

[Sign in](#)[Get started](#)[Follow](#)

598K Followers

·

[Editors' Picks](#)[Features](#)[Deep Dives](#)[Grow](#)[Contribute](#)[About](#)

# Word Level English to Marathi Neural Machine Translation using Encoder-Decoder Model

A Guide to build Sequence to sequence models using LSTM



Harshall Lamba · Feb 9, 2019 · 13 min read

## Table of Contents

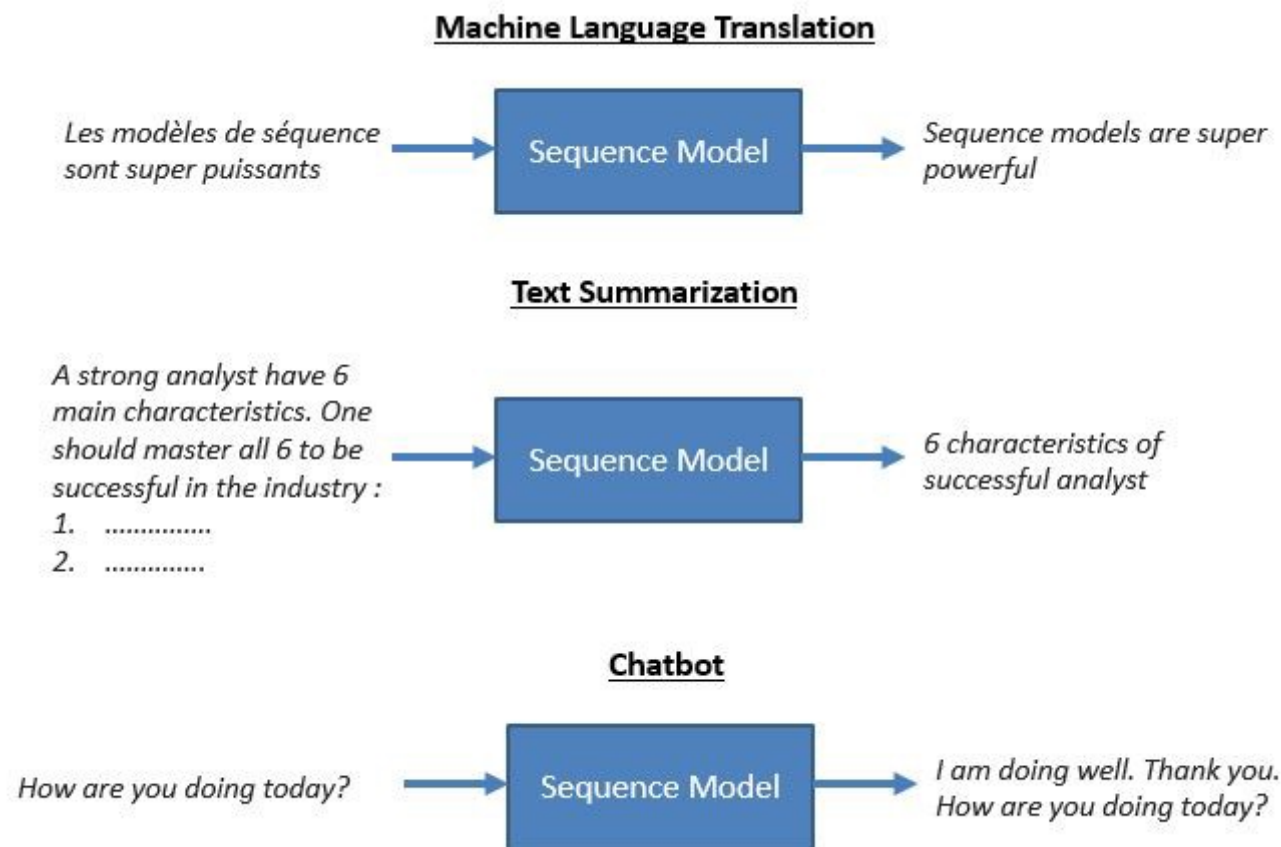
1. Introduction
2. Prerequisites
3. Encoder — Decoder Architecture
4. Encoder LSTM
5. Decoder LSTM — Training mode
6. Decoder LSTM — Inference mode
7. Code Walk through
8. Results and Evaluation
9. Future Work
10. End Notes
11. References

## 1. Introduction

Recurrent Neural Networks (or more precisely LSTM/GRU) have been found to be very effective in solving complex sequence related problems given a large amount of data. They have real time applications in speech recognition, Natural Language Processing (NLP) problems, time series forecasting, etc. This [blog](#) nicely explains some of these applications.

Sequence to Sequence (often abbreviated to seq2seq) models are a special class of Recurrent Neural Network architectures typically used (but not restricted) to solve complex Language related problems like Machine

Translation, Question Answering, creating Chat-bots, Text Summarization, etc.



Source <https://www.analyticsvidhya.com/blog/2018/04/sequence-modelling-an-introduction-with-practical-use-cases/>

The purpose of this blog post is to give a detailed explanation on how sequence to sequence models are built and to give an intuitive understanding of how they solve these complex tasks.

We will take the problem of Machine Translation (translating a text from one language to another, in our case from English to Marathi) as the running example in this blog. However the technical details apply to any sequence to sequence problem in general.

Since we are using Neural Networks to perform Machine Translation, more commonly it is called as Neural Machine translation (NMT).

## 2. Prerequisites

This post assumes that you:

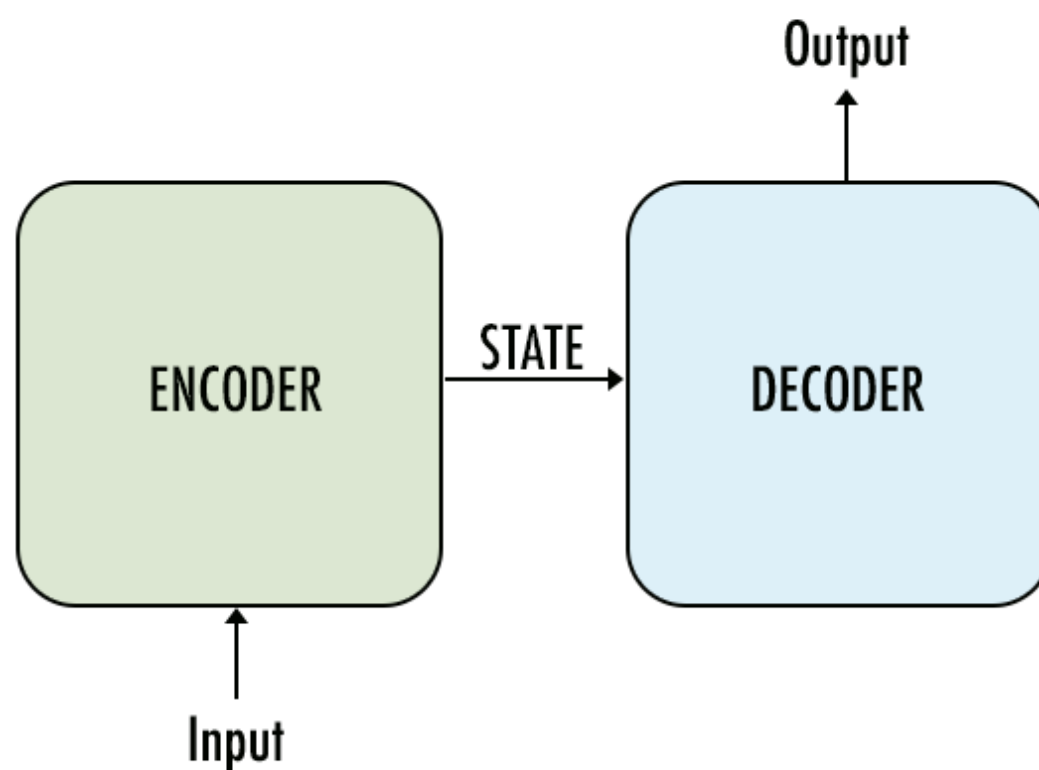
- Know Fundamental concepts in Machine Learning and Neural Networks
- Know High School Linear Algebra and Probability
- Have working knowledge of LSTM networks in Python and Keras

*The Unreasonable Effectiveness of Recurrent Neural Networks* (explains how RNNs can be used to build language models) and *Understanding LSTM Networks* (explains the working of LSTMs with solid intuition) are two

brilliant blogs that I strongly suggest to go through if you haven't. The concepts explained in these blogs are extensively used in my post.

### 3. Encoder — Decoder Architecture

The most common architecture used to build Seq2Seq models is the Encoder Decoder architecture. This is the one we will use for this post. Below is a very high level view of this architecture.



Source: <https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d>

Points to note:

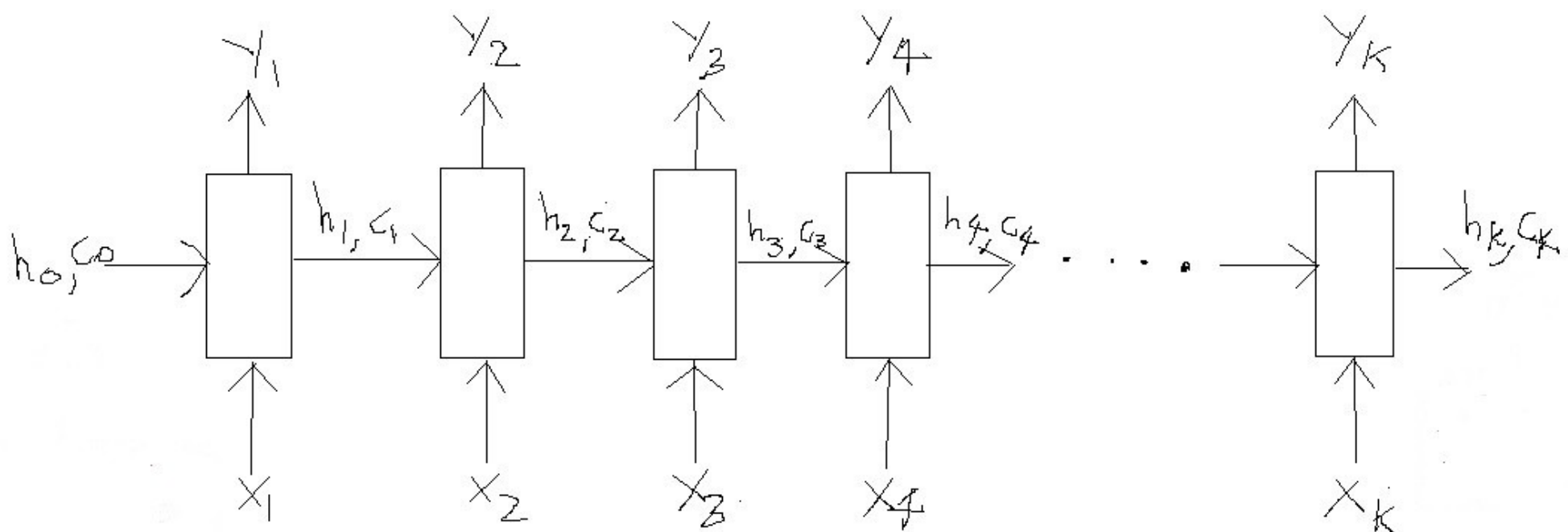
- Both encoder and the decoder are typically LSTM models (or sometimes GRU models)
- Encoder reads the input sequence and summarizes the information in something called as the internal state vectors (in case of LSTM these are called as the hidden state and cell state vectors). We discard the outputs of the encoder and only preserve the internal states.
- Decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM. Using these initial states, decoder starts generating the output sequence.
- The decoder behaves a bit differently during the training and inference procedure. During the training, we use a technique call teacher forcing which helps to train the decoder faster. During inference, the input to the decoder at each time step is the output from the previous time step.

e. Intuitively, the encoder summarizes the input sequence into state vectors (sometimes also called as Thought vectors), which are then fed to the decoder which starts generating the output sequence given the Thought vectors. The decoder is just a language model conditioned on the initial states.

Now we will understand all the above steps in detail by considering the example of translating an English sentence (input sequence) into its equivalent Marathi sentence (output sequence).

## 4. Encoder LSTM

This section provides a brief overview of the main components of the Encoder LSTM. I will keep this intuitive without going into the Mathematical arena. So here's what they are:



LSTM processing an input sequence of length 'k'

The LSTM reads the data one sequence after the other. Thus if the input is a sequence of length 'k', we say that LSTM reads it in 'k' time steps (think of this as a for loop with 'k' iterations).

Referring to the above diagram, below are the 3 main components of an LSTM:

a.  $X_i$  => Input sequence at time step  $i$

b.  $h_i$  and  $c_i$  => LSTM maintains two states ('h' for hidden state and 'c' for cell state) at each time step. Combined together these are internal state of the LSTM at time step  $i$ .

c.  $Y_i$  => Output sequence at time step  $i$

**Important:** Technically all of these components ( $X_i$ ,  $h_i$ ,  $c_i$  and  $Y_i$ ) are actually vectors of floating point numbers (explained below)

Let's try to map all of these in the context of our problem. Recall that our problem is to translate an English sentence to its Marathi equivalent. For the purpose of this blog we will consider the below example. Say, we have the following sentence

Input sentence (English) => "Rahul is a good boy"

Output sentence (Marathi) => "राहुल चांगला मुलगा आहे"

For now just focus on the input i.e. the English sentence

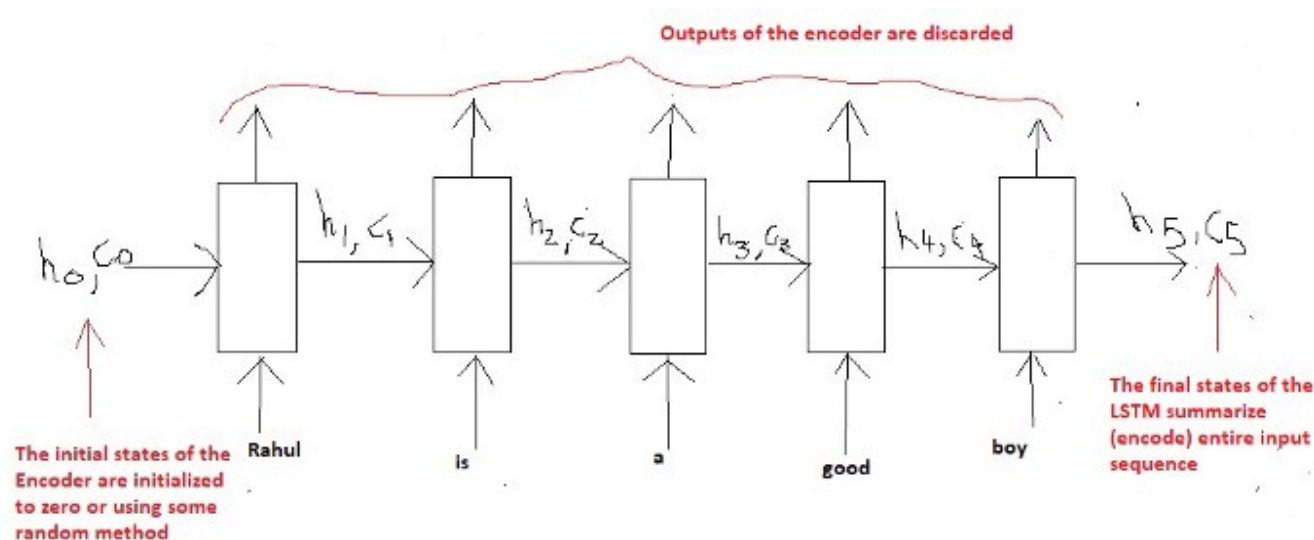
### Explanation for $X_i$ :

Now a sentence can be seen as a sequence of either words or characters. For example in case of words, the above English sentence can be thought of as a sequence of 5 words ('Rahul', 'is', 'a', 'good', 'boy'). And in case of characters, it can be thought of as a sequence of 19 characters ('R', 'a', 'h', 'u', 'l', ' ', 'i', 's', ' ', 'a', ' ', 'g', 'o', 'o', 'd', ' ', 'b', 'o', 'y').

We will break the sentence by words as this scheme is more common in real world applications. Hence the name 'Word Level NMT'. So, referring to the diagram above, we have the following input:

$X_1$  = 'Rahul',  $X_2$  = 'is',  $X_3$  = 'a',  $X_4$  = 'good',  $X_5$  = 'boy'.

The LSTM will read this sentence word by word in 5 time steps as follows



Encoder LSTM

But one question that we must answer is how to represent each  $X_i$  (each word) as a vector?

There are various word embedding techniques which map (embed) a word into a fixed length vector. I will assume the reader to be familiar with the concept of word embeddings and won't cover this topic in detail. However,

we will use the built-in Embedding Layer of the Keras API to map each word into a fixed length vector.

### Explanation for $h_i$ and $c_i$ :

The next question is what is the role of the internal states ( $h_i$  and  $c_i$ ) at each time step?

In very simple terms, they remember what the LSTM has read (learned) till now. For example:

$h_3, c_3 \Rightarrow$  These two vectors will remember that the network has read “Rahul is a” till now. Basically its the summary of information till time step 3 which is stored in the vectors  $h_3$  and  $c_3$  (thus called the states at time step 3).

Similarly, we can thus say that  $h_5, c_5$  will contain the summary of the entire input sentence, since this is where the sentence ends (at time step 5). These states coming out of the last time step are also called as the “**Thought vectors**” as they summarize the entire sequence in a vector form.

Then what about  $h_0, c_0$ ? These vectors are typically initialized to zero as the model has not yet started to read the input.

**Note:** The size of both of these vectors is equal to number of units (neurons) used in the LSTM cell.

### Explanation for $Y_i$ :

Finally, what about  $Y_i$  at each time step? These are the output (predictions) of the LSTM model at each time step.

But what type of a vector is  $Y_i$ ? More specifically in case of word level language models each  $Y_i$  is actually a probability distribution over the entire vocabulary which is generated by using a softmax activation. Thus each  $Y_i$  is a vector of size “vocab\_size” representing a probability distribution.

Depending on the context of the problem they might sometimes be used or sometimes be discarded.

In our case we have nothing to output unless we have read the entire English sentence. Because we will start generating the output sequence (equivalent Marathi sentence) once we have read the entire English sentence. Thus we will discard the  $Y_i$  of the Encoder for our problem.

### Summary of the encoder:



We will read the input sequence (English sentence) word by word and preserve the internal states of the LSTM network generated after the last time step  $h_k, c_k$  (assuming the sentence has 'k' words). These vectors (states  $h_k$  and  $c_k$ ) are called as the **encoding** of the input sequence, as they encode (summarize) the entire input in a vector form. Since we will start generating the output once we have read the entire sequence, outputs ( $Y_i$ ) of the Encoder at each time step are discarded.

Moreover you must also understand what type of vectors are  $X_i, h_i, c_i$  and  $Y_i$ . What are their sizes (shapes) and what do they represent. If you have any confusion understanding this part, then you need to first strengthen your understanding of LSTM and language models.

## 5. Decoder LSTM — Training Mode

Unlike the Encoder LSTM which has the same role to play in both the training phase as well as in the inference phase, the Decoder LSTM has a slightly different role to play in both of these phases.

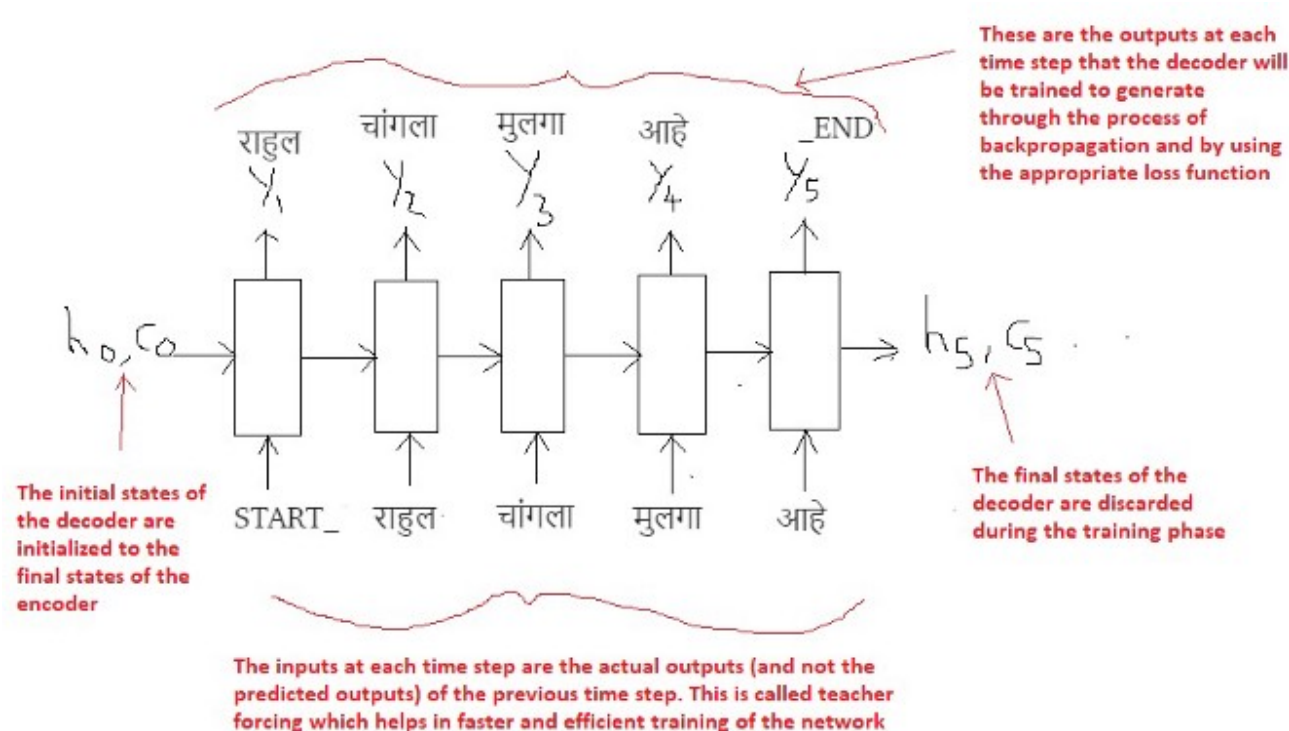
In this section we'll try to understand how to configure the Decoder during the training phase, while in the next section we'll understand how to use it during inference.

Recall that given the input sentence "Rahul is a good boy", the goal of the training process is to train (teach) the decoder to output "राहुल चांगला मुलगा आहे". Just as the Encoder scanned the input sequence word by word, similarly the Decoder will generate the output sequence word by word.

For some technical reasons (explained later) we will add two tokens in the output sequence as follows:

Output sequence => "START\_ राहुल चांगला मुलगा आहे \_END"

Now consider the diagram below:



## Decoder LSTM — Training Mode

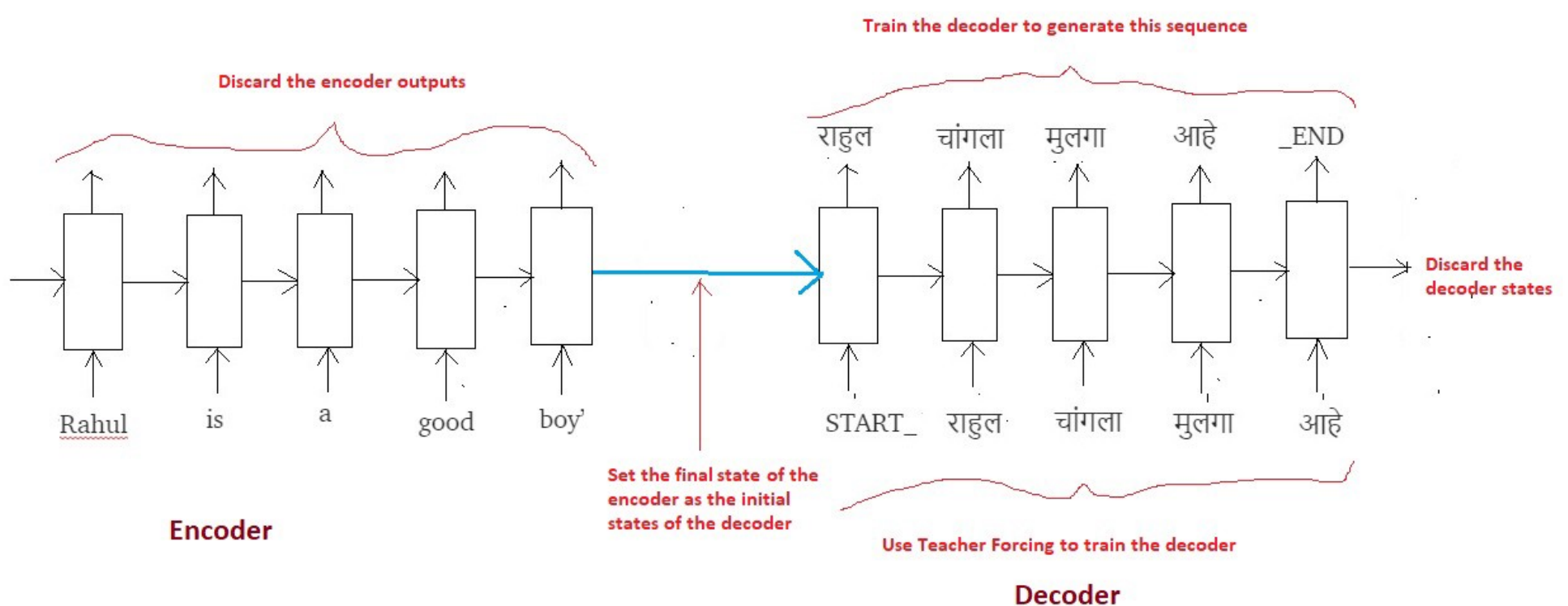
The most important point is that the initial states ( $h_0$ ,  $c_0$ ) of the decoder are set to the final states of the encoder. This intuitively means that the decoder is trained to start generating the output sequence depending on the information encoded by the encoder. Obviously the translated Marathi sentence must depend on the given English sentence.

In the first time step we provide the `START_` token so that the decoder starts generating the next token (the actual first word of Marathi sentence). And after the last word in the Marathi sentence, we make the decoder learn to predict the `_END` token. This will be used as the stopping condition during the inference procedure, basically it will denote the end of the translated sentence and we will stop the inference loop (more on this later).

We use a technique called “Teacher Forcing” wherein the input at each time step is given as the **actual** output (and not the predicted output) from the previous time step. This helps in more faster and efficient training of the network. To understand more about teacher forcing, refer this [blog](#).

Finally the loss is calculated on the predicted outputs from each time step and the errors are back propagated through time in order to update the parameters of the network. Training the network over longer period with sufficiently large amount of data results in pretty good predictions (translations) as we’ll see later.

The entire training process (Encoder + Decoder) can be summarized in the below diagram:



Summary of the training process



## 6. Decoder LSTM — Inference Mode

Let's now try to understand the setup required for inference. As already stated the Encoder LSTM plays the same role of reading the input sequence (English sentence) and generating the thought vectors ( $h_k, c_k$ ).

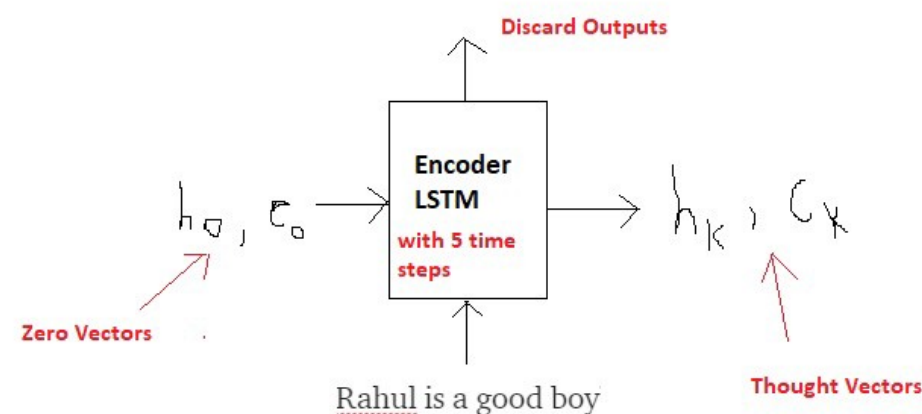
However, the decoder now has to predict the entire output sequence (Marathi sentence) given these thought vectors.

Let's try to visually understand by taking the same example.

Input sequence => "Rahul is a good boy"

(Expected) Output Sequence => "राहुल चांगला मुलगा आहे"

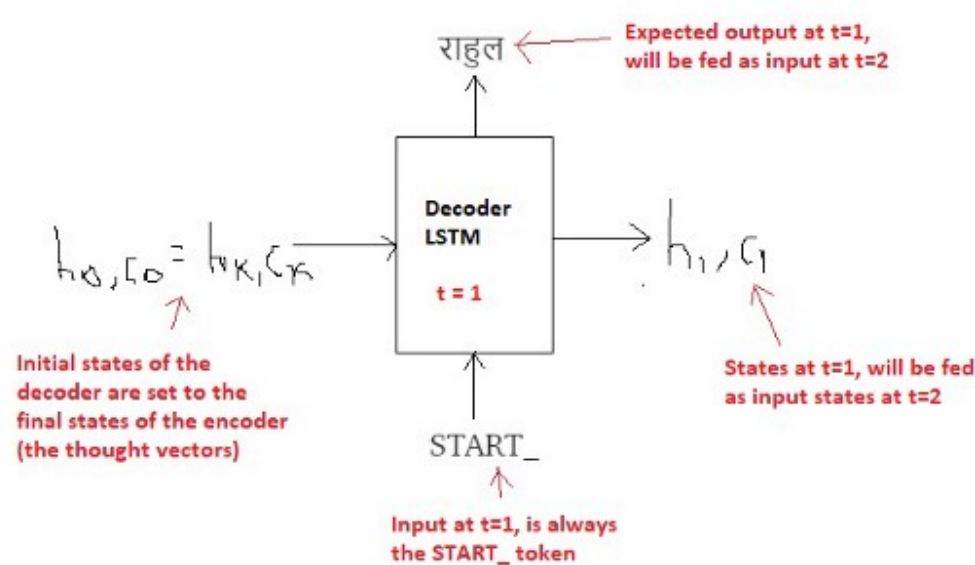
**Step 1:** Encode the input sequence into the Thought Vectors:



Encode the input sequence into thought vectors

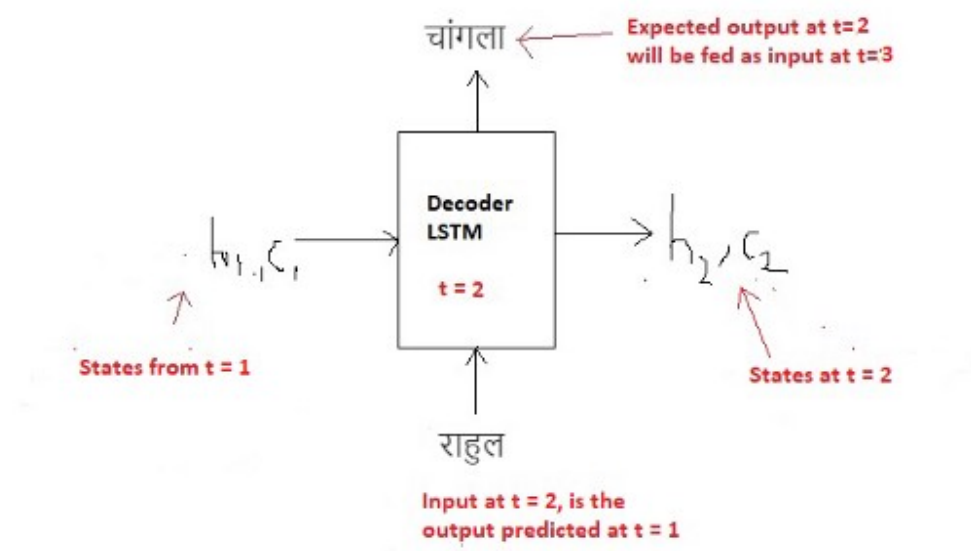
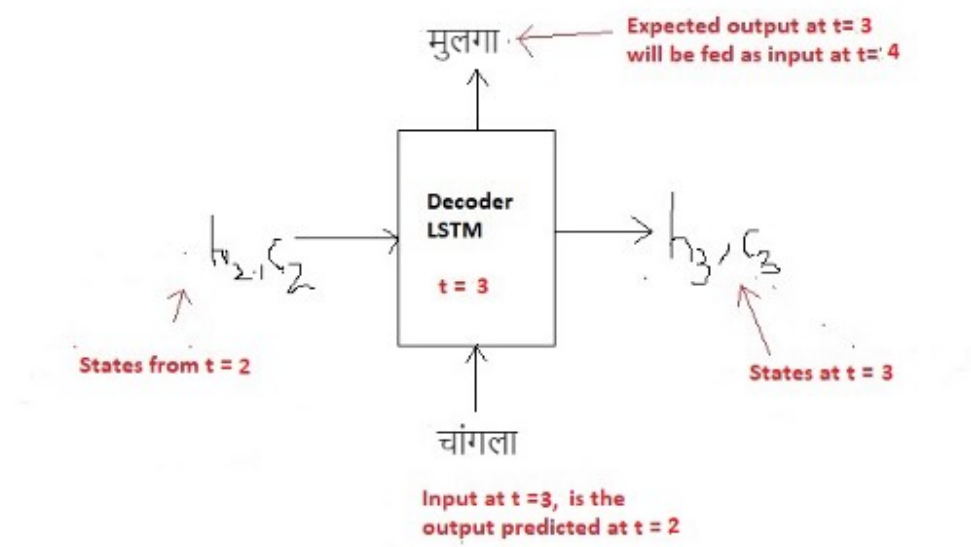
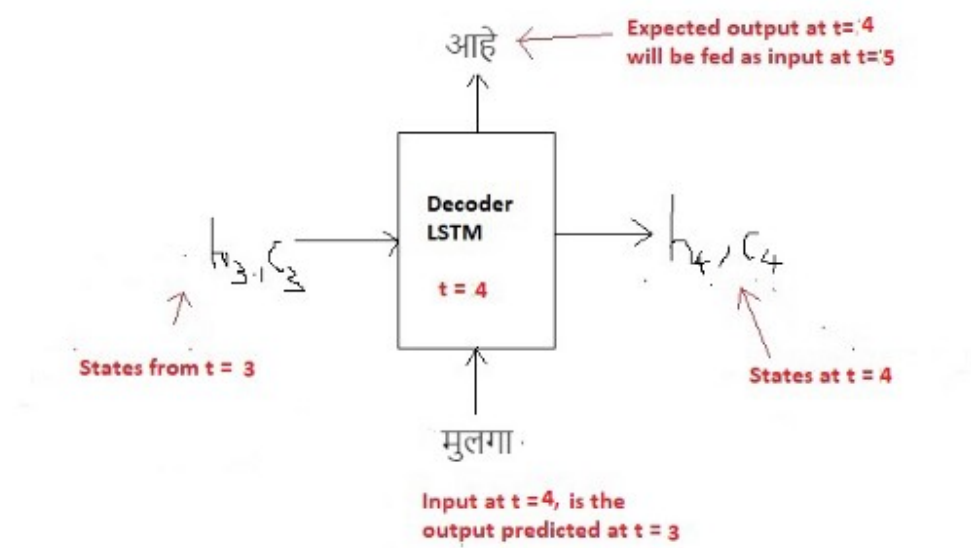
**Step 2:** Start generating the output sequence in a loop, word by word:

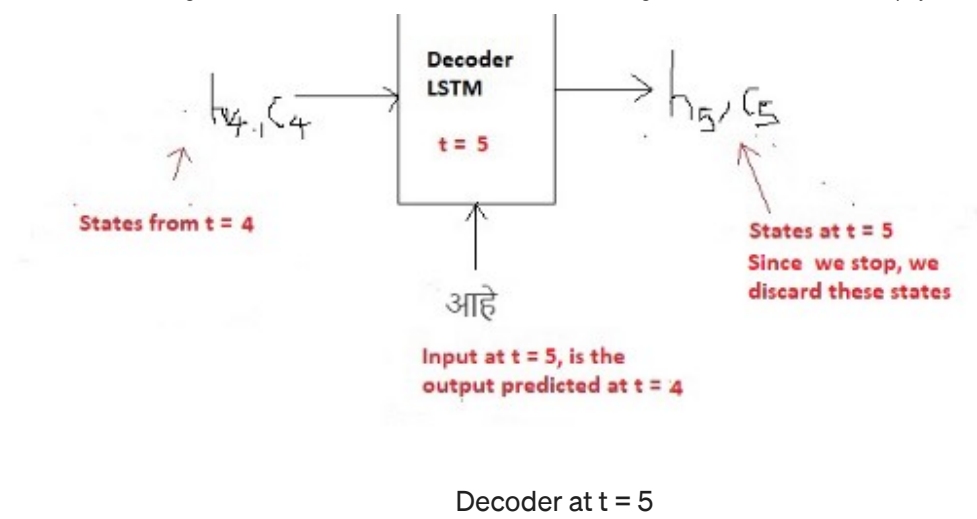
At  $t = 1$



Decoder at  $t=1$

At  $t = 2$

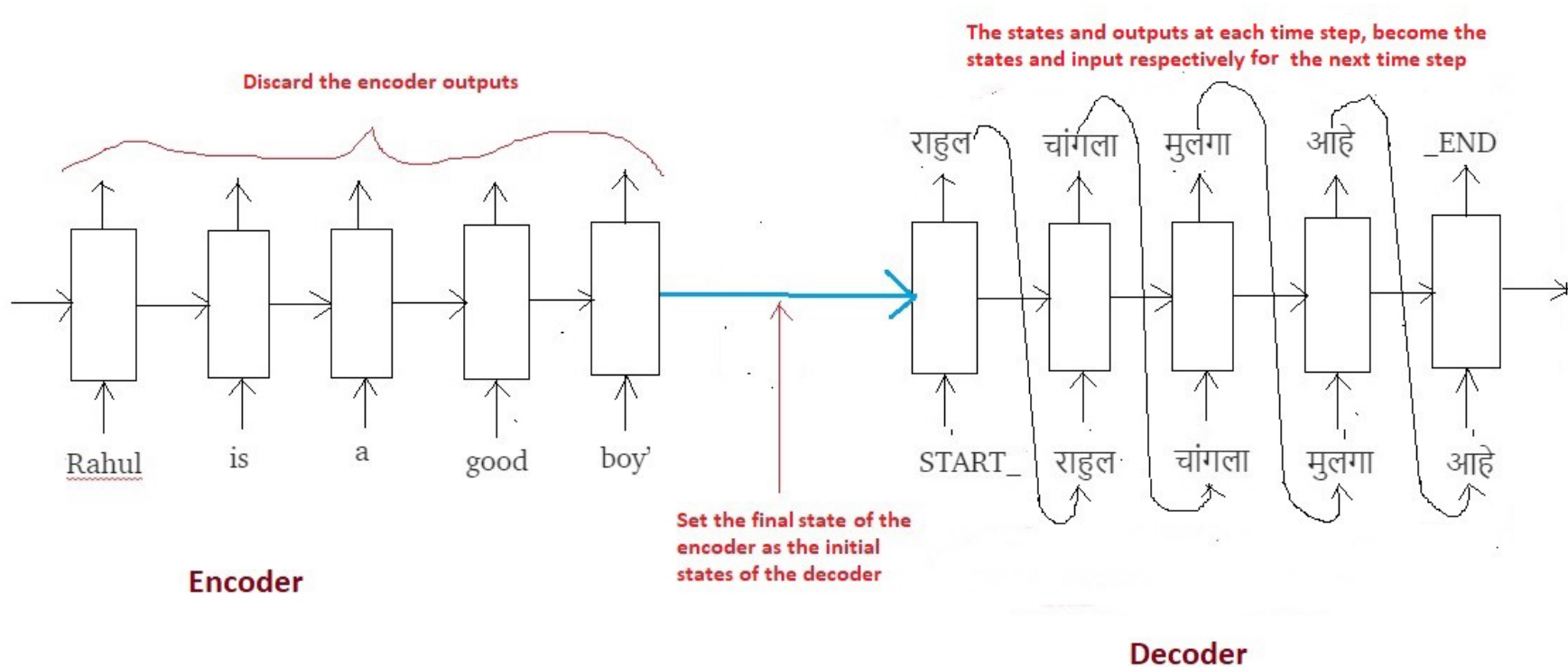
Decoder at  $t=2$ At  $t=3$ Decoder at  $t=3$ At  $t=4$ Decoder at  $t=4$ At  $t=5$ 



### Inference Algorithm:

- During inference, we generate one word at a time. Thus the Decoder LSTM is called in a loop, every time processing only one time step.
- The initial states of the decoder are set to the final states of the encoder.
- The initial input to the decoder is always the `START_` token.
- At each time step, we preserve the states of the decoder and set them as initial states for the next time step.
- At each time step, the predicted output is fed as input in the next time step.
- We break the loop when the decoder predicts the `END_` token.

The entire inference procedure can be summarized in the below diagram:



## 7. Code Walk through

Nothing beats the understanding developed when we actually implement the code, no matter how much efforts are put in to understand the theory (that does not however mean that we do not discuss any theory, but what I mean to say is theory must always be followed by implementation).

## Dataset:

Download and unzip mar-eng.zip file from [here](#).

Before we start building the models, we need to perform some data cleaning and preparation. Without going into too much details, I will assume the reader to understand the below (self explanatory) steps that are usually a part of any language processing project.

```

1  lines= pd.read_table('mar.txt', names=['eng', 'mar'])
2
3  # Lowercase all characters
4  lines.eng=lines.eng.apply(lambda x: x.lower())
5  lines.mar=lines.mar.apply(lambda x: x.lower())
6
7  # Remove quotes
8  lines.eng=lines.eng.apply(lambda x: re.sub('"', '', x))
9  lines.mar=lines.mar.apply(lambda x: re.sub('"', '', x))
10 exclude = set(string.punctuation) # Set of all special characters
11
12 # Remove all the special characters
13 lines.eng=lines.eng.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
14 lines.mar=lines.mar.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
15
16 # Remove all numbers from text
17 remove_digits = str.maketrans('', '', digits)
18 lines.eng=lines.eng.apply(lambda x: x.translate(remove_digits))
19 lines.mar = lines.mar.apply(lambda x: re.sub("[२३०८१५७९४६]", "", x))
20
21 # Remove extra spaces
22 lines.eng=lines.eng.apply(lambda x: x.strip())
23 lines.mar=lines.mar.apply(lambda x: x.strip())
24 lines.eng=lines.eng.apply(lambda x: re.sub(" +", " ", x))
25 lines.mar=lines.mar.apply(lambda x: re.sub(" +", " ", x))
26
27 # Add start and end tokens to target sequences
28 lines.mar = lines.mar.apply(lambda x : 'START_ '+ x + ' _END')
```

data\_cleaning.py hosted with ❤ by GitHub

[view raw](#)

Code to perform Data Cleaning

Below we compute the vocabulary for both English and Marathi. We also compute the vocabulary sizes and the length of maximum sequence for both the languages. Finally we create 4 Python dictionaries (two for each language) to convert a given token into an integer index and vice-versa.

```

1  # Vocabulary of English
2  all_eng_words=set()
3  for eng in lines.eng:
4      for word in eng.split():
5          if word not in all_eng_words:
6              all_eng_words.add(word)
7
8  # Vocabulary of French
```

```

9  all_marathi_words=set()
10 for mar in lines.mar:
11     for word in mar.split():
12         if word not in all_marathi_words:
13             all_marathi_words.add(word)
14
15 # Max Length of source sequence
16 lenght_list=[]
17 for l in lines.eng:
18     lenght_list.append(len(l.split(' ')))
19 max_length_src = np.max(lenght_list)
20
21 # Max Length of target sequence
22 lenght_list=[]
23 for l in lines.mar:
24     lenght_list.append(len(l.split(' ')))
25 max_length_tar = np.max(lenght_list)
26
27 input_words = sorted(list(all_eng_words))
28 target_words = sorted(list(all_marathi_words))
29
30 # Calculate Vocab size for both source and target
31 num_encoder_tokens = len(all_eng_words)
32 num_decoder_tokens = len(all_marathi_words)
33 num_decoder_tokens += 1 # For zero padding
34
35 # Create word to token dictionary for both source and target
36 input_token_index = dict([(word, i+1) for i, word in enumerate(input_words)])
37 target_token_index = dict([(word, i+1) for i, word in enumerate(target_words)])
38
39 # Create token to word dictionary for both source and target
40 reverse_input_char_index = dict((i, word) for word, i in input_token_index.items())
41 reverse_target_char_index = dict((i, word) for word, i in target_token_index.items())

```

data\_prep\_nmt.py hosted with ❤ by GitHub

[view raw](#)

## Code for Data Preparation

Then we make a 90–10 train and test split and write a Python generator function to load the data in batches as follows:

```

1  def generate_batch(X = X_train, y = y_train, batch_size = 128):
2      ''' Generate a batch of data '''
3      while True:
4          for j in range(0, len(X), batch_size):
5              encoder_input_data = np.zeros((batch_size, max_length_src),dtype='float32')
6              decoder_input_data = np.zeros((batch_size, max_length_tar),dtype='float32')
7              decoder_target_data = np.zeros((batch_size, max_length_tar, num_decoder_tokens),dtype='float32')
8              for i, (input_text, target_text) in enumerate(zip(X[j:j+batch_size], y[j:j+batch_size])):
9                  for t, word in enumerate(input_text.split()):
10                     encoder_input_data[i, t] = input_token_index[word] # encoder input seq
11                 for t, word in enumerate(target_text.split()):
12                     if t<len(target_text.split())-1:
13                         decoder_input_data[i, t] = target_token_index[word] # decoder input seq
14                     if t>0:
15                         # decoder target sequence (one hot encoded)
16                         # does not include the START_ token
17                         # Offset by one timestep
18                         decoder_target_data[i, t - 1, target_token_index[word]] = 1.
19             yield([encoder_input_data, decoder_input_data], decoder_target_data)

```

◀

▶

batch\_generator\_nmt.py hosted with ❤ by GitHub

[view raw](#)

## Code for loading Batches of Data

Then we define the model required for training as follows:

```
1  # Encoder
2  encoder_inputs = Input(shape=(None,))
3  enc_emb = Embedding(num_encoder_tokens, latent_dim, mask_zero = True)(encoder_inputs)
4  encoder_lstm = LSTM(latent_dim, return_state=True)
5  encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
6
7  # We discard `encoder_outputs` and only keep the states.
8  encoder_states = [state_h, state_c]
9
10 # Set up the decoder, using `encoder_states` as initial state.
11 decoder_inputs = Input(shape=(None,))
12 dec_emb_layer = Embedding(num_decoder_tokens, latent_dim, mask_zero = True)
13 dec_emb = dec_emb_layer(decoder_inputs)
14
15 # We set up our decoder to return full output sequences,
16 # and to return internal states as well. We don't use the
17 # return states in the training model, but we will use them in inference.
18 decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
19 decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
20
21 # Use a softmax to generate a probability distribution over the target vocabulary for each time
22 decoder_dense = Dense(num_decoder_tokens, activation='softmax')
23 decoder_outputs = decoder_dense(decoder_outputs)
24
25 # Define the model that will turn
26 # `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
27 model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
28
29 # Compile the model
30 model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
```

train\_model\_nmtl.py hosted with ❤ by GitHub

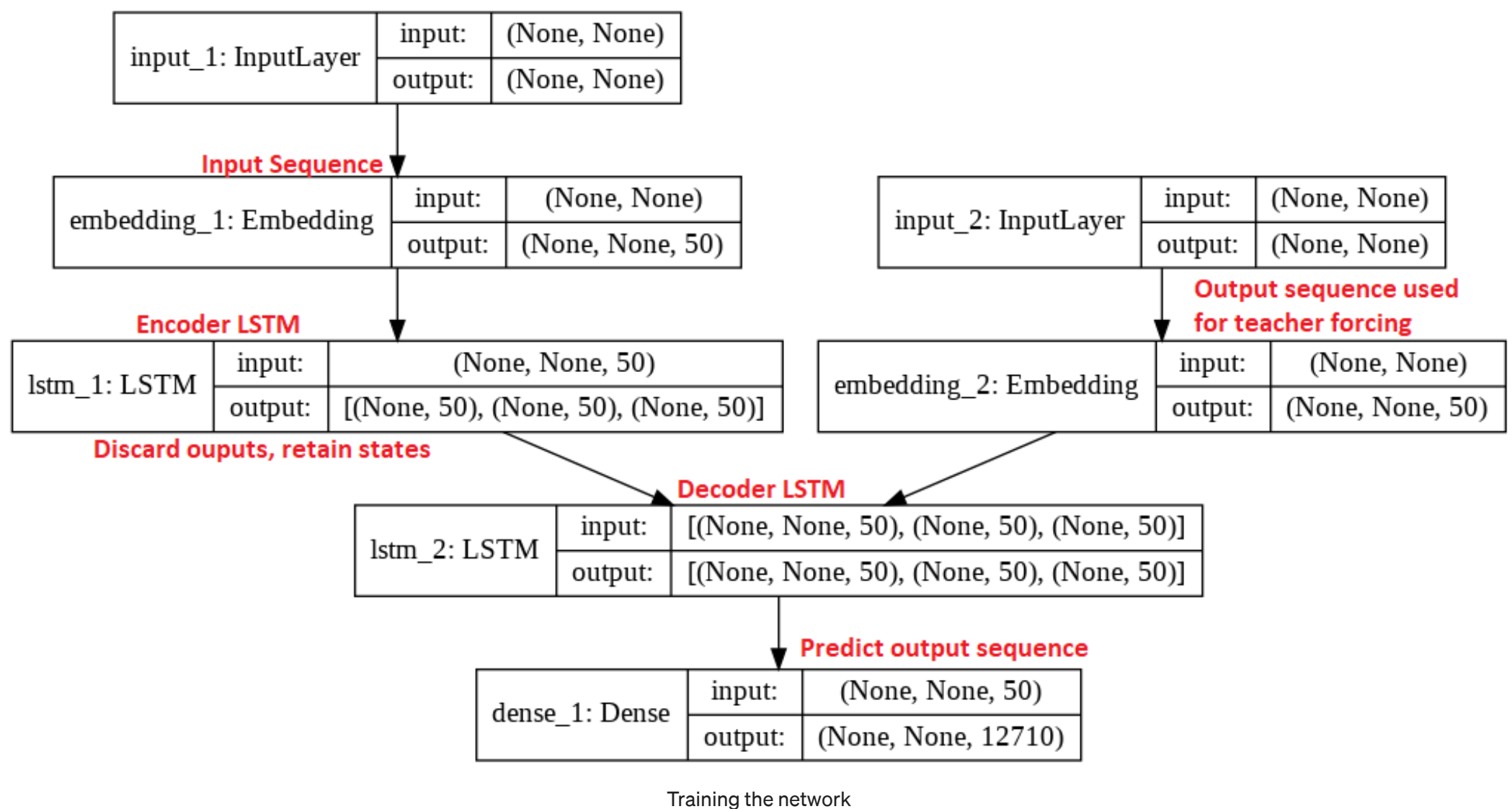
[view raw](#)

Code to define the Model to be trained

You should be able to conceptually connect each and every line with the explanation I have provided in sections 4 and 5 above.

Let's look at the model architecture generated from the plot\_model utility of the Keras.





We train the network for 50 epochs with a batch size of 128. On a P4000 GPU, it takes slightly more than 2 hours to train.

## Inference Setup:

```

1  # Encode the input sequence to get the "thought vectors"
2  encoder_model = Model(encoder_inputs, encoder_states)
3
4  # Decoder setup
5  # Below tensors will hold the states of the previous time step
6  decoder_state_input_h = Input(shape=(latent_dim,))
7  decoder_state_input_c = Input(shape=(latent_dim,))
8  decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
9
10 # Get the embeddings of the decoder sequence
11 dec_emb2= dec_emb_layer(decoder_inputs)
12
13 # To predict the next word in the sequence, set the initial states to the states from the previous time step
14 decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_input)
15 decoder_states2 = [state_h2, state_c2]
16
17 # A dense softmax layer to generate prob dist. over the target vocabulary
18 decoder_outputs2 = decoder_dense(decoder_outputs2)
19
20 # Final decoder model
21 decoder_model = Model(
22     [decoder_inputs] + decoder_states_inputs,
23     [decoder_outputs2] + decoder_states2)

```

inference\_setup\_nmt.py hosted with ❤ by GitHub

[view raw](#)

Code for setting up Inference

Finally, we generate the output sequence by invoking the above setup in a loop as follows:

```

1  def decode_sequence(input_seq):
2      # Encode the input as state vectors.
3      states_value = encoder_model.predict(input_seq)
4
5      # Generate empty target sequence of length 1.
6      target_seq = np.zeros((1,1))
7
8      # Populate the first character of target sequence with the start character.
9      target_seq[0, 0] = target_token_index['START_']
10
11     # Sampling loop for a batch of sequences
12     # (to simplify, here we assume a batch of size 1).
13     stop_condition = False
14     decoded_sentence = ''
15
16     while not stop_condition:
17         output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
18         # Sample a token
19         sampled_token_index = np.argmax(output_tokens[0, -1, :])
20         sampled_char = reverse_target_char_index[sampled_token_index]
21         decoded_sentence += ' '+sampled_char
22
23         # Exit condition: either hit max length or find stop token.
24         if (sampled_char == '_END' or len(decoded_sentence) > 50):
25             stop_condition = True
26
27         # Update the target sequence (of length 1).
28         target_seq = np.zeros((1,1))
29         target_seq[0, 0] = sampled_token_index
30
31         # Update states
32         states_value = [h, c]
33
34     return decoded_sentence

```

decode\_sequence.py hosted with ❤ by GitHub

[view raw](#)

Code to decode the output sequence in a loop

At this point you must be able to conceptually connect each and every line of the code in the above two blocks with the explanation provided in section 6.

## 8. Results and Evaluation

The purpose of this blog post was to give an intuitive explanation on how to build basic level sequence to sequence models using LSTM and not to develop a top quality language translator. So keep in mind that the results are not world class (and you don't start comparing with google translate) for many reasons. The most important reason being is that the dataset size is very small, only 33000 pairs of sentences (yes these are too few). If you want to improve the quality of translations, I will list some suggestions towards the end of this blog. However for now, let's see some results generated from the above model (they are not too bad either).

### Evaluation on train dataset:

```

1  Input English sentence: it is a holiday tomorrow
2  Actual Marathi Translation:  उद्या सुट्टी आहे
3  Predicted Marathi Translation:  उद्या नाताळ आहे

```

|  |   |
|--|---|
| 4  |   |
| 5  | Input English sentence: i will give you this book                       |
| 6  | Actual Marathi Translation: मी तुम्हाला हे पुस्तक देईन                  |
| 7  | Predicted Marathi Translation: मी तुला हे पुस्तक तुला दिलं              |
| 8  |   |
| 9  | Input English sentence: this sentence has five words                    |
| 10   | Actual Marathi Translation: या वाक्यात पाच शब्द आहेत                    |
| 11   | Predicted Marathi Translation: या पाच शब्द आहेत                         |
| 12   |   |
| 13   | Input English sentence: did you clean your room                         |
| 14   | Actual Marathi Translation: तुझी खोली साफ केलीस का                      |
| 15   | Predicted Marathi Translation: तुझी खोली साफ केली का                    |
| 16   |   |
| 17   | Input English sentence: she wrapped herself in a blanket                |
| 18   | Actual Marathi Translation: त्यांनी स्वतःला एका चादरीत गुंडाळून घेतलं   |
| 19   | Predicted Marathi Translation: तिने एका छोट्या चादर घातली               |
| 20   |   |
| 21   | Input English sentence: he is still alive                               |
| 22   | Actual Marathi Translation: तो अजूनही जिवंत आहे                         |
| 23   | Predicted Marathi Translation: ते अजूनही जिवंत आहेत                     |
| 24   |   |
| 25   | Input English sentence: you cant speak french can you                   |
| 26   | Actual Marathi Translation: तुला फ्रेंच बोलता येत नाही ना               |
| 27   | Predicted Marathi Translation: तुला फ्रेंच बोलता येत नाही का नाही माहित |
| eval_train_nmt hosted with ❤ by GitHub <span>view raw</span> |   |

Evaluation on Training Dataset

Evaluation on test dataset:

|   |   |
|---|---|
| 1   | Input English sentence: my wife isnt beautiful yours is             |
| 2   | Actual Marathi Translation: माझी बायको सुंदर नाहीये तुझी आहे        |
| 3   | Predicted Marathi Translation: माझी तुझी गर्लफ्रेंड सुंदर आहे भेटलो |
| 4   |   |
| 5   | Input English sentence: who lives in this house                     |
| 6   | Actual Marathi Translation: या घरात कोण राहतं                       |
| 7   | Predicted Marathi Translation: या घरात कोणी असतो                    |
| 8   |   |
| 9   | Input English sentence: somethings happened to tom                  |
| 10  | Actual Marathi Translation: टॉमला काहीतरी झालंय                     |
| 11  | Predicted Marathi Translation: टॉमला काहीतरी झालं आहे               |
| 12  |   |
| 13  | Input English sentence: the dog jumped over a chair                 |
| 14  | Actual Marathi Translation: कुत्र्याने खुर्चीवरून उडी मारली         |
| 15  | Predicted Marathi Translation: एक कुत्र्याला दरवाजा बंद केला        |
| 16  |   |
| 17  | Input English sentence: i can prove who the murderer is             |
| 18  | Actual Marathi Translation: खुनी कोण आहे हे मी सिद्ध करू शकतो       |
| 19  | Predicted Marathi Translation: मी जिंकू शकतो हे मला माहित आहे       |
| 20  |   |
| 21  | Input English sentence: everyone could hear what tom said           |
| 22  | Actual Marathi Translation: टॉम जे म्हणाला ते सर्वांना ऐकू येत होतं |
| 23  | Predicted Marathi Translation: टॉमने काय म्हटलं म्हटलं तर खरं       |
| 24  |   |
| 25  | Input English sentence: those are my books                          |
| 26  | Actual Marathi Translation: ती माझी पुस्तकं आहेत                    |
| 27  | Predicted Marathi Translation: ती माझी पुस्तकं आहेत                 |
| eval_test_nmt hosted with ❤ by GitHub <span>view raw</span> |   |

Evaluation on Test Dataset

What can we conclude?

Even though the results are not the best, they are not that bad as well. Certainly much better than what a randomly generated sequence would result in. In some sentences we can even note that the words predicted are not correct but they are semantically quite close to the correct words.

Also, another point to be noticed is that the results on training set are a bit better than the results on test set, which indicates that the model might be over-fitting a bit.

## 9. Future Work

If you are interested to improve the quality, you can try out below measures:

- a. Get much more data. Top quality translators are trained on millions of sentence pairs.
- b. Build more complex models like Attention.
- c. Use dropout and other forms of regularization techniques to mitigate over-fitting.
- d. Perform Hyper-parameter tuning. Play with learning rate, batch size, dropout rate, etc. Try using bidirectional Encoder LSTM. Try using multi-layered LSTMs.
- e. Try using beam search instead of a greedy approach.
- f. Try BLEU score to evaluate your model.
- g. The list is never ending and goes on.

## 10. End Notes

If the article appeals you, do provide some comments, feedback, constructive criticism, etc.

Full code on my GitHub repo [here](#).

If you like my explanations, you can follow me as I plan to release some more interesting blogs related to Deep Learning and AI.

You can connect on LinkedIn [here](#)

## 11. References

- a. <https://medium.com/@dev.elect.iitd/neural-machine-translation-using-word-level-seq2seq-model-47538cba8cd7>

b. <https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html>

c. <https://arxiv.org/abs/1409.3215>

d. <https://arxiv.org/abs/1406.1078>

---

## Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Get this newsletter

[Machine Learning](#)   [Deep Learning](#)   [Lstm](#)   [Seq2seq](#)   [Machine Translation](#)

[About](#)   [Write](#)   [Help](#)   [Legal](#)