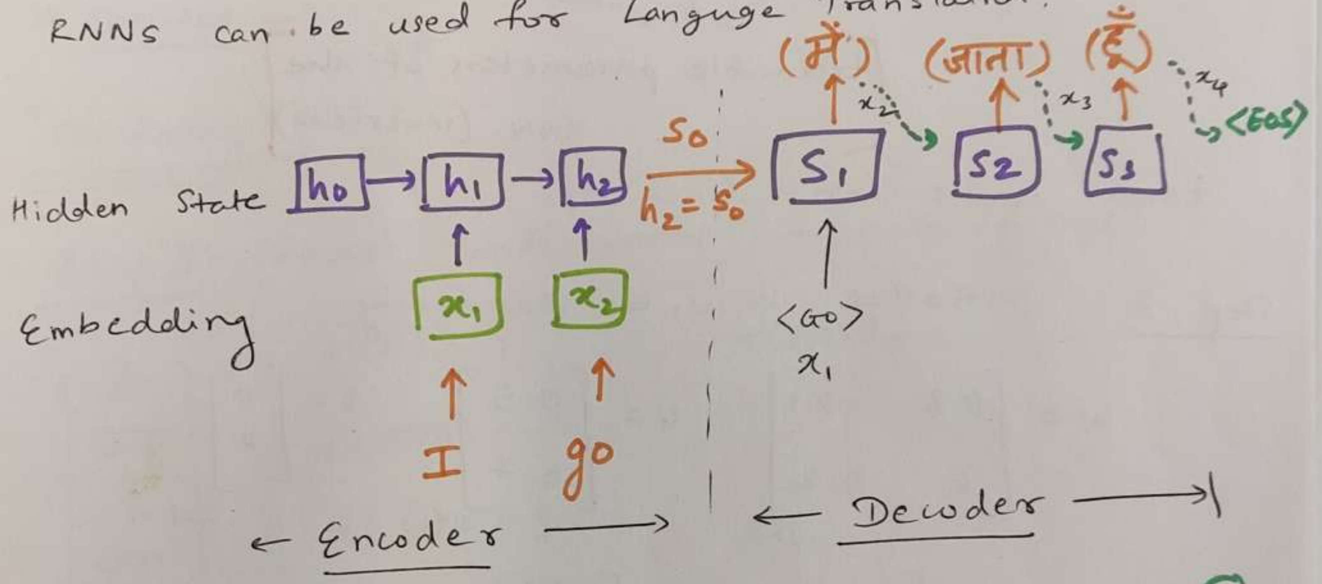
RNNs

Lecture No. 3

"Build RNN from scratch"

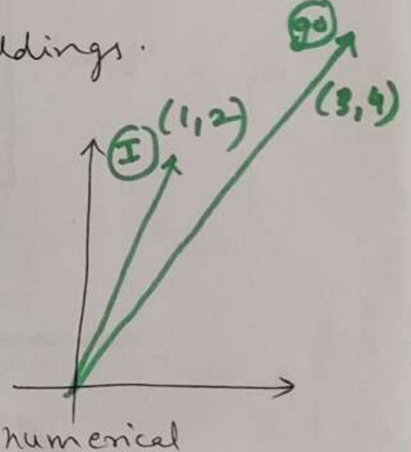
RNNS can be used for Langruge Translation.



Step 1 : convert I/P words into embeddings.

$$x(I) = x_1 = 1$$

$$x(go) = x_2 = 2$$



numerical representation.

assume every word is in 1D.

Step 2 : Decide hidden state size.

assume hidden state size $= 2$

$$h_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{2\times1}$$

Mathematical relation b/w hidden states;

$$\underset{(2\times1)}{h_1} = \tanh(W \cdot h_0 + U \cdot x_1 + b)$$

$$\underset{(2\times1)}{h_2} = \tanh(W \cdot h_1 + U \cdot x_2 + b)$$

[trainable parameters of the RNN. (matrices)].

tanh adds non-linearity.

Step 3 : initialize w, u, b randomly

$$W = \begin{bmatrix} 0.3 & -0.1 \\ 0 & 0.2 \end{bmatrix}_{2\times2} \quad U = \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix}_{2\times1} \quad b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{2\times1}$$

$$h_1 = \tanh\begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 0.46 \\ 0.60 \end{bmatrix}_{2\times1}$$

$$h_2 = \tanh\left\{\begin{bmatrix} 0.3 & -0.1 \\ 0 & 0.2 \end{bmatrix}\boxed{\begin{bmatrix} 0.46 \\ 0.6 \end{bmatrix}} + \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix} 2 + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\}.$$

$$= \tanh\begin{bmatrix} 0.078 + 1 \\ 1.52 \end{bmatrix} = \tanh\begin{bmatrix} 1.078 \\ 1.52 \end{bmatrix} = \begin{bmatrix} 0.79 \\ 0.90 \end{bmatrix}$$

memory trans.

"passed to decoder"

- that is why the pressure solely lies on h_2 in RNN.

<GO> → some point of start.

* Decoder –

"Vocabulary"

Step 1: Convert Hindi words into embeddings.

$\begin{bmatrix} \text{मैं} \\ \text{जाता} \\ \text{हूं} \\ \text{<EOS>} \end{bmatrix}$

$x(\text{मैं}) = 1$

$x(\text{जाता}) = 1.1$

$x(\text{हूं}) = 0.9$

$x(EOS) = 0.0$.

$$s_0 = h_2 = \begin{bmatrix} 0.79 \\ 0.90 \end{bmatrix}$$

$$S_t = \tanh\left(W_{dec} \cdot S_{t-1} + U_{dec} \cdot \underbrace{x_t}_{\text{input}} + b_{dec}\right)$$

* o/p of S_1 is I/p of S_2, o/p of S_2 is I/p to S_3 . . . .

Step 3 : Calculate decoder hidden states.

$$s_1 = \tanh\left\{ W_{dec} \begin{bmatrix} 0.7 \\ 0.9 \end{bmatrix} + U_{dec}\, 0.5 + b_{dec} \right\}$$

$x\langle GO\rangle = 0.5$ (chosen random)

W_{dec}, U_{dec} & b_{dec} → randomly initialized.

$$s_1 = \begin{bmatrix} 0.27 \\ 0.58 \end{bmatrix}_{2\times1} \quad \text{— ①}$$

$(4\times2)\ (2\times1) = 4\times1$ matrix.

- take this 2×1 matrix and project into 4×1 ~~matrix~~ ~~of~~ probability matrix. because voc. is 4×1 matrix.

Step 4 : output matrix / Logits matrix.

$$\underset{(4\times1)}{\text{logits}_t} = \underset{(4\times2)}{W_{out}} \cdot \underset{(2\times1)}{S_t} + b_{out}$$

① trainable para.

$$W_{out} = \begin{bmatrix} 0.2 & 0.1 \\ 0. & 0.2 \\ -0.1 & -0.2 \\ -0.2 & -0.1 \end{bmatrix}_{4\times2} \qquad b_{out} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{4\times1}$$

$$\text{logits}_t = \begin{bmatrix} 0.2 & 0.1 \\ 0 & 0.2 \\ -0.1 & -0.2 \\ -0.2 & -0.1 \end{bmatrix}_{4\times2} \begin{bmatrix} 0.27 \\ 0.58 \end{bmatrix}_{2\times1} = \boxed{\begin{bmatrix} 0.11 \\ 0.12 \\ -0.14 \\ -0.03 \end{bmatrix}_{4\times1}}$$

Logits matrix indicates what translated word is.

~~We~~ But we somehow need to turn these values into "probability".

⊙ We make use of $\boxed{\text{softmax function}}$, to give higher probabilities to higher values & penalize the lower values. other normalization functions cannot do that. ⊛

we cannot be ~~confi~~ completely confident with predictions.

① Positive numbers.

② Sum = 1

③ Get more confidence in our predictions.

⊛ imp.

Step 5:

$$\begin{bmatrix} 0.11 \\ 0.12 \\ -0.14 \\ -0.03 \end{bmatrix} \Rightarrow \text{softmax} \Rightarrow \begin{bmatrix} 0.275 \\ \boxed{0.276} \\ 0.212 \\ 0.238 \end{bmatrix}$$

✓ Highest probability (जाता)

Next word prediction : "जाता"

Step 6 : Continue decoding -

x_2 = "जाता" embedding = 1.1.

$$s_2 = \tanh(w_{dec} \cdot s_1 + U_{dec} \cdot x_2 + b_{dec})$$

• Trainable parameters.

$$= \tanh\left(\underline{\underline{w_{dec}}} \cdot \begin{bmatrix} 0.27 \\ 0.58 \end{bmatrix} + \underline{\underline{U_{dec}}} * 1.1 + \underline{\underline{b_{dec}}}\right)$$

$$= \begin{bmatrix} 0.21 \\ 0.49 \end{bmatrix} \longrightarrow \text{softmax} \Rightarrow \begin{bmatrix} 0.0927 \\ \boxed{0.0978} \\ -0.12 \\ -0.02 \end{bmatrix}$$

✓ जाता

जाता is predicted again!!

⊙ word 2 vec → ~~conv~~ takes in words & converts to vector tokens · embeddings.

for. eg: $\boxed{\text{मैं}}$ → $\boxed{0.5}$ $\boxed{\text{जाता}}$ → $\boxed{1.1}$

or one can train them separately.

in RNNS both can be done.

⊙ RNNs are very specific to tasks.

whereas LLMs can do anything & everything together.

All the trainable parameters are trained during back propogation using "cross entropy loss" (*).

* Parameters marked in orange are trainable parameters.

Lecture 4

12/03/2025

"Code RNN from Scratch"

$\boxed{\text{Encoder block}}$ → main aim is ~~to~~ find h_1 & h_2 hidden state.

$$h_t = \tanh(W \cdot h_{t-1} + U \cdot x_t + b)$$

① W ② U ③ b

nn. parameter → (trainable) tell pytorch they are trainable

torch. randn → initialize randomly.

(hidden size, hidden size) → ~~size~~ of dim. of W matrix.

self. embedding → easy finding of embedding value for a given token.

For decoder –

$$s_n = \tanh(W_{dec}\, s_{n-1} + U_{dec} \cdot x_n + b_{dec})$$

① W_{dec} ② U_{dec} ③ b_{dec}

$$\underline{\underline{logit_t}}_{(4\times1)} = W_{out} \cdot s_t + b_{out}$$

④ W_{out}: 4×2 → 4: vocabulary size, 2: no. of hidden state size.

⑤ b_{out}: 2×1

logit → Matrix of probability.

LSTM – Long Short Term Memory

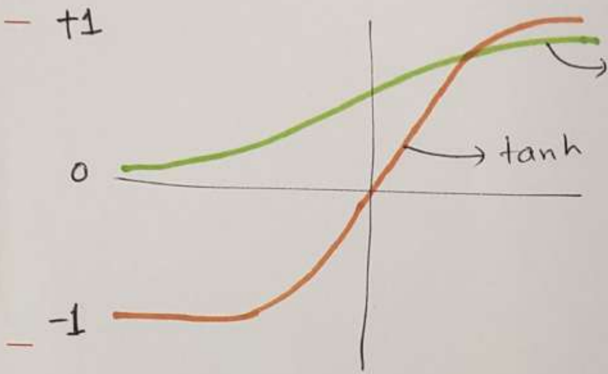
Lecture 5 -

(1997) Long Short term Memory

Neural Networks + Memory = RNN.

→ Huge info ⇒ cannot be taken by RNN.
(long texts cannot be computed by RNNs).

LSTM → solved both issues of RNNs.



sigmoid (retain fraction of value)

→ activation functions to introduce non-linearity.

→ they are very expressive.

when values are want to be pressed between 0 & 1 use sigmoid

① Gating mechanism:

$\boxed{\text{Value}}$ ⟶ (σ sigmoid) - - -> only fraction of the I/p value passed through.

0 to 1.

② Cell state: (missing in RNNs)

Long memory aspect of LSTM. carried info from way back.

consider it like a conveyor belt. carrying info from past we decide what to keep at each stage of cell state & what to remove.

$\boxed{\text{3 Gates}}$

① Forget Gate

what info to throw away from the <u>old cell state</u>.

Till now, input was only Harry (male).

Now, Hermoine enters (Female)

so imp to Harry should diminish.

Harry $\longrightarrow$ $\boxed{\text{Gate}}$ $\Rightarrow$ imp to Harry reduces by some factor f_t.

C_{t-1}

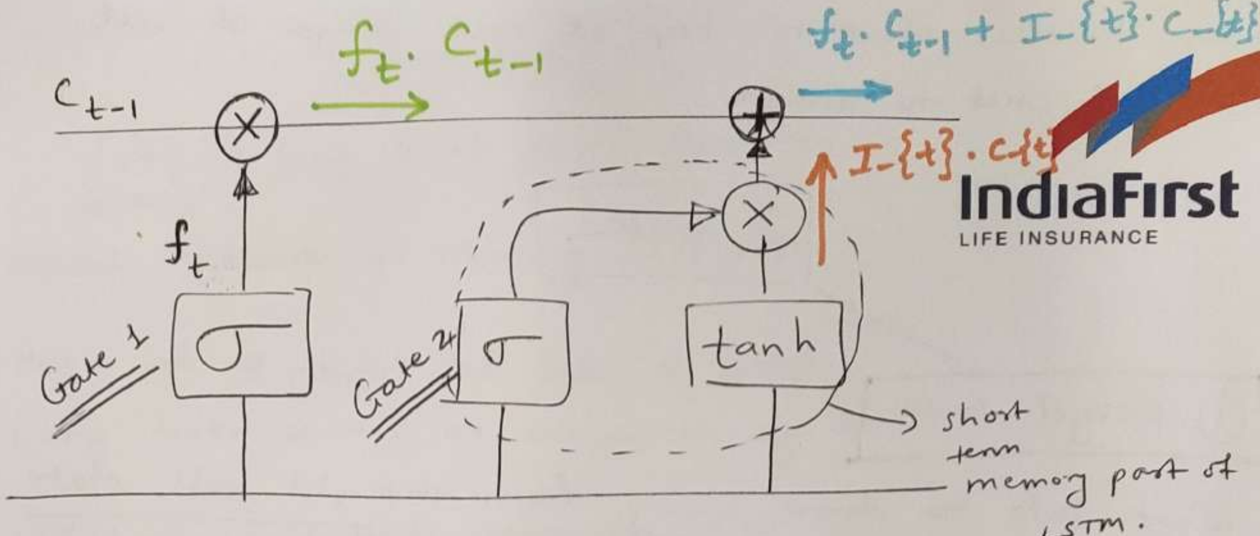
② Input gate. C_{t}.

Decides how much <u>new info</u> to let inside cell state.

↳ letsay about (Hermoine)

decided by i/p gate factor I_{t}

③ New <u>cell state</u> $C_t = \underbrace{f_t \cdot C_{t-1}}_{\substack{①\\ \text{How much of Harry to retain/forget}}} + \underbrace{\overbrace{\underbrace{I\_\{t\}}_{\substack{0-1 \\ \text{(sigmoid)}}} \cdot \underbrace{C\_\{t\}}_{\text{candidate vector}}}^{②}}_{\text{How much of Hermoine to add.}}$



C_{t} can lie between -1 & +1

⊛ I_{t} decides how much of C_{t} should be let in.

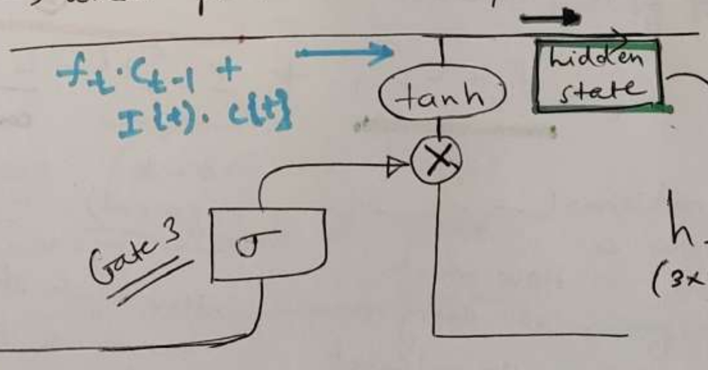
④ Update the cell state
using the derived formula in ③

⑤ $\boxed{\text{Output gate}}$ – lies btw 0 & 1

o/p = ~~cell states~~ filtered version of cell state.

→ only some part will be displayed
and this is controlled through o/p gate.

hidden state ultimately predicts the next word.
↳ what part to retain / remove.



$$\underbrace{h_t}_{(3\times1)} = \underbrace{\underline{\underline{o_t}}}_{\substack{\text{o/p gate} \\ (3\times1)}} \cdot \underbrace{\tanh(C_t)}_{(3\times1)}$$

element multiplication

O/P gate

$$h_t = O_t * \tanh(C_t)$$

(3x1) (3x1) (3x1)

not matrix multi

but element wise multiplication.

Forget gate (3x1) | (3x1) | I/P gate (3x1) | (3x1)

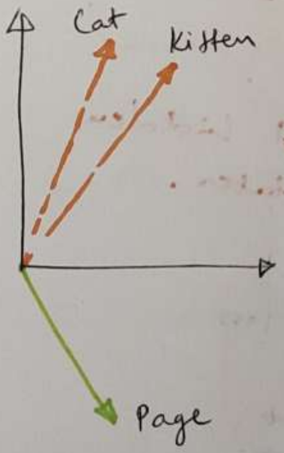
$$C_t = f_t * C_{t-1} + I\_\{t\} * C\_\{t\}$$

- f_t: sigmoid function
- C_{t-1}: prev. cell state
- $I\_\{t\}$: sigmoid function (value to lie btw 0 & 1)
- $C\_\{t\}$: tanh

W - Weight projection matrix

U - input projection matrix. → takes input and converts it into hidden state dimension.

more dimensions we have for words to embedding, the more context we can capture from them



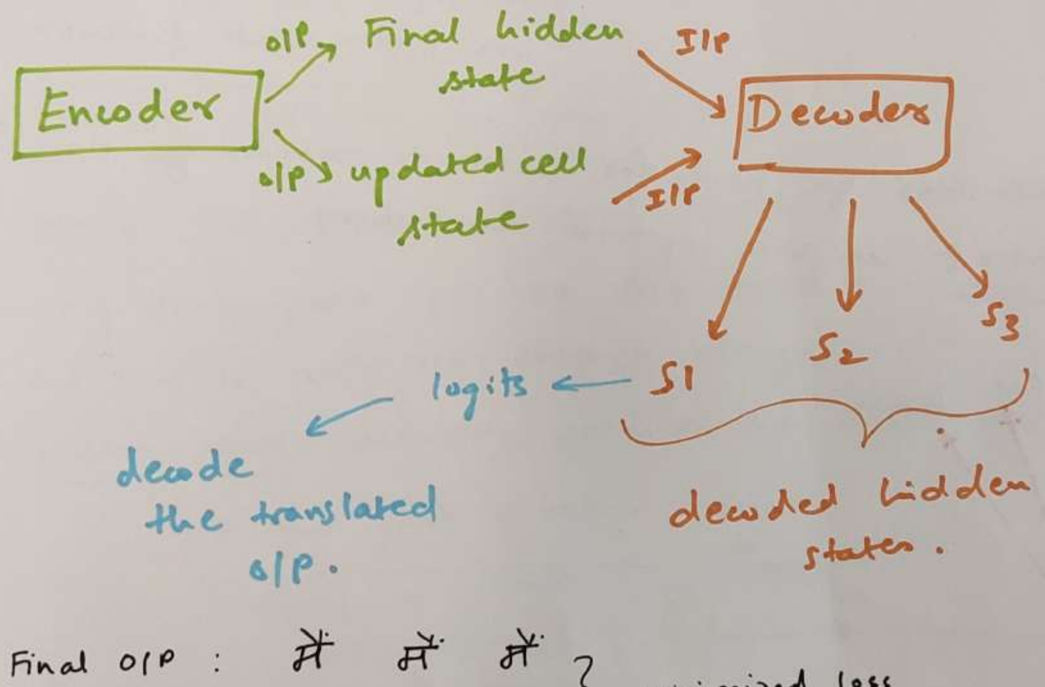
* For LSTMs,

two things are passed to the decoder.
the hidden state and cell state.
whereas in RNNs, only hidden state
was passed to decoder.

12 matrices → encoder
12 matrices → decoder
2 : logits

} LSTM. translator

all parameters optimized through ~~opti~~ back propogation.



Final o/p : मैं मैं मैं
wanted o/p : मैं जाता हूँ

} minimized loss through Gradient Descent.

LSTM for text generation -

We are studying Data Science $\boxed{?}$

① I/P (Predictors)

We [11]	are [2] → vector
We are [11 2]	studying 33
We are studying [11 2 33]	Data [44]
We are studying Data [11 2 33 44]	Science [77]

Target (Labels)

- Autoregressive
- model creates it on own

Each I/P - target pair is given to the LSTM.

Step 1 : Data Collection.

↓

Step 2 : Create vocabulary of words → tokenization. (token IDs)

word level "the" → 1 "Texas" = 3

"dog" → 2 ⋮

Step 3 : I/P target pairs for each headline.

Step 4: LSTM architecture

- convert token into vector embedding.
- Find the hidden state (pass it through architecture)
- logit matrix (Dense NN)

Step 5: Loss for each iteration.

Step 6: Use gradient descent optimization.

padding function = makes sure that all inputs are of the same size

if context length = 9.

we are studying Data Science

We [2]	are [11]
We are [2 11]	studying [100]
We are st. [2 11 100]	Data [60]
we are st. Data [2 11 100 60]	Science [90]

predictors

Label / Target state

[0 0 0 0 2 11 100 60 | 90]

(1) (2) (3) (4) (5) (6) (7) (8) (9)

No. of hidden states = Context size.

4,00,000
50,000