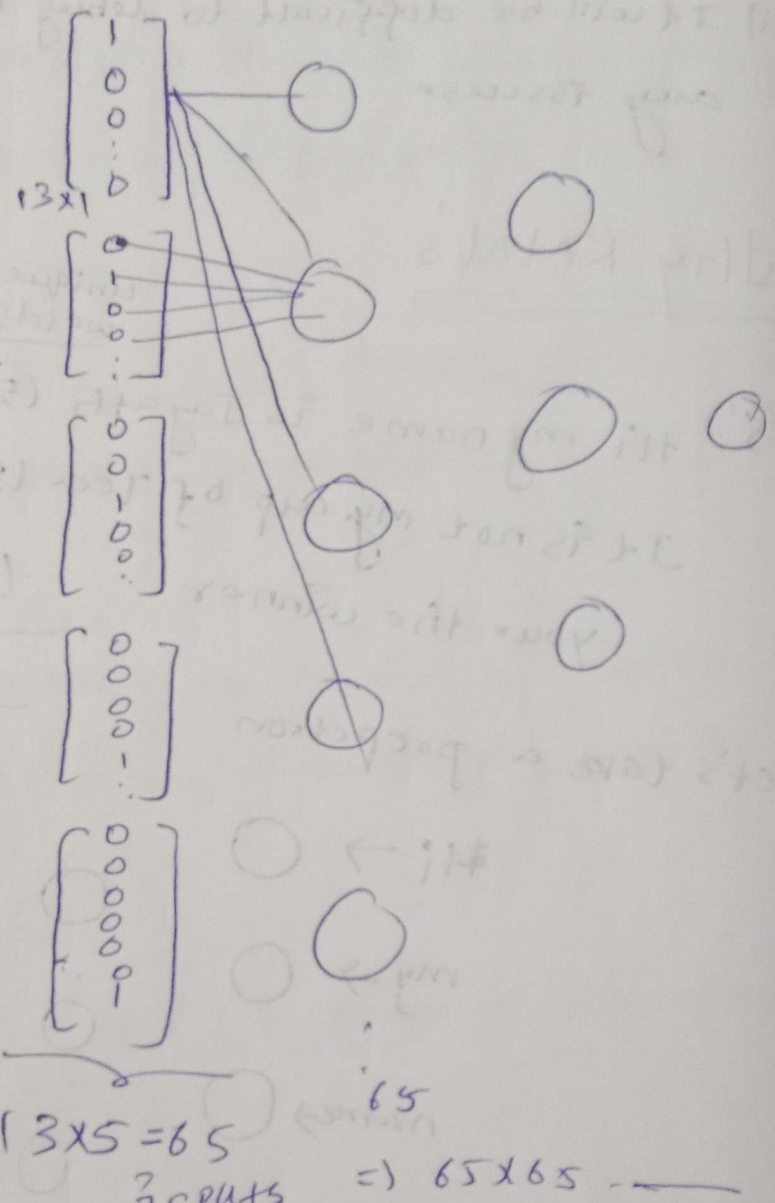


* on doing one hot encoding, we convert into vectors.

hi = [1,0,0,0, ... 0] (13)

my = [0,1,0 ... 0] (13)

is = [0,0,1 ... 0]



problems with ANNs

1) Text input varying size

2) Zero padding ⇒ unnecessary computation

3) If you did zero padding also, what if the test or new text as different no of unique words. Then we get prediction problem

4) Totally disregarding the sequence information.

Then they thought of Different architecture called "RNN".

Applications of RNN's

1. NLP:
   i) language modelling
   ii) machine translation
   iii) sentiment analysis
   iv) named entity recognition
   v) Text generation
2. Speech Recognition (speech to text)
3. Time series Analysis
4. Image captioning
5. Anomaly detection in text (like behaviour)

etc so ... on.

RNN's :- Rnn's are the special type of Neural networks which has memory feature. That is why they work great on "sequential Data".

Data for RNN :-

format ⇒ (Timesteps (stamps), input-features)

movie = [1,0,0,0,0]

was = [0,1,0,0,0]

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

No. of unique words = 5

good = [0,0,0,1,0] .. so on.

Now Review-1 can be written as

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

R-1 ⇒ [Timestamps, input features]

next t = 1sec for movie to go into archi

t = 2sec for was

t = 3sec for movie all.

∴ R-1 ⇒ [3, 5] ↳ no. of unique features

Similarly R-2 ⇒ [3, 5]

R-3 ⇒ [4, 5]

* In keras → simpleRNN →

(batch size, timestamps, input-features)

For above ex:- (3, 4, 5)

- 3 → As we have 3 Reviews
- 4 → max no of timestamps among the Reviews
- 5 → input-unique-words

How it WORKS?

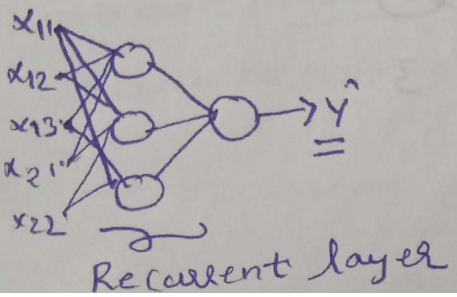
Above example: Reviews

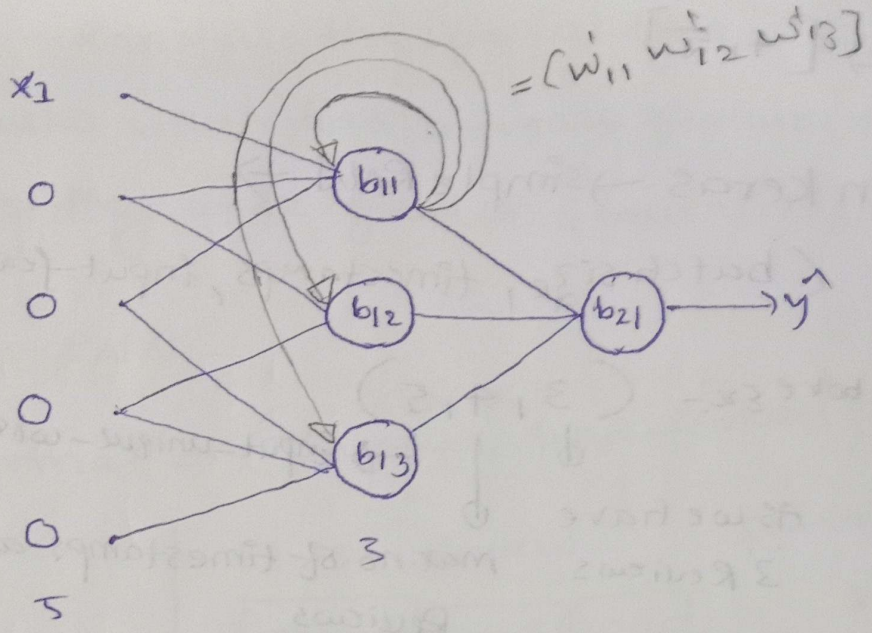
					sentiment
$R_1 \Rightarrow$	movie	was	good		1
	x_{11}	x_{12}	x_{13}		
$R_2 \Rightarrow$	movie	was	bad		0
	x_{21}	x_{23}	x_{33}		
$R_3 \Rightarrow$	movie	was	not	good	0
	x_{31}	x_{32}	x_{33}	x_{34}	

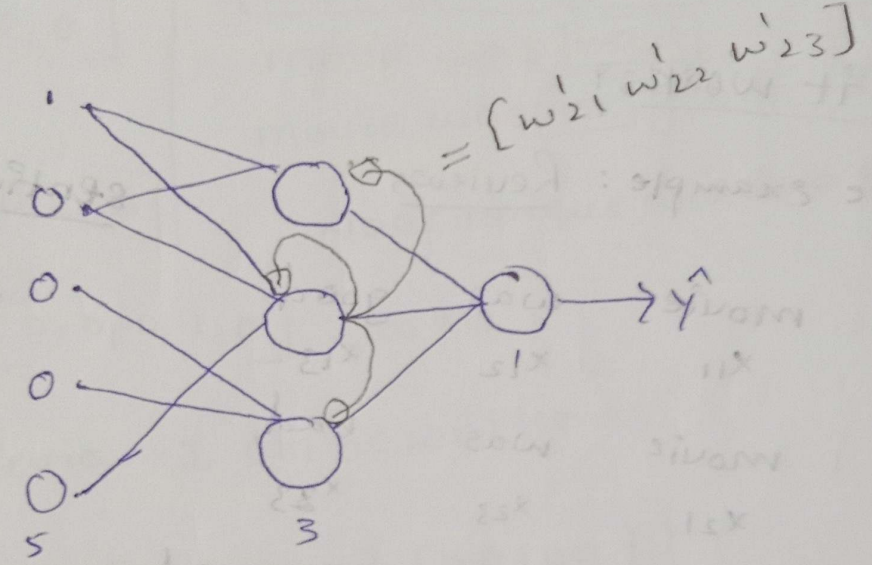
$x_{11} = [10000]$ $x_{12} = [01000]$

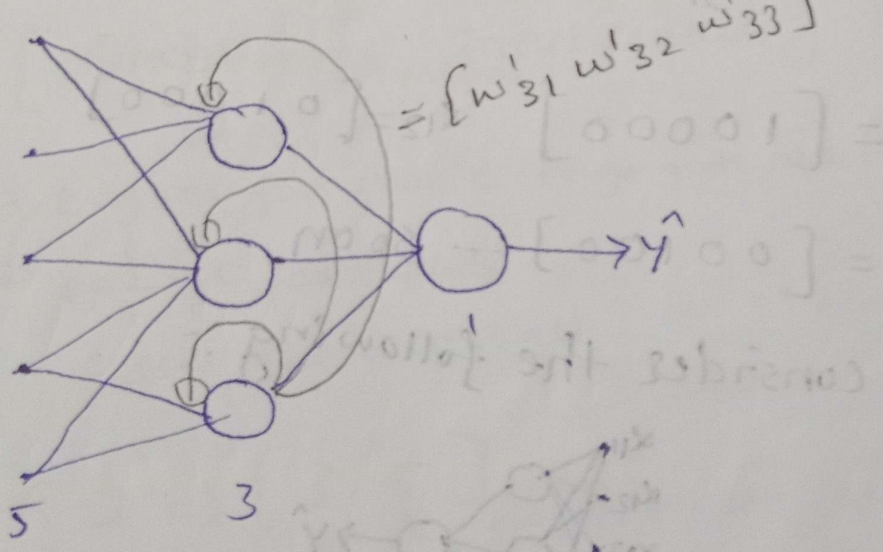
$x_{13} = [00100]$ --- soon.

Lets consider the following:-




x_1
0
0
0
0
5
b_{11}
b_{12}
b_{13}
3
b_{21}
$\hat{y}$
$= [w'_{11}\ w'_{12}\ w'_{13}]$


1
0
0
0
0
5
3
1
$\hat{y}$
$= [w'_{21}\ w'_{22}\ w'_{23}]$


5
3
1
$\hat{y}$
$= [w'_{31}\ w'_{32}\ w'_{33}]$

Total trainable parameters =

$W_1 = 5 \times 3 = 15$ weights

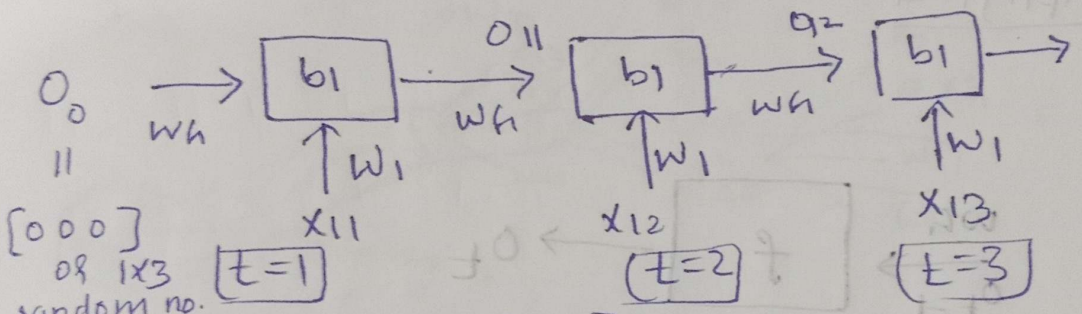
$W_h = 3 \times 3 = 9$ weights hidden (Recurrent layer)

$W_2 = 3 \times 1 = 3$ weights

27 weights

biases = 4 $\therefore$ **T. = 31** trainable parameters

Forward propagation :-



$X_{11} = [1 \times 5]$; $W_1 = [5 \times 3]$

$O_{11} = f(W_1 X_{11} + b_1 + W_h O_0) = (1 \times 3)$

$O_{12} = f(W_1 X_{12} + O_1 W_h + b_1) = (1 \times 3)$

$O_{13} = f(W_1 X_{13} + O_2 W_h + b_1) = (1 \times 3)$

Now $W_2 = [3 \times 1]$

$O_{21} = \therefore Y = f(W_2 \times O_3 + b_2) = [1 \times 1] =$ scaler

In RNN $\Rightarrow$ At every time stamp we send the current input and previous output for the next network

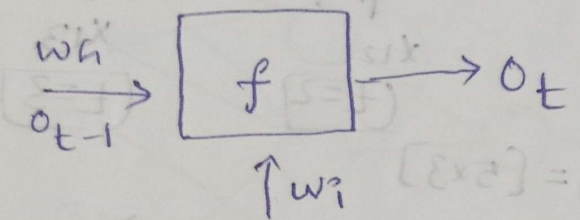
Recurring (or) Reusing it as feedback for entire sentence

That is why we call Recurrent NN.

* we are using the same weights and baias for complete sentence forward prediction; That implies we using the concept of parameter sharing or weight sharing.

* RNN is actually Remembering the previous words so that the sequence or the context of sentence is maintained.

simplified!



i = row

t = timestamp

$$o_t = f(w_i x_{it} + w_h o_{t-1} + b_1)$$

If t is last time stamp then

$$g(o_t w_o + b_2) = \text{scaler} = y^{\wedge}$$

g = sigmoid or softmax or linear

Backpropagation:

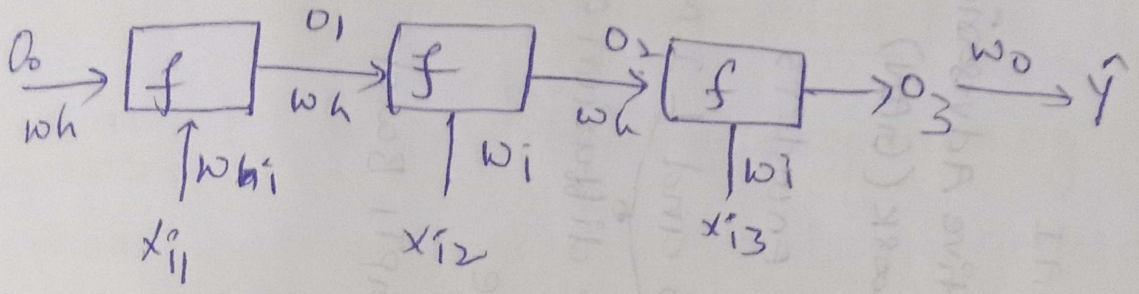
$$o_1 = f(x_{i1} w_i + o_0 w_h)$$

$$o_2 = f(x_{i2} w_i + o_1 w_h)$$

$$o_3 = f(x_{i3} w_i + o_2 w_h)$$

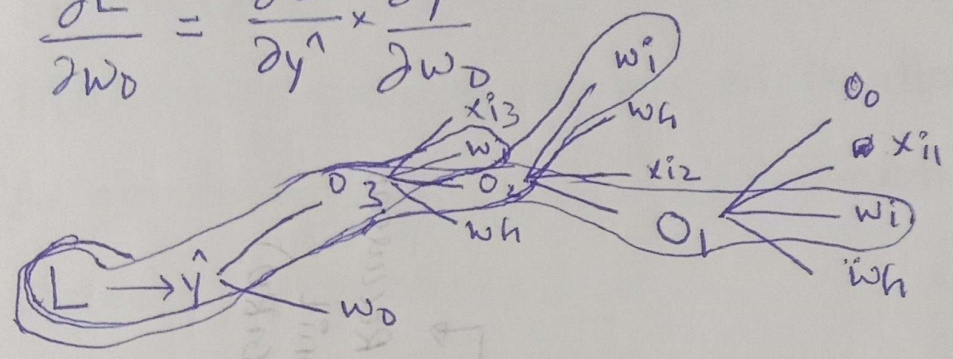
Aim: $\frac{\partial L}{\partial w_o}, \frac{\partial L}{\partial w_h}, \frac{\partial L}{\partial w_i}$

$$y^{\wedge} = \sigma(o_3 \cdot w_o)$$



$$L = -y_i \log \hat{y_i} - (1-y_i) \log (1-\hat{y_i})$$

$$\frac{\partial L}{\partial w_o} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_o}$$



$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial O_2} \cdot \frac{\partial O_2}{\partial w_i}$$

$$+ \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial O_2} \times \frac{\partial O_2}{\partial O_1} \times \frac{\partial O_1}{\partial w_i}$$

$$\Rightarrow \frac{\partial L}{\partial w_i} = \sum_{j=1}^{3 \text{ (As we took 3 words)}} \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_j} \times \frac{\partial O_j}{\partial w_i}$$

$$\text{If } \frac{\partial L}{\partial w_h} = \sum_{j=1}^{3} \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_j} \cdot \frac{\partial O_j}{\partial w_h}$$

$$\therefore \frac{\partial L}{\partial w_i} \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_j} \cdot \frac{\partial O_j}{\partial w_i}$$

n = no. of time stamps

Problems with RNN

i) Long term dependencies caused by vanishing gradient problem

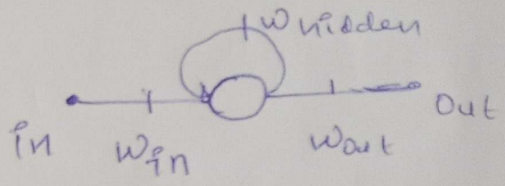
ii) Stagnated Training ⇒ reason ⇒ exploding gradient problem

#problem 1:- Vanishing Gradient descent

For ex:

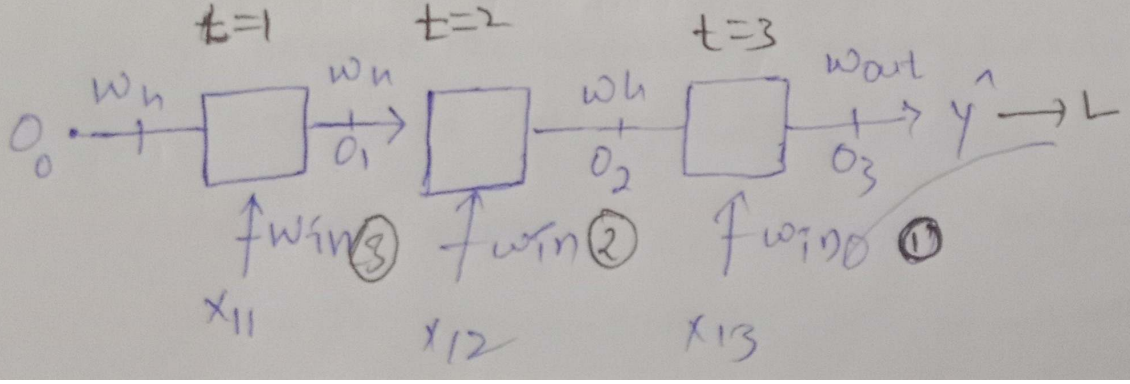
Input	output
1 0 1	1
0 0 1	0
0 0 0	0
1 1 1	1

Simple RNN:-



$$x = [1, 0, 1]_{1\times 3}$$

remember we have only one neuron in rnn, but 3 input vectors are there, so we write 3 times at different time stamps.



$$w_{in} = w_{in} - \frac{\partial L}{\partial w_{in}}$$

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

$$w_{out} = w_{out} - \frac{\partial L}{\partial w_{out}}$$

we apply chain rule:-

$$\frac{\partial L}{\partial w_{in}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial w_{in}} +$$

① short term dependencie

②

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_{in}}$$

③

[see diagram in front page]

Long term dependencie

we got it for only

3 time stamps.

what if 100 time stamps !, It will look like this

$$\Rightarrow \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \frac{\partial o_{100}}{\partial o_{99}} \cdots\cdots \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_{in}}$$ (A)

Imagine how much small it becomes!

(A) $\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \left[\frac{\partial o_{100}}{\partial o_{99}} \cdots \frac{\partial o_2}{\partial o_1} \right] \frac{\partial o_1}{\partial w_{in}} =$

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \left[\prod_{t=2}^{100} \left(\frac{\partial o_t}{\partial o_{t-1}} \right) \right] \frac{\partial o_1}{\partial w_{in}}$$

$$o_1 = \tanh\left(x_{1,1} w_{in} + 0 \cdot w_h \right)$$

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

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

Depends on o_t.

Range : 0 - 1

$$(A) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \left[\prod_{t=2}^{100} \left(\tanh'\left(x_{1,t} w_{in} + o_{t-1} w_h \right) w_h \right) \right] \frac{\partial o_1}{\partial w_{in}}$$

0-1 0-1

Assume you have given w_h : 0 to 1

Image Imagine how small it will be

Gradient is Vanishing $\simeq 0$

$\therefore (A) \simeq 0$

∴ The change in L w.r.t W_{in} will be only contributed by short term dependencies

Solution:

1) using different activation function instead of tanh (Range: 0 to 1) like Relu / leaky relu

2) Better weight intialization (not from 0-1)

3) Skip Rnns

4) LSTMS

#problem 2: Exploding Gradient descent

If we are using "tanh" and intializing weights from 0-1 then we are getting vanishing gradient problem.

Assume you have used Relu and intialized weights greater than 1

Then (A) will dominate the short term dependencies. I mean

$$\frac{\partial L}{\partial w_{in}} = \frac{\partial L}{\partial y^1}\frac{\partial y^1}{\partial o_3}\frac{\partial o_3}{\partial w_{in}} + \frac{\partial L}{\partial y^1}\frac{\partial y^1}{\partial o_3}\frac{\partial o_3}{\partial o_2}\frac{\partial o_2}{\partial w_{in}}$$

$$+ \frac{\partial L}{\partial y^1}\frac{\partial y^1}{\partial o_3}\frac{\partial o_3}{\partial o_2}\frac{\partial o_2}{\partial w_{in0,1}}\frac{\partial o_1}{\partial w_{in}}$$

(A) — These will become neglig... compared to (I)

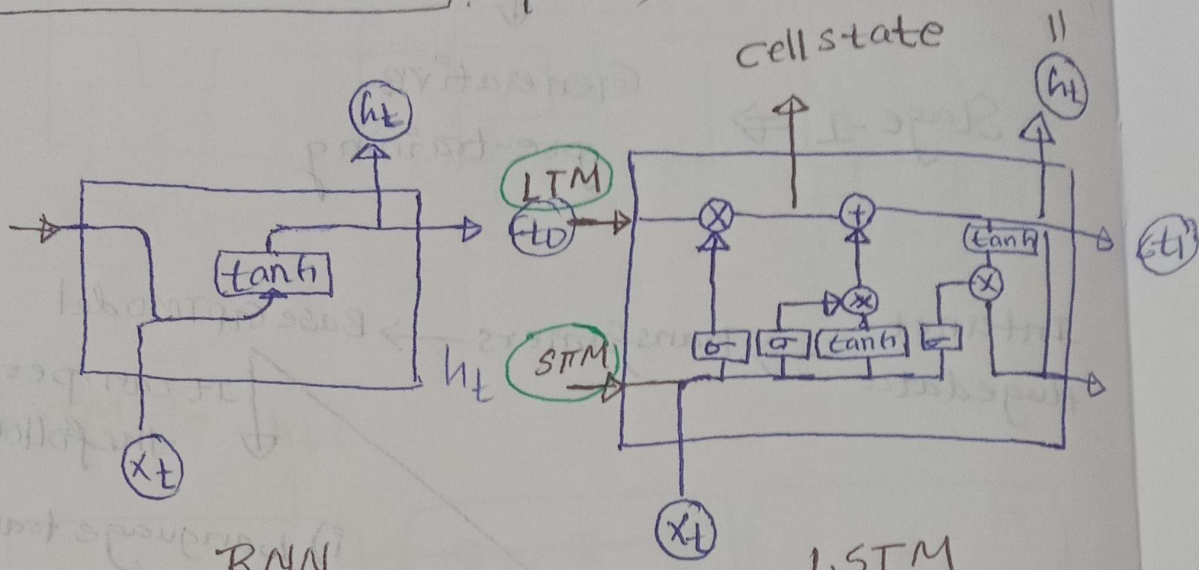
(I) become very very large number if we take 100 time stamps!

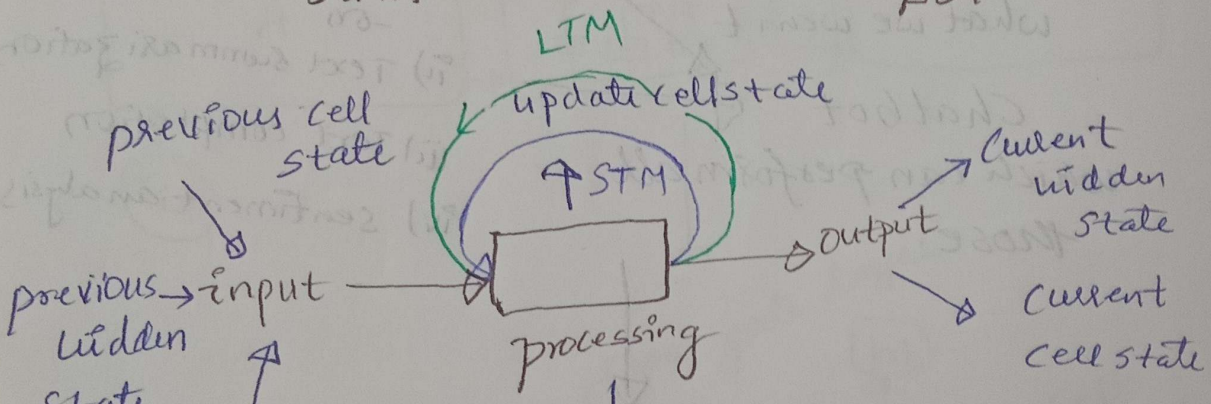
Solutions:

1) Gradient clipping
2) Controlled learning rate
3) LSTMs

LSTM's : Long Short Term Memory

- The what
- The architecture :- (How)

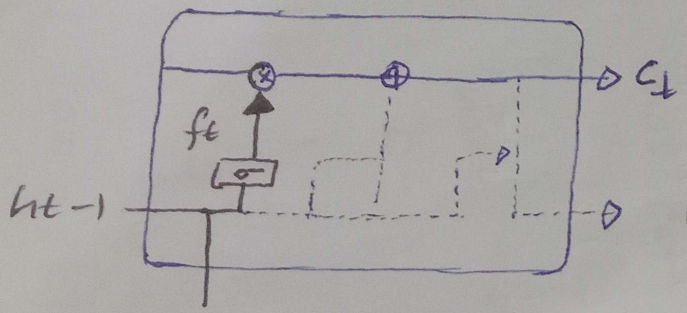




Basically 3 steps involved in

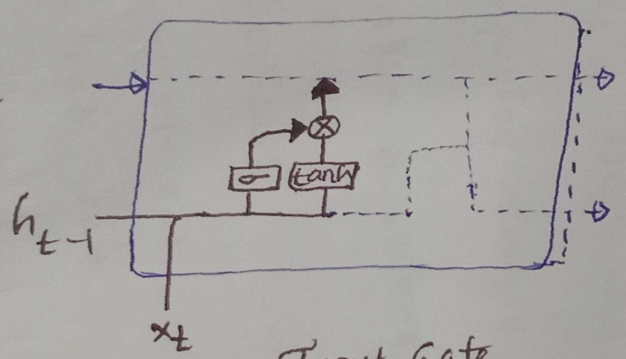
Core Idea: In RNNs we have only one path (h_t) which captures the short term dependencies but in LSTM scientists added another path cell state which also store Long term dependencies (LTM

Gates



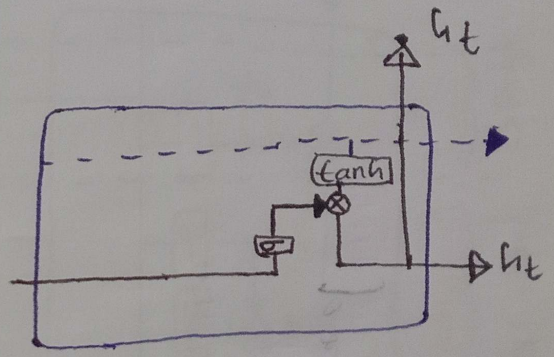
forget gate

To remove something from C_t



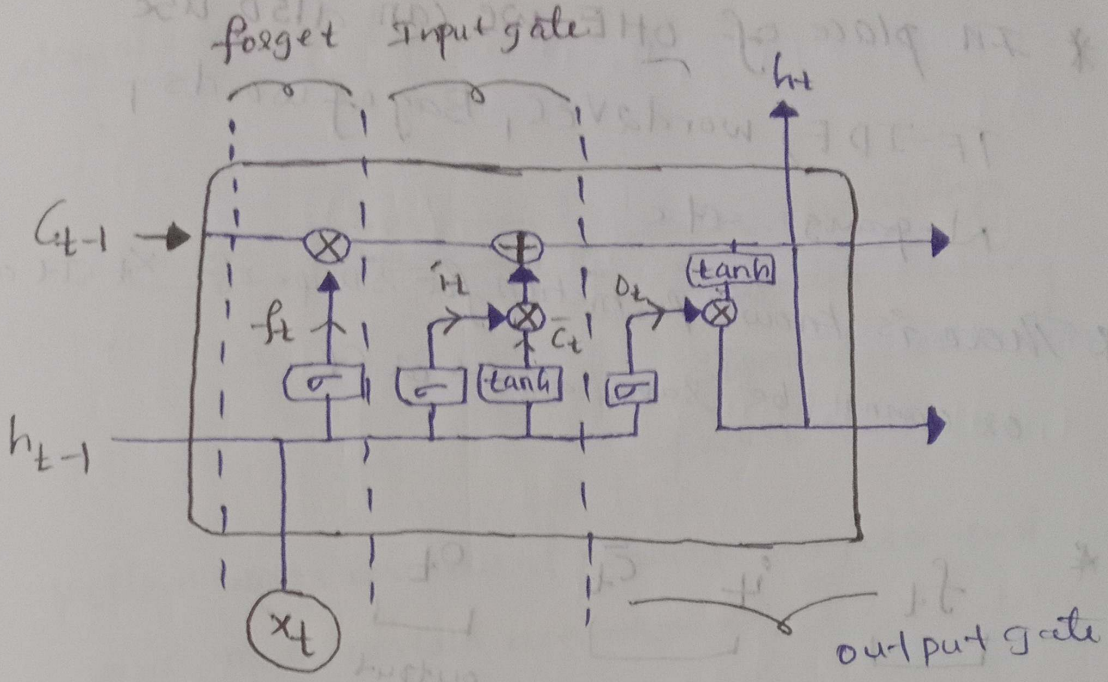
Input Gate

To add something to C_t



output gate

To calculate hidden state h_t



- C_{t-1} & h_{t-1} both are vectors of same dimension always. Ily C_t & h_t
- x_t is also a vector like:

Reviews	Sentiment
cat mat rat	0
cat rat rat	0
mat mat cat	1

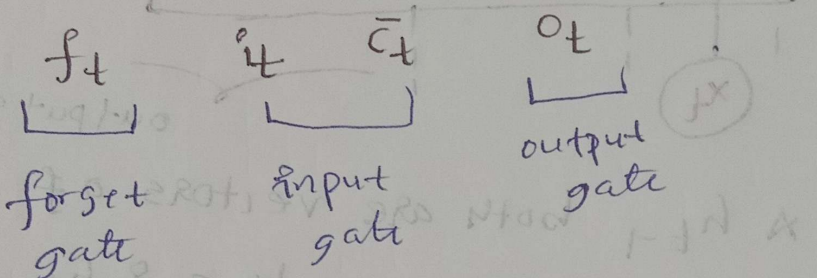
one

cat	mat	rat
1	0	0
0	1	0
0	0	1

⇒ The word 'cat' = [1 0 0]

* In place of OHE we can also use TF-IDF, word2vec, Bag of words, N-grams. etc.

* There is know restriction for shape of x_t. It can or cannot be same as h_t or C_t.



* f_t — forget gate; i_t, $\bar{C}_t$ — input gate; o_t — output gate

All the above are vectors

$\Rightarrow$ The shape is same as C_t $\rightarrow$ h_t

pointwise operations

$\otimes \rightarrow$

Ex:- $C_{t-1} = [4 \;\; 5 \;\; 6]$

$f_t = [1 \;\; 2 \;\; 3]$

$C_{t-1} \otimes f_t \Rightarrow$ vector

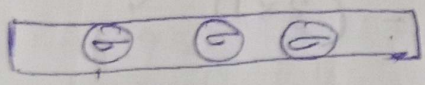
$= [4 \;\; 10 \;\; 18]$

$\oplus \rightarrow$

tanh $\rightarrow$ [Tanh(4), Tanh(5), Tanh(6)]

C_{t-1} saf $= [0.12 \;\; 0.36 \;\; 0.21]$

Neural Network layer

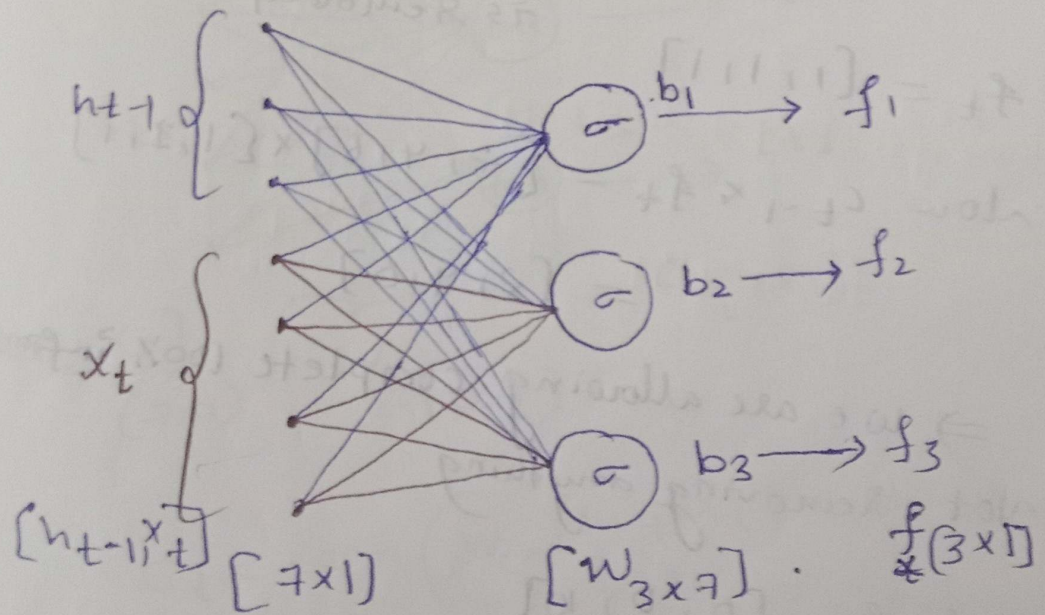


→ layer containing nodes having sigmoid activation functions.

* Yellow boxes are neural network layers and each node contain activation fn.

The Forgate :-

Let suppose x_t is 4 dimensional vector and h_{t-1} is a 3 dimensional vector



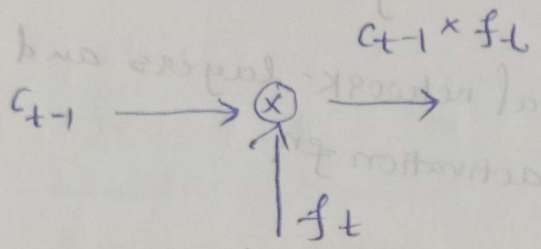
$$\Rightarrow (3\times7)(7\times1) = (3\times1)$$

$$f_t = [f_1\ f_2\ f_3]$$

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

Now $C_{t-1} \times f_t = (3\times1) \times (3\times1)$

Intution behind word "Removal" used for ⊗



i) Let's say $C_{t-1} = (2, 4, 6)$

and $f_t = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$

Now $C_{t-1} \times f_t = (2, 4, 6) \times \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$

$= (1, 2, 3) \Rightarrow$ only 50% is removed

ii) $f_t = [1, 1, 1]$

Now $C_{t-1} \times f_t = [2, 4, 6] \times [1, 1, 1]$

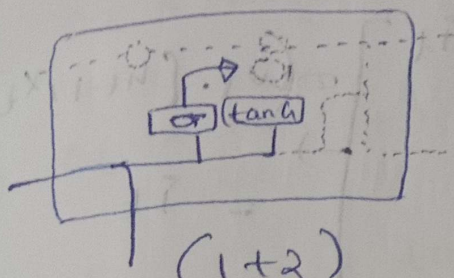
$= [2, 4, 6]$

$\Rightarrow$ we are allowing complete 100% information or not removing anything

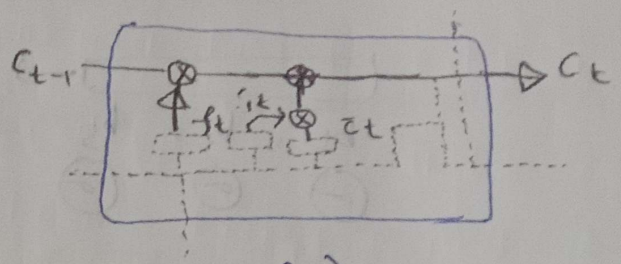
iii) If $f_t = [0, 0, 0]$

Now $f_{t-1} \times f_t = [0, 0, 0] \Rightarrow$ 100% information is removed.

The Input gate :- To Add new imp info to C_t



(1+2)



(3)

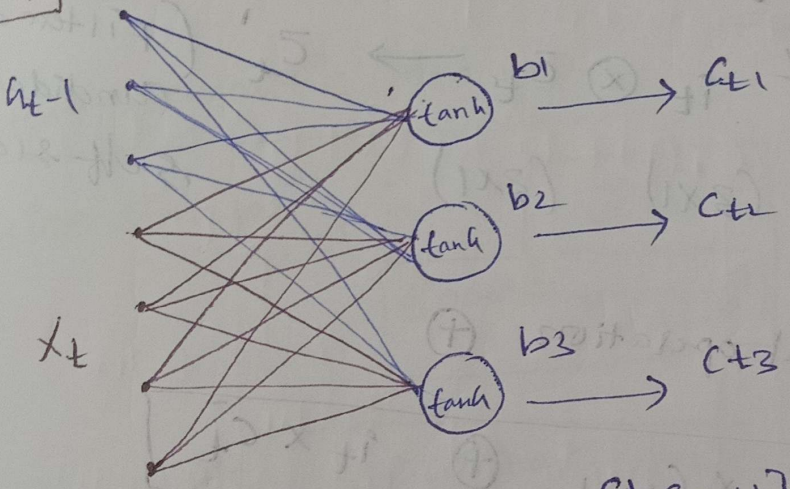
stages Cell

i) $\bar{C}_t$ candidate state

ii) $i_t \Rightarrow$ It is basically filtering

iii) updatating cell state

1+2



C_t (3×1)

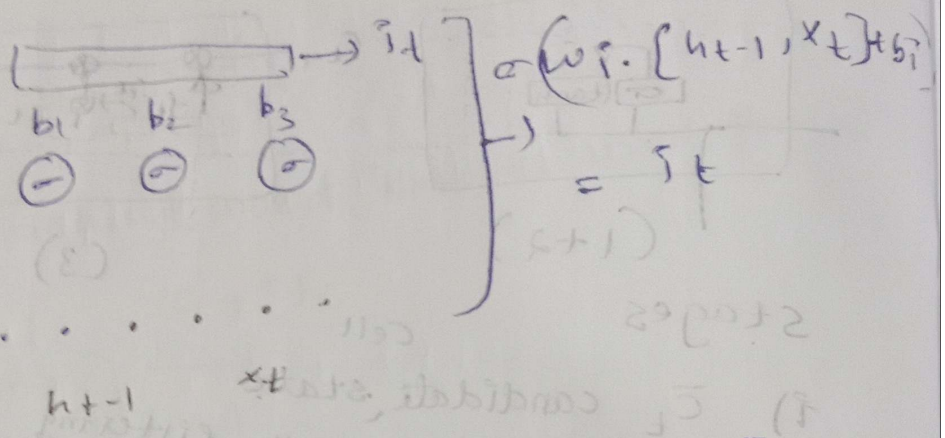
$[h_{t-1}, x_t]$ (7×1) $\quad W_c$ (3×7)

$$\therefore \bar{C}_t = \tanh\left(\underset{3\times 7}{W_c} \cdot \underset{7\times 1}{[h_{t-1}, x_t]} + \underset{3\times 1}{b_c}\right)$$

$\Rightarrow_{3\times 1}$

$\therefore \bar{C}_t$ $_{3\times 1}$

Same for it also



Here it acts as Filter for $\bar{c}_t$

3 - focusing on 3rd stage

pointwise: $i_t \otimes \bar{c}_t \longrightarrow \bar{c}_t$ (Filtered candidate self state)

$(3 \times 1) \quad (3 \times 1)$

Now final operation $\oplus$

$$c_t = f_t \times c_{t-1} \oplus i_t \times \bar{c}_t$$

The output gate:-

goal:- $c_t \rightarrow h_t$

i) $\tanh(c_t) \rightarrow [-1\ 1] \Rightarrow$ Now we apply filteration

ii) For filteration we require a vector called o_t

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

iii) $o_t \times \tan(c_t)$ - pointwise multiplication

$$h_t = o_t \times \tanh(c_t)$$

(3×1) (3×1) - element wise multiplication

Word2Vec

queen - girl + boy = king

*

Latent semantic Analysis

cosine similarity :-

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|}$$

$$|\vec{a}| = \sqrt{a_1^2 + a_2^2 + a_3^2 \cdots a_n^2}$$

$$|\vec{b}| = \sqrt{b_1^2 + b_2^2 + b_3^2 + \cdots b_n^2}$$

Topic modelling :-

* The process to find the topics from documents in an unsupervised manner.
* Text mining approach to find recurring patterns

the text documents. (Nothing but a row)

* Algorithms or techniques for topic modelling:

1. LDA - Latent Dirichlet Allocation
2. NNMF - Non-Negative Matrix Factorization
3. LSA - Latent Semantic Allocation.

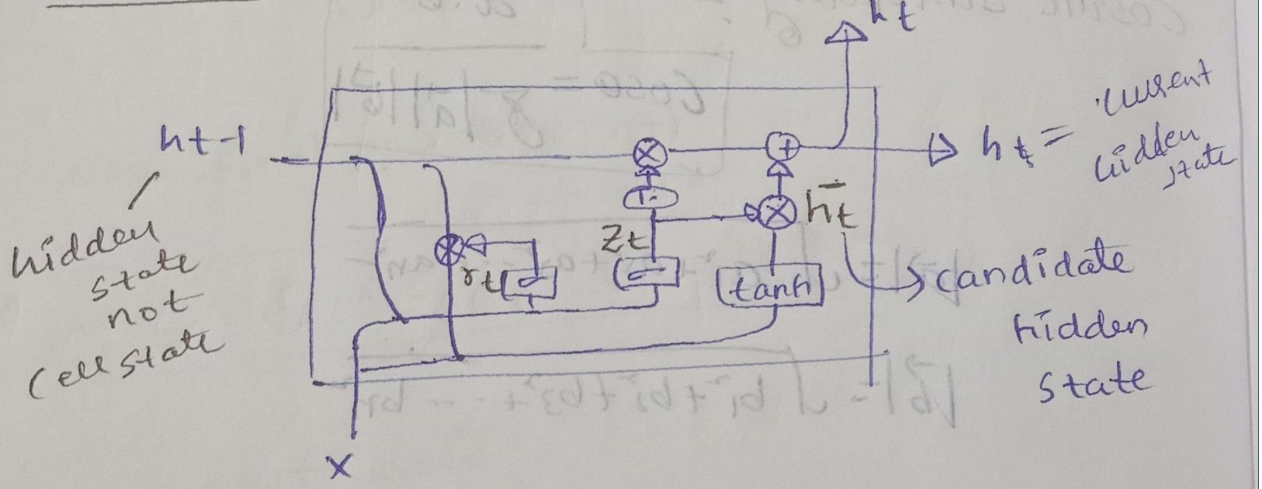
Aim of LDA:

i) what are the most important topics
ii) what are the topics assigned to each document

OR

i) what is the topic to the term distribution (Nothing but finding how many topics are there like food, animals)
ii) Document to topic distribution. [Nothing but annotating words in document with topics].

GRU's :- GATED RECURRENT UNIT



Why?

* It is less complex than LSTM's
* ⇒ It's training time will be less than LSTM's
* GRU ≅ LSTM's
* only two gates (Reset gate & update gate)

* we manipulate the hidden state so that it can carry both long term memory & short term memory.

Goal: h_{t-1}, x_t → h_t current hidden state (?)

→ h_{t-1}, h_t, r_t, x_t, z_t, $\bar{h}_t$ → All are vectors and except x_t all these dimension is same.

→ σ tanh ⇒ These are neural networks with activation fn sigmoid & tanh

σ → ○ ○ ○ ○

Rem: No. of Neurons will same in σ & tanh

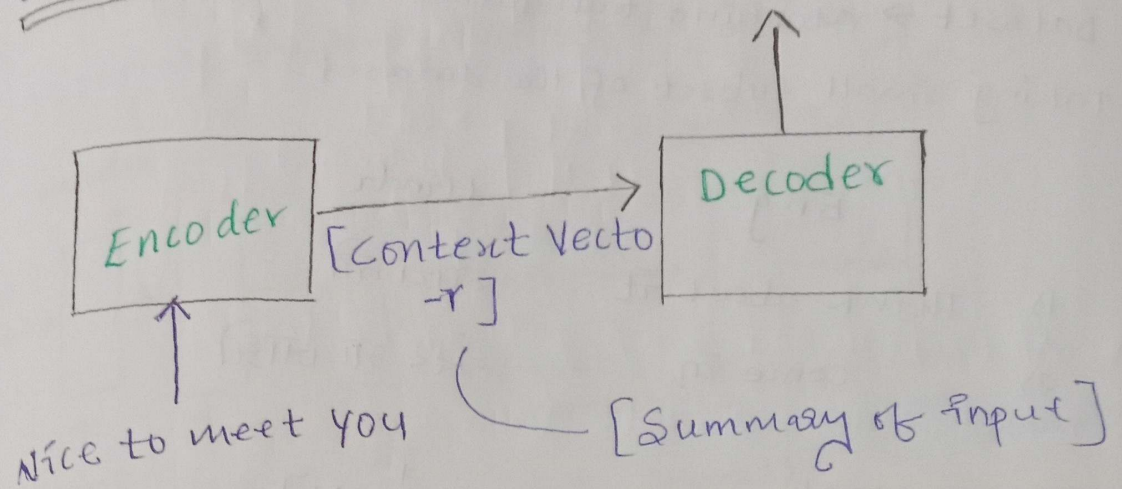
Number of training steps =

$$\left(\frac{\text{No. of rows (or) examples}}{\text{Batch size}}\right) \times \text{Number of epochs}$$

EDA - Encoder Decoder Architecture

overview:-

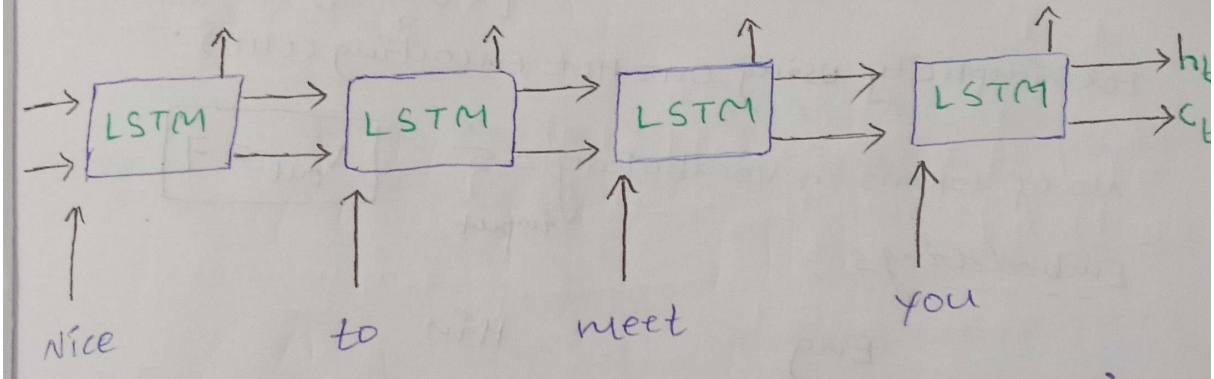
आप से मिल कर अच्छा लगा



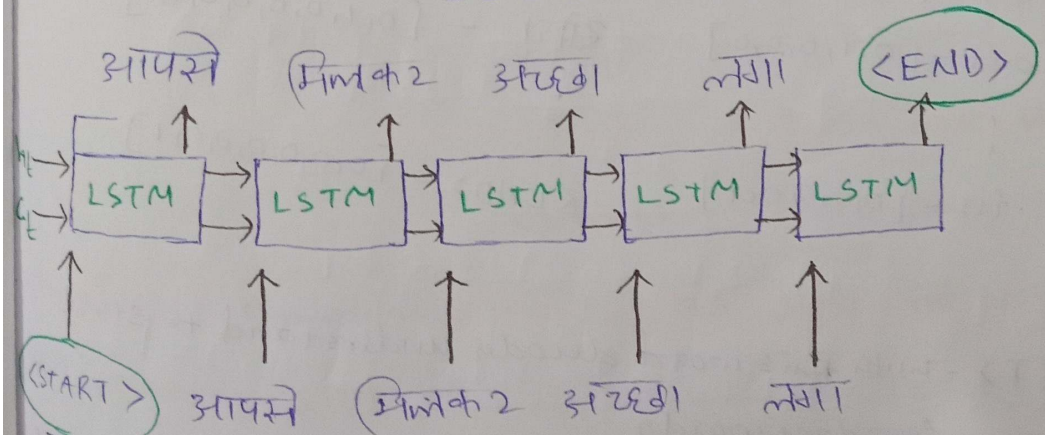
What's under the hood?

Encoder:

These outputs are ignored



Decoder:



Considering Machine translation task

Training the architecture using Backpropagation

Dataset → Machine translation

* Taking small subset of the dataset

	Eng	Hindi
1)	Think about it	सोच तो
2)	come in	अंदर आ जाओ

Tokenization:-

[think, about, it] | [<START>]
[come, in] | [सोच, तो]
| [अंदर, आ, जाओ]
| [<END>]

For simplicity using One Hot Encoding (OHE):

No. of tokens in vocabulary = 5 (V_{input}) | $V_{out} = 7$

Embeddings :-

Eng	Hin
think - [1,0,0,0,0]	<START> - [1,0,0,0,0,0,0]
about - [0,1,0,0,0]	सोच - [0,1,0,0,0,0,0]
⋮	⋮
in - [0,0,0,0,1]	<END> - [0,0,0,0,0,0,1]

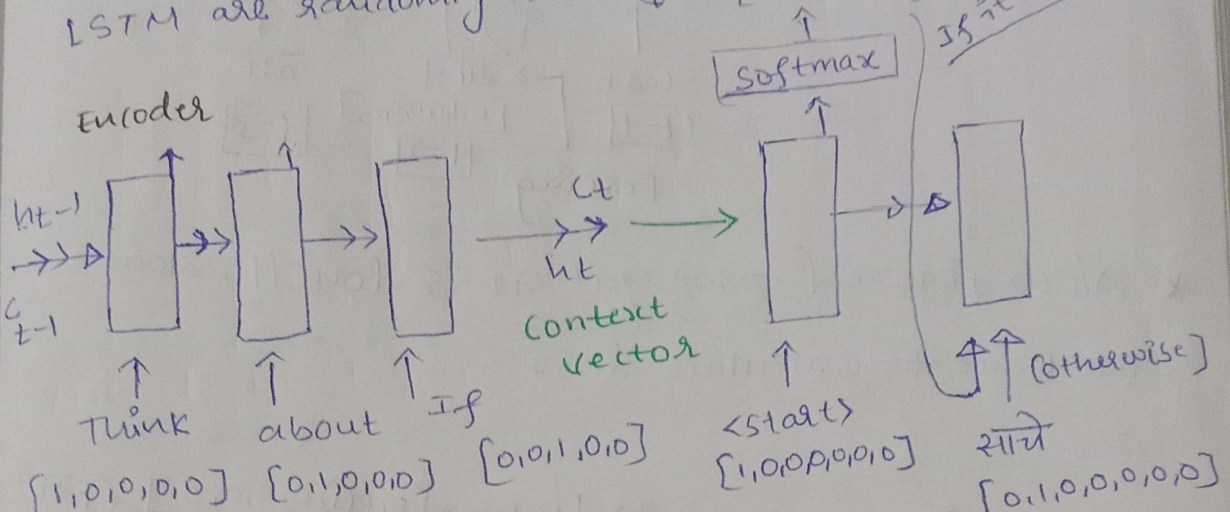
<START> - with this word decoder understand to start decoding words

<END> - After decoding all the words, decoder will through 'END' word

Training:-

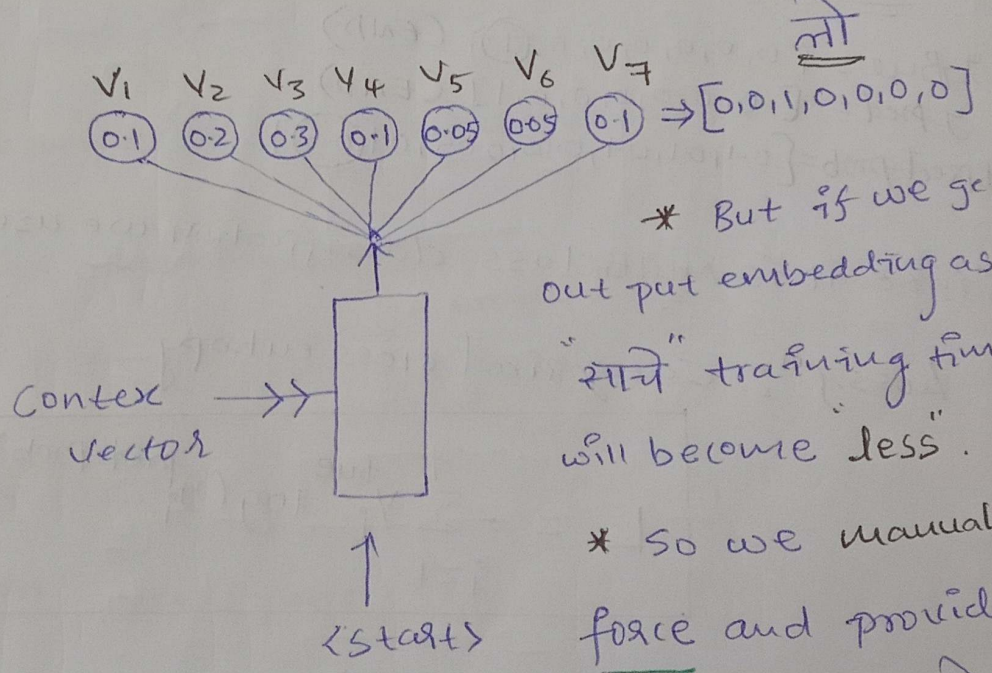
Taking row-1 : [Think about it → सोच तो]

* Initially as we know weights and biases are of LSTM are randomly intialized [embedding]



Softmax → It contains neurons, No. of Neurons = V_{out}

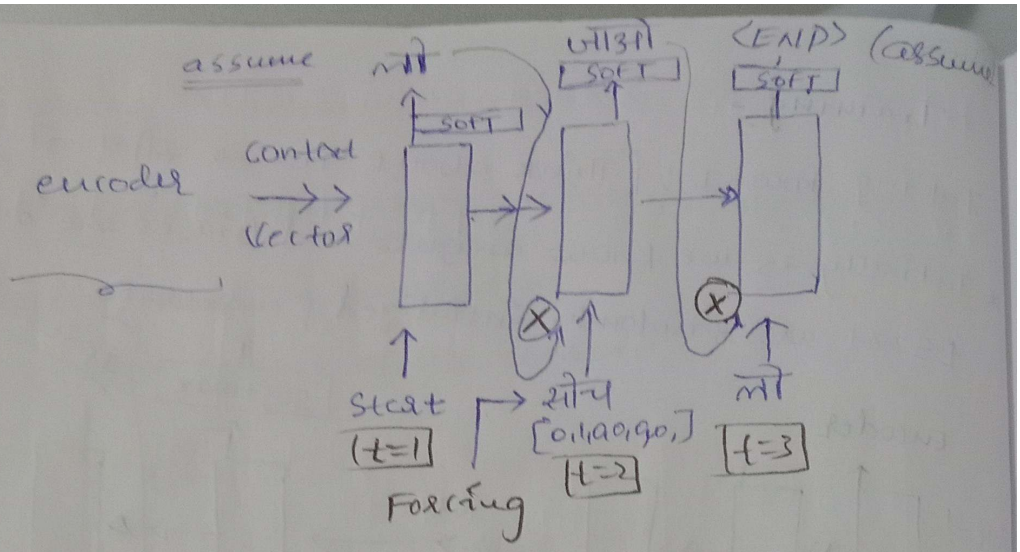
For Eg:-



* But if we get out put embedding as "साचे" training time will become less.

* So we manually force and provide the input embedding vector as सोच instead तो, for next lstm, this is known as Teacher forcing

⇒ This makes the convergence fast.



* Now forward propagation is done!! Now we have to calculate loss.

y_true = [0 1 0,0,0,0,0,0] (सोचे)
y_pred = [0,0,1,0,0,0,0,0] (तो)
y_pred_prob = [0,1 0.2 0.3,0.1,0.05,0.05,0.1]

t=1

y_true = [0,0 1 0,0,0,0,0] (तो)
y_pred = [0,0,0,0,0,0,0.1,0] (जाऊंगी)
y_pred_prob = [0.1,0.05 0.05 0.1,0.2,0.4,0.1]

t=2

y_true = [0,0,0,0,0,0,0,1] (END)
y_pred = [0,0,0,0,0,0,0,1] (END)
y_pred_prob = [0.1,0.1,0.1,0.05,0.05,0.1,0.5]

Since it is Multiclass classification we use

Loss fn: Categorical cross entropy

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

$L_{t=1} = (-1)(0.2)\log(0.2) = 0.69$

$L_{t=2} = (-)(1)(0.05) = (-)(1)\log(0.05) = 1.3$

$L_{t=3} = (-1)(0.5) = (-)(1)\log(0.5) = 0.3$

$L_T = 2.29$

* Now we know loss, what next ?

Backpropagation :-

Two steps :-

1) calculating gradient

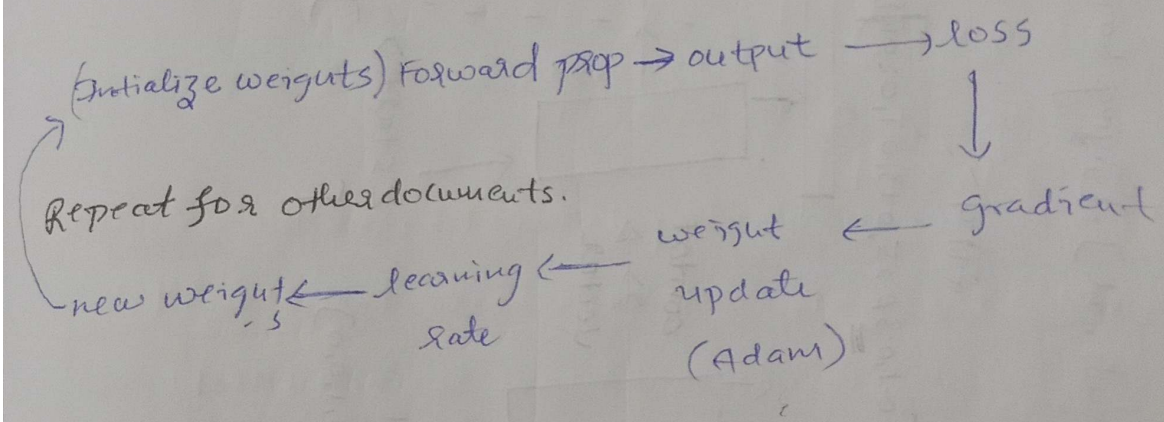
2) update parameters

we have to take the gradient of all the weights & biases of LSTM & SOFT MAX Layer w.r.t LOSS

then we update them using adam or rms-prop

.. etc optimizers.

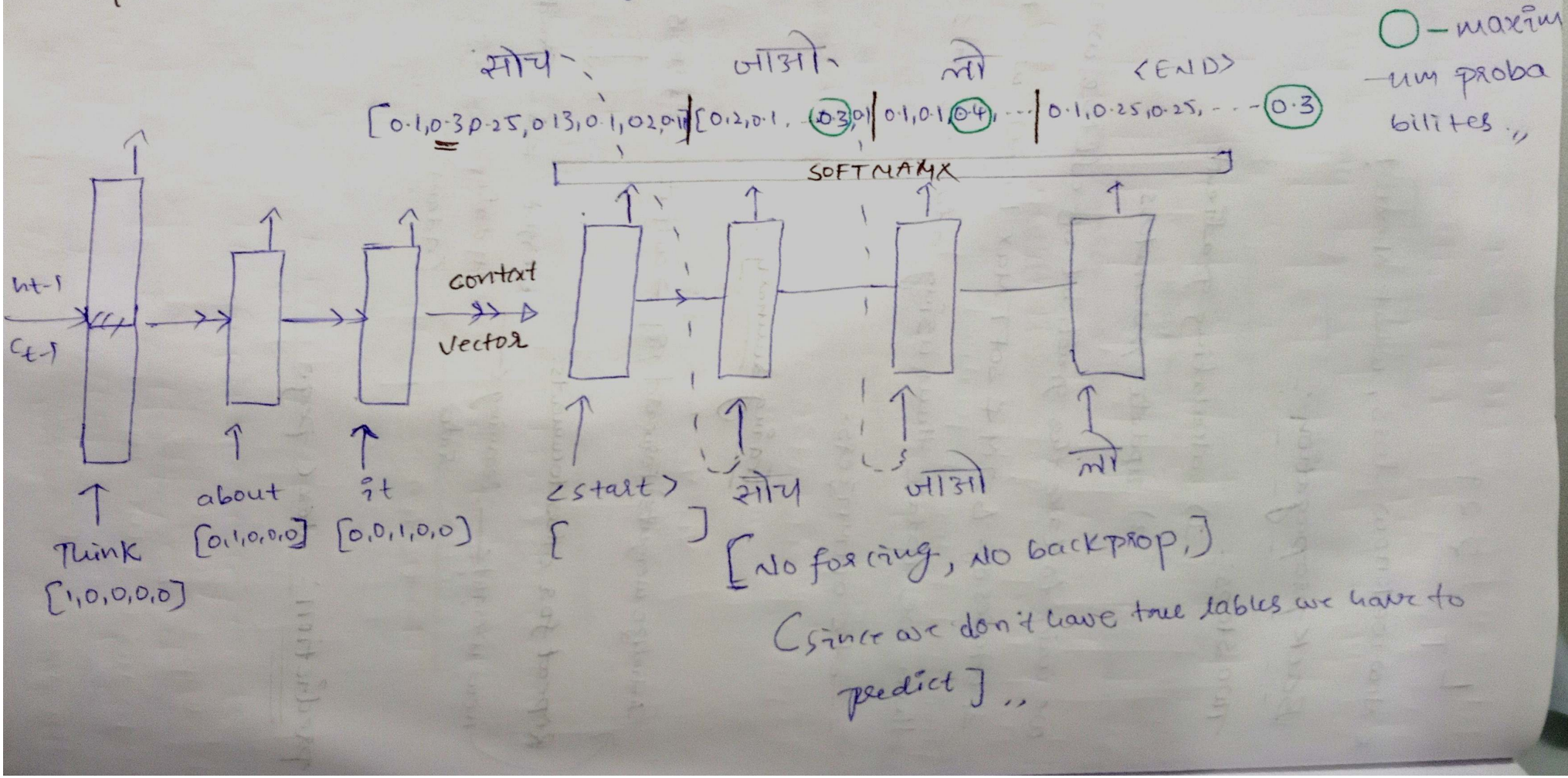
Traing Summary :



prediction : next page.

prediction: <start> → ① सोच ② लो ③, सं द २ → ④, आ ⑤, जाओ → ⑥, <end> - ⑦

input: [Think about, it] - [giving having data document itself]



[No forcing, No backprop,)

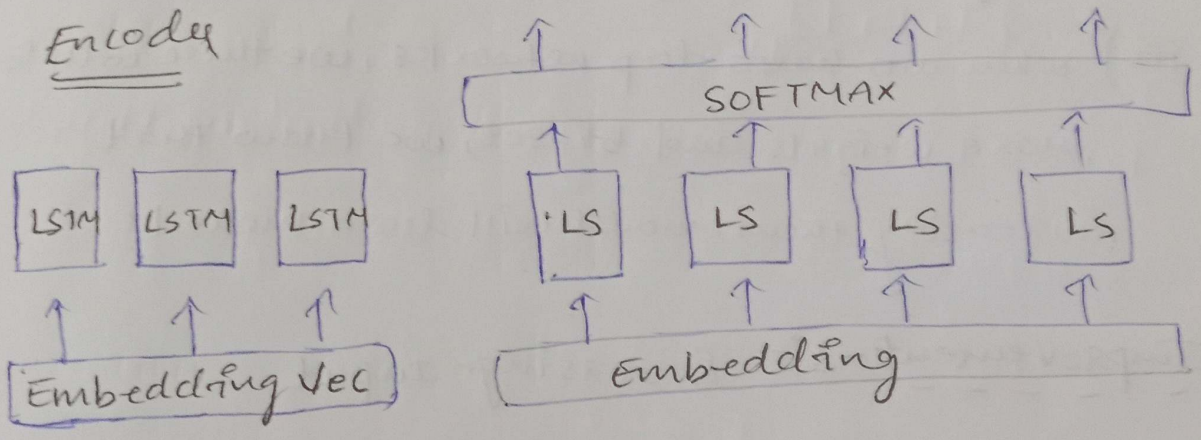
(since we don't have true labels we have to predict]..

Improvement-1: Embeddings

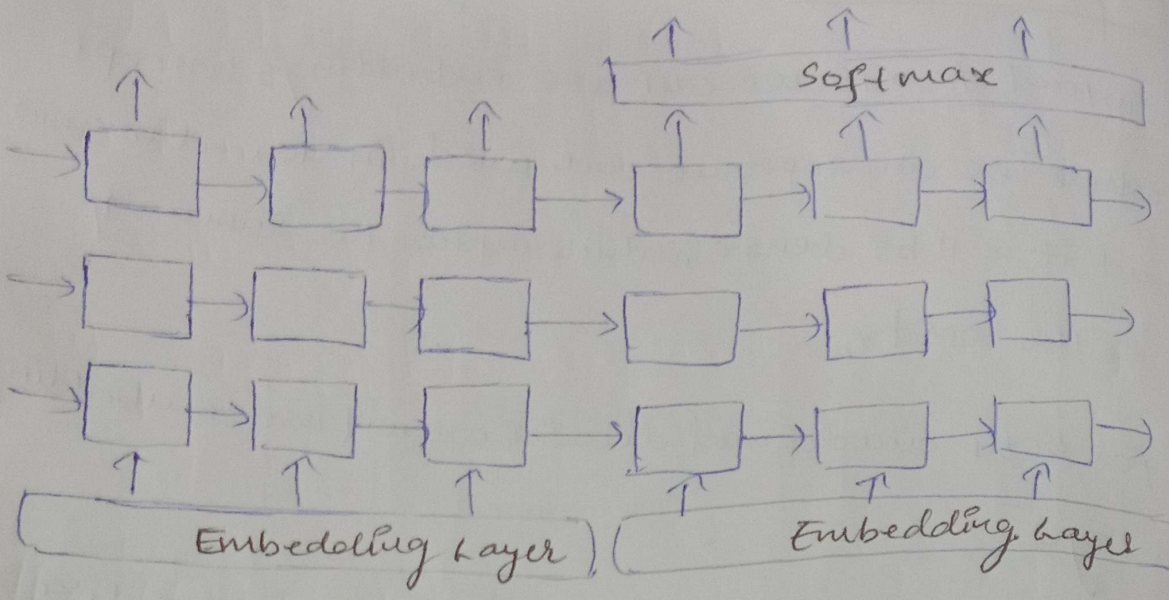
* Instead of OHE we can use embeddings concept where we can represent each word in low-dimension and it will be dense, which capture the summary of the word."
* For both encoder and decoder we will use embedding layer.
* We can use pretrained embedding techniques like word2vec, glove or custom embedding on training the data

Final architecture :-

Screen shot



Improvement 2- Deep LSTMS



3 Benefits

i) Long term dependencies

ii) layered representation:

layer-3 :- paragraph

layer-2 ⇒ sentences

layer-1 ⇒ lower words

iii) when we have deep networks, we there will be more weights and biases, we know that increasing them, model will learn more!!

Improvement-3:- Reversing Input

* when we reverse the input the starting words will be close to the corresponding output

* I.t. will not work always, there are some language pairs, there we like english-french, ..etc, It will work

where initial words should have more context than last words!