Encoder-Decoder Seq2Seq Models

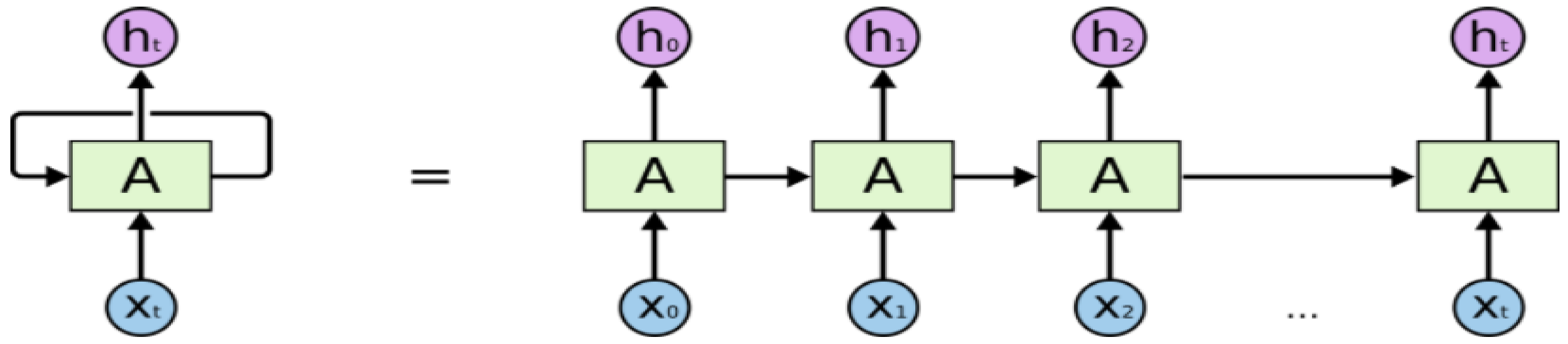
Introduction

- The traditional Deep Neural Networks (DNNs) are powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition and visual object recognition.
- But they can only be used for large labeled data where inputs and targets can be sensibly encoded with vectors of fixed dimensionality.
- They cannot be used to map sequences-to-sequences (for eg. machine translation)
- This was the motivation behind coming up with an architecture that can solve general sequence-to-sequence problems and so encoder-decoder models were born.

Sequence Modelling Problems

- Consider a very simple problem of predicting whether a movie review is positive or negative. Here our input is a sequence of words and output is a single number between 0 and 1. If we used traditional DNNs, then we would typically have to encode our input text into a vector of fixed length.
- But note that here the sequence of words is not preserved and hence when we feed our input vector into the model, it has no idea about the order of words and thus it is missing a very important piece of information about the input.

- Thus to solve this issue, RNNs came into the picture. In essence, for any input X = (x_0, x_1, x_2, … x_t) with a variable number of features, at each time-step, an RNN cell takes an item/token x_t as input and produces an output h_t while passing some information onto the next time-step. These outputs can be used according to the problem at hand.

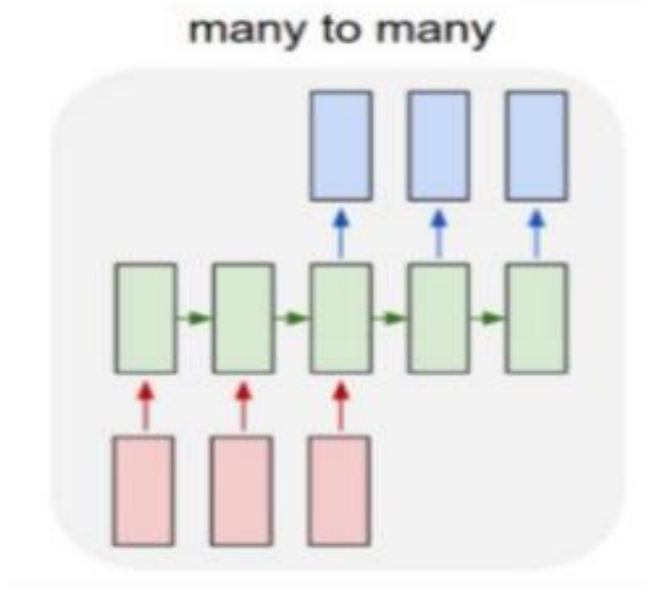


An unrolled recurrent neural network.

- The movie review prediction problem is an example of a very basic sequence problem called many to one predictions.

Sequence-to-Sequence Problems

- Sequence-to-Sequence (Seq2Seq) problems is a special class of Sequence Modelling Problems in which both, the input and the output is a sequence.
- Encoder-Decoder models were originally built to solve such Seq2Seq problems.



The Architecture of Encoder-Decoder models

- The Neural Machine Translation Problem : In neural machine translation, the input is a series of words, processed one after another. The output is, likewise, a series of words.
- **Input :**

  English sentence: “nice to meet you”
- **Output :**

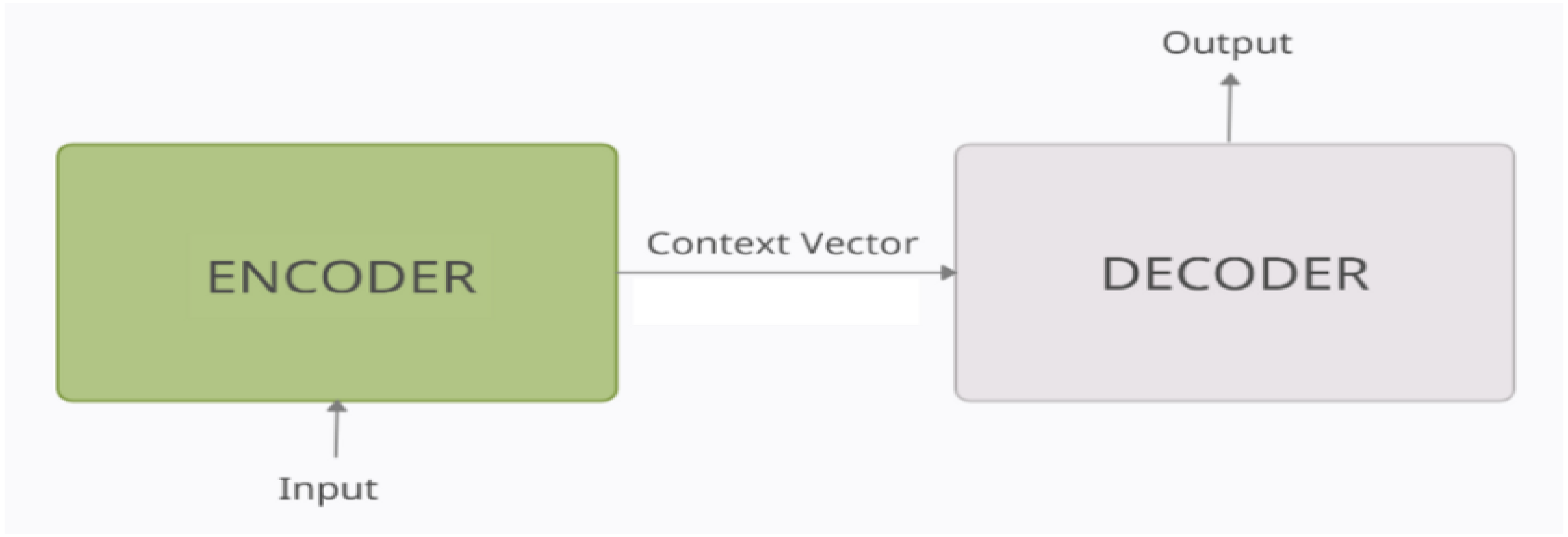
  French translation: “ravi de vous rencontrer”

Terms Used

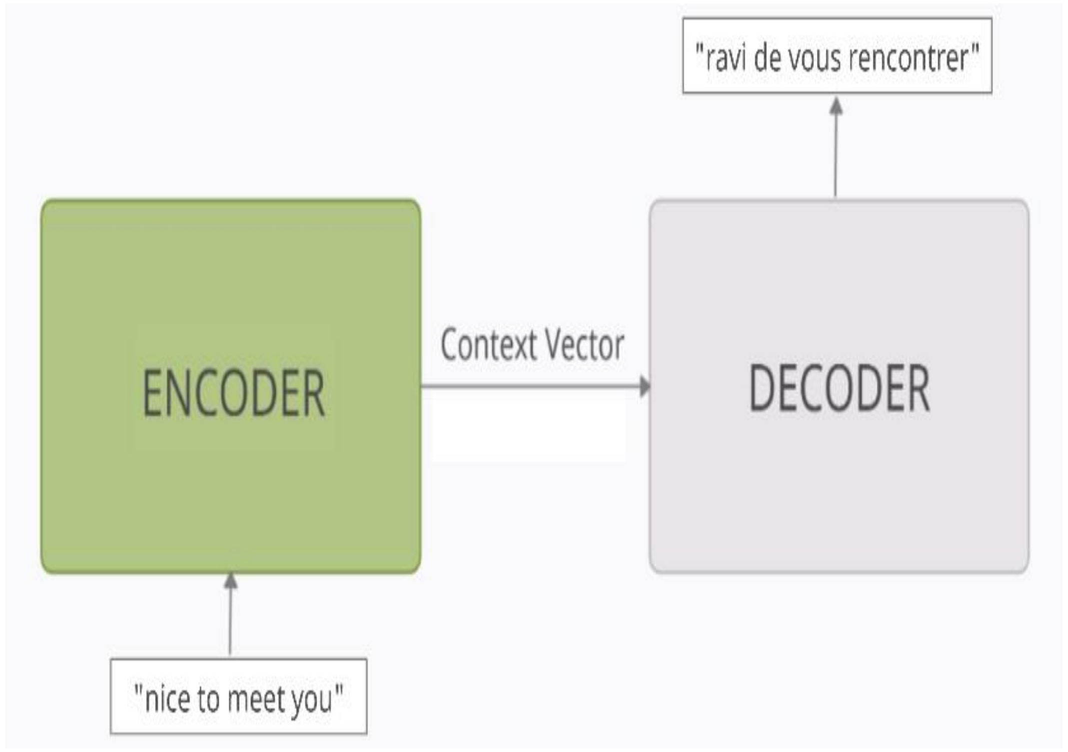
- **Input sentence** “nice to meet you” as **X/input-sequence.**
- **Output sentence** “ravi de vous rencontrer” as **Y_true/target-sequence** → This is what we want our model to predict (the ground truth).
- The **predicted output sentence** of the model as **Y_pred/predicted-sequence**
- The **individual words** of the English and French sentence are referred to as **tokens**

High-Level Overview

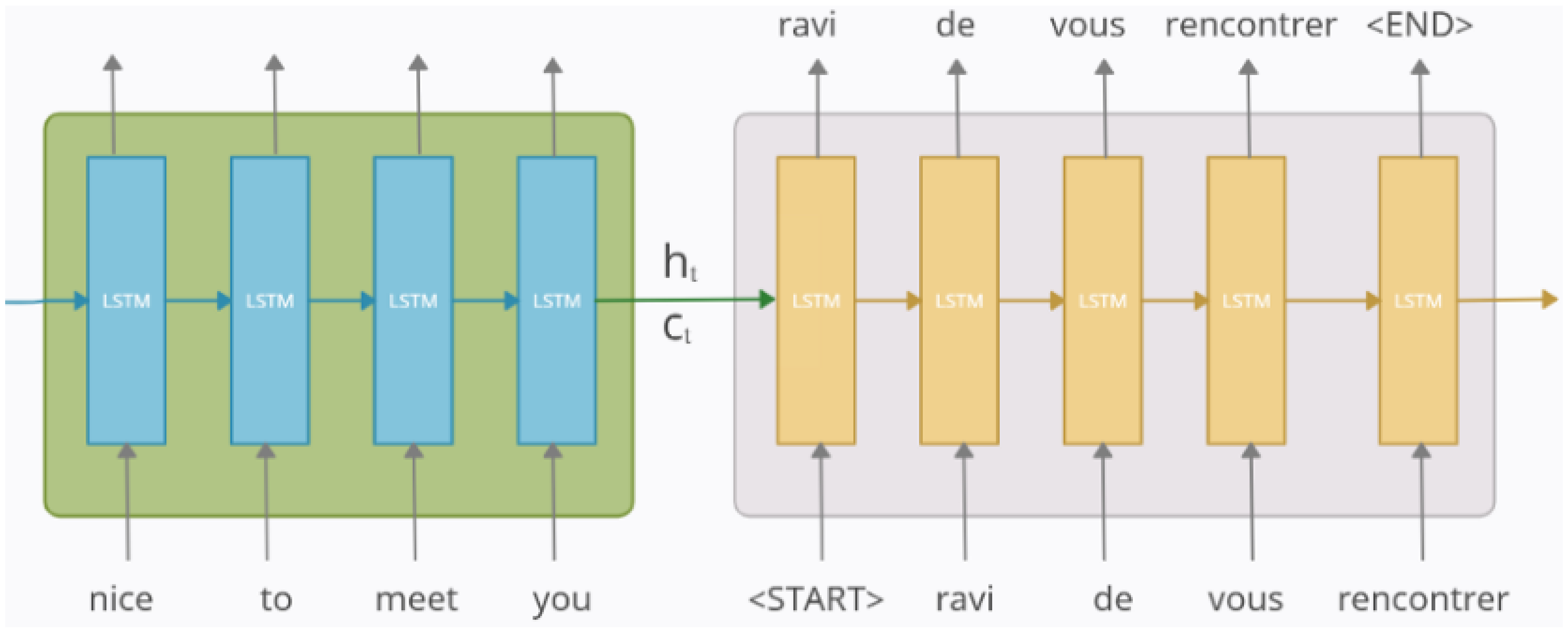
- At a very high level, an encoder-decoder model can be thought of as two blocks, the encoder and the decoder connected by a vector which we will refer to as the 'context vector'.



- **Encoder**: The encoder processes each token in the input-sequence. It tries to cram all the information about the input-sequence into a vector of fixed length i.e. the ‘context vector’. After going through all the tokens, the encoder passes this vector onto the decoder.
- **Context vector**: The vector is built in such a way that it's expected to encapsulate the whole meaning of the input-sequence and help the decoder make accurate predictions.
- **Decoder**: The decoder reads the context vector and tries to predict the target-sequence token by token.

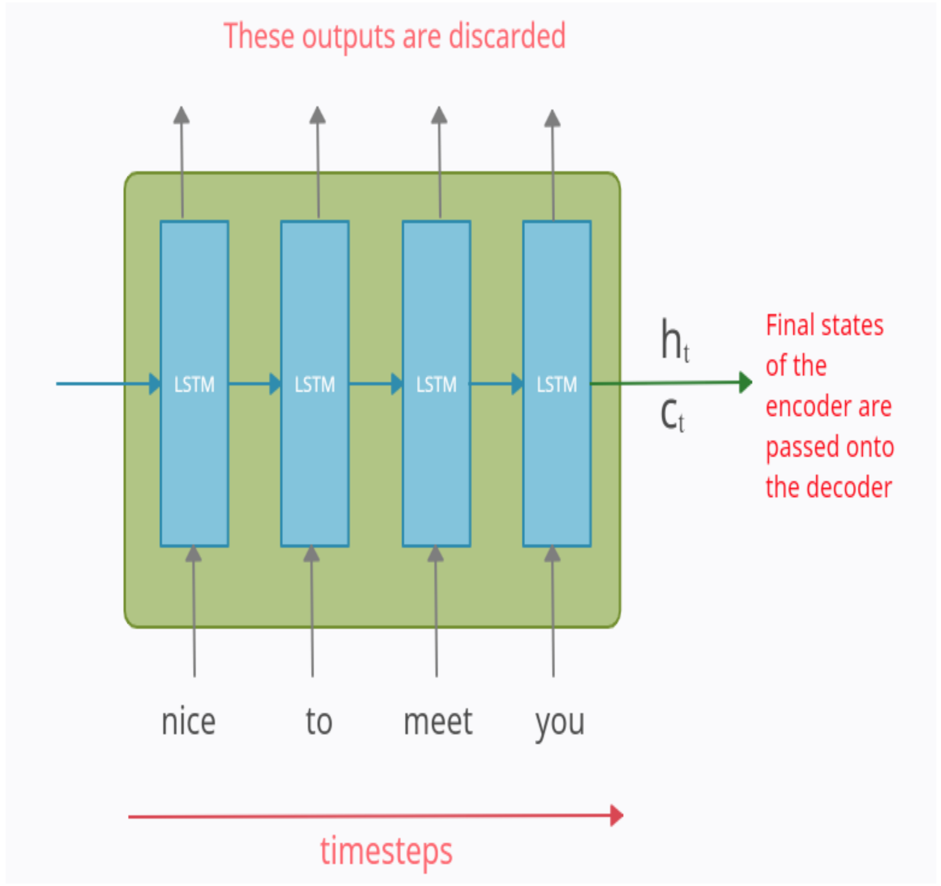


- The internal structure of both the blocks would look something like this:



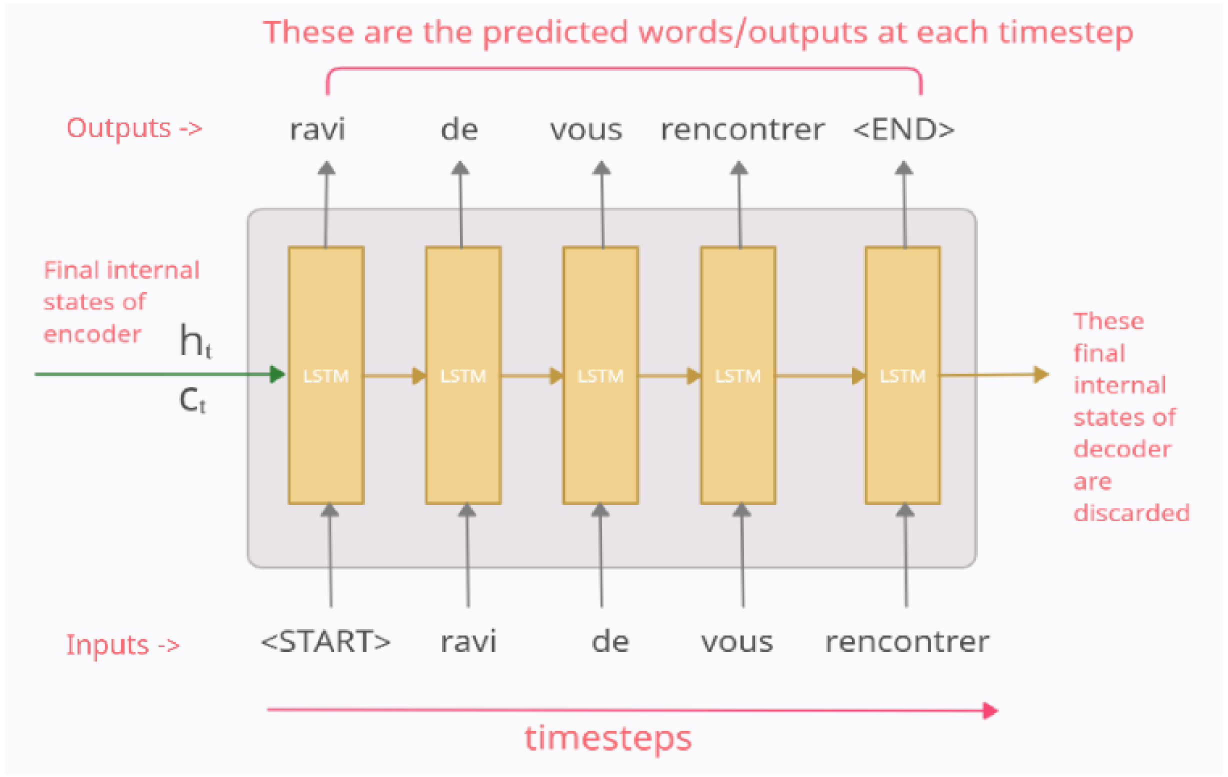
The Encoder Block

- The encoder part is an LSTM cell. It is fed in the input-sequence over time and it tries to encapsulate all its information and store it in its final internal states h_t (hidden state) and c_t (cell state).
- The internal states are then passed onto the decoder part, which it will use to try to produce the target-sequence. This is the 'context vector' which we were earlier referring to.
- The outputs at each time-step of the encoder part are all discarded.

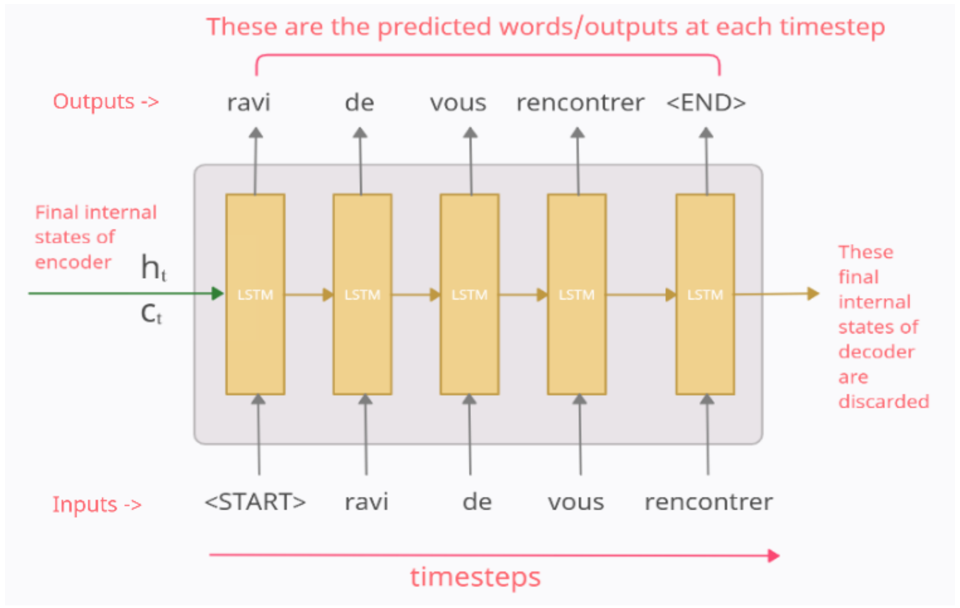

These outputs are discarded
LSTM
LSTM
LSTM
LSTM
h_t
c_t
Final states of the encoder are passed onto the decoder
nice
to
meet
you
timesteps

The Decoder Block

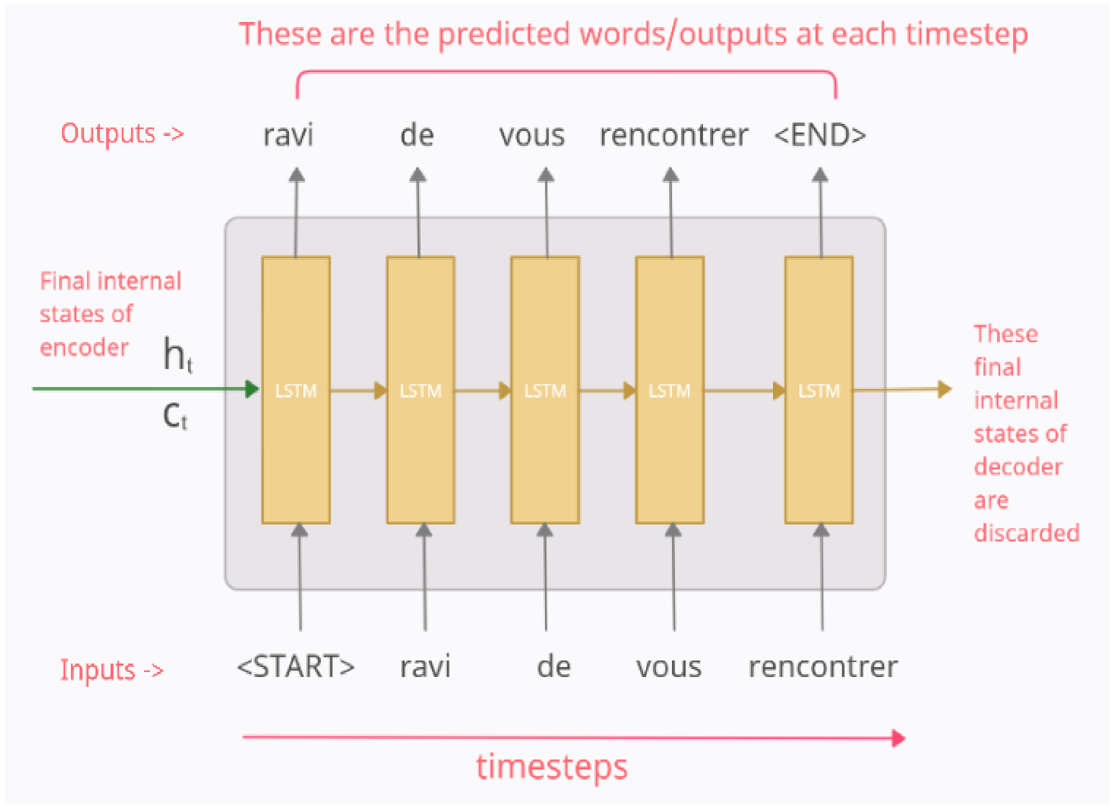
- So after reading the whole input-sequence, the encoder passes the internal states to the decoder and this is where the prediction of output-sequence begins.


These are the predicted words/outputs at each timestep
Outputs ->
ravi
de
vous
rencontrer
<END>
Final internal states of encoder
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
These final internal states of decoder are discarded
Inputs ->
<START>
ravi
de
vous
rencontrer
timesteps

- The decoder block is also an LSTM cell. The main thing to note here is that the initial states (h_0, c_0) of the decoder are set to the final states (h_t, c_t) of the encoder. These act as the 'context' vector and help the decoder produce the desired target-sequence.
- Now the way decoder works, is, that its output at any time-step t is supposed to be the t^{th} word in the target-sequence/Y_true ("ravi de vous rencontrer").

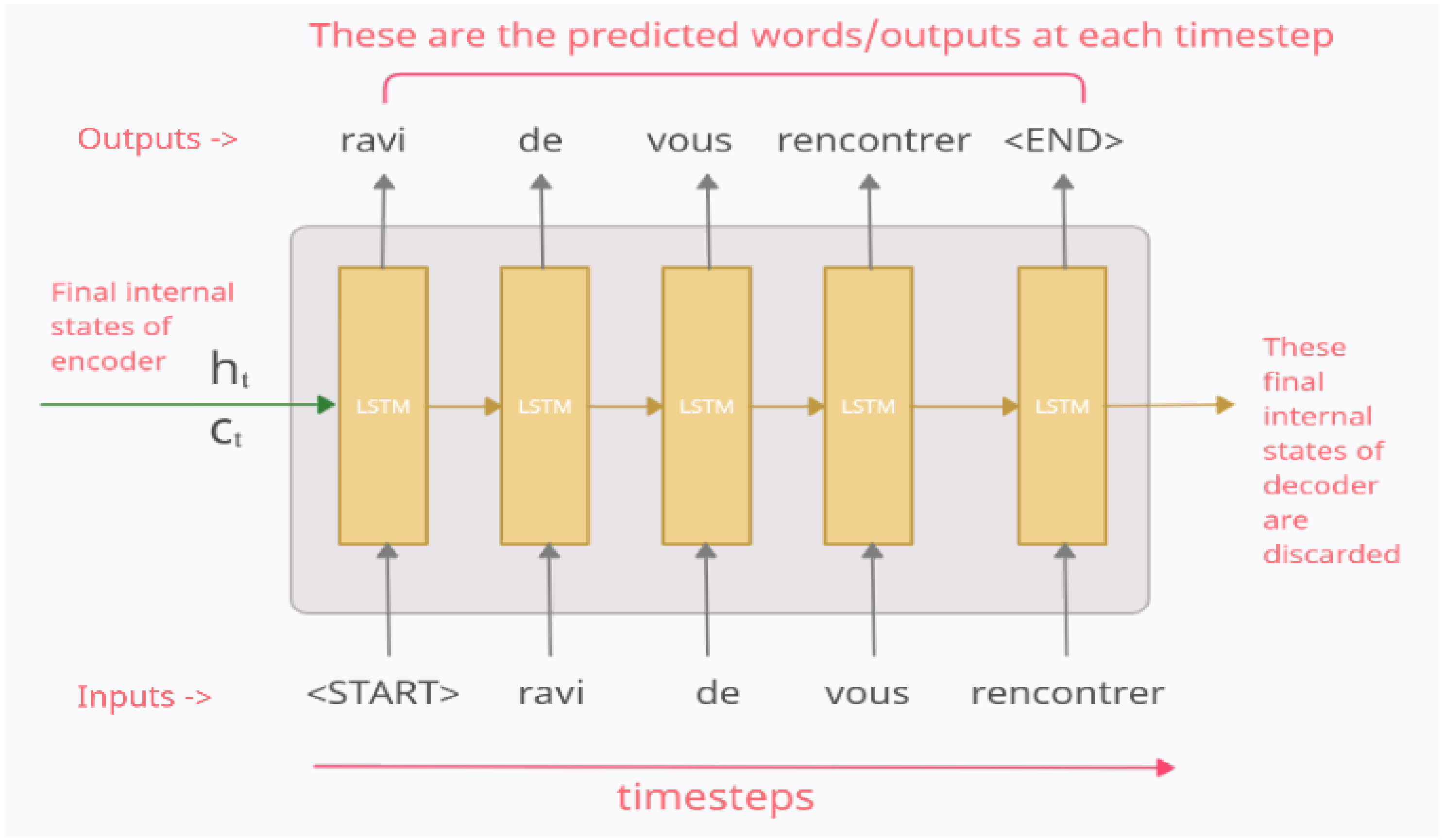

These are the predicted words/outputs at each timestep
Outputs ->
ravi
de
vous
rencontrer
<END>
Final internal states of encoder
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
These final internal states of decoder are discarded
Inputs ->
<START>
ravi
de
vous
rencontrer
timesteps

- At time-step 1
- The input fed to the decoder at the first time-step is a special symbol "<START>". This is used to signify the start of the output-sequence. Now the decoder uses this input and the internal states (h_t, c_t) to produce the output in the 1st time-step which is supposed to be the 1st word/token in the target-sequence i.e. 'ravi'.
- At time-step 2
- At time-step 2, the output from the 1st time-step "ravi" is fed as input to the 2nd time-step. The output in the 2nd time-step is supposed to be the 2nd word in the target-sequence i.e. 'de'



These are the predicted words/outputs at each timestep
Outputs ->
ravi
de
vous
rencontrer
<END>
Final internal states of encoder
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
These final internal states of decoder are discarded
Inputs ->
<START>
ravi
de
vous
rencontrer
timesteps

- And similarly, the output at each time-step is fed as input to the next time-step. This continues till we get the “<END>” symbol which is again a special symbol used to mark the end of the output-sequence. The final internal states of the decoder are discarded.


These are the predicted words/outputs at each timestep
Outputs ->
ravi
de
vous
rencontrer
<END>
Final internal states of encoder
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
These final internal states of decoder are discarded
Inputs ->
<START>
ravi
de
vous
rencontrer
timesteps

- Note that these special symbols need not necessarily be “<START>” and “<END>” only. These can be any strings given that these are not present in our data corpus so the model doesn’t confuse them with any other word.

- NOTE: The process mentioned above is how an ideal decoder will work in the testing phase. But in the training phase, a slightly different implementation is required, to make it train faster.

Vectorizing our data

- The raw data that we have is

  X = “nice to meet you” → Y_true = “ravi de vous rencontrer”

- Now we put the special symbols “<START>” and “<END>” at the start and the end of our target-sequence

  X = “nice to meet you” → Y_true = “<START> ravi de vous rencontrer <END>”

- Next the input and output data is vectorized using one-hot-encoding (ohe). Let the input and output be represented as

  X = (x1, x2, x3, x4) → Y_true = (y0_true, y1_true, y2_true, y3_true, y4_true, y5_true)

- where xi’s and yi’s represent the ohe vectors for input-sequence and output-sequence respectively. They can be shown as:
- For input X

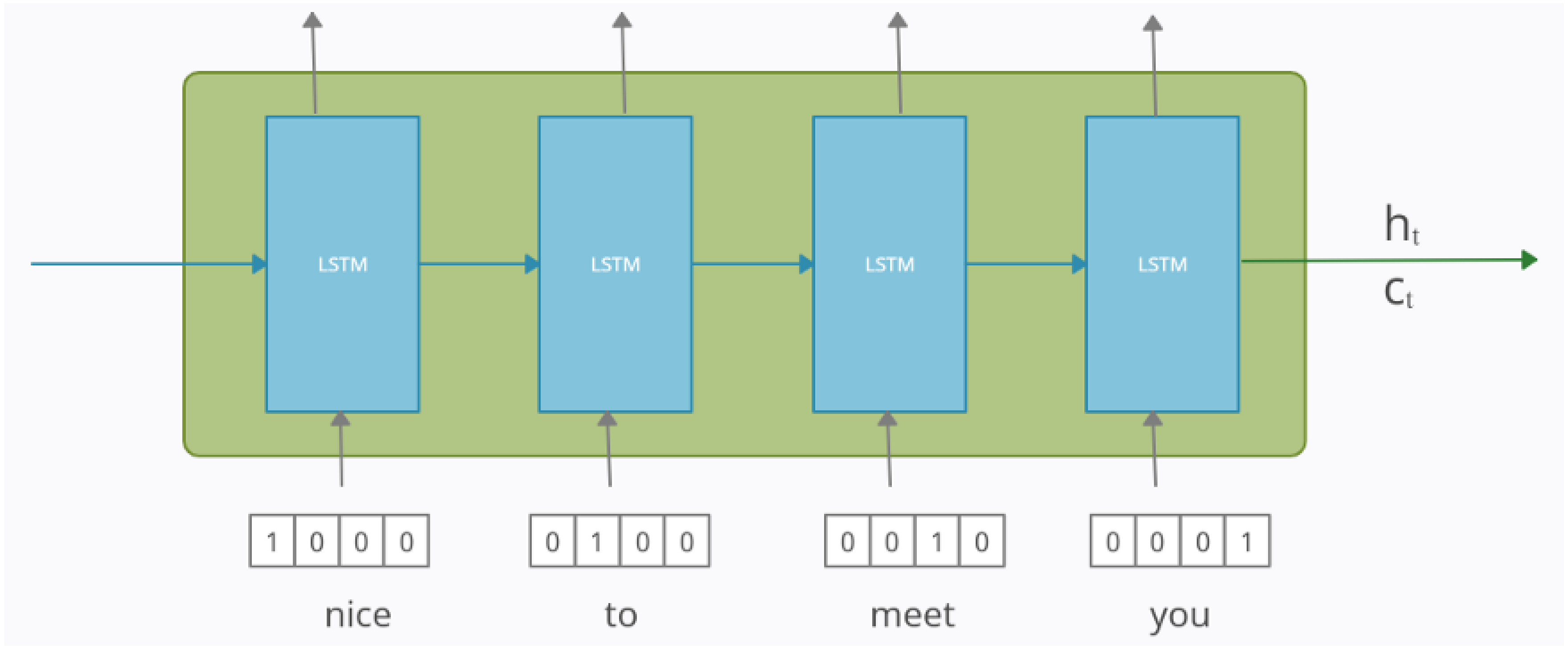
‘nice’	→ x1 : [1 0 0 0]
‘to’	→ x2 : [0 1 0 0]
‘meet’	→ x3 : [0 0 1 0]
‘you’	→ x4 : [0 0 0 1]

- For Output Y_true

‘<START>’	→ y0_true : [1 0 0 0 0 0]
‘ravi’	→ y1_true : [0 1 0 0 0 0]
‘de’	→ y2_true : [0 0 1 0 0 0]
‘vous’	→ y3_true : [0 0 0 1 0 0]
‘rencontrer’	→ y4_true : [0 0 0 0 1 0]
‘<END>’	→ y5_true : [0 0 0 0 0 1]

Training and Testing of Encoder

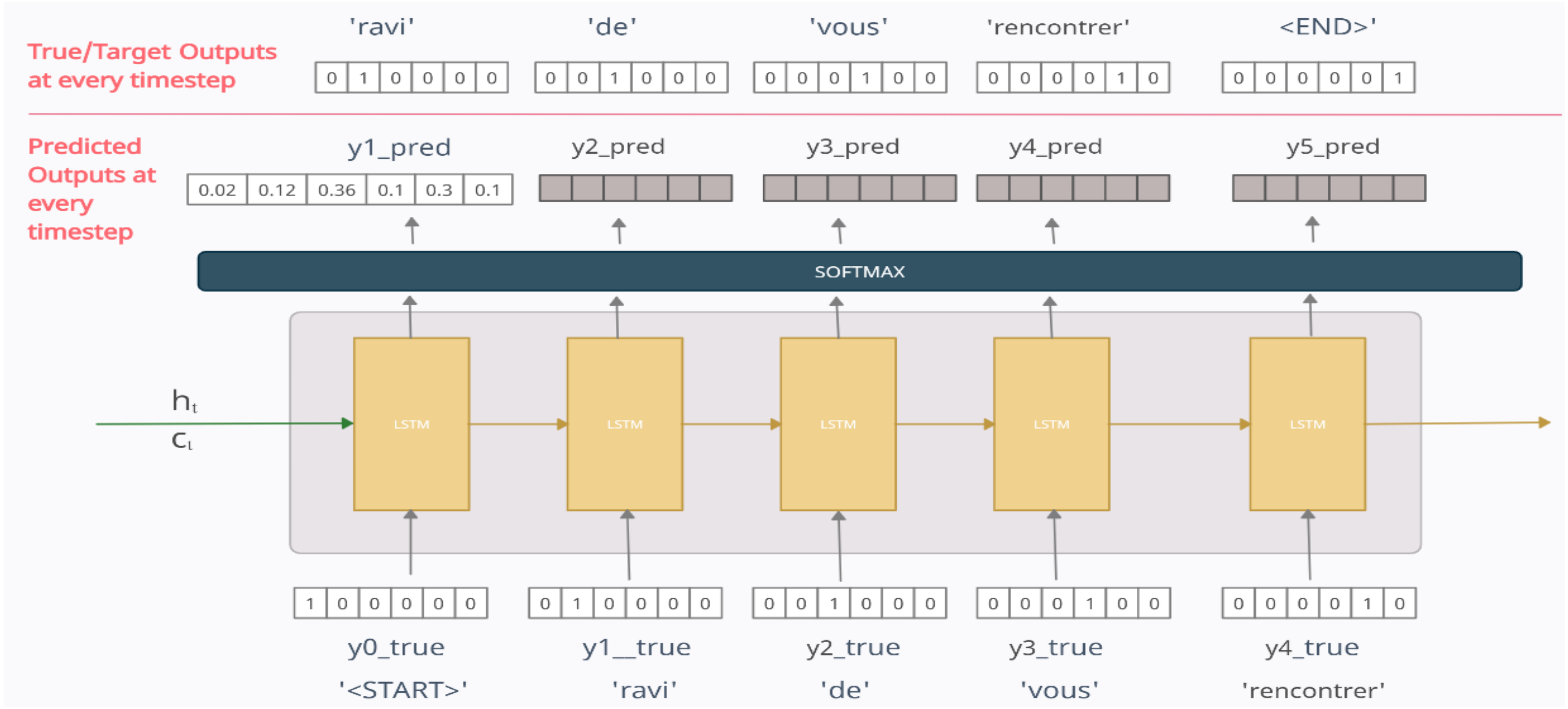
- The working of the encoder is the same in both the training and testing phase. It accepts each token/word of the input-sequence one by one and sends the final states to the decoder. Its parameters are updated using backpropagation over time.



The Decoder in Training Phase: Teacher Forcing

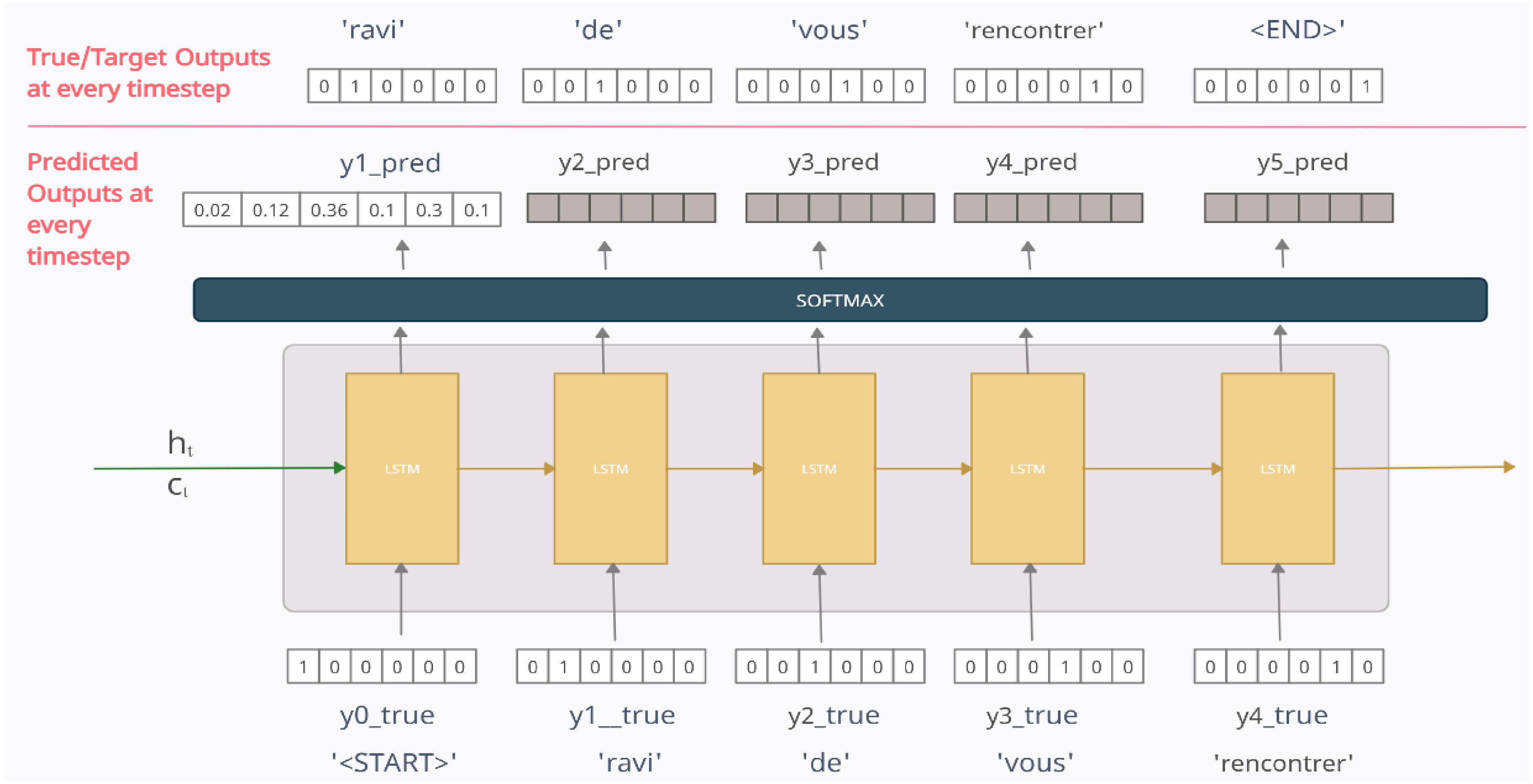
- The working of the decoder is different during the training and testing phase, unlike the encoder part.
- To train our decoder model, we use a technique called "Teacher Forcing" in which we feed the true output/token (and not the predicted output/token) from the previous time-step as input to the current time-step.

- We have fed our input-sequence to the encoder, which processes it and passes its final internal states to the decoder. Now for the decoder part, refer to the diagram below.



- In the decoder, at any time-step t, the output yt_pred is the probability distribution over the entire vocabulary in the output dataset which is generated by using the Softmax activation function. The token with the maximum probability is chosen to be the predicted word.

For eg. referring to the diagram, y1_pred = [0.02 0.12 0.36 0.1 0.3 0.1] tells us that our model thinks that the probability of 1st token in the output-sequence being '<START>' is 0.02, 'ravi' is 0.12, 'de' is 0.36 and so on. We take the predicted word to be the one with the highest probability. Hence here the predicted word/token is 'de' with a probability of 0.36



At time-step 1

- The vector [1 0 0 0 0 0] for the word '<START>' is fed as the input vector.
- Now here model wants to predict the output as y1_true=[0 1 0 0 0 0] but since the model has just started training, it will output something random.
- Let the predicted value at time-step 1 be y1_pred=[0.02 0.12 0.36 0.1 0.3 0.1] meaning it predicts the 1st token to be 'de'.
- Now, should we use this y1_pred as the input at time-step 2?.
- We can do that, but in practice, it was seen that this leads to problems like slow convergence, model instability, and poor skill which is quite logical if you think.

- Thus, teacher forcing was introduced to rectify this.
- In which we feed the true output/token (and not the predicted output) from the previous time-step as input to the current time-step.
- That means the input to the time-step 2 will be y1_true=[0 1 0 0 0 0] and not y1_pred.
- Now the output at time-step 2 will be some random vector y2_pred. But at time-step 3 we will be using input as y2_true=[0 0 1 0 0 0] and not y2_pred. Similarly at each time-step, we will use the true output from the previous time-step.

- Finally, the loss is calculated on the predicted outputs from each time-step and the errors are backpropagated through time to update the parameters of the model. The loss function used is the categorical cross-entropy loss function between the target-sequence/Y_true and the predicted-sequence/Y_pred such that

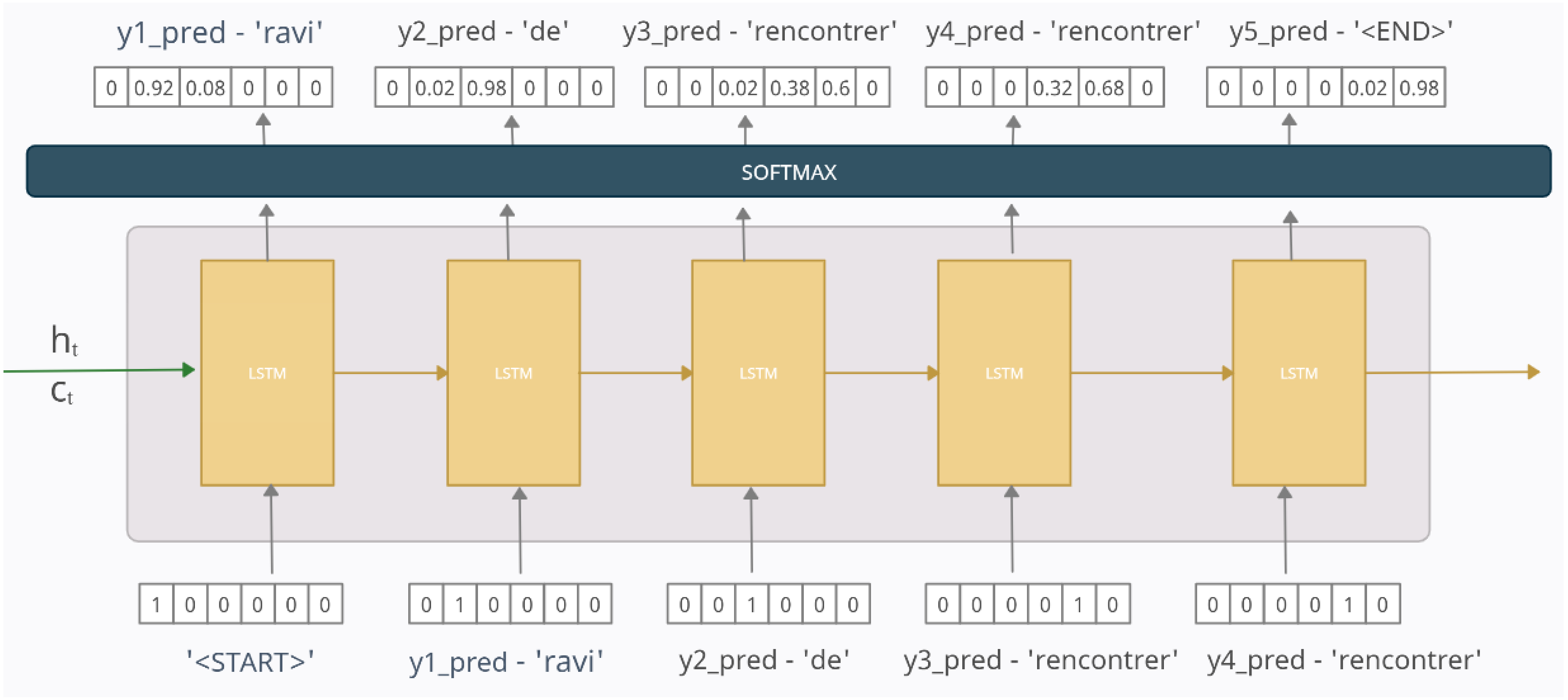
Y_true = [y0_true, y1_true, y2_true, y3_true, y4_true, y5_true]

Y_pred = ['<START>', y1_pred, y2_pred, y3_pred, y4_pred, y5_pred]

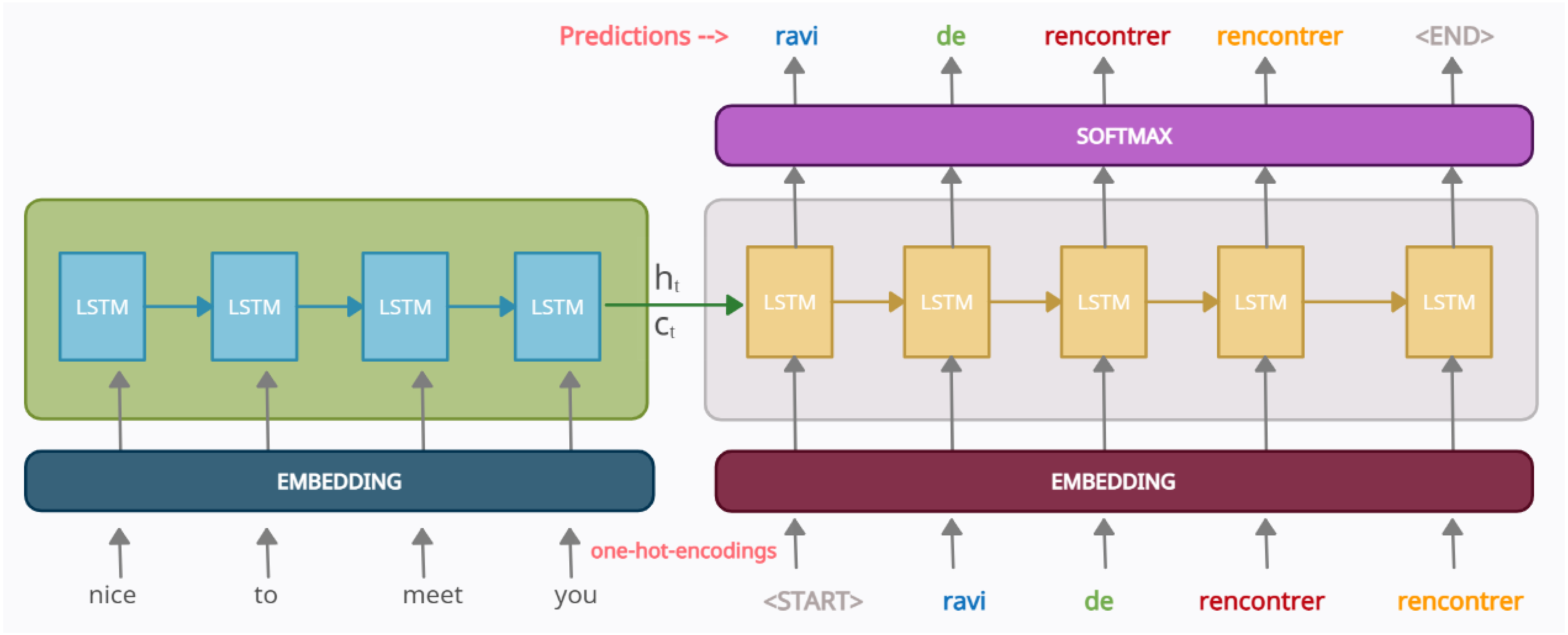
- The final states of the decoder are discarded

The Decoder in Test Phase

- In a real-world application, we won't have Y_true but only X.
- Thus we can’t use what we did in the training phase as we don’t have the target-sequence/Y_true. Thus when we are testing our model, the predicted output (and not the true output unlike the training phase) from the previous time-step is fed as input to the current time-step.


y1_pred - 'ravi'
y2_pred - 'de'
y3_pred - 'rencontrer'
y4_pred - 'rencontrer'
y5_pred - '<END>'
0 | 0.92 | 0.08 | 0 | 0 | 0
0 | 0.02 | 0.98 | 0 | 0 | 0
0 | 0 | 0.02 | 0.38 | 0.6 | 0
0 | 0 | 0 | 0.32 | 0.68 | 0
0 | 0 | 0 | 0 | 0.02 | 0.98
SOFTMAX
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
1 | 0 | 0 | 0 | 0 | 0
0 | 1 | 0 | 0 | 0 | 0
0 | 0 | 1 | 0 | 0 | 0
0 | 0 | 0 | 0 | 1 | 0
0 | 0 | 0 | 0 | 1 | 0
'<START>'
y1_pred - 'ravi'
y2_pred - 'de'
y3_pred - 'rencontrer'
y4_pred - 'rencontrer'

The Final Visualization at test time


Predictions -->
ravi
de
rencontrer
rencontrer
<END>
SOFTMAX
LSTM
LSTM
LSTM
LSTM
h_t
c_t
LSTM
LSTM
LSTM
LSTM
LSTM
EMBEDDING
EMBEDDING
one-hot-encodings
nice
to
meet
you
<START>
ravi
de
rencontrer
rencontrer