Project Report on

**BLOOD BANK**

for course Cloud Computing Lab by

Name of student Class Roll No.

Ravi Kumar BE-3 19

Deep Mehta BE-3 21

Under the guidance of

PROF. PRIYANKA GHULE



**DEPARTMENT OF COMPUTER ENGINEERING**

**SHAH AND ANCHOR KUTCHHI ENGINEERING COLLEGE**

CHEMBUR, MUMBAI – 400088.

2019 – 2020

ABSTRACT

Web-based Blood Donation Management System is a management system website that enables individuals who want to donate blood to help the needy. It also enables hospitals to record and store the data for people who want to communicate with them, and it also provides a centralized blood bank database. The system is developed by using HTML, Flask, and MongoDB as a database system to manage and store the data. The application is deployed on Heroku Cloud Service for web hosting. The system targets 2 types of users: registered users who wants to donate blood, the recipients who can search and request for blood. A blood center acts as a central link between both the users. The main objectives for developing the website is to educate the community on the benefits of blood donation, develop a Web-Based Blood Bank System to manage the records of donors and recipients, and encourage voluntary blood donation, easily accessing any information about blood type and the distribution of the blood in various hospitals in a given area.

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| SR.NO | TOPIC | PAGE NO |
| 1 | INTRODUCTION | 1 |
| 2 | LITERATURE SURVEY | 3 |
| 3 | PROPOSED SYSTEM | 4 |
| 4 | IMPLEMENTATION | 10 |
| 5 | RESULTS AND ANALYSIS | 16 |
| 6 | CONCLUSION AND FUTURE SCOPE | 18 |
| 7 | REFERENCES | 19 |

**CHAPTER 1**

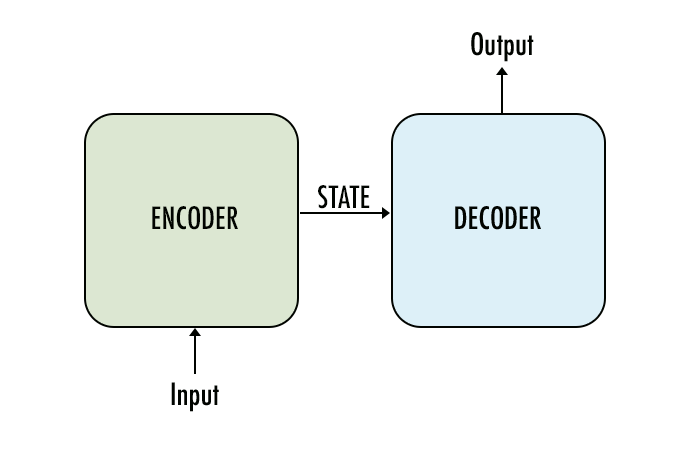
**INTRODUCTION**

Machine Translation (MT) can be described as the task of translating text or speech from one natural language to another, with as little human effort as possible. MT aims to achieve quality translations which are semantically equivalent to the source sentence and syntactically correct in the target language. MT performs simple substitution of words on a ground level, but that alone is not enough, as recognition of whole phrases and their closest counter-parts in the target language are necessary. Different approaches have been proposed for MT rule based translation, knowledge based translation, corpus based translation and hybrid translation . Each of these approaches has its own advantages and disadvantages.

Statistical Machine Translation (SMT) (which can be subcategorized under corpus based translation) was widely used as it produced better results compared to other methods. SMT also requires little human intervention, as it learns everything from the parallel corpus. Neural networks in MT are also popular lately—a novel technique in MT called Neural Machine Translation (NMT) has emerged. Though a lot of work has been done on MT, it is limited to European and other foreign languages. Little has been done on Indian languages. We find no work giving benchmarks on Indian language pairs, which makes us work on the translation from European language to Indian language.

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. 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.

The most common architecture used to build Seq2Seq models is the Encoder Decoder architecture. The one we will use for this post is as shown in figure 1.1

**Figure 1.1 - Encoder — Decoder Architecture**

**ENCODER-DECODER ARCHITECTURE**

a. Both encoder and the decoder are typically LSTM models.

b. 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.

c. 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.

d. The decoder behaves a bit differently during the training and inference procedure. During the training, we use a technique ca ll 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

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1: SURVEY EXISTING SYSTEM**

Deep Neural Networks (DNNs) are extremely powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition and visual object recognition. DNNs are powerful because they can perform arbitrary parallel computation for a modest number of steps. A surprising example of the power of DNNs is their ability to sort N N-bit numbers using only 2 hidden layers of quadratic size. So, while neural networks are related to conventional statistical models, they learn an intricate computation. Furthermore, large DNNs can be trained with supervised backpropagation whenever the labelled training set has enough information to specify the network’s parameters. Thus, if there exists a parameter setting of a large DNN that achieves good results (for example, because humans can solve the task very rapidly), supervised backpropagation will find these parameters and solve the problem.

**2.2: LIMITATION OF EXISTING SYSTEM**

Despite their flexibility and power, DNNs can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality. It is a significant limitation, since many important problems are best expressed with sequences whose lengths are not known a-priori. For example, speech recognition and machine translation are sequential problems. Likewise, question answering can also be seen as mapping a sequence of words representing the question to a 1 sequence of words representing the answer. It is therefore clear that a domain-independent method that learns to map sequences to sequences would be useful.

**2.3: PROBLEM STATEMENT**

The aim of our project is to translate a given sentence in European Language English to Indian Language Marathi. The technologies used in our project ENGLISH TO MARATHI NEURAL MACHINE TRANSLATION are RNN, LSTM, ENCODER-DECODER ARCHITECTURE.

**2.4: OBJECTIVE**

Language translators are very important in day to day life as we presented basic translator which is sentence to sentence translator. In an office there are many circulars and document which has to translate into local language. In that there is need to translate documents with less human interference.

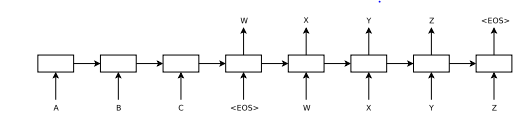
**CHAPTER 3**

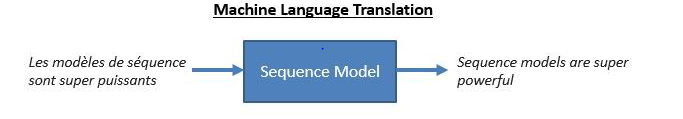
**PROPOSED SYSTEM**

**3.1 ANALYSIS/FRAMEWORK/ALGORITHM**

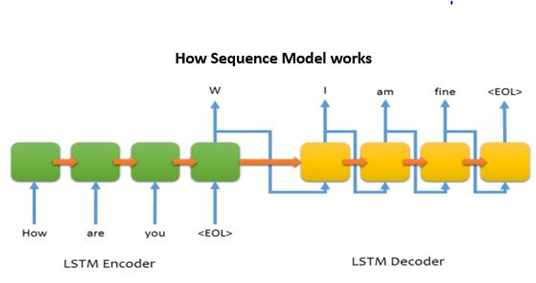
**ANALYSIS:**

In this report, we show that a straightforward application of the Long Short-Term Memory (LSTM) architecture [16] can solve general sequence to sequence problems. The idea is to use one LSTM to read the input sequence, one time step at a time, to obtain large fixed dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector (fig. 3.1). The second LSTM is essentially a recurrent neural network language model [28, 23, 30] except that it is conditioned on the input sequence. The LSTM’s ability to successfully learn on data with long range temporal dependencies makes it a natural choice for this application due to the considerable time lag between the inputs and their corresponding outputs (fig. 3.1).

**Figure 3.1.a: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token.**

**ALGORITHM:** 

**Figure 3.1.b.**



**Figure 3.1.c**

**Seq to Seq**:

For Encoder and Decoder which uses LSTM cell has hidden vector size of 30. Encoder takes inputs from embedding layer which is size 50 and from previous hidden and cell state of encoder and output new hidden and cell vectors:- Decoder takes inputs from previous dense layer and from attention layer concatenate it and form vector of (50+30) dimension inputs size where 50 is target embedding size and 30 is attention vector and from previous hidden and cell state of encoder and output new hidden and cell vectors.

**Attention mechanism**:

This layer process for every time step at decoder and it takes inputs from encoder network for all sequences and from decoder network for particular time step as a context vector and applying following operation which given in equation below for computation alpha and it use as a weights and compute weighted sum of all hidden vector in encoder sequence for every time step this implies hidden vector in encoder network which has high alpha has high contribution means high attention.

**3.2: DETAILS OF HARDWARE AND SOFTWARE**

**SOFTWARE**

1. Windows 7, Windows 8 or Windows 10
2. Mac OSX 10.8, 10.9, 10.10 or 10.11
3. Chrome/ IE/ Mozilla/ Safari
4. Keras.
5. Tensor.

**Hardware:**

1. Processor (CPU) with 2 gigahertz (GHz) frequency or above
2. A minimum of 4 GB of RAM
3. Internet Connection Broadband (high-speed) Internet connection with a speed of 4 Mbps or higher
4. Keyboard and a Microsoft Mouse or some other compatible pointing device
5. Windows system with 16 GB RAM
6. CUDA dependencies for GPU acceleration
7. CUDA-enabled Nvidia graphical processing unit (GPU)

**3.3: METHODOLOGY**

**DATA PRE-PROCESSING**

1. We processed the dataset so as to remove the punctuations present there using
2. Regular expression library.
3. Spell Corrections for English words: - Using autocorrect library in order to make
4. Corrections, if any Misspelled words are present.
5. Convert all upper-case letter to lower case counterparts in order to make our dataset homogeneous.
6. Remove all the special characters.
7. Remove all single and double quotes if any.
8. Remove all numbers from the text
9. Remove all the extra spaces.
10. Add start and end tokens to the target sequences.

**DATASET DETAILS**

We used data obtained from the Tatoeba Project. Tatoeba is a collection of sentences and translations. It's collaborative, open and free. These are sorted by length, with the shortest sentences first. the dataset consists of 33000 pairs of sentences. We allocated 90 % and 10% percent of the data for training and testing respectively

**Simple Encoder-Decoder Architecture**

A basic network in NMT consists of an encoder and decoder. The natural choice for encoder and decoder is a Recurrent Neural Network (RNN) because RNNs can easily map sequences to sequences when the alignment between inputs and outputs is known ahead of time. RNN is a natural extension of feed forward neural networks with the difference being RNN has a memory, i.e., it takes into account previous outputs.

Let X and Y be the source and target sentence pairs respectively. The encoder RNN converts the source sentence x1,x2...xn into vectors of fixed dimensions. The decoder outputs one word at a time using conditional probability

P(Y |X) = P(Y |X1,X2,X3, ...,XM)

Here X1, .....XM in the equation are the fixed size vectors encoded by the encoder. Using chain rule the above equation is converted to the equation below where, while decoding, next word is predicted using symbols that are predicted till now and source sentence vectors. The above expression then becomes

P(Y |X) = P(yi|y0,y1,y2, ....,yi−1;X1,X2,X3, ...,XM)

Each term in the distribution is represented with a softmax4 over all the words in the vocabulary.

**Long Short Term Memory (LSTM)**

A variant of RNN called Long Short Term Memory (LSTM) is known to learn problems with long range temporal dependencies, so RNNs are sometimes replaced with LSTMs in MT networks, because they may succeed better in this setting. In Figure 1, the model reads an input sentence “AB” and produces “WXY” as the output sentence. The model stops making predictions after outputting the end-of-sentence (EOS) token.



**Figure 3.3**

**Attention Mechanism**

Even though LSTMs are used to deal with the problem of long term dependencies, they do not solve the problem completely solved by them, because the model can still suffer from failures in long sentence translation due to its incapability in capturing long term dependencies. This is largely due to the fact that the encoder of this basic approach needs to compress a whole source sentence into a single vector. We use an attention mechanism to solve this problem. The basic idea of attention mechanisms is that instead of encoding a sentence into a single vector, we encode each word in the sentence into a vector, and reference these vectors while decoding. Since the number of vectors available while decoding is equivalent to the number of words in the sentence, long sentences have many vectors and short sentences have few vectors. This makes it feasible to represent sentences in an efficient way, avoiding the problems arising because of the inefficient representation if we encode the whole sentence by a single vector.

We now encode each word in the sentence by passing it into an RNN in both directions. The output from both the directions is concatenated. The notations used below are from



j are outputs of a word from an RNN in both forward and backward directions respectively. These vectors are concatenated into a single matrix for all the words.



Every column in the matrix represents one word. This matrix is of variable number of columns but the decoder accepts a vector of fixed dimension while decoding to get the context. So that implies context vector should be of fixed length. So we multiply this ma- trix with the attention vector

*ct* = *HF αt*

*HF* is the matrix, *ct* is the context vector and *αt* is the attention vector.

The basic idea behind the attention vector is that it indicates how much we are “focusing” on a particular source word at a particular instant of time. The larger the value of *αt*, the more impact a word has when predicting the next word in the output sentence.

To calculate the attention vector, we first calculate attention scores. These attention scores can be computed with an arbitrary function that takes two vectors as inputs, and outputs a score be- tween 0 and 1 indicating how much to focus on this particular in-

put word encoding *h*(*F* ) at the time step *h*(*e*).

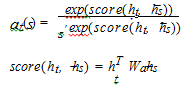
We then normalize this to get the actual attention vector itself by taking a softmax over the scores.

*αt* = *softmax*(*at*)

This attention vector is then used to weight the encoded representation *HF* to create a context vector *ct* for the current time step.

We calculate attention scores using the following formulae. Here

*hs* is source hidden state and *ht* is target state



These context vectors are used while decoding. They give bet- ter translations by selectively focusing on words in the source sentence during translation

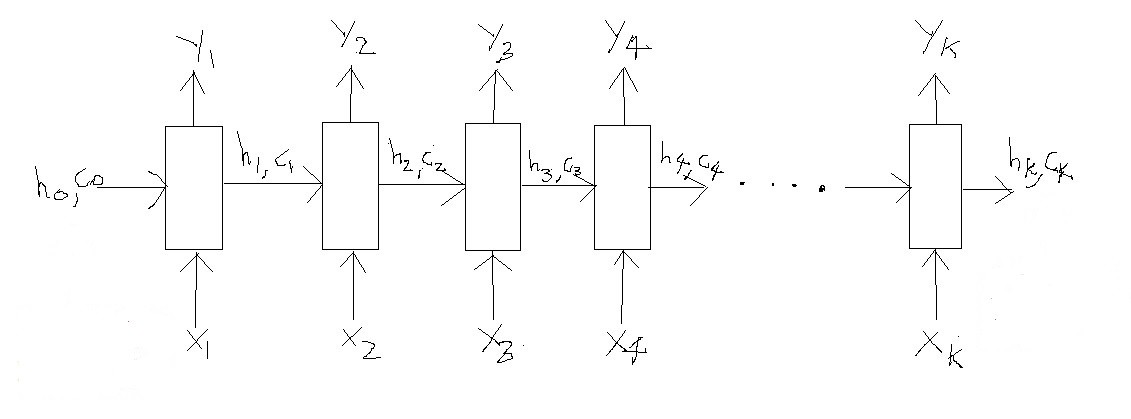
**CHAPTER 4**

**IMPLEMENTATION DETAILS**

**4.1: MODULE AND DESCRIPTION**

**Encoder LSTM**

The Encoder LSTM has the same role to play in both the training phase as well as in the inference phase.

 **Figure 4.1.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:

1. Xi => Input sequence at time step i
2. hi and ci => 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.
3. Yi => Output sequence at time step i

All of these components (Xi, hi, ci and Yi) are the actual vectors of floating point numbers the mapping of all these vectors in the context of our problem is as follows. For example, we have the following sentence.

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

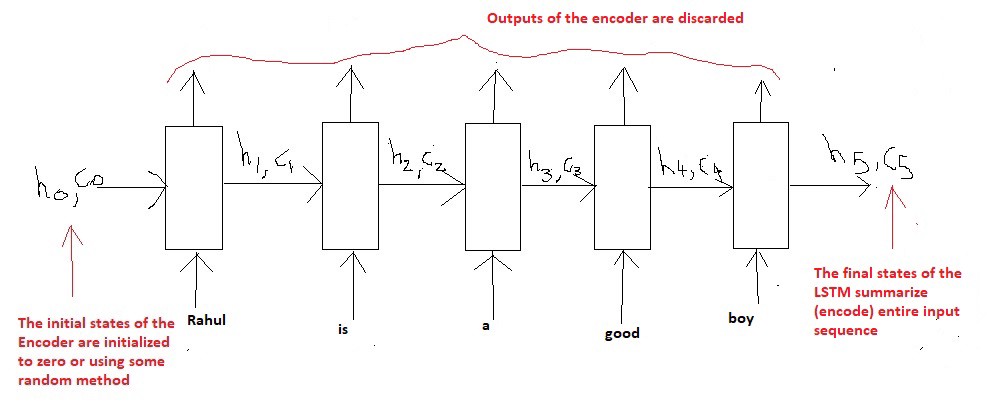
Output sentence (Marathi) => “राहुल चांगला मुलगा आहे”

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

**Explanation for Xi:**

X1 = ‘Rahul’, X2 = ‘is’, X3 = ‘a’, X4 = ‘good, X5 = ‘boy’.

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



**Figure 4.2**

**Explanation for hi and ci:**

The role of the internal states (hi and ci) at each time step is explained as follows. In very simple terms, they remember what the LSTM has read (learned) till now. For example:h3, c3 =>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 h3 and c3 (thus called the states at time step 3).Similarly, we can thus say that h5, c5 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. The vectors h0,c0 are typically initialized to zero as the model has not yet started to read the input

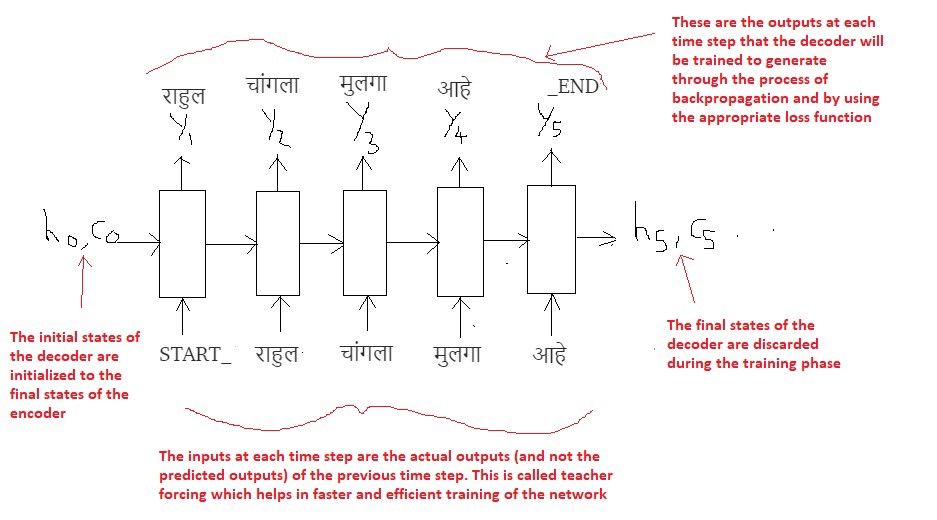
**Explanation for Yi:**

The Yi are the output (predictions) of the LSTM model at each time step. More specifically in case of word level language models each Yi is actually a probability distribution over the entire vocabulary which is generated by using a softmax activation. Thus each Yi 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 Yi of the Encoder for our problem.

**DECODER LSTM- TRAINING MODEL**

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. Consider the given 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.

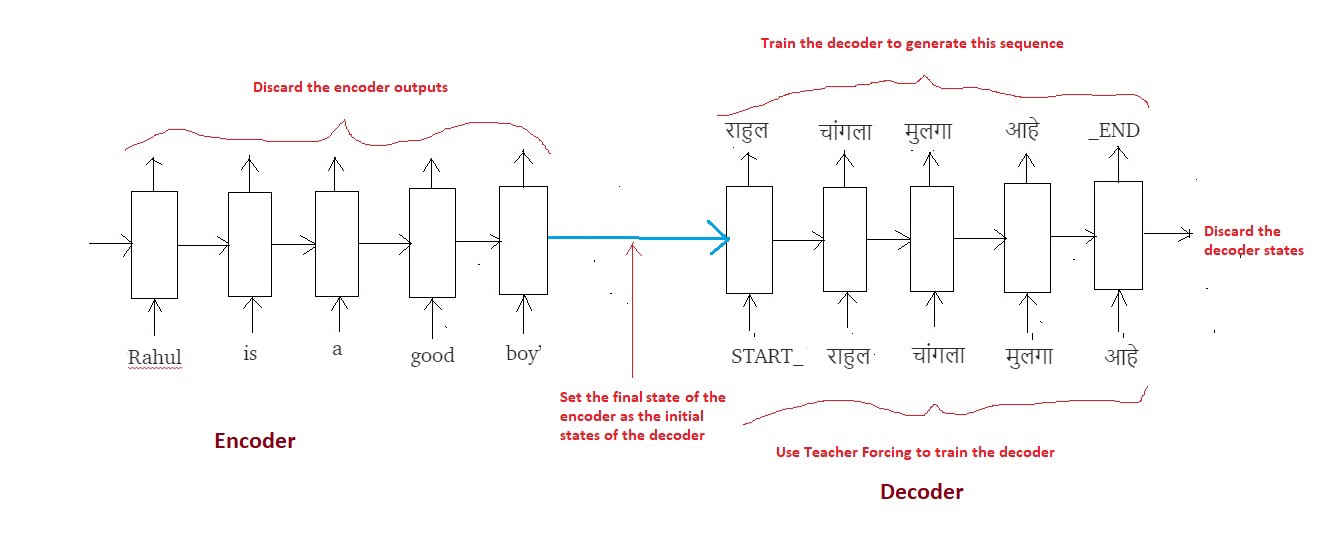
Output sequence => “START\_ राहुल चांगला मुलगा आहे \_END”



**Figure4.3 Decoder LSTM — Training Mode**

The most important point is that the initial states (h0, c0) 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.

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.



**Figure 4.4 Summary of the training process**

The use of 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

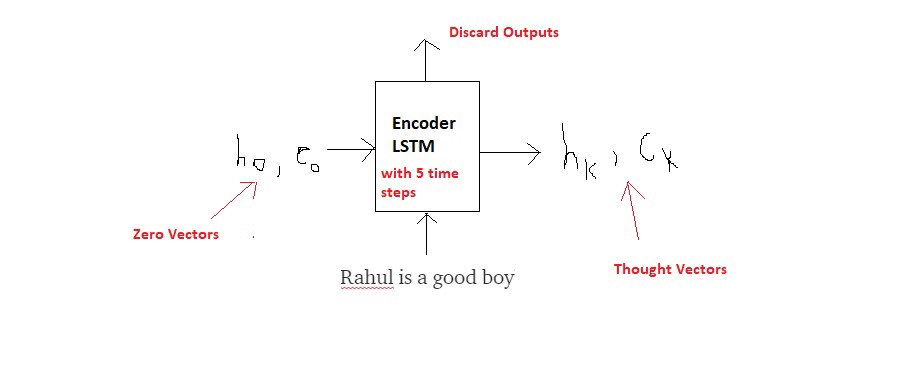
**Decoder LSTM — Inference Mode**

The Encoder LSTM plays the role of reading the input sequence (English sentence) and generating the thought vectors (hk, ck).However, the decoder LSTM now has to predict the entire output sequence (Marathi sentence) given by these thought vectors generated by the encoder LSTM. The visual understanding of the decoder LSTM by taking the same example as before.

Input sequence => “Rahul is a good boy”

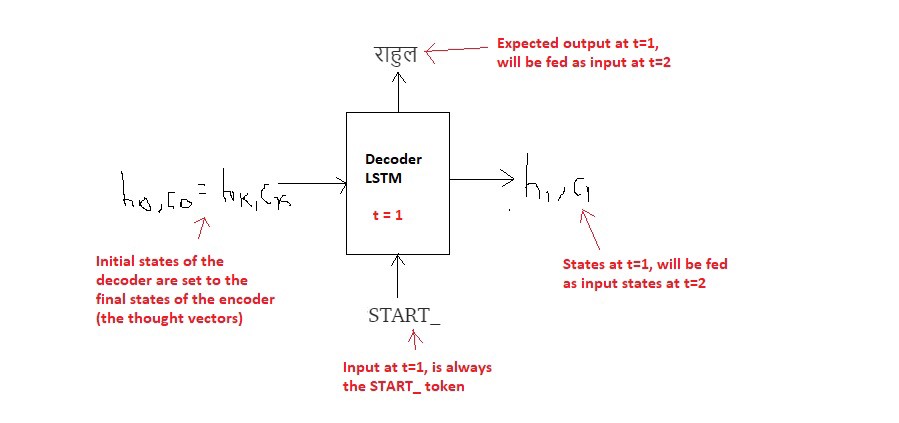
(Expected) Output Sequence => “राहुल चांगला मुलगा आहे”

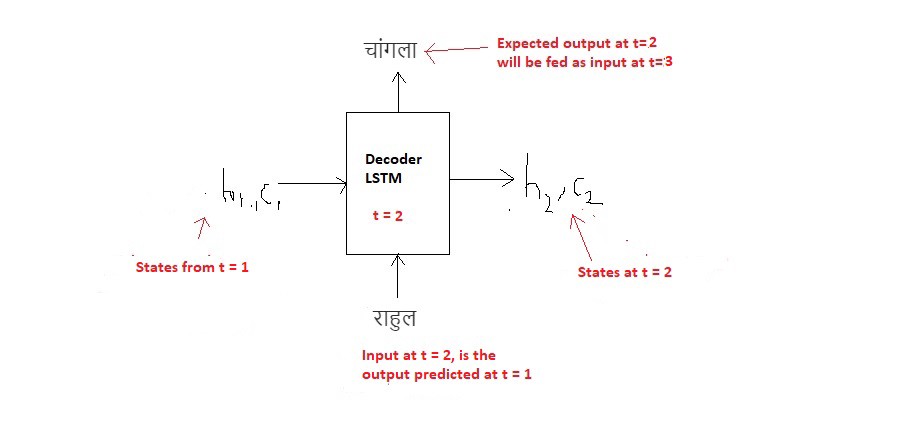
**Step 1**: Encode the input sequence into the Thought Vectors:



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

At t = 1

At t = 2

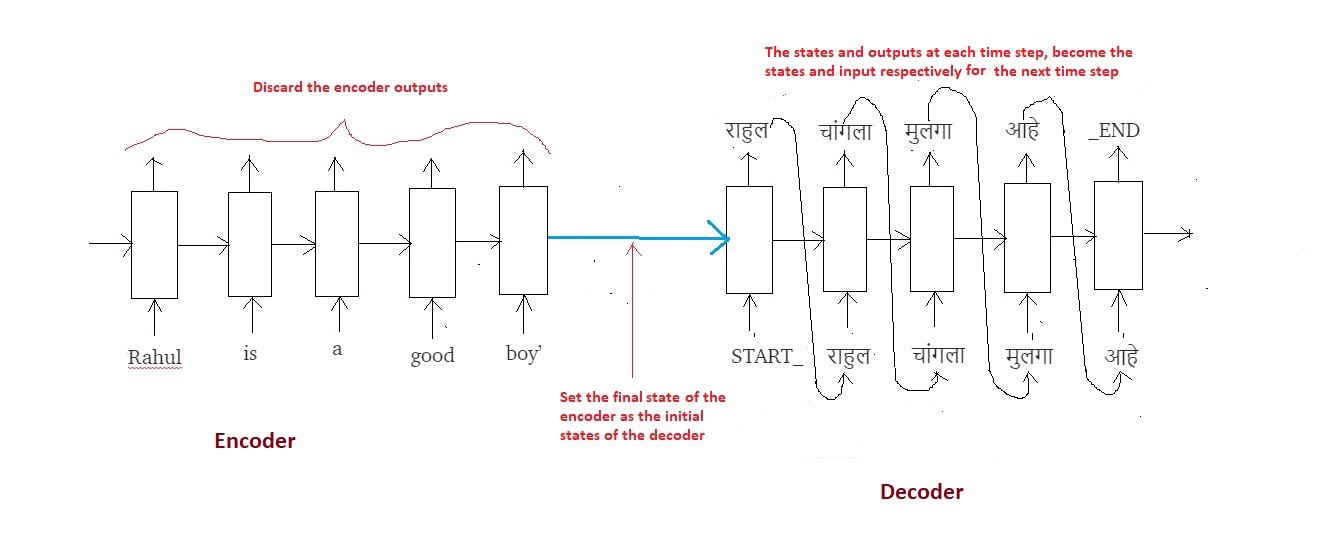


In the similar manner, for t=3,t=4,t=5, the decoder LSTM –Inference mode takes place.

**Inference Algorithm:**

1. 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.
2. The initial states of the decoder are set to the final states of the encoder.
3. The initial input to the decoder is always the START\_ token.
4. At each time step, we preserve the states of the decoder and set them as initial states for the next time step.
5. At each time step, the predicted output is fed as input in the next time step.
6. We break the loop when the decoder predicts the END\_ token.

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

 **Figure 4.5 Summary of Inference Algorithm**

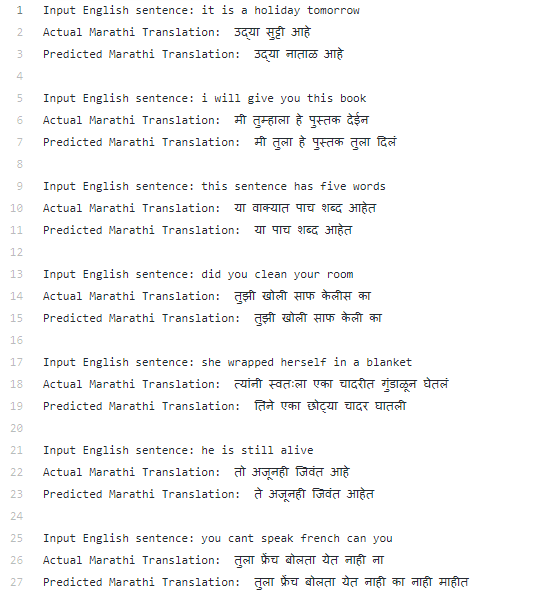
**CHAPTER 5**

**RESULTS AND ANALYSIS**

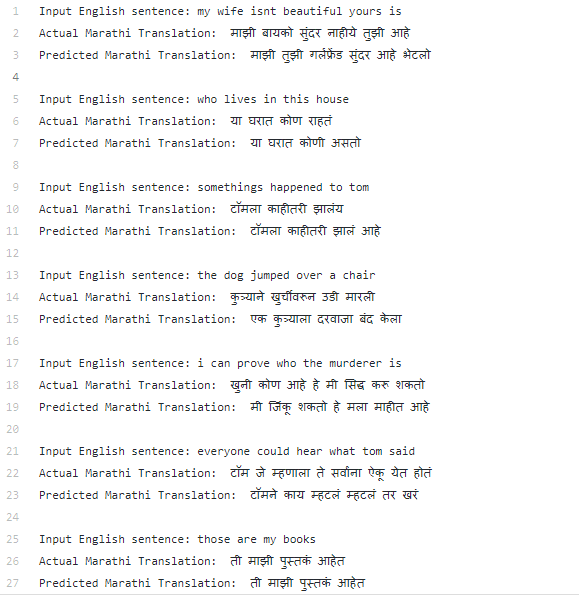
The main purpose of our project was to give an intuitive explanation on how to build basic level sequence to sequence models using LSTM. In our project, the dataset consists of 33000 pairs of sentences.

In this work, we showed that a large deep LSTM, that has a limited vocabulary and that makes almost no assumption about problem structure can outperform a standard SMT-based system whose vocabulary is unlimited on a large-scale MT task. The success of our simple LSTM-based approach on MT suggests that it should do well on many other sequence learning problems, provided they have enough training data.

Evaluation on train dataset:



## Evaluation on test dataset:



**CHAPTER 6**

**CONCLUSION AND FUTURE SCOPE**

In this report, we proposed a new neural network architecture, called an RNN Encoder–Decoder that is able to learn the mapping from a sequence of an arbitrary length to another sequence, possibly from a different set, of an arbitrary length. The proposed RNN Encoder–Decoder is able to either score a pair of sequences (in terms of a conditional probability) or generate a target sequence given a source sequence. Our qualitative analysis of the trained model shows that it indeed captures the linguistic regularities in multiple levels i.e. at the word level as well as phrase level. This suggests that there may be more natural language related applications that may benefit from the proposed RNN Encoder– Decoder.

In this work we have applied neural machine translation techniques on European language (English) to Indian Language (Marathi). Even though the results are not the best, they are not that bad either. We could see certainly much better results 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. One of the reasons for this could be the small dataset for training.

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 as the corpus is not that large enough. This indicates the requirement for more variety of sentences for training purpose.

To improve the accuracy of the results in the training and testing part the following this can be done in future.

1. Get much more data.
2. Build more complex models like Attention.
3. Use dropout and other forms of regularization techniques to mitigate over-fitting.
4. Perform Hyper-parameter tuning. Play with learning rate, batch size, dropout rate, etc. Try using bidirectional Encoder LSTM. Try using multi-layered LSTMs.
5. Try using beam search instead of a greedy approach.
6. Try BLEU score to evaluate your model.

**CHAPTER 7**

**REFERENCES**

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

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

[3][Ilya Sutskever](https://arxiv.org/search/cs?searchtype=author&query=Sutskever%2C+I), [Oriol Vinyals](https://arxiv.org/search/cs?searchtype=author&query=Vinyals%2C+O), [Quoc V. Le](https://arxiv.org/search/cs?searchtype=author&query=Le%2C+Q+V), “Sequence to Sequence Learning with Neural Networks”,14 Dec 2014.

[4][Kyunghyun Cho](https://arxiv.org/search/cs?searchtype=author&query=Cho%2C+K), [Bart van Merrienboer](https://arxiv.org/search/cs?searchtype=author&query=van+Merrienboer%2C+B), [Caglar Gulcehre](https://arxiv.org/search/cs?searchtype=author&query=Gulcehre%2C+C), [Dzmitry Bahdanau](https://arxiv.org/search/cs?searchtype=author&query=Bahdanau%2C+D), [Fethi Bougares](https://arxiv.org/search/cs?searchtype=author&query=Bougares%2C+F), [Holger Schwenk](https://arxiv.org/search/cs?searchtype=author&query=Schwenk%2C+H), [Yoshua Bengio](https://arxiv.org/search/cs?searchtype=author&query=Bengio%2C+Y), “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”, 3 Sep 2014.