

A Project Report
on
NEURAL MACHINE TRANSLATION
FOR INDIAN LANGUAGES

Submitted in partial fulfillment of the requirements for the award of the degree
of

BACHELOR OF TECHNOLOGY
in
INFORMATION TECHNOLOGY
by

A. Bhavishya (16WH1A1202)

B. Sandhya (16WH1A1210)

G. DivyaLata (16WH1A1215)

V. Bhanu Samera (16WH1A1248)

Under the esteemed guidance of

Mr. L. Naveen Kumar
Assistant Professor



Department of Information Technology
BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN
(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and
Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090
April 2020

DECLARATION

We hereby declare that the work presented in this project entitled **“NEURAL MACHINE TRANSLATION FOR INDIAN LANGUAGES**” submitted towards completion of the major project in IV year of B.Tech IT at **“BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN”**, is an authentic record of our original work carried out under the esteem guidance of Mr. L. Naveen Kumar, Assistant Professor, IT department.

A.Bhavishya
(16WH1A1202)

B. Sandhya
(16WH1A1210)

G. DivyaLata
(16WH1A1215)

V. Bhanu Samera
(16WH1A1248)

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Information Technology



Certificate

This is to certify that the Project report on “**Neural Machine Translation for Indian Languages**” is a bonafide work carried by **A.Bhavishya (16WH1A1202), B.Sandhya (16WH1A1210), G.Divyalata (16WH1A1215), V.BhanuSamera (16WH1A1248)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide

Mr. L. Naveen Kumar

Assistant Professor

Department of IT

Head of the Department

Dr. ArunaRao S L

Professor and HOD

Department of IT

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal**, BVRIT HYDERABAD, for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. ArunaRao S L, Head**, Dept. of IT, BVRIT HYDERABAD for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr.L. Naveen Kumar, Assistant Professor, Department of IT**, BVRIT HYDERABAD for his constant guidance, encouragement and moral support throughout the project.

Finally, We would also like to thank our project co-coordinator, all the faculty and staff of IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

A.Bhavishya
(16WH1A1202)

B. Sandhya
(16WH1A1210)

G. Divya Lata
(16WH1A1215)

V. Bhanu Samera
(16WH1A1248)

ABSTRACT

Neural Machine Translation (NMT) is a variation of machine translation system. It learns how to translate data itself by using training data. This translation is based on the two ideas, one is Recurrent Neural Network and another is Encoding. Recurrent Neural Network takes the inputs, along with this inputs it will take previous output as a input. Finally gives the output by taking all these inputs. This output is a input for next translation. This project aim is to create a Neural Network to translate from English to Hindi. For this an extension of RNN (Recurrent Neural Network) i.e. LSTM (Long Short-Term Memory) is being used then. This project is implemented in six modules they are as follows: Data Preprocessing, Tokenization and Padding, Word Embeddings, Creating a model, Training a model and Testing a model. This LSTM was able to translate English input sentences to Hindi output sentences.

LIST OF FIGURES

Figure No	Figure Name	Page No
4.1	Example of a Neural Network for predicting house price	4
4.2	Model of a Recurrent Neural Network	5
4.3	Encoding a sentence into a series of 128 numbers	6
4.4	Recurrent Neural Networks have loops	11
4.5	Chain-like structure of a recurrent neural network	11
4.6	RNN with gap between relevant info and the point	12
4.7	Repeating modules in a standard RNN	13
4.8	Module in an LSTM network	13
4.9	Model predicting the next word	14
4.10	Model of storing information in cell state	15
4.11	Model of dropping old and loading new information	15
4.12	Generating the output model	16
5.1	Modules in the project	17
6.1	Importing the packages	25
6.2	Output of the data set	25
6.3	preprocessing the dataset	26
6.4	Tokenization process	26
6.5	Generating the model	27
6.6	Generating the encoder	27
6.7	Generating the decoder	27
6.8	Training the model	28
6.9	Process of Training	28
6.10	Training with encoder and decoder	29
6.11	Predict Function code	29
6.12	Testing the model	30
6.13	Final Output	30

LIST OF ABBREVIATIONS

Word	Abbreviation
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
R2NN	Recursive Recurrent Neural Network
MT	Machine Translation
NMT	Neural Machine Translation

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	V
	LIST OF FIGURES	Vi
1.	INTRODUCTION	1-5
	1.1 PROBLEM IN EXISTING SYSTEM	5
	1.2 SOLUTION	5
	1.3 FEATURES	5
2	LITERATURE SURVEY	6-7
3	REQUIREMENT SPECIFICATION	8
	3.1 SOFTWARE REQUIREMENT	8
	3.2 HARDWARE REQUIREMENT	8
4	DESIGN OF THE SYSTEM	9-17
5	MODULES	18
	5.1 DATA PREPROCESSING	19-24
	5.2 TOKENIZATION AND PADDING	25-27
	5.3 CREATING THE MODEL	27-28
	5.4 TRAINING	29-31
	5.5 TESTING	31-33
6	IMPLEMENTATION	34-41
7	CONCLUSION AND FUTURE SCOPE	42
	8.1 CONCLUSION	42
	8.2 FUTURE SCOPE	42
8	REFERENCES	43

1. INTRODUCTION

Machine translation, sometimes referred to by the abbreviation MT (not to be confused with computer-aided translation, machine-aided human translation (MAHT) or interactive translation), is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another.

On a basic level, MT performs simple substitution of words in one language for words in another, but that alone usually cannot produce a good translation of a text because recognition of whole phrases and their closest counterparts in the target language is needed. Solving this problem with corpus statistical, and neural techniques is a rapidly growing field that is leading to better translations, handling differences in linguistic typology, translation of idioms, and the isolation of anomalies.

Current machine translation software often allows for customization by domain or profession (such as weather reports), improving output by limiting the scope of allowable substitutions. This technique is particularly effective in domains where formal or formulaic language is used. It follows that machine translation of government and legal documents more readily produces usable output than conversation or less standardized text.

Improved output quality can also be achieved by human intervention: for example, some systems are able to translate more accurately if the user has unambiguously identified which words in the text are proper names. With the assistance of these techniques, MT has proven useful as a tool to assist human translators and, in a very limited number of cases, can even produce output that can be used as is (e.g., weather reports).

Machine Translation is a subbranch of natural language processing. It is defined as the use of computers in translation of text from one language to another language. For example, if for a given input of English text, translation is needed in Hindi or some other language, then the usual option is to refer to a dictionary. Today, the job is being done by technological means. In other words, a computer or a smartphone can use a machine

translation-based application to give the output text in Hindi.

There are three main approaches to machine translation. These are as follows:

1. Linguistic or Rule-based approach
 - a) Direct Machine translation
 - b) Inter-lingua Machine translation
 - c) Transfer based Machine translation
2. Non-linguistic approaches
 - a) Dictionary based Machine translation
 - b) Corpus-based Machine translation
 - c) Example-based Machine translation
 - d) Statistical-based Machine translation
3. Hybrid

1. Linguistic or Rule Based approach

Linguistic or Rule based approach uses grammar and computer programs to translate from one language to another.

a) Direct Machine translation:

Direct MT (Machine translation) makes use of two-way dictionaries. First, it extracts root words from the source language. Second, it looks up these words in the bilingual dictionaries. Third, it syntactically rearranges the words in target language.

b) Inter lingua Machine translation:

Interlingua MT is used to translate to more than one language. This is done by first translating the text from source language to an intermediate language called universal language. Then, it can be translated from this universal language to any target language.

c) Transfer based Machine translation:

Transfer-based MT, a database with translation rules is used. This is done in three phases: First, in analysis phase, morphological and syntactic information of source

language is analyzed, and a base form of the source language is produced. A hierarchical syntax tree for the source language is generated by grammar rules. Second, in transfer phase, base form of the source language is text is translated into target language by using rearrangement rules. Finally, in the generation phase, the exact translation of source text to target language is taken place.

2. Non-linguistic approach

Non-Linguistic approach does not require any linguistic knowledge to translate from source language to target language. It makes use of dictionary for dictionary-based approach or bilingual corpus for corpus-based approach.

a) Dictionary based Machine translation:

In Dictionary based approach, the dictionary is used to translate the text word by word from the source language to target language.

b) Corpus-based Machine translation:

The corpus-based approach does not require explicit linguistic knowledge to translate from source language to target language. The system is trained using bilingual corpus and the monolingual corpus of the target language to translate the sentence.

c) Example-based Machine translation:

In example-based approach, huge bilingual corpus of the language pairs of source and target language is used. It works based on the human being approach of problem solving. The main problem is divided into sub problems. Each subproblem is solved based on the experience gathered in the previous round of solving the same kind of problem and finally integrating the solutions of the sub problems to solve the main problem.

d) Statistical-based Machine translation:

In Statistical approach, bilingual corpora and statistical methods are used to translate the sentence from source language to target language. The parameters from bilingual corpora and monolingual corpus is analysed and determined for creating translation and language models.

Statistical translation involves 3 steps:

1. Estimating the language probability $p(t)$
2. Estimating the translation model probability $p(s,t)$
3. Devise a product of them to achieve an efficient search of target text
Sentence T is found, for which $p(s,t)$ is maximum.

3. Hybrid

Hybrid machine translation is the combination of Rule-based as well as statistical -based translation systems. One of the examples for hybrid machine translation approach is Rule based system with post-processing by statistical approach.

Neural Machine Translation (NMT) is a variation of machine translation system. Here an artificial neural network is used to increase the fluency and accuracy of translation. NMT is based on a simple encoder-decoder based network. The type of neural networks used in NMT is Recurrent Neural Networks (RNN). The reason for selecting RNN for the task is its basic structure, which involves cyclic structure and enables the learning of repeated sequences.

1.1 PROBLEM IN EXISTING SYSTEM

A Recurrent Neural Network remembers only the immediately preceding state. While this is useful for some operations such as mathematical calculations, this becomes a disadvantage in text prediction, translation, etc. For example, the meaning of the word 'for' is not fixed. It may vary according to its position in the sentence. To correctly translate a word the network needs long term memory.

1.2 SOLUTION

The above problem is solved using Long Short-Term Memory configuration of RNN. In LSTM, the state is carried through by default until it is modified by relevant gate in the network.

1.3 FEATURES

- In LSTM more and more controlling knobs will be there, it will control the flow and mixing of inputs as per trained weights.
- LSTM brings more flexibility in controlling the outputs.
- LSTM offers the most Control-ability because of which, it will give better results.
- LSTM can handle more Complexity.
- Operating cost of LSTM is high.

2. LITERATURE SURVEY

Saini and others (2018) explored different configurations of a Neural Machine Translation System for Indian language Hindi. They have experimented with eight different architecture combinations of NMT for English to Hindi. Compared with conventional machine translation techniques, their results showed that NMT requires very less amount of data size for training and thus exhibits satisfactory translation for few thousands of training sentences.

Neural Machine Translation for English to Hindi, by Sandeep Saini and Vineet Shaula, Conference Paper, 2018. URL: : <https://www.researchgate.net/publication/327717152>. Wu et al (2016) presented the Google Neural Machine Translation (NMT) system consisting of a deep LSTM network with 8 encoder and 8 decoder layers using attention and residual connections. To decrease training time, their attention mechanism connected the bottom layer of the decoder to the top layer of the encoder. To accelerate the translation speed, they used low-precision arithmetic during inference computations. To improve handling of rare words, they divided words into a limited set of common sub-word units ("word pieces") for both input and output. This method provides flexibility and naturally handles translation of rare words, ultimately improving the overall accuracy of the system.

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, by Yonghui Wu, 2016. URL: <https://arxiv.org/abs/1609.08144>

Sennrich (2015) and others attempted to overcome the fixed-vocabulary limitation of NMT models by encoding rare and unknown words as sequences of sub word units. This is based on the idea that various word classes are translatable via smaller units than words, such as names (via character copying or transliteration), compounds (via compositional translation), and cognates and loanwords (via phonological and morphological transformations). The authors discussed the suitability of different word segmentation techniques, including simple character n-gram models and a segmentation based on the byte pair encoding compression algorithm. They empirically showed that sub word models

are an improvement over a back-off dictionary baseline for the WMT 15 translation tasks English-German and English Russian by 1.1 and 1.3 BLEU, respectively. Neural Machine Translation of Rare Words with Sub word Units, by Rico Sennrich, Barry Haddow, Alexandra Birch, 2015. URL: <https://arxiv.org/abs/1508.07909>

Gulcehre (2015) found from recent work on end-to-end neural network-based architectures for machine translation for En-Fr and En-De that one of the major factors behind the success has been the availability of high-quality parallel corpora. They investigated how to leverage abundant monolingual corpora for neural machine translation. Compared to a phrase-based and hierarchical baseline, they obtain up to 1.96 BLEU improvement on the low-resource language pair Turkish-English, and 1.59 BLEU on the focused domain task of Chinese-English chat messages. While their method was initially targeted toward such tasks with less parallel data, they indicate that it extends to high resource languages such as Cs-En and De-En where an improvement of 0.39 and 0.47 BLEU scores were obtained over the neural machine translation baselines, respectively. On Using Monolingual Corpora in Neural Machine Translation, by Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loic Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, Yoshua Bengio (2015) URL: <https://arxiv.org/abs/1503.03535>

3. REQUIREMENT SPECIFICATIONS

Software requirements deal with software and hardware deals with resources that need to be installed on a server which provides optimal functioning for the application. These software and hardware requirements need to be installed before the packages are installed. These are the most common set of requirements defined by any operation system. These software and hardware requirements provide a compatible support to the operation system in developing an application.

3.1 SOFTWARE REQUIREMENTS

The software requirements specify the use of all required software products like data management system. The required software product specifies the numbers and version. Each interface specifies the purpose of the interfacing software as related to this software product.

- Operating system : Linux, Windows XP
- Coding Language : Python 3
- Development Kit : Anaconda, jupyter notebook, Google Colab
- Software : J2SE, ADT plug-in

3.2 HARDWARE REQUIREMENTS

The hardware requirement specifies each interface of the software elements and the hardware elements of the system. These hardware requirements include configuration characteristics.

- System : Pentium IV with 2 GHZ
- Hard Disk : 40 GB
- RAM : 1 GB

4. DESIGN OF THE SYSTEM

Recurrent Neural Network

A regular (or, non-recurrent) neural network is a generic machine learning algorithm that takes in a list of numbers and calculates a result (based on previous training). Neural networks can be used as a black box to solve lots of problems.

For example, we can use a neural network to calculate the approximate value of a house based on attributes of that house.

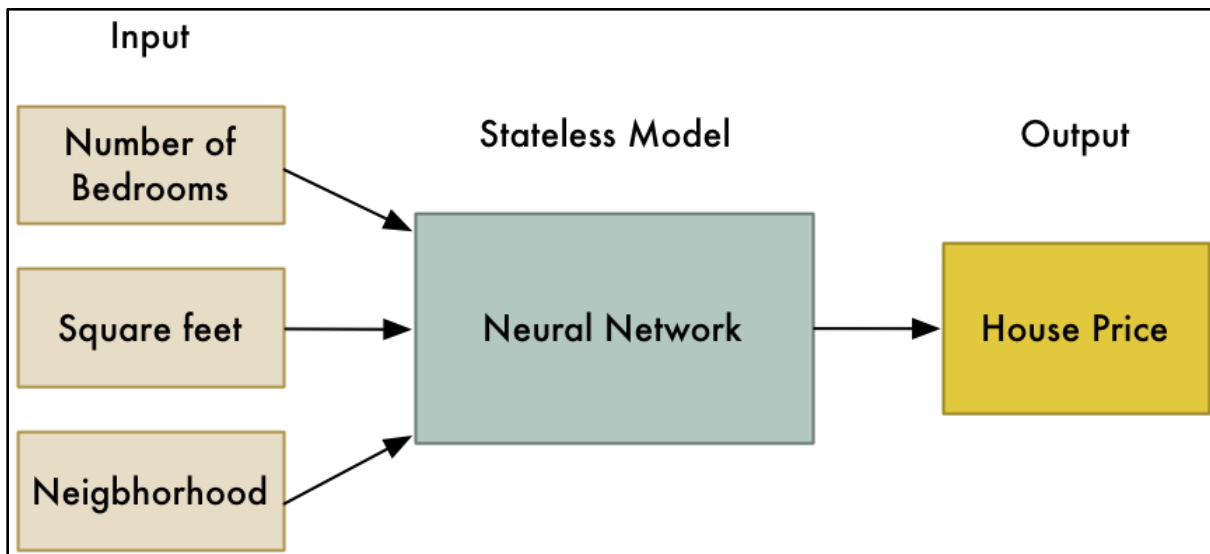


Fig. 4.1 Example of a Neural Network for predicting house price

But like most machine learning algorithms, neural networks are *stateless*. It takes a list of numbers and calculates a result. If the same numbers are fed, it will always calculate the same result. It has no memory of past calculations. In other words, $2 + 2$ always equals 4.

A **Recurrent Neural Network (RNN)** on the other hand is a slightly modified version of a regular neural network. Here, the previous state of the neural network is one of the inputs to the next calculation. This means that previous calculations can change the result of future calculations, as shown in the chart below.

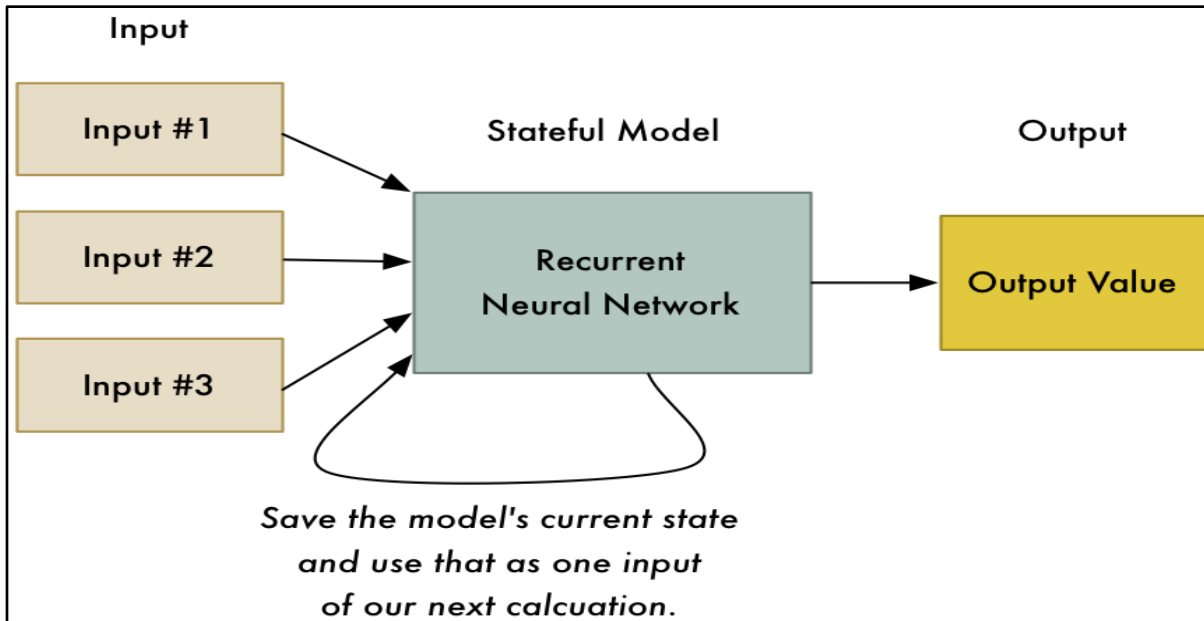


Fig. 4.2 Model of a Recurrent Neural Network

This method of using the state of a calculation as an input for the next calculation allows recurrent neural networks to learn patterns in a sequence of data. RNNs are also useful to detect patterns in data. Because human language is just one big, complicated pattern, RNNs are increasingly used in many areas of natural language processing.

Encoding

The idea of turning a face into a list of measurements is an example of an encoding. We are taking raw data and turning it into a list of measurements that represent it (the encoding).

It lets us represent something very complicated such as sentence, word, or a picture of a face with something simple (128 numbers). Now comparing two different sentences / words / faces is much easier because we only have to compare these 128 numbers for each input instead of comparing full images.

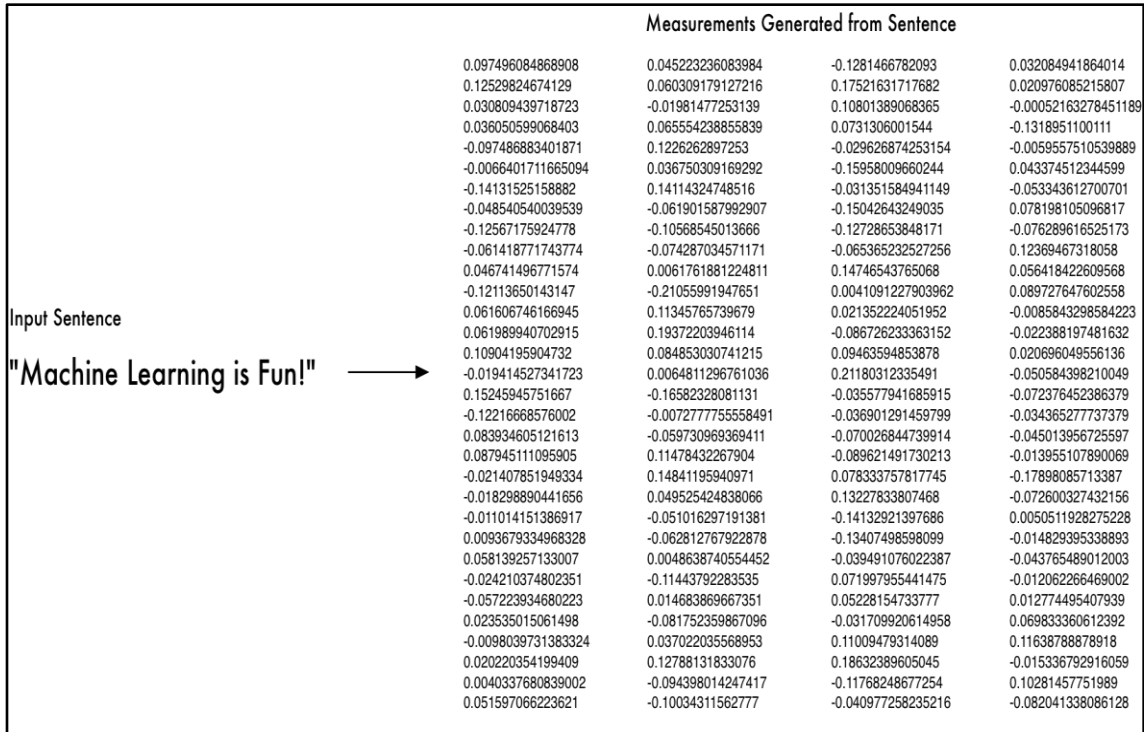


Fig. 4.3 Encoding a sentence into a series of 128 numbers

This list of numbers represents the English sentence “Machine Learning is Fun!”. A different sentence would be represented by a different set of numbers.

To generate this encoding, we’ll feed the sentence into the RNN, one word at time. The final result after the last word is processed will be the values that represent the entire sentence. Because the RNN has a “memory” of each word that passed through it, the final encoding it calculates represents all the words in the sentence.

This project makes use of a special variation in Recurrent Neural Network to address the need for long term dependencies in machine translation. This variation is called Long Short-Term Memory (LSTM). The Recurrent Neural Networks can store ‘state’ values and pass them on to the next calculation, using loops as shown below, allowing information to persist.

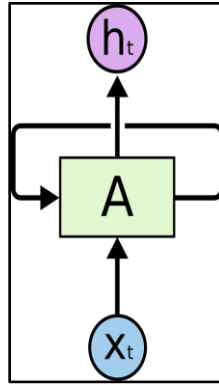


Fig. 4.4 Recurrent Neural Networks have loops

In the above diagram, a chunk of neural network, A , looks at some input x_t and outputs a value h_t . A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

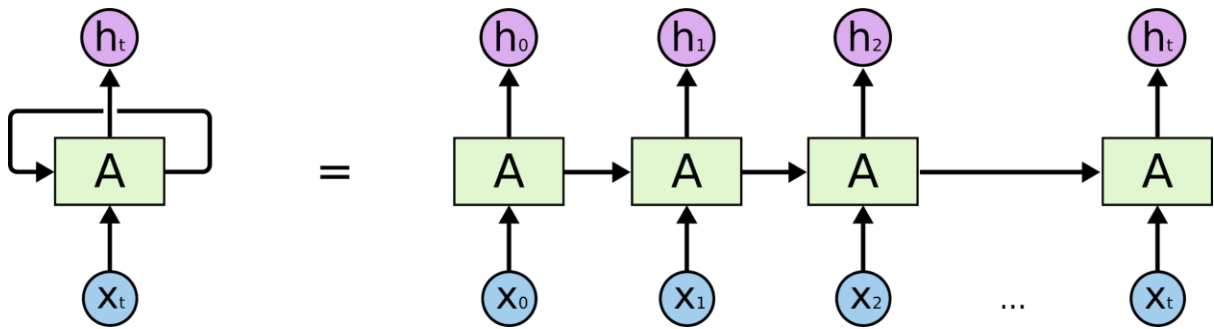


Fig. 4.5 Chain-like structure of a recurrent neural network

Long-Term Dependencies

One of the advantages of RNNs is the idea that they might be able to connect previous information to the present task. If RNNs could do this, they would be very useful. In some cases, the RNN needs to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If the problem is to predict the last word in “the clouds are in the sky,” the network can easily predict that the next word is going to be sky. In such cases, where the gap

between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

But there are also cases more context is needed. Consider trying to predict the last word in the text “I grew up in France... I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but to narrow down which language, the network needs the context of France, from further back. It's possible for the gap between the relevant information and the point where it is needed to become very large. As that gap grows, RNNs become unable to learn to connect the information.

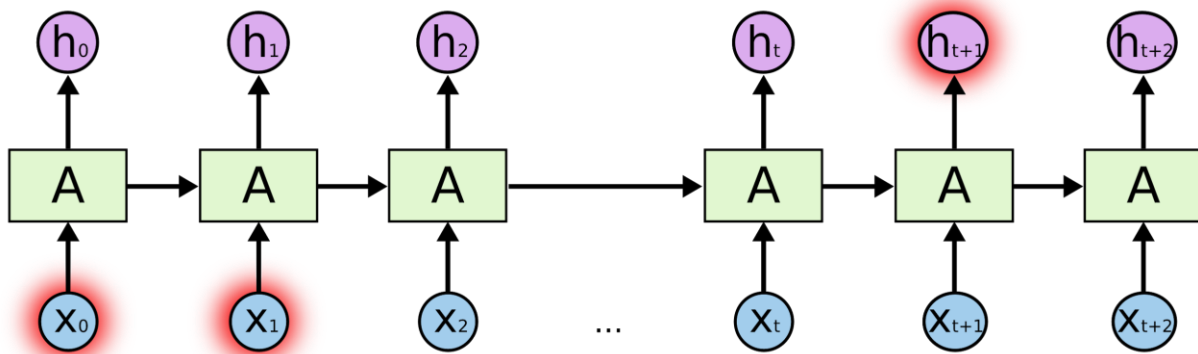


Fig. 4.6 RNN with gap between relevant info and the point where it is needed

In theory, RNNs are capable of handling such “long-term dependencies.” A human could pick parameters for them to solve toy problems of this form. In practice, RNNs don't seem to be able to learn them. LSTMs don't have this problem. Because of this reason in our project recurrent neural networks not being used.

LSTM Networks

Long-Short-Term Memory Networks are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997). LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is their default behavior. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this

repeating module will have a very simple structure, such as a single tanh layer.

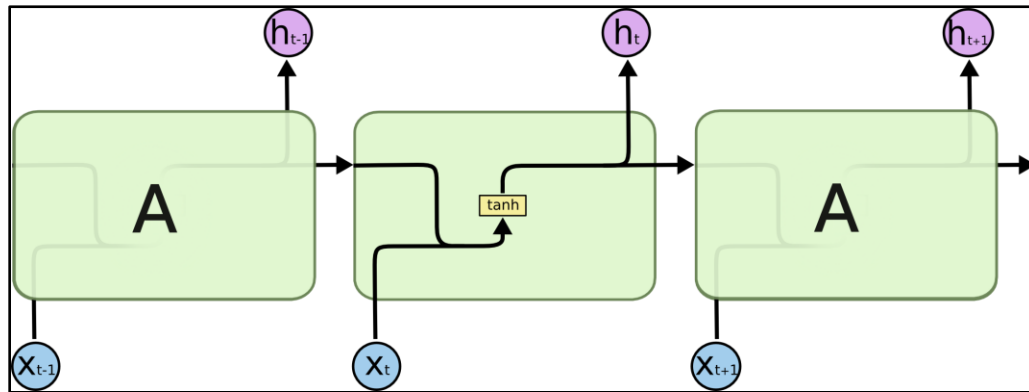


Fig. 4.7 Repeating modules in a standard RNN

The repeating module in a standard RNN contains a single layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way. The repeating module in an LSTM contains four interacting layers.

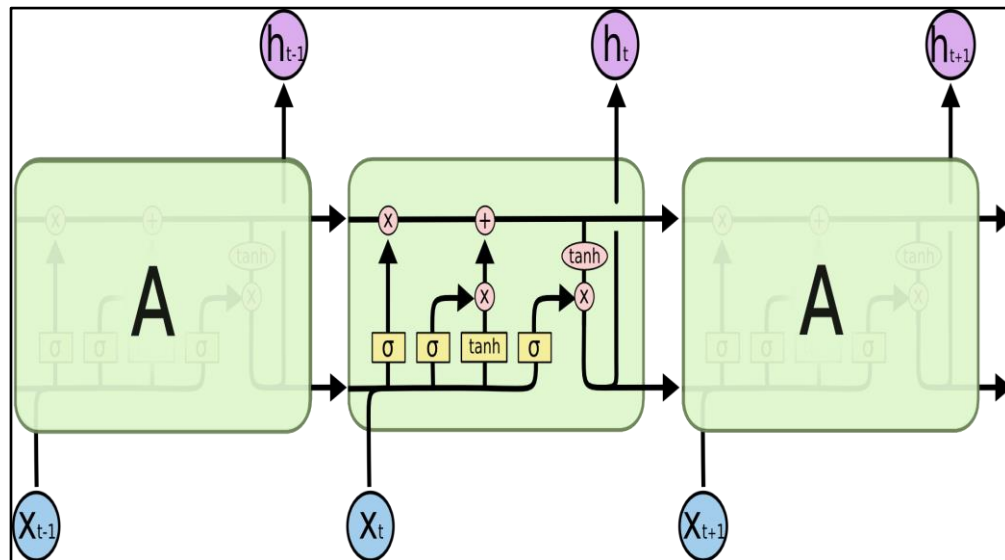
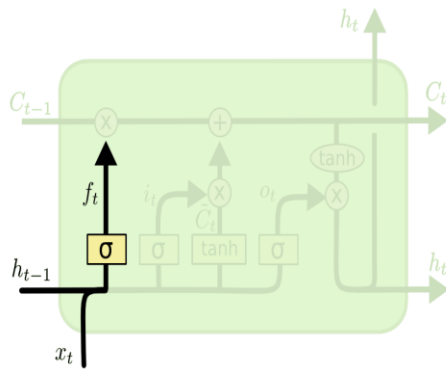


Fig. 4.8 Module in an LSTM network

Step-by-Step Process of LSTM

The first step in our LST is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer". It looks at h_{t-1} and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1} . A 1 represents "completely keep this" while a 0 represents "completely get rid of this". Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Fig. 4.9 Model predicting the next word

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a TanH layer creates a vector of new candidate values, \tilde{C}_t , that could be added to the state. In the next step, we'll combine these two to create an update to the state. In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

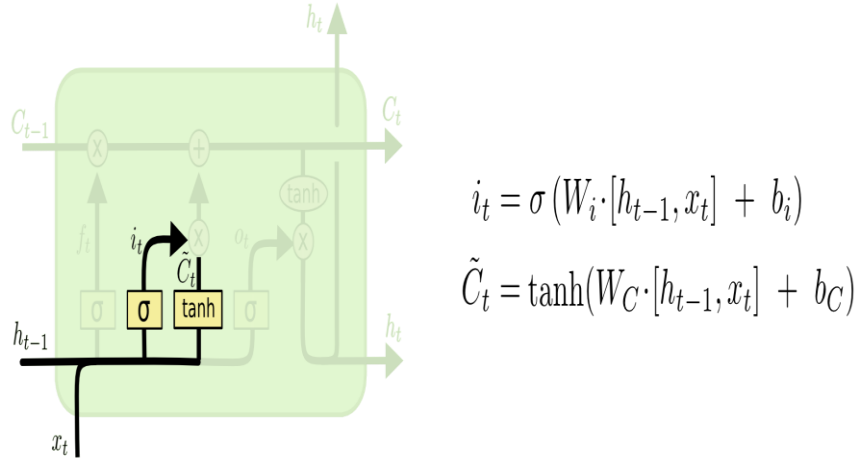


Fig. 4.10 Model of storing information in cell state

It's now time to update the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to do it. We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we'd drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

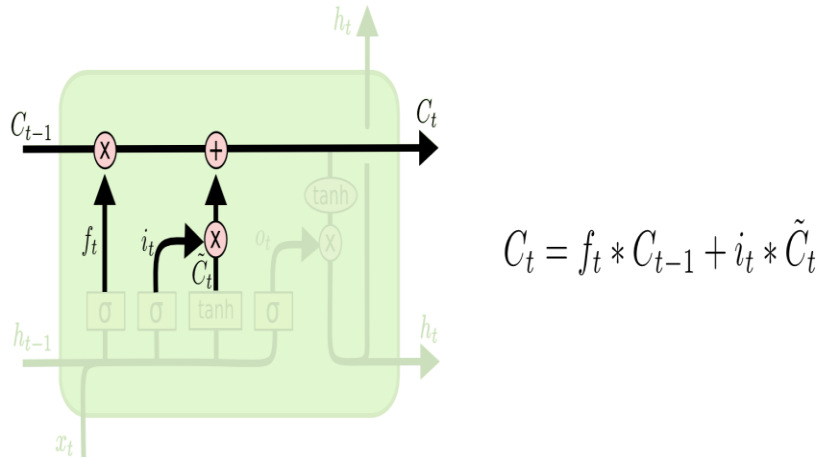


Fig. 4.11 Model of dropping old and loading new information

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what

parts of the cell state we're going to output. Then, we put the cell state through \tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

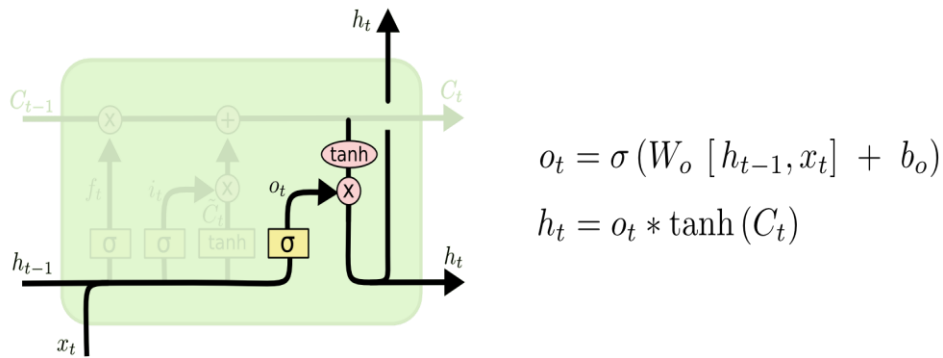


Fig. 4.12 Generating the output model

5. MODULES

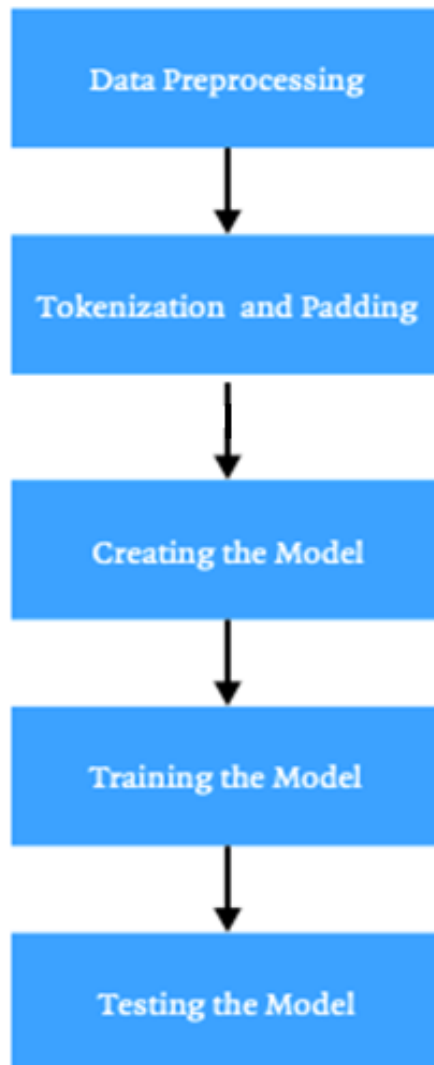


Fig. 5.1 Modules in the project

5.1 Data Preprocessing

Data Set

A dataset is the information or data required in data science or machine learning. The data is normally obtained from historical observations. There are normally two or three datasets in a project: a training dataset, and either a development dataset or a validation and a test dataset.

- Training datasets are used for building models.
- The other datasets are used for fine-tuning and picking the best performing model, and for checking how well the chosen model generalizes well to unseen examples.

In dataset, process the input is not required, however, two copies of the translated sentence need to be generated one with the start-of-sentence token and the other with the end-of-sentence token. In this project, we used a corpus of 25000 size of dataset which consist of Hindi and English corpus.

Data Preprocessing

Data preprocessing/preparation/cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a dataset, or and refers to identifying incorrect, incomplete, irrelevant parts of the data and then modifying, replacing, or deleting the dirty or coarse data. Data pre-processing has a significant impact on the performance of supervised learning models (Kotsiantis et al., 2006) because unreliable samples probably lead to wrong outputs. Although fashion sales data are usually noisy and influenced by various unpredictable external factors, previous studies in fashion sales forecasting did not consider data pre-processing of sales data. Effective data pre-processing methods are applied in this study to avoid the effects of noisy and unreliable data. Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Data preprocessing is used database-driven applications such as customer relationship

management and rule-based applications (like neural networks). Data goes through a series of steps during preprocessing:

- **Data Cleaning:** Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data.
- **Data Integration:** Data with different representations are put together and conflicts within the data are resolved.
- **Data Transformation:** Data is normalized, aggregated and generalized.
- **Data Reduction:** This step aims to present a reduced representation of the data in a data warehouse.
- **Data Discretization:** Involves the reduction of a number of values of a continuous attribute by dividing the range of attribute intervals.

Data Cleaning

Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.[1] Data cleansing may be performed interactively with data wrangling tools, or as batch processing through scripting.

After cleansing, a data set should be consistent with other similar data sets in the system. The inconsistencies detected or removed may have been originally caused by user entry errors, by corruption in transmission or storage, or by different data dictionary definitions of similar entities in different stores. Data cleaning differs from data validation in that validation almost invariably means data is rejected from the system at entry and is performed at the time of entry, rather than on batches of data.

Data Editing

Data editing is defined as the process involving the review and adjustment of collected survey data. The purpose is to control the quality of the collected data. Data editing can be performed manually, with the assistance of a computer or a combination of both.

Methods of editing

1. Interactive Editing:

The term interactive editing is commonly used for modern computer-assisted manual editing. Most interactive data editing tools applied at National Statistical Institutes (NSIs) allow one to check the specified edits during or after data entry, and if necessary to correct erroneous data immediately. Several approaches can be followed to correct erroneous data:

- Recontact the respondent
- Compare the respondent's data to his data from previous year
- Compare the respondent's data to data from similar respondents
- Use the subject matter knowledge of the human editor

Interactive editing is a standard way to edit data. It can be used to edit both categorical and continuous data. Interactive editing reduces the time frame needed to complete the cyclical process of review and adjustment.

2. Selective editing

Selective editing is an umbrella term for several methods to identify the influential errors and outliers. Selective editing techniques aim to apply interactive editing to a well-chosen subset of the records, such that the limited time and resources available for interactive editing are allocated to those records where it has the most effect on the quality of the final estimates of publication figures. In selective editing, data is split into two streams:

- The critical stream
- The non-critical stream

The critical stream consists of records that are more likely to contain influential errors. These critical records are edited in a traditional interactive manner. The records in the non-critical stream which are unlikely to contain influential errors are not edited in a computer assisted manner.

3. Macro editing

There are two methods of macro editing:

- Aggregation method:

This method is followed in almost every statistical agency before publication: verifying whether figures to be published seem plausible. This is accomplished by comparing quantities in publication tables with same quantities in previous publications. If an unusual value is observed, a micro-editing procedure is applied to the individual records and fields contributing to the suspicious quantity.

- Distribution method

Data available is used to characterize the distribution of the variables. Then all individual values are compared with the distribution. Records containing values that could be considered uncommon (given the distribution) are candidates for further inspection and possibly for editing.

- Automatic editing:

In automatic editing records are edited by a computer without human intervention[9]. Prior knowledge on the values of a single variable or a combination of variables can be formulated as a set of edit rules which specify or constrain the admissible values.

Data Reduction

Data reduction is the transformation of numerical or alphabetical digital information derived empirically or experimentally into a corrected, ordered, and simplified form. The basic concept is the reduction of multitudinous amounts of data down to the meaningful parts. When information is derived from instrument readings there may also be a transformation from analog to digital form. When the data are already in digital form the 'reduction' of the data typically involves some editing, scaling, encoding, sorting, collating, and producing tabular summaries. When the observations are discrete, but the underlying phenomenon is continuous then smoothing and interpolation are often needed. Often the data reduction is undertaken in the presence of reading or measurement errors. Some idea of the nature of these errors is needed before the most likely value may be determined.

Data wrangling

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. A data wrangler is a person who performs these transformation operations.

Neural machine translation models are often based on the sequence to sequence architecture. The seq2seq architecture is an encoder-decoder architecture which consists of two LSTM networks: the encoder LSTM and the decoder LSTM. The input to the encoder LSTM is the sentence in the original language; the input to the decoder LSTM is the sentence in the translated language with a start-of-sentence token. The output is the actual target sentence with an end-of-sentence token. In dataset, process the input is not required, however, two copies of the translated sentence need to be generated one with the start-of-sentence token and the other with the end-of-sentence token. Firstly, we need to mount the drive to the project, so that the data set is linked to the project.

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

Now we need to import the packages that are needed in the project. Here we are using Keras model, sklearn.utils, sklearn.model_selection, keras.layers, re files.

```
import re
import numpy as np
import pandas as pd
from keras.models import Model
```

Now we need to read the data set from the drive. As we are using Google colab, we can access the drive directly. Here we are reading the csv file by declaring a variable "data"

```
data = pd.read_csv('/content/drive/My Drive/data.csv')
data.head()
data = data[data['source'] == 'ted']
data = data.sample(n=25000, random_state=42)
print("num samples input:", len(data))
```

Here we are writing the code to know the size of the data present in the data set. From the

above sentences of the code we get the following output.

```
num samples input: 25000
```

Now we need to convert all the text into single format. Then remove the unnecessary symbols, special characters, numbers and extra spaces.

```
# convert to lower case
data['english_sentence'] = data['english_sentence'].apply(lambda x : x.lower())
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x : x.lower())
# remove extra spaces
data['english_sentence'] = data['english_sentence'].apply(lambda x: x.strip())
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: x.strip())
```

Now we need to get all the unique words in English and Hindi language. And we need to find out the number of words in each sentence. So we write the following code

```
all_eng_words=set()

for eng in data['english_sentence']:

    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

data['length_hin_sentence'] = data['hindi_sentence'].apply(lambda x:len(x.split(" ")))
```

The length of the sentence should be fixed. Because the English sentence may have ‘n’ number of words and Hindi sentence will have ‘n’ number of words. If the both lengths of the sentences do not match, they may not produce accurate output. To avoid this problem, we fix the length of the sentences manually.

5.2 Tokenization and Padding

Tokenization

The purpose of tokenization is to swap out sensitive data—typically payment card or bank account numbers—with a randomized number in the same format but with no intrinsic value of its own. This differs from encryption, where a number is mathematically changed, but its original pattern is still stored within the new code—known as format-preserving encryption. Tokenization is the process of removing sensitive data from your business systems by replacing it with an undecipherable token and storing the original data in a secure cloud data vault. Encrypted numbers can be decrypted with the appropriate key. Tokens, however, cannot be reversed, because there is no mathematical relationship between the token and its original number. Detokenization is the reverse process, exchanging the token for the original number. Detokenization can be done only by the original tokenization system. There is no other way to obtain the original number from just the token. Tokens can be single-use (a one-time debit card transaction) that are not retained or multi-use (a credit card number of a repeat customer) that are stored in a database for recurring transactions.

The goal of a tokenization platform is to remove any original sensitive payment or personal data from your business systems, replace each value with an undecipherable token, and store the original data in a secure cloud data vault separate from your data environment. For example, when you process a payment using the token stored in your systems, only the original tokenization system can swap the token with the corresponding PAN (primary account number) and send it to the payment processor for authorization. Your systems never record, transmit, or store the PAN—only the token.

Padding

Padding may refer to any of the following:

- Padding is a term used to describe the process of filling a field with pad characters. For example, if a name field required ten characters and your name was "Bob" (3 characters) the field would be "Bob0000000" where the 0's are the padding characters.

- Padding is white space immediately surrounding an element or another object on a web page. The picture below helps demonstrate the difference between padding and a margin when working with CSS. As can be seen, the padding is in the border, and the margin is outside the border. For example, with a table cell, cellpadding can be added to the <table> tag to add white space around the text in a cell.

The next step is tokenizing the original and translated sentences and applying padding to the sentences that are longer or shorter than a certain length, which in case of inputs will be the length of the longest input sentence. And for the output this will be the length of the longest sentence in the output.

For tokenization, the Tokenizer class from the keras. Preprocessing text library can be used. The tokenizer class performs two tasks:

- It divides a sentence into the corresponding list of word
- Then it converts the words to integers

This is extremely important since deep learning and machine learning algorithms work with numbers. The following script is used to tokenize the input sentences:

```
input_words = sorted(list(all_eng_words))
num_encoder_tokens = len(all_eng_words)
input_word2index = dict([(word, i+1) for i, word in enumerate(input_words)])
input_index2word = dict((i, word) for word, i in input_word2index.items())
```

From the comparison of the number of unique words in the input and the output, it can be concluded that English sentences are normally shorter and contain a smaller number of words on average, compared to the translated Hindi sentences.

Next, the input needs to be padded. The reason behind padding the input and the output is that text sentences can be of varying length, however LSTM (the algorithm that we are going to train our model) expects input instances with the same length. Therefore, we need to convert our sentences into fixed-length vectors. One way to do this is via padding.

In padding, a certain length is defined for a sentence. In our case the length of the longest sentence in the inputs and outputs will be used for padding the input and output sentences,

respectively. The longest sentence in the input contains 6 words. For the sentences that contain less than 6 words, zeros will be added in the empty indexes.

5.3 Creating the model

The first thing to be done is to define our outputs, since the output will be a sequence of words. Recall that the total number of unique words in the output are 2500. Therefore, each word in the output can be any of the 2500 words. The length of an output sentence is 26. And for each input sentence, a corresponding output sentence is needed. Therefore, the final shape of the output will be:

(number of inputs, length of the output sentence, the number of words in the output)

The following script creates the empty output array:

```
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2,
random_state=42)
```

To make predictions, the final layer of the model will be a dense layer, therefore the outputs need to be in the form of one-hot encoded vectors, since we will be using softmax activation function at the dense layer.

```
def generate_batch(x, y, batch_size = 128):
    while True:
        for j in range(0, len(x), batch_size):

            # encoder input seq
            for t, word in enumerate(target_text.split()):
                if t < len(target_text.split()) - 1:
                    decoder_input_data[i, t] = target_word2index[word]
            ] # decoder input seq
            if t > 0:
                decoder_target_data[i, t -
1, target_word2index[word]] = 1.
```

Next, encoder and decoders are created. The input to the encoder will be the sentence in

English and the output will be the hidden state and cell state of the LSTM.

The following script defines the encoder:

```
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(num_encoder_tokens, 300, mask_zero=True)(encoder_inputs)
```

The next step is to define the decoder. The decoder will have two inputs: the hidden state and cell state from the encoder and the input sentence, which actually will be the output sentence with an token appended at the beginning.

The following script creates the decoder LSTM:

```
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, 300, mask_zero=True)
dec_emb = dec_emb_layer(decoder_inputs)
```

Finally, the output from the decoder LSTM is passed through a dense layer to predict decoder outputs. The next step is to compile the model:

```
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

This step deals with making predictions using English sentences as inputs. In the tokenization steps, words are converted to integers. The outputs from the decoder will also be integers. However, the output has to be a sequence of words in the Hindi language. To do so, the integers have to be converted back to words. New dictionaries are created for both inputs and outputs where the keys will be the integers and the corresponding values will be the words.

5.4 Training the model

A Training data is basically a type of data used for training a new application, model or system through various methods depending on the project's feasibility and requirements. And training data for AI or ML is slightly different, as they are labeled or annotated with certain techniques to make it recognizable to computer that helps machines to understand the objects. Most of the training data contains the pair of input gathered from the various resources and then organized and annotated with certain techniques with accuracy.

The data could be different due to model algorithms and the field for which it is developed while ensuring the accuracy level to make sure the prediction should be accurate.

Types of Training Data for Machine Learning:

In machine learning training data is the key factor to make the machines recognize the objects or certain patterns and make the right prediction when used in real-life. Basically, there are three types of training data used in machine learning model development and each data has its own importance and role in building a ML model.

- **Training Data:**

Training data is the main and most important data which helps machines to learn and make the predictions. This data set is used by machine learning engineer to develop your algorithm and more than 70% of your total data used in the project. A huge quantity of datasets are used to train the model at best level to get the best results.

- **Validation Data:**

This is the second type of data set used to validate the machine learning model before final delivery of project. ML model validation is important to ensure the accuracy of model prediction to develop a right application. Using this type of data helps to know whether model can correctly identify the new examples or not.

- **Testing Data:**

This is the final and last type of data helps to check the prediction level of machine learning and AI model. Its is similar to validation data in testing the model accuracy but don't help to improve the prediction level. It is basically used to test the model whether it

will work well in real-life use and final test in the moment of truth for the model, if it works perfectly.

Importance of Training the data:

It's true, without training data AI or ML is not possible. The quality, relevancy and availability of your data directly affects the goals of AI model. Incomplete or inaccurate data sets will train your AI model like a illiterate human that can't understand his environment better. Hence, choosing the right data for your model will also help you to get the accurate results. Hence, your AI deserves best data that are precisely annotated and labeled that can only help your AI model to achieve the best level of accuracy at affordable cost. If you are looking for such high-quality data sets for your machine learning or AI model you can get in touch with Cogito which is providing machine learning training datasets in various forms as per the needs and adaptability of the project. It is involved in text, video and image annotation services to give the precisely annotated data at low-cost while ensuring the privacy and security of data till the deliver of the projects.

```
generator = generate_batch(train_x, train_y),
steps_per_epoch = train_samples // 128,
epochs = 10,

encoder_model = Model(encoder_inputs, encoder_states)

decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_inputs)
```

In the script above the input sequence to the encoder_model is passed, which predicts the hidden state and the cell state, which are stored in the states_value variable. Next, a variable target_seq, is defined which is a 1 x 1 matrix of all zeros. The target_seq variable contains the first word to the decoder model, which is <sos>. After that, the eos variable is initialized, which stores the integer value for the <eos> token. In the next line, the output_sentence list is defined, which will contain the predicted translation.

```
def predict(input_seq):  
    states_value = encoder_model.predict(input_seq)  
    target_seq = np.zeros((1,1))  
    target_seq[0, 0] = index
```

Next, a for loop is executed. The number of execution cycles for the for loop is equal to the length of the longest sentence in the output. Inside the loop, in the first iteration, the decoder_model predicts the output and the hidden and cell states, using the hidden and cell state of the encoder, and the input token. The index of the predicted word is stored in the idx variable. If the value of the predicted index is equal to the <eos> token, the loop terminates. Else if the predicted index is greater than zero, the corresponding word is retrieved from the idx2word dictionary and is stored in the word variable, which is then appended to the output_sentence list. The states_value variable is updated with the new hidden and cell state of the decoder and the index of the predicted word is stored in the target_seq variable. In the next loop cycle, the updated hidden and cell states, along with the index of the previously predicted word, are used to make new predictions. The loop continues until the maximum output sequence length is achieved or the <eos> token is encountered. Finally, the words in the output_sentence list are concatenated using a space and the resulting string is returned to the calling function.

5.5 Testing the model

When considering the strategy of Machine Learning testing, think accuracy and efficiency as a primary goal in the quality assurance. Benefits such as detecting redundant unsuccessful tests, and keeping untested code out of production, prediction, and prevention ultimately reduce much of the risk in the deployment phase. Some of the critical contribution's quality assurance include –

Defect alerts Enhanced analytics Faster predictions Improved optimization Cleaner traceability Real-time feedback

The sooner can implement an in-house AI platform to assist in application testing; will discover a more accurate and efficient deployment with reduced effort. Using defined test metrics and analytics will launch application development to new heights. The Machine Learning field provides tools to make decisions automatically from data to achieve any goal or requirement. Some problems resist a manually specified solution. Machine learning matters as it provides methods to create solutions for complex problems. Machine learning promises to solve problems automatically, faster and more accurately than a manually specified solution and at a larger scale. Machine learning applications are not 100% accurate, and approx. never will be. There are some of the reasons why testers cannot ignore learning about Machine learning and Deep learning. The fundamental reason is that these applications learning limited by data they have used to build algorithms. As Machine learning apps are managing almost daily activity performed by humans – one error can lead to severe losses. You may also love to read more about System Testing in this insight.

Best Practices for ML and DL Model Testing

Both testing practices and results have changed to accommodate applications that don't behave the same as traditional software. Traditional testing techniques based on fixed inputs. Testers believe that given inputs x and y, the output will be z and this will be constant until the application changes. It is not true in machine learning systems. The output does not fix. It will change over time. It is similar to the model built on Machine Learning system evolving as more data fed. It forces the testing professional to think differently and adapt test strategies that are very different from traditional testing techniques.

Critical activities that will be essential to test machine learning systems –

- Developing training data sets – This refers to a data set of examples used for training the model. In this data set, you have the input data with the expected output. This data usually prepares by collecting data in a semi-automated way. The goal of such a continuous learning system is to ensure the highest possible quality of the exposed model. The feedback data store is used to evaluate the quality of the server model, automatically retrain the deployed model, and finally to re-deploy the

new model version.

- Developing test data sets – It is a subset or part of the training dataset that is built to test all the possible combinations, also estimates how well the model trains. Based on the test data set results, the model fine-tuned.

To test the code, a sentence is randomly chosen from the input sentences list, retrieve the corresponding padded sequence for the sentence, and will pass it to the predict() method. The method will return the translated sentence as shown below.

Here is the script to test the functionality of the mode

```
rain_gen = generate_batch(train_x, train_y, batch_size = 1)
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = predict(input_seq)

print('Input English sentence:', train_x[0:1].values[0])
print('Actual Hindi Translation:', train_y[0:1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-3])
```

Finally, we evaluate the models using appropriate metrics. The output as follows,

```
Input English sentence: for eight billion people living in cities
Actual Hindi Translation: े शहरों में रहने वाले अरब लोगों को
Predicted Hindi Translation: ऐसे शहरों में रहने वाले अरब लोगों
```

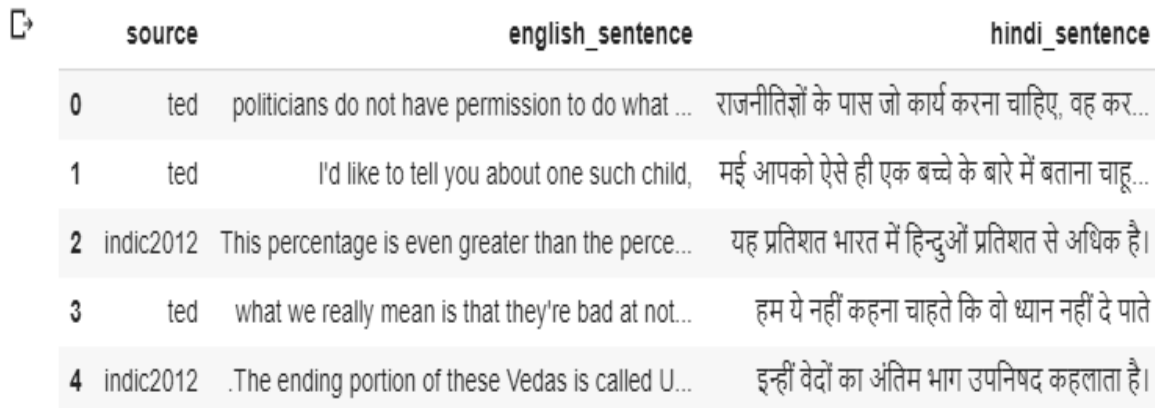
6. IMPLEMENTATION

DATA PREPROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

import re
import numpy as np
import pandas as pd
from keras.models import Model
from sklearn.utils import shuffle
from string import digits, punctuation
from sklearn.model_selection import train_test_split
from keras.layers import Input, LSTM, Embedding, Dense
data = pd.read_csv('/content/drive/My Drive/data.csv')
data.head()
```



	source	english_sentence	hindi_sentence
0	ted	politicians do not have permission to do what ...	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह कर...
1	ted	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहू...
2	indic2012	This percentage is even greater than the perce...	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।
3	ted	what we really mean is that they're bad at not...	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते
4	indic2012	.The ending portion of these Vedas is called U...	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।

Figure 6.1 Output of the data set

```
data = data[data['source'] == 'ted']
data = data[~pd.isnull(data['english_sentence'])]
data.drop_duplicates(inplace=True)

data = data.sample(n=25000, random_state=42)
print("num samples input:", len(data))
```



Fig 6.2 Size of data set

```
# convert to lower case
data['english_sentence'] = data['english_sentence'].apply(lambda x :
x.lower())
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x : x.lo
wer())

# remove quotes
data['english_sentence'] = data['english_sentence'].apply(lambda x: r
e.sub("'", '', x))
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: re.su
b("'", '', x))

# remove special characters
puncs = set(punctuation)
data['english_sentence']=data['english_sentence'].apply(lambda x: ''.
join(c for c in x if c not in puncs))
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: ''.join
(c for c in x if c not in puncs))

# remove all numbers from text
remove_digits = str.maketrans('', '', digits)
data['english_sentence'] = data['english_sentence'].apply(lambda x: x
.translate(remove_digits))
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: x.tra
nslate(remove_digits))
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: re.su
b("[२३०८१५७९४६]", "", x))

# remove extra spaces
```

```
data['english_sentence'] = data['english_sentence'].apply(lambda x: x
.strip())
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: x.str
ip())
data['english_sentence'] = data['english_sentence'].apply(lambda x: r
e.sub(" +", " ", x))
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: re.su
b(" +", " ", x))
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x : '<S>
'+ x + ' <E>')
all_eng_words=set()
for eng in data['english_sentence']:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

all_hindi_words=set()
for hin in data['hindi_sentence']:
    for word in hin.split():
        if word not in all_hindi_words:
            all_hindi_words.add(word)

data['length_eng_sentence'] = data['english_sentence'].apply(lambda x
:len(x.split(" ")))
data['length_hin_sentence'] = data['hindi_sentence'].apply(lambda x:l
en(x.split(" ")))
data = data[data['length_eng_sentence']<=20]
data = data[data['length_hin_sentence']<=20]
max_len_hin = max(data['length_hin_sentence'])
max_len_eng = max(data['length_eng_sentence'])
```

TOKENISATION

```
input_words = sorted(list(all_eng_words))

target_words = sorted(list(all_hindi_words))

num_encoder_tokens = len(all_eng_words)

num_decoder_tokens = len(all_hindi_words) + 1 # 1 for padding
```

```
input_word2index = dict([(word, i+1) for i, word in enumerate(input_ords)])
target_word2index = dict([(word, i+1) for i, word in enumerate(target_ords)])
```

```
input_index2word = dict((i, word) for word, i in input_word2index.items())
target_index2word = dict((i, word) for word, i in target_word2index.items())
```

CREATING THE MODEL

```
data = shuffle(data)
```

```
x, y = data['english_sentence'], data['hindi_sentence']
```

```
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
def generate_batch(x, y, batch_size = 128):
    while True:
        for j in range(0, len(x), batch_size):
            encoder_input_data = np.zeros((batch_size, max_len_eng), dtype='float32')
            decoder_input_data = np.zeros((batch_size, max_len_hin), dtype='float32')
            decoder_target_data = np.zeros((batch_size, max_len_hin, num_decoder_tokens), dtype='float32')
            for i, (input_text, target_text) in enumerate(zip(x[j:j + batch_size], y[j:j + batch_size])):
                for t, word in enumerate(input_text.split()):
                    encoder_input_data[i, t] = input_word2index[word]
            # encoder input seq
            for t, word in enumerate(target_text.split()):
                if t < len(target_text.split()) - 1:
                    decoder_input_data[i, t] = target_word2index[word]
            # decoder input seq
            if t > 0:
                decoder_target_data[i, t - 1, target_word2index[word]] = 1.

            yield([encoder_input_data, decoder_input_data], decoder_target_data)
```

```
encoder_inputs = Input(shape=(None,))
```

```
enc_emb = Embedding(num_encoder_tokens, 300, mask_zero=True)(encoder_inputs)
encoder_lstm = LSTM(300, return_state=True)
encoder_outputs, h, c = encoder_lstm(enc_emb)
encoder_states = [h, c]

decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, 300, mask_zero=True)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = LSTM(300, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

TRAINING

```
train_samples = len(train_x)
test_samples = len(test_x)
print(model.summary())

model.fit_generator(
    generator = generate_batch(train_x, train_y),
    steps_per_epoch = train_samples // 128,
    epochs = 10,

    validation_data = generate_batch(test_x, test_y),
    validation_steps = test_samples // 128
)
```

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, None)	0	
input_2 (InputLayer)	(None, None)	0	
embedding_1 (Embedding)	(None, None, 300)	4209000	input_1[0][0]
embedding_2 (Embedding)	(None, None, 300)	5262300	input_2[0][0]
lstm_1 (LSTM)	[(None, 300), (None, 721200)]		embedding_1[0][0]
lstm_2 (LSTM)	[(None, None, 300), 721200]		embedding_2[0][0] lstm_1[0][1] lstm_1[0][2]
dense_1 (Dense)	(None, None, 17541)	5279841	lstm_2[0][0]

Total params: 16,193,541
 Trainable params: 16,193,541
 Non-trainable params: 0

Fig.6.3 Summary of the model

None

Epoch 1/10

154/154 [=====] - 33s 215ms/step - loss: 0.0793 - val_loss: 7.7335

Epoch 2/10

154/154 [=====] - 32s 209ms/step - loss: 0.0734 - val_loss: 7.7513

Epoch 3/10

154/154 [=====] - 32s 211ms/step - loss: 0.0710 - val_loss: 7.7541

Epoch 4/10

154/154 [=====] - 33s 211ms/step - loss: 0.0664 - val_loss: 7.7920

Epoch 5/10

154/154 [=====] - 33s 212ms/step - loss: 0.0635 - val_loss: 7.7988

Epoch 6/10

154/154 [=====] - 33s 212ms/step - loss: 0.0628 - val_loss: 7.8195

Epoch 7/10

154/154 [=====] - 33s 212ms/step - loss: 0.0592 - val_loss: 7.8334

Epoch 8/10

154/154 [=====] - 33s 212ms/step - loss: 0.0579 - val_loss: 7.8358

Epoch 9/10

154/154 [=====] - 33s 211ms/step - loss: 0.0567 - val_loss: 7.8434

Epoch 10/10

154/154 [=====] - 33s 213ms/step - loss: 0.0553 - val_loss: 7.8638

<keras.callbacks.History at 0x7f47e9592898>

Fig. 6.4 Training the model

```
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(300,))
decoder_state_input_c = Input(shape=(300,))

decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c
]

dec_emb2= dec_emb_layer(decoder_inputs)

decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial
_state=decoder_states_inputs)

decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2)

decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2
)
```

TESTING

As we work with datasets, a machine learning algorithm works in two stages. One is training stage and the other is testing stage. Under supervised learning, we usually use two different files, one for training the data and the other for testing the data.

```
def predict(input_seq):
    states_value = encoder_model.predict(input_seq)

    target_seq = np.zeros((1,1))
    target_seq[0, 0] = target_word2index['<S>']

    sentence = ''
    while True:
        output_tokens, h, c = decoder_model.predict([target_seq] + st
ates_value)

        index = np.argmax(output_tokens[0, -1, :])
        word = target_index2word[index]
```



```
if word == '<E>' or len(sentence) > 50:
    break

sentence += ' ' + word

target_seq = np.zeros((1,1))
target_seq[0, 0] = index
states_value = [h, c]

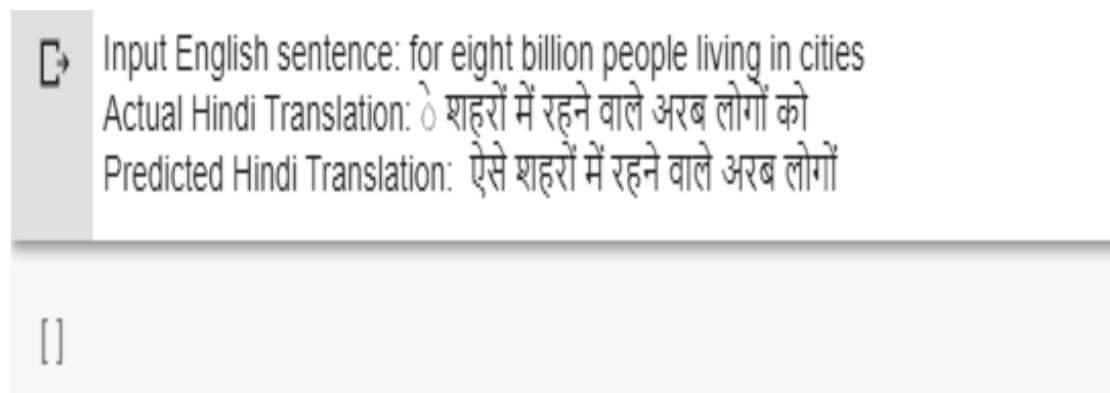
return sentence

train_gen = generate_batch(train_x, train_y, batch_size = 1)

(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = predict(input_seq)

print('Input English sentence:', train_x[0:1].values[0])
print('Actual Hindi Translation:', train_y[0:1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-3])
```

The Final obtained output is:



```
[ ]
```

Input English sentence: for eight billion people living in cities
Actual Hindi Translation: े शहरों में रहने वाले अरब लोगों को
Predicted Hindi Translation: ऐसे शहरों में रहने वाले अरब लोगों

Fig. 6.5 Final Output

8. CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION

India is linguistically rich and diverse country, So there will always be a requirement for translation from one language to another language. Machine Translation, translates text from English language to Hindi language text in domain of administration which translates documents such as government appointments, circulars, office orders etc. There are mainly three approaches for Machine Translation. They are Linguistic or rule based, Non-Linguistic and Hybrid. However, these methods have limitations in terms of scope and efficiency. That is, these can translate when input sentences are grammatically and syntactically correct, and the translated output may not be in commonly used language / style.

A new approach based on Deep Learning algorithm has been created by Google. This uses a Recurrent Neural Network plus an Encoder at input end and a Decoder. In this model, the Recurrent Neural Network is a self-learning algorithm that learns patterns in a sequence of input data. The Encoder converts this input into a unique sequence of digits. The Decoder takes these unique numbers and converts them into sentences based on its own self-learning algorithm. So, to translate from English to Hindi, the first Recurrent Neural Network and Encoder has to be “trained” on English, so that it recognize an input sentence in English, and converted into unique data, whereas the Decoder and the second Recurrent Neural Network have to be “trained” on Hindi to take the unique data and give a Hindi sentence as output.

8.2 FUTURE SCOPE

The future scope of the project is to improve performance and accuracy of the output sentence, while working with large dataset.

9. REFERENCES

1. Dubey P., Overcoming the Digital Divide through Machine Translation, Translation Journal, Volume 15, 2011, http://translationjournal.net/journal/55mt_india.htm [Dec 2011].
2. International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-2S4, July 2019
3. Sandeep Saini ,Department of Electronics and Communication Engineering The LNM Institute of Information Technology,Jaipur, India,sandeep.saini@lnmiit.ac.in
4. Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. CoRR, abs/1409.3215, 2014.
5. <https://www.researchgate.net/publication/327717152> NMT Conference Paper March 2018
6. Negi, Sapna, and P. Buitelaar. "Suggestion mining from opinionated text" Sentiment Analysis in Social Networks(2017): 129-139.
7. survey <http://www.cfilt.iitb.ac.in/resources/surveys/NMT-survey-paper.pdf>
8. <https://towardsdatascience.com/neural-machine-translation-15ecf6b0b>
9. Natural Language Processing (NLP) - <https://mlwhiz.com/nlpseries/>