Encoder Decoder

Seq 2 Seq Data :

A kind of data in which input & output both will be in sequence.

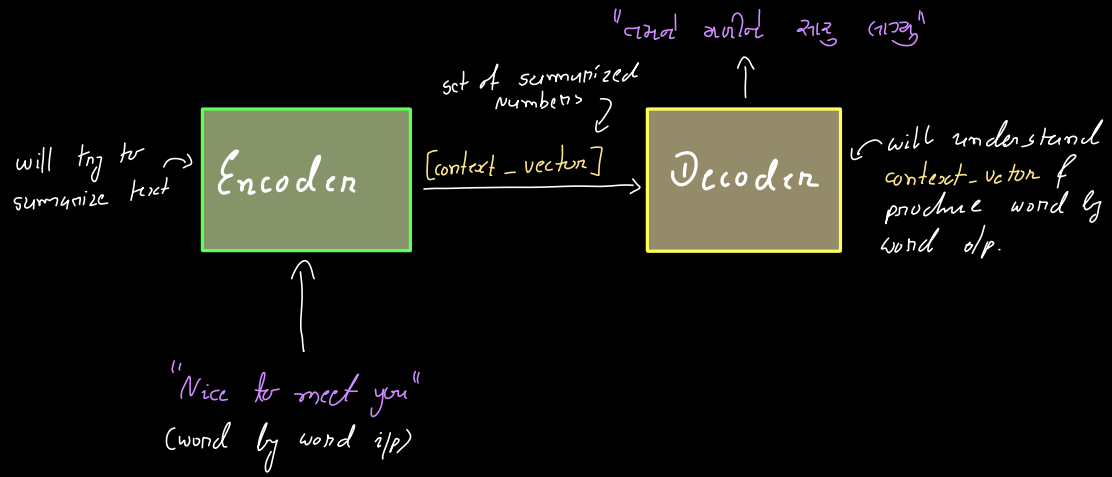
Example

Machine Translation

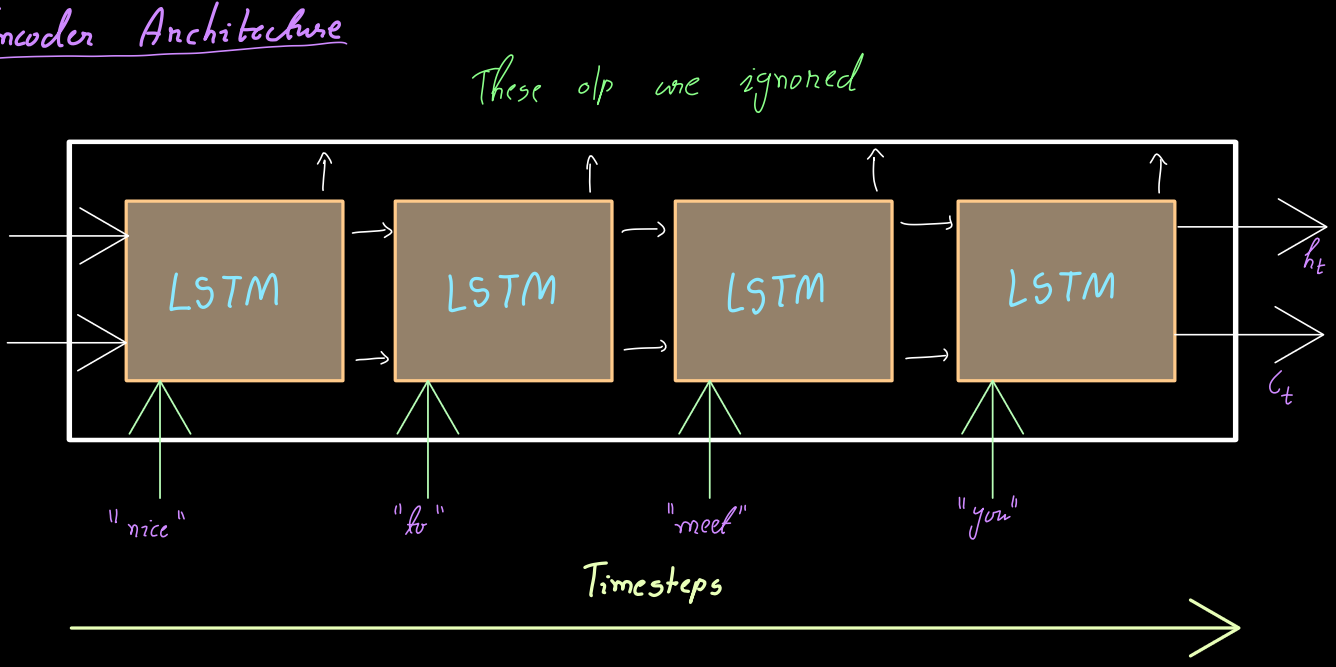
Challanges

input → Sentence → Variable length
output → Sentence → Variable length

basic architecture



Encoder Architecture

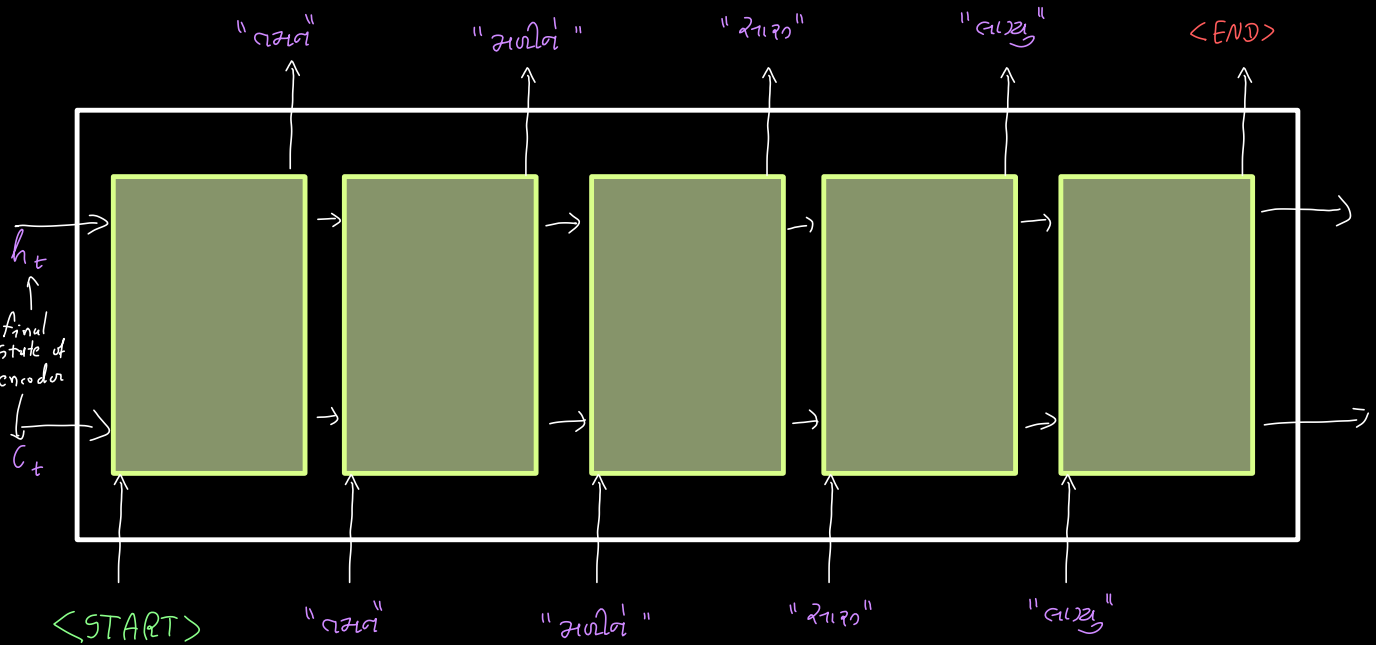


→ Encoder is just a single LSTM cell which we have unfolded in timesteps.

→ h_t and C_t at o/p side are final representation of sentence (context_vector).

→ Instead of LSTM cell, we can also use GRU. we don't use RNN due to vanishing gradient problem.

Decoder Architecture



t=1
i/p = context vector + special symbol <START>
(gives flag to decoder to start producing o/p).

→ Each o/p word will be i/p for next timestep.
→ Repeat until decoder gets termination symbol "<END>".

Training the Architecture using Backpropagation

→ If we're trying to solve machine translation kind of problems, then we need to use supervised dataset, which looks like this.

Inp_lang	Olp_lang
Jump	इछांगा
Hello!	नमस्ते!
Cheers!	वाह वाह!
⋮	⋮

Sample dataset

eng	hindi
Think about it	सोच लो
Come in	अंदर आ जाओ

[1,0,0,0,0] ← Think
[0,1,0,0,0] ← about
[0,0,1,0,0] ← it
[0,0,0,1,0] ← come
[0,0,0,0,1] ← in

[Think, about, it]	[सोच, लो]
[Come, in]	[अंदर, आ, जाओ]

[1,0,0,0,0,0,0] ← <START>
[0,1,0,0,0,0,0] ← सोच
[0,0,1,0,0,0,0] ← लो
[0,0,0,1,0,0,0] ← अंदर
[0,0,0,0,1,0,0] ← आ
[0,0,0,0,0,1,0] ← जाओ
[0,0,0,0,0,0,1] ← <END>

during one-hot-encoding of hindi words, we will add 2 more flag words <START> & <END> also.

Starting training process

Row_2 : [Think, about, it] → [सोच, लो]

→ Initially weights & bias will be assigned randomly.

y-true [0 1 0 0 0 0 0] each neuron will return probability for each word & max probability corresponding to word will be final o/p word.

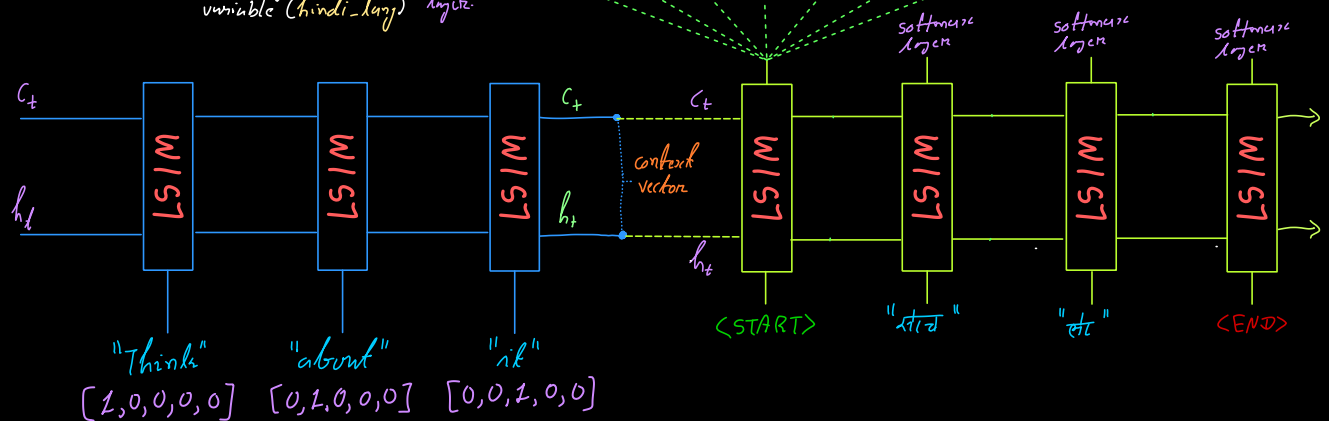
y-pred [0 0 1 0 0 0 0]

[0.1 0.2 **0.3** 0.1 0.2 0.2 0.01]

number of neurons = dictionary of o/p variable (hindi_lang)

softmax activation layer.

<START> सोच लो अंदर या गहरी <END>



→ As we know that this architecture can't gave any hindi word as o/p. so it will return a vector as o/p.

→ As we have seen example above that we can't pass o/p a current timestep on next timestep. because because wrong prediction at first time step will lead wrong o/p at next timestamp also.

→ Due to this problem instead of sending o/p of current timestep on next, we will pass correct o/p at related timestep intentionally. this process is called as Teacher forcing.

→ Teacher forcing will be apply during training, on testing current predicted word will be i/p for next timestep.

→ Due to teacher forcing method, training will be faster

→ Till now we have completed forward propagation, now let's calculate loss & move for backward propagation

Loss

	t=1	t=2	t=3
y_true	[0,1,0,0,0,0,0]	[0,0,1,0,0,0,0]	[0,0,0,0,0,0,1]
y_pred	[0.2,0.1,0.3,0.2,0.1,0.1,0.2]	[0.1,0.2,0.1,0.15,0.3,0.5,0.05]	[0.1,0.2,0.2,0.1,0.3,0.1,0.4]

loss function : categorical_crossentropy (multiclass classification)

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

$L_{t=1} = -1 \times \log(0.1) = 1$

$L_{t=2} = -1 \times \log(0.1) = 1$

$L_{t=3} = -1 \times \log(0.4)$

total loss = 2.39

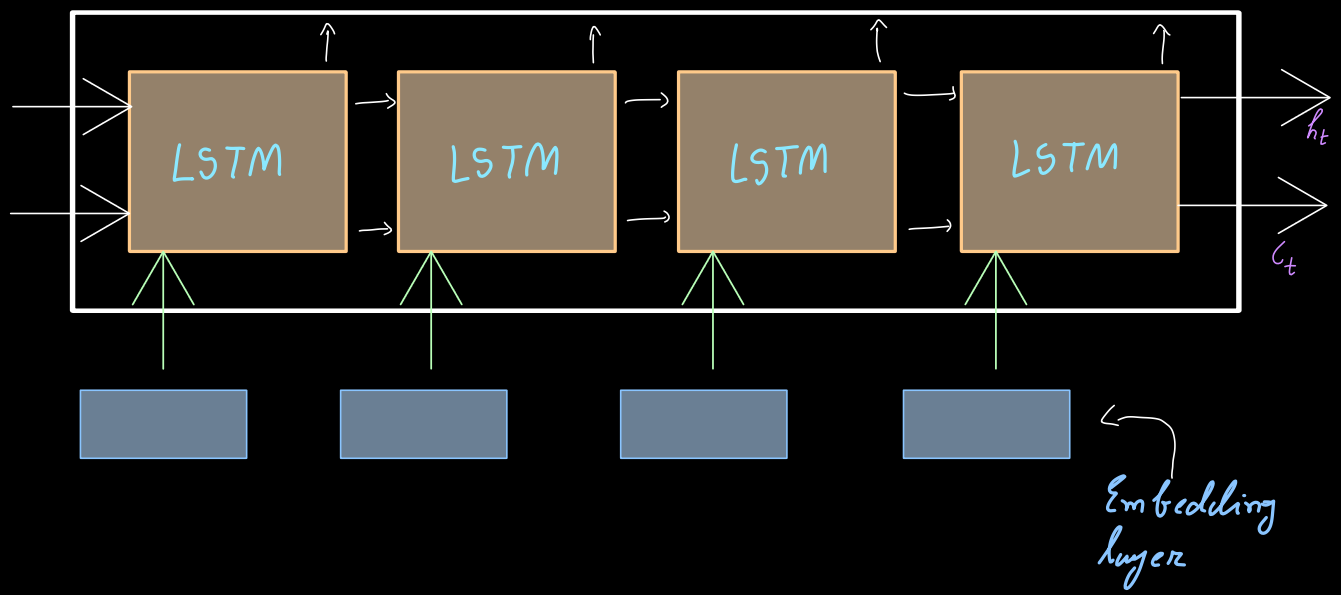
average loss = 0.7966

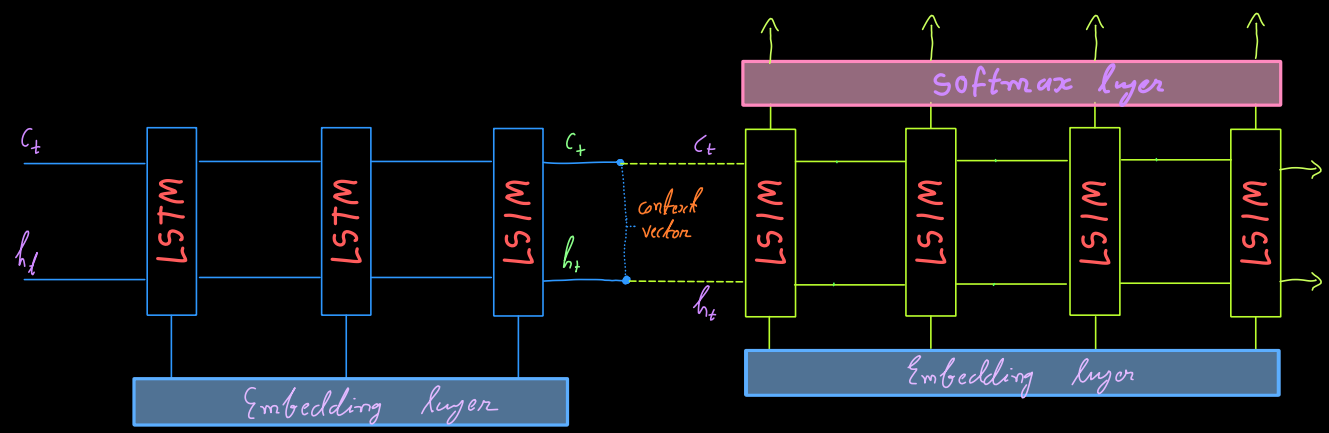
Back propagation

1. Gradient Calculation
2. Update weights

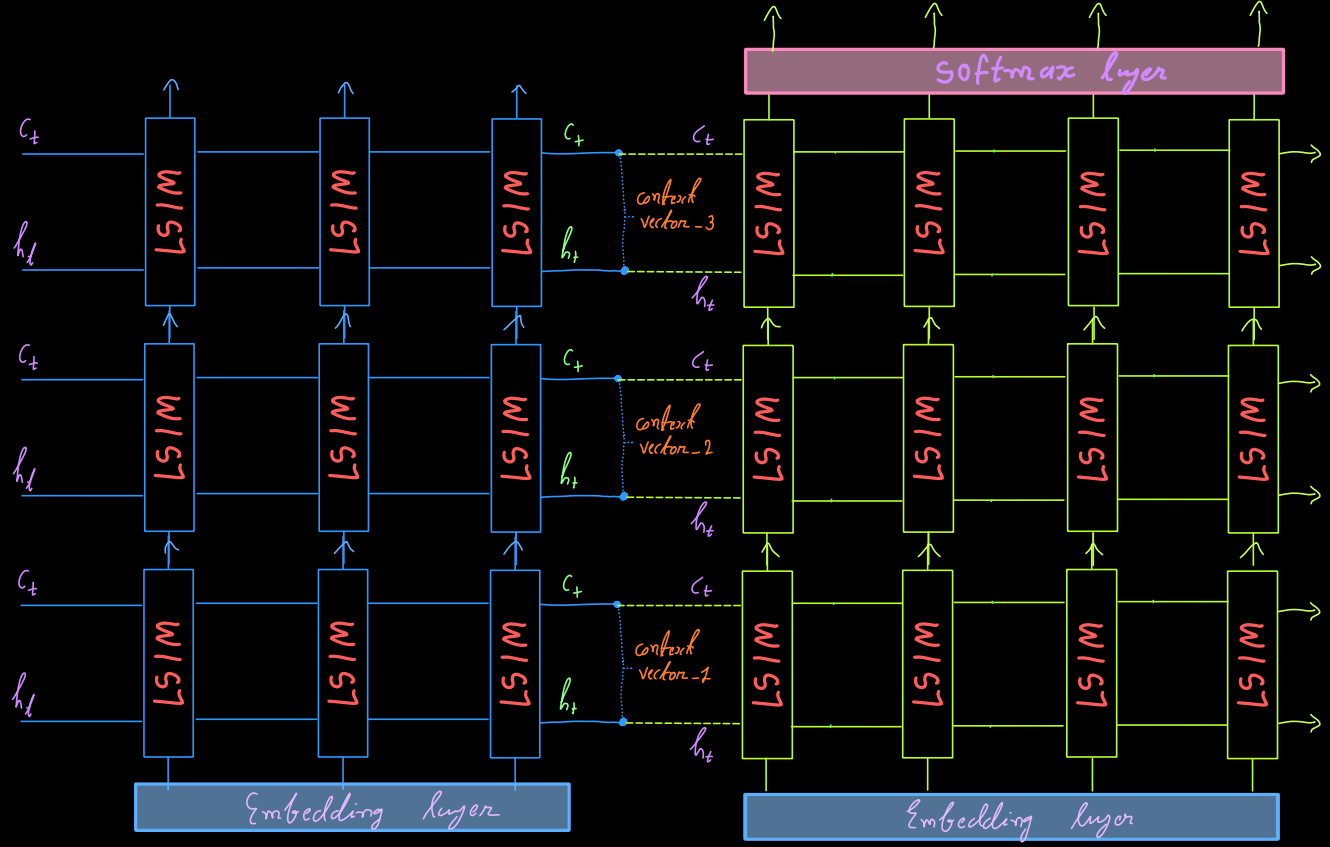
forward propagation → output → loss → gradient calculation → update weights → learning_rate → new_weights

Improvement - 1 (Embedding)





Improvement - 2 (Deep LSTMs)



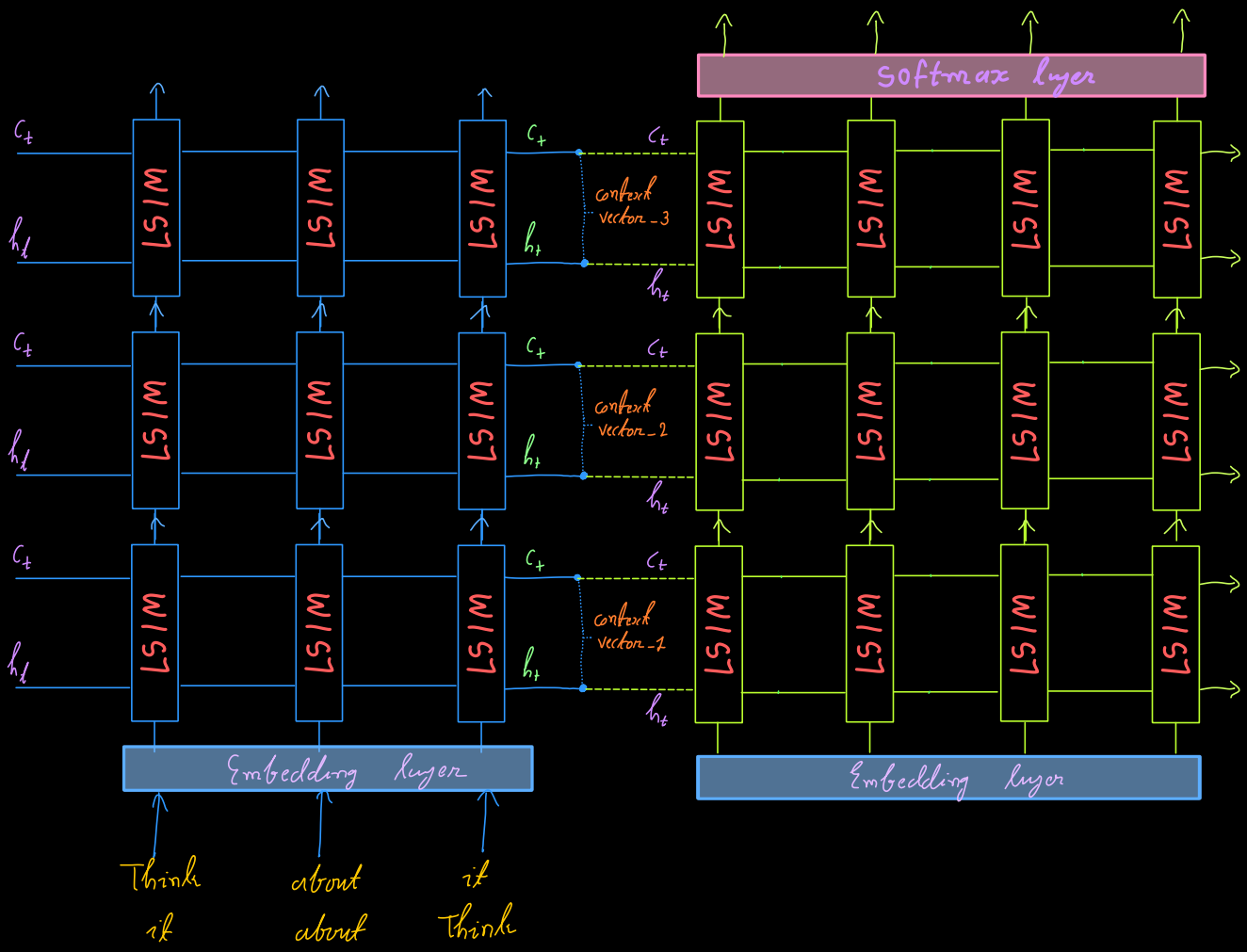
Benefits

→ Can easily handle long-term dependencies.

→ Layered representation easy to understand hierarchical representation.

→ Improves learning capability

Improvement -3 (Reversing the input)



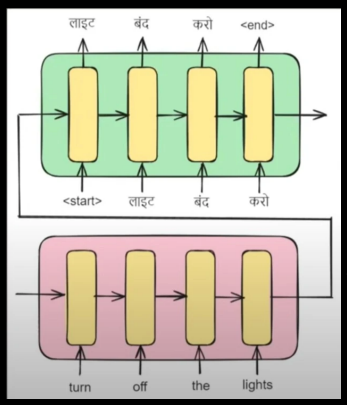
→ The input reversing word technique won't work in all cases.

→ This technique works in such i/p where initial words contains more context language

Attention Mechanism

why....?

→



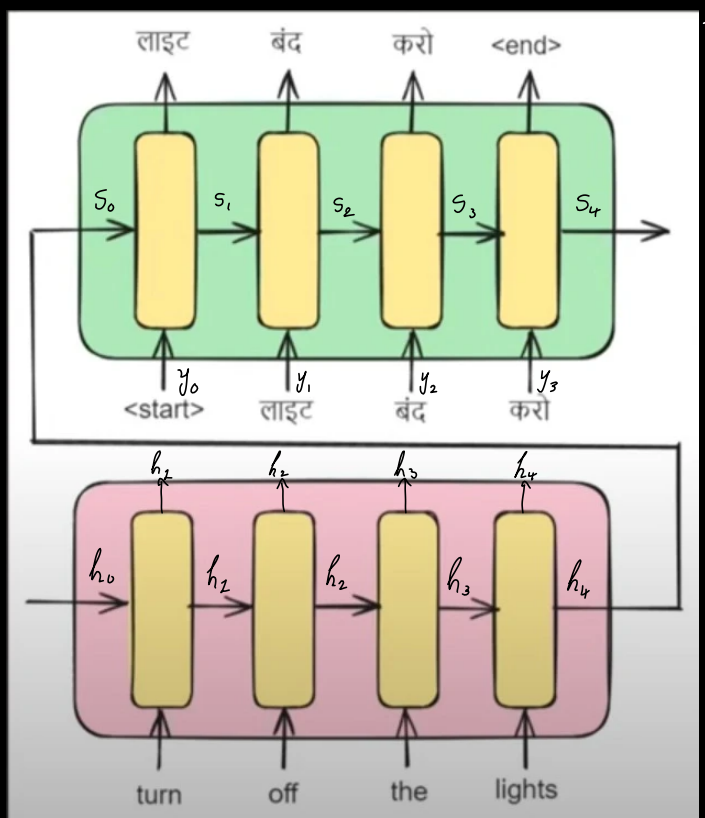
→ Suppose we have a paragraph with almost 50 words. which creates more pressure on encoder to summarize entire paragraph in few numbers which is very hard to manage context.

→ At any timestamp of decoder. we don't need entire sentence, we just need only few words/numbers from encoder to translate at given timestamp.

→ But in this encoder-decoder architecture, entire data in the form of context vector will be passed at each time step. and creates a pressure to decoder for generate accurate prediction.

→ If we talk about how human memorize & translate sentences. we won't translate entire paragraph at once. instead we will translate it sentence by sentence by creating an attention window to give more/less attention on each part of sentence rather than giving equal importance to entire paragraph

→ As we move forward, region around region window gets visible & the translation process will occur based on visibility.



→ At each time step of decoder, we also need to pass the most important timestamp of encoder at current timestamp.

→ At $t=1$ of decoder, $t=4$ of encoder is important At $t=2$ of decoder, $t=1$ and $t=2$ of encoder is important.

→ At any timestep t_i i/p will be $[s_j, y_j]$ additionally we will also pass one more piece of information, which is most useful -ll timestep of encoder at current decoder's timestep.

→ Now the input at current timestep t_i i/p will be $[s_j, y_j, c_k]$

→ So at particular time t_i i/p will be $[s_{i-1}, y_{i-1}, c_k]$

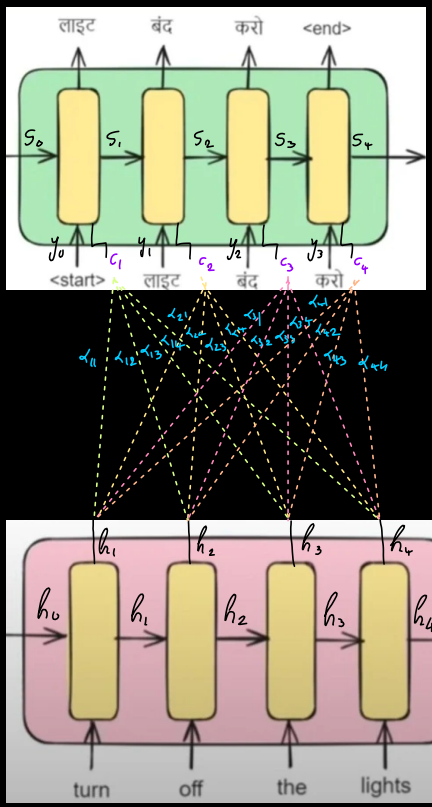
↳ attention i/p

→ Here c_k will be either Vector, Scaler or matrix. depends on y_i

→ The dimention of c_k will equal to h_i. in case of multipal context vectors h_i we will do weighted sum of them.

Q. How to identify most relevent hidden state of encoder for decoder at any given time-stemp....?

→ Seppose at $t=1$ of decoder we want most relevent hidden state that we will assign weights α to the o/p at each timestemp of encoder & will perform element-wise operation like....



$$c_2 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3 + \alpha_{14} h_4$$

where:
- α = scaler weights
- h = vectors

hence if h_2 are 4D vector-space the o/p c_1 will also be 4D vector space.

→ This process will be repeated for c_2, c_3 and c_4 also.

→ While calculating c_1, which ever weight have highest value, will be transfuse for further use

→ Now suppose if decoder timestep is denoted by i and encoder timestemp is denoted by j.

$$c_i = \sum_{i,j=1}^{n} \alpha_{ij} h_j$$

→ for each h_j we need to create different α_{ij} so the total number of αs will be $i \times j = 16$.

α calculation

→ α is called as alignment/similarity score.

→ Example

for $\hat{y}_2$ calculation via c_2. hence let's seppose we pick α_{21}. which depends on h_1 and S_1. (previous hidden-state of decoder)

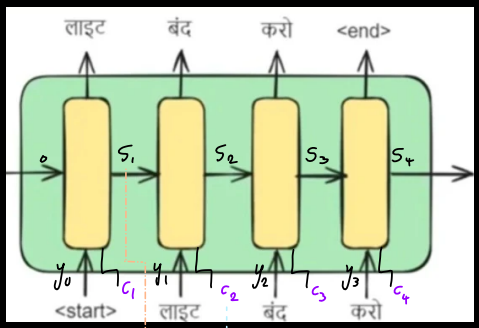
→ S_0, using attention mechanism, we wants to find that, "based on previous completed state S_1, tell me now, how much h_1 is important to calculate current o/p at $t=2$ timestemp."

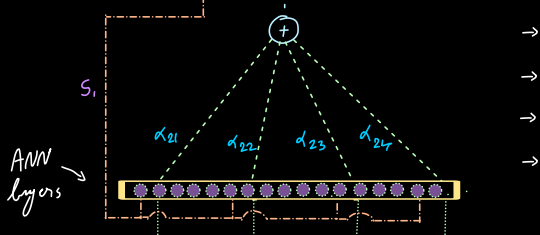
$\alpha_{21} \rightarrow f(h_1, S_1)$
$\alpha_{22} \rightarrow f(h_2, S_1)$

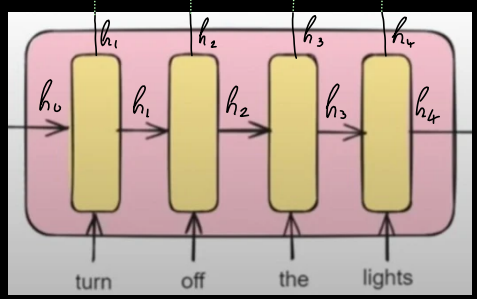
$\alpha_{ij} \rightarrow f(h_j, S_{i-1})$

→ this mathematical function f is approximation of ANN.

→ So the architecture may looks like this.







	i/p	o/p
→ At t=1	h_1 & S_1	α_{21}
→ At t=2	h_2 & S_1	α_{22}
→ At t=3	h_3 & S_1	α_{23}
→ At t=4	h_4 & S_1	α_{24}

	Turn	off	the	lights
लाइट	α_{11}	α_{12}	α_{13}	α_{14}
बंद	α_{21}	α_{22}	α_{23}	α_{24}
करो	α_{01}	α_{32}	α_{33}	α_{34}
<END>	α_{41}	α_{42}	α_{43}	α_{44}

→ Visulization purpose

→ Researchers have used Bi-directional LSTM only in encoder.

→ Now let's see 2 different type of α calculation technique.

1. Bahdanau Attention
2. Luong Attention

#1 Bahdanau Attention

Q. What is α..?

A. α is alignment score.

→ It shows that at any particular timestamp of decoder, what is the importance of any hidden state of encoder.

α_{11} ⟶ लाइट ⟶ turn

α_{12} ⟶ लाइट ⟶ off

$\alpha_{11} = f(h_1, S_0)$

$\alpha_{21} = f(h_1, S_1)$

$\alpha_{31} = f(h_1, S_2)$

α says that, based on till now decoded data, o/p of current timestamp. how much it depends on each hidden state of encoder.

S_i stores context of decoded data till current timestamp.

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

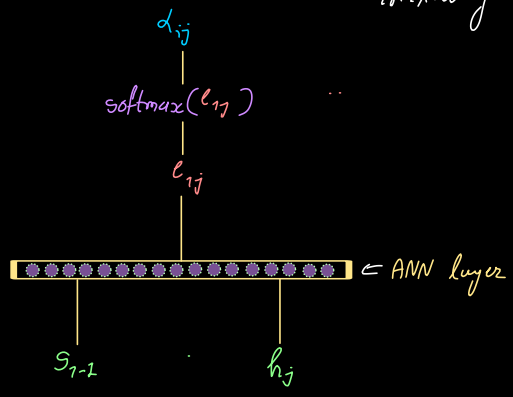
→ alignment score will be a mathematical function of 2 things. softmax, exp, sin, cos etc....

- hidden state of encoder
- previous hidden state of decoder

→ Here, selection of mathematical function Bahdanau Attention will help. it says that we can approximate this function using Feed Forward Neural-Networks.

→ Because Feed Forward Neural-Networks are Universal Function Approximators. if we provide enough data to ANNs.

initially suppose $h_0 = [a, b, c, d]$ & $S_0 = [e, f, g, h]$



- encodder side
  - ↳ h_1, h_2, h_3, h_4
- decodder side
  - ↳ $c_i = \sum \alpha_{ij} h_j$

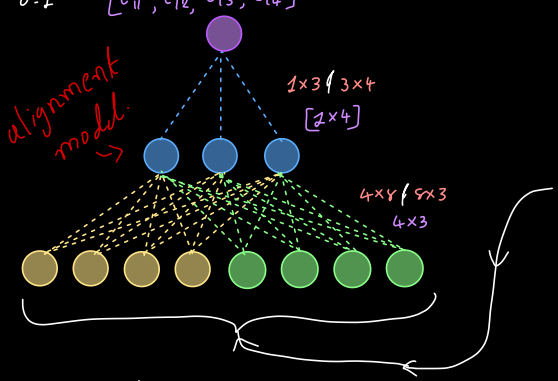
$\alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3 + \alpha_{14} h_4$

- giving i/p to ANN.

$\alpha_{11} = (h_1, S_0)$

$= (h_1, S_0 = [e, f, g, h])$

we need to concate S_0 with h_1, h_2, h_3 and h_4.



$$\begin{bmatrix} S_{01} & S_{02} & S_{03} & S_{04} & h_{11} & h_{12} & h_{13} & h_{14} \\ S_{01} & S_{02} & S_{03} & S_{04} & h_{21} & h_{22} & h_{23} & h_{24} \\ S_{01} & S_{02} & S_{03} & S_{04} & h_{31} & h_{32} & h_{33} & h_{34} \\ S_{01} & S_{02} & S_{03} & S_{04} & h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix}_{4\times 8}$$

→ Now we will pass each row to NN one-by-one.

→ Or we can pass entire matrix as batch operation.

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

$$\begin{bmatrix} S_{11} & S_{12} & S_{13} & S_{14} & h_{11} & h_{12} & h_{13} & h_{14} \\ S_{11} & S_{12} & S_{13} & S_{14} & h_{21} & h_{22} & h_{23} & h_{24} \\ S_{11} & S_{12} & S_{13} & S_{14} & h_{31} & h_{32} & h_{33} & h_{34} \\ S_{11} & S_{12} & S_{13} & S_{14} & h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix}$$

pass to ANN → [$\alpha_{21}, \alpha_{22}, \alpha_{23}, \alpha_{24}$] → c_2

c_2, S_1, Y_1 → LSTM → Y_2 → S_2

→ At each timestemp we will get different α values.

→ In decoder, this FNN will be same at each timestemp. hence it is called as

Time distributed Fully-connected Neural Network.

→ The parameters of this network will be updated during back propagation.

$$c_{ij} = \sum \alpha_{ij} h_j$$

[softmax(c_{ij})]

[tanh($W \cdot [S_{i-1} ; h_j] + b$) · V]

weights of i/p layer & first hidden layer.

weights between first

→ So, above shown mathematical process is called as **Bahdanau Attention**. also called as **Additive Attention**.

#2 Luong Attention / Multiplicative Attention

→ In this technique, we will calculate attention score.

→ Process will be same as bahdanau attention. but the α calculation process is slightly different.

$$\alpha_{ij} = f(h_j, s_i)$$

↳ we're not dependent on previous hidden state.

here, function f does the dot product of h_j and s_i.

$s_i = [a, b, c, d]$ $h_j = [e, f, g, h]$ $s_i^T \cdot h_j = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \cdot [e, f, g, h] = ae + bf + cg + dh$

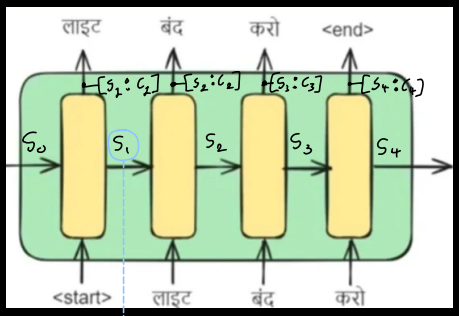
scaler / attention value

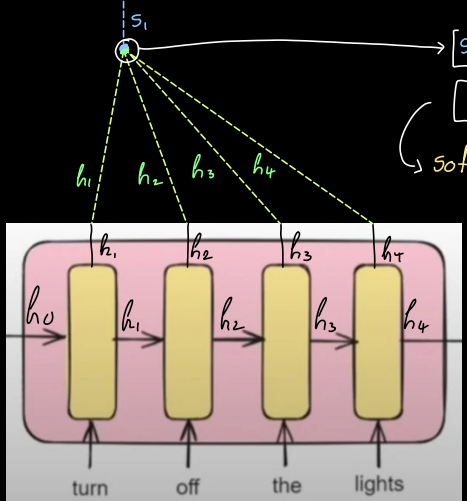
e_{ij}

$\text{softmax}(e_{ij})$

α_{ij}

→ here, we're using s_i instead of s_{i-1} because, by using s_i we will get more updated hidden state.





$[s_1 h_1, s_1 h_2, s_1 h_3, s_1 h_4]$

$[e_{11}, e_{12}, e_{13}, e_{14}]$

$\text{softmax}([e_{11}, e_{12}, e_{13}, e_{14}])$

$[\alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{14}]$

c_1 (this c_1 will be concatinated with s_1)

$[s_2 : c_1] = \tilde{s}_1 \rightarrow FNN \rightarrow \text{softmax}$

→ This process will continue for s_2, s_3 and s_4.