MACHINE TRANSLATION FROM ENGLISH TO HINDI

Enrollment No.(s) -16103020, 16103234, 16103242

Name of Students -Shubham Dwivedi, Aman Singh Chauhan, Ishan Agarwal

Name of supervisor -Asst. Prof. K. Vimal Kumar

AUGUST 2019 – DECEMBER 2019

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

in

Computer Science Engineering

MAJOR PROJECT REPORT

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

TABLE OF CONTENTS

Chapter No.	Topics	Page No.
Chapter-1	**Introduction** 1.1 General Introduction 1.2 Problem Statement 1.3 Significance/Novelty of the problem 1.4 Empirical Study 1.5 Brief Description of the Solution Approach 1.6 Comparison of existing approaches to the problem framed	9 to 14
Chapter-2	**Literature Survey** 2.1 Summary of papers studied 2.2 Integrated summary of the literature studied	15 to 19
Chapter 3:	**Requirement Analysis and Solution Approach** 3.1 Overall description of the project 3.2 Requirement Analysis 3.3 Solution Approach	20 to 34
Chapter-4	**Modelling and Implementation Details** 4.1 Design 4.2 Implementation details	35 to 42
Chapter-5	**Testing (Focus on Quality of Robustness and Testing)** 5.1 Testing Cases for each Module 5.2 Limitations	43 to 44
Chapter-6	**Findings, Conclusion, and Future Work** 6.1 Findings 6.2 Conclusion 6.3 Future Work	45 to 46
References		47 to 50

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Place: Jaypee Institute of Information Technology, Noida

Date: 16 December 2019

Name: Shubham Dwivedi
Enrollment No: 16103020
Signature:

Name: Aman Singh Chauhan
Enrollment No: 16103234
Signature:

Name: Ishan Agarwal
Enrollment No: 16103242
Signature:

CERTIFICATE

This is to certify that the work titled "**MACHINE TRANSLATION FROM ENGLISH TO HINDI**" submitted by "Shubham Dwivedi, Ishan Agarwal and Aman Singh Chauhan" in partial fulfilment for the award of degree of B. Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree.

Signature of Supervisor:
Name of Supervisor: K. Vimal Kumar
Designation: Assistant Professor
Date: 16 December 2019

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude and deep regards to our mentor Asst. Prof. K. Vimal Kumar for his exemplary guidance, monitoring and constant encouragement throughout the course of this major project. We also take this opportunity to express deep sense of gratitude to my group members who helped me in completing this task through various stages.

Name of Students: Shubham Dwivedi (16103020)
Aman Singh Chauhan (16103234)
Ishan Agarwal (16103242)

Date: 16/12/19

SUMMARY

Machine translation is the task of automatically converting source text in one language to text in another language. Given a sequence of text in a source language, there is no one single best translation of that text to another language. This is because of the natural ambiguity and flexibility of human language. This makes the challenge of automatic machine translation difficult, perhaps one of the most difficult in artificial intelligence: Classical machine translation methods often involve rules for converting text in the source language to the target language. The rules are often developed by linguists and may operate at the lexical, syntactic, or semantic level. This focus on rules gives the name to this area of study: Rule-based Machine Translation, or RBMT. The key limitations of the classical machine translation approaches are both the expertise required to develop the rules, and the vast number of rules and exceptions required. So, we have used Neural Machine Translation technique that uses RNNs and attention layers for translating. There are more than 30 common languages in the world. Hindi and English are the most common languages spoken in India therefore we have made a translation system that translates English to Hindi.

Supervisor's sign:
Name: Asst. Prof. K. Vimal Kumar
Date- 16/12/19

Signature of Students
Shubham Dwivedi:
Aman Singh Chauhan:
Ishan Agarwal:

LIST OF FIGURES

CHAPTER NO	FIGURE NO	FIGURE NAME	PAGE NO
Chapter 3	Figure 1	The Training process for one layer neural network	22
Chapter 3	Figure 2	Predicting a label using a RNN	25
Chapter 3	Figure 3	One hot encoding vector	25
Chapter 3	Figure 4	Encoder portion of NMT	26
Chapter 3	Figure 5	First step of Decoder	27
Chapter 3	Figure 6	Decoder portion of NMT	28
Chapter 3	Figure 7	First step of LSTM	29
Chapter 3	Figure 8	Encoder of NMT with LSTM	30
Chapter 3	Figure 9	Step 1 of Decoder using Global Attention	33
Chapter 6	Figure 10	Training results	44

LIST OF TABLES

CHAPTER NO	TABLE NUMBER	TABLE NAME	PAGE NO
Chapter 1	Table 1	Comparison of Existing Approaches to the Problem	12 – 13
Chapter 2	Table 2	Review of Research Paper	15 - 18

CHAPTER I: INTRODUCTION

As technology continues to advance, the world becomes more and more connected. Physical distance is no longer as big of a roadblock as it once was; allowing people to connect with one another over a number of different social platforms. Yet, while distance may not divide us like it once did, the vast number of spoken languages continues to make universal communication difficult. With over 30 languages with 50 million or more speakers in the world,language barriers are a common occurrence that hinder communication and collaboration all across the globe. And while translators have been effectively overcoming these barriers for centuries, the demand for translation far outweighs the supply of translators available. However, just as it helped overcome the roadblock of physical distance, technology is continually improving our ability to overcome language barriers as well. Currently, the most effective manner by which technology is able to accomplish this task is through neural machine translation (NMT).

PROBLEM STATEMENT

A huge amount of valuable resources is available on the web in English, which are often translated into local languages to facilitate knowledge sharing among local people who are not much familiar with English. However, translating such content manually is very tedious, costly, and time-consuming process. To this end, machine translation is an efficient approach to translate text without any human involvement. Neural machine translation (NMT) is one of the most recent and effective translation techniques amongst all existing machine translation systems.

SIGNIFICANCE AND NOVELTY PROBLEM

According to research firm Common Sense Advisory, 72.1 percent of the consumers spend most or all of their time on sites in their own language, 72.4 percent say they would be more likely to buy a product with information in their own language and 56.2 percent say that the ability to obtain information in their own language is more important than price. These are just a few of the many reasons that translation has become essential in the modern and ever more globalized world that we live in.

Machine translation is a tool that can help businesses and individuals in many ways. While machine translation is unlikely to totally replace human beings in any application where quality is really important, there are a growing number of cases that show how effective and useful machine translation can be.

Machine translation can be used on its own or in conjunction with human proofreaders and post-editors. Depending on the nature of content, it is not uncommon to use 3 approaches together (even on one project) – human translation only, machine translation only and a combination of machine translation with human post-editing.

Machine translation is useful in many areas, including:

Highly repetitive content where productivity gains from machine translation can dramatically exceed what is possible with just using translation memories alone (e.g. automotive manuals)

Content that is similar to translation memories but not exactly the same (e.g. government policy documents.)

Content that would not get translated otherwise due to cost, scale and volume limitations of human translations (e.g. millions of Chinese, Japanese, Korean and other language patents made available in English.)

High value content that is changing every hour and every day there is time sensitivity (e.g. stock market news.)

Content that does not need to be perfect but just approximately understandable (e.g. any website for a quick review.)

Knowledge content that facilitates and enhances the global spread of critical knowledge (e.g. customer support.)

Content that is created to enhance and accelerate communication with global customers who prefer a self-service model (e.g. knowledgebase.)

Content that would normally be too expensive or too slow to translate with a human only translation approach (e.g. many projects that have insufficient budget for a human only approach.)

Increasing the amount of content that can be translated within a budget (e.g. Dell has doubled the amount of content they translate without increasing their translation budget.)

Real-time communications where it would not be practical for a human to translate.

EMPIRICAL STUDY

The idea of machine translation has been studied on-and-off for several decades. Warren Weaver was the first to propose the idea of machine translation in 1949. Weaver was inspired by the concepts of code breaking from WWII and the idea of an underlying similarity between all languages. Since the conception of machine translation in 1949, research on the topic has transitioned through several active and stagnant periods. Before NMT, statistical machine translation (SMT) provided the most state-of-the art results. While many initially believed that SMT would eventually become the answer to machine translation, several issues, including the number of components that went into a single translation model and the lack of generalizability of a model, stagnated SMT progress and prevented SMT from ever providing perfect translations. After multiple decades of SMT research, in the past five years NMT has arose as a far superior method at performing the task of machine translation. The rise of neural machine translation is due in large part to the increasing popularity of deep learning and the advanced power of GPUs and computers. Cho et. al was the first group to propose the concept of NMT in their paper "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In the paper, Cho et. al proposed the use of two separate RNN networks to perform machine translation, a structure they called an RNN Encoder-Decoder. While the paper initially garnered little attention, it eventually led to a revolution in machine translation. The power of NMT was first introduced to the public at the 2015 OpenMT machine translation competition, where the model was so successful that over 90% of teams utilized the NMT structure in the following year's 2016 OpenMT competition. Now, five years since its conception, NMT has taken off, capturing the attention of tech giants like Google and Facebook as the cutting edge technology in machine translation. This project will investigate the mechanics and mathematical foundations of the NMT Encoder-Decoder architecture. Furthermore, we will look into the recent trend of attention-mechanisms and the added benefit they provide to NMT.

BRIEF DESCRIPTION OF THE SOLUTIONS APPROACH

Within NMT, the encoder-decoder structure is quite a popular RNN architecture. This architecture consists of two components: an encoder network that consumes the input text and a decoder network that generates the translated output text. The task of the encoder is to extract a fixed sized dense representation of the different length input texts. The task of the decoder is to generate the corresponding text in the destination language, based on the dense representation from the encoder. Usually both networks are RNNs, often LSTMs.

A common way of representing RNNs is to unroll them into a sequence of copies of the same static network A, each one fed by the hidden state of the previous copy $h(t-1)$ and by the current input $x(t)$.

An LSTM network is a particular type of RNN, relying on internal gates, to selectively remove (*forget*) or add (*remember*) information from/to the cell state $C(t)$ based on the input values $x(t)$ and the hidden state $h(t-1)$ at each time step t.
LSTM networks are RNNs, which could be represented as a sequence of copies of static network A connected via their hidden states. Below is a representation of an LSTM network trained to generate free text.

COMPARISION OF EXISTING APPROACHES TO THE PROBLEM SOLVED

Methods	Description
Rule-based machine translation (RBMT)	Rule-based machine translation (RBMT) is machine translation systems based on linguistic information about source and target languages basically retrieved from (unilingual, bilingual or multilingual) dictionaries and grammars covering the main semantic, morphological, and syntactic regularities of each language respectively. Having input sentences (in some source language), an RBMT system generates them to output sentences (in some target language) on the basis of morphological, syntactic, and semantic analysis of both the source and the target languages involved in a concrete translation task.
Direct Machine Translation	This is the most straightforward type of machine translation. It divides the text into words, translates them, slightly corrects the morphology, and harmonizes syntax to make the whole thing sound right, more or less. When the sun goes down, trained linguists write the rules for each word.
Transfer Based Machine Translation	In contrast to the simple direct model of MT, transfer MT breaks translation into three steps: analysis of the source language text to determine its grammatical structure, transfer of the resulting structure to a structure suitable for generating text in the target language, and finally generation of this text. Transfer-based MT systems are thus capable of using knowledge of the source and target languages.
Interlingual Machine Translation	In this approach, the source language, i.e. the text to be translated is transformed into an interlingua, i.e., an abstract language-independent representation. The target language is then generated from the interlingua.

Statistical Machine Translation Technology	Statistical machine translation utilizes statistical translation models whose parameters stem from the analysis of monolingual and bilingual corpora. Building statistical translation models is a quick process, but the technology relies heavily on existing multilingual corpora. A minimum of 2 million words for a specific domain and even more for general language are required. Statistical machine translation is CPU intensive and requires an extensive hardware configuration to run translation models for average performance levels.
Phrase-based SMT	This method is based on all the word-based translation principles: statistics, reordering, and lexical hacks. Although, for the learning, it split the text not only into words but also phrases. These were the n-grams, to be precise, which were a contiguous sequence of n words in a row. Thus, the machine learned to translate steady combinations of words, which noticeably improved accuracy.
Syntax Based Machine Translation	The proponents of syntax-based translation believed it was possible to merge it with the rule-based method. It's necessary to do quite a precise syntax analysis of the sentence — to determine the subject, the predicate, and other parts of the sentence, and then to build a sentence tree. Using it, the machine learns to convert syntactic units between languages and translates the rest by words or phrases. That would have solved the word alignment issue once and for all.

TABLE-I : Comparison of Existing Approaches to the Problem

CHAPTER II: LITERATURE SURVEY

SUMMARY OF PAPERS

Most of the papers which were published earlier focused on rule-based and statistical machine translation techniques. These approaches did not perform well but just helped in carrying forward the research in the area of machine translation. Most of the research papers focused on use of neural networks that mainly constituted of Recurrent Neural Networks. RNNs helped in memorizing the previous operations for predicting outputs based on previous inputs and outputs. Use of neural network and RNNs produced very great results with less computation requirement than the previous approaches. But there were some problems with RNNs such as gradient vanishing, exploding problems and processing for very long sequences. Due to these reasons LSTMs were discovered that helped in remembering information for a long period. Many research papers used LSTMs and GRUs that gave better results than RNNs. Some of the research paper made hybrid model using SMT , NMT which produced better results for some datasets but the model required a lot of training time. Some papers discussed about attention layers that overcame cons of LSTMs that are prone to overfitting.

PAPER - I

TITLE OF PAPER	A Hybrid Approach For Hindi-English Machine Translation
AUTHORS	Omkar Dhariya_, Shrikant Malviya*y* and Uma Shanker Tiwary
YEAR OF PUBLICATION	2016
SUMMARY	The first step in translation is to split the sentence into words or simple sentences (if the input sentence is complex/compound sentence or consists of phrases). If a part of the input is in the example database then it kept that part as it is and the remaining part is segmented into words. When the segmentation is done, the chunks or segments are actually translated and tagged independently. If the segment is example based, it is directly converted to English. Further, the remaining words are translated using the parallel Hindi-English Dictionary. Tagging is performed at the time of translation with the help of English POS tagger. SMT is used to resolve the ambiguity if a word in Hindi with multiple meaning in English is present. Pronouns are identified on the basis of its absence in the Hindi dictionary. Such kind of words are translated through transliteration rules and tagged as pronoun. On finishing the tagging of all the segments, RBMT rearranges them and constructs a valid sentence of it by applying proper grammar.

TABLE - II

PAPER - II

TITLE OF PAPER	Minimum Error Rate Training in Statistical Machine Translation
AUTHORS	Franz Josef Och
YEAR OF PUBLICATION	2015
SUMMARY	Described a new algorithm for efficient training an unsmoothed error count. Shows that significantly better results can often be obtained if the final evaluation criterion is taken directly into account as part of the training procedure. Presented alternative training criteria for log linear statistical machine translation models which are directly related to translation quality: an unsmoothed error count and a smoothed error count on a development corpus. For the unsmoothed error count, they presented a new line optimization algorithm which can efficiently find the optimal solution along a line. As a result, expect that actual 'true' translation quality is improved, as previous work has shown that for some evaluation criteria there is a correlation with human subjective evaluation of fluency and adequacy. However, the different evaluation criteria yield quite different results on their Chinese–English translation task and therefore we expect that not all of them correlate equally well to human translation quality.

TABLE - III

PAPER - III

TITLE OF PAPER	Building an effective MT system for English – Hindi Using RNNs
AUTHORS	Ruchit Agrawal and Dipti Misra Sharma
YEAR OF PUBLICATION	2017
SUMMARY	This paper discusses about the effectiveness of RNNs for the task of English to Hindi Machine Translation. Various experiments using different neural network architectures - employing Gated Recurrent Units, Long Short Term Memory Units and Attention Mechanism has been done with report for each architecture. 1) Recurrent Neural Network (RNN) deals with variable-length input and output that conventional feed forward network can't do. 2) Whenever a single sample is fed into a feed-forward neural network, the network's internal state, or the activations of the hidden units, is computed from scratch and is not influenced by the state computed from the previous sample while an RNN maintains its internal state while reading a sequence of inputs. 3) The model is able to produce grammatically fluent translations, as opposed to traditional approaches. 4) A bi-directional LSTM model with attention mechanism shows improvement over normal RNN's by solving problem of presence of repeated tokens and unknown or unmapped words.

TABLE - IV

PAPER - IV

TITLE OF PAPER	Incorporating Statistical Machine Translation Word Knowledge Into Neural Machine Translation
AUTHORS	Xing Wang , Zhaopeng Tu, and Min Zhang
YEAR OF PUBLICATION	2018
SUMMARY	This paper proposes a model that incorporates SMT word knowledge into NMT to reduce some limitations of NMT. The NMT decoder makes more accurate word prediction by referring to the SMT word recommendations in both training and testing phases. The SMT model offers informative word recommendations based on the NMT decoding information. This model uses the SMT word predictions as prior knowledge to adjust the NMT word generation probability, which unitizes a neural network based classifier to digest the discrete word knowledge. 1) The SMT decoder finishes the translation process when all source segments (words or phrases) are covered by the translation rules. It guarantees that each word in source sentence has been translated. 2) Words are treated as discrete symbols rather than continuous real-valued vectors in SMT, which ensures SMT models translate a source word to a semantic-related target word, alleviating imprecise translation problem. 3) Words are explicitly memorized in SMT translation rules. Therefore there are no UNK symbols in the SMT translation rules.

TABLE - V

CHAPTER III: REQUIREMENT ANALYSIS AND SOLUTION APPROACH

REQUIREMENT ANALYSIS

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

System Requirements for Training

Processor : Core i5 7^{th} generation
8 GB RAM
GPU : Nvidia Geforce

Dataset Description

A text file containing pairs of input language sentences (English) with their output language (Hindi) conversion downloaded from Kaggle.

OVERALL DESCRIPTION OF THE PROJECT

The RNN Encoder-Decoder structure is composed of two separate RNN networks: the encoder and the decoder. Towards understanding how this architecture works, it is first essential to understand the basics behind neural networks and recurrent neural networks (RNN). Therefore, this background section will first cover the fundamentals of a neural network and the key distinctions that make a neural network a recurrent neural network. From there, this section will take a brief look at the data processing technique of One Hot Encoding before finally diving into the Encoder-Decoder structure. Lastly, this section will discuss Long-Short Term Memory RNNs and Attention mechanisms; two concepts which vastly improve the effectiveness of NMT

SOLUTION APPROACH

Neural Network

The objective of any neural network is to train a set of learnable parameters to perform prediction tasks on inputted data. Any dataset used to train a model is made up of input data x composed of L_x features and an output target, y. In simpler machine learning tasks, this output target, y, is a single number corresponding to a value or class. However, in more complicated machine learning tasks, such as machine translation, this output target, y, is composed of L_y features. Before training begins, preprocessing of the data is performed. In general, for any machine learning task using a neural network the steps are as follows:

(1) Any preprocessing steps needed to make non-numeric data interpretable by computers are performed (see One Hot Encoding).
(2) The data is split into Train, Validation, and Test sets. A typical split is around 70%:20%:10%.
(3) Train data is split into mini-batches for training purposes.

With the data prepared, the training process can begin. At the most basic level, the process of training a machine learning model is made up of 4 separate steps. When working with a train set of size n, the first step of a neural network is to take an input xi for i ∈ [1, n] and predict a probability vector, yˆi , in the following way:

$$\hat{y}_i = \text{softmax}(Wx_i) \tag{1}$$

The softmax function in (1) is defined as follows: If v is a vector in R^n , p = softmax(v) is the vector in R^n defined as:

$$\vec{p} = \text{softmax}(\vec{v}) = \begin{bmatrix} \frac{e^{v_1}}{\sum_j e^{v_j}} \\ \vdots \\ \frac{e^{v_N}}{\sum_j e^{v_j}} \end{bmatrix} \tag{2}$$

In (1), W is referred to as a weight matrix, and is composed of a number of learnable parameters. Since the objective of W is to predict y_i for x_i , the dimensions of W in the above example are (Ly, Lx). In this way, the matrix-vector multiplication Wx_i transforms the input vector of length L_x (the number of input features) into a vector of length Ly (the number of classes). With these predicted classes, the second step in the training process is to compute a loss l_i from the computed y^i as shown below:

$$li = LossFunction(y\hat{}_i , y_i) \tag{3}$$

The LossFunction can be a number of different functions that analyze the accuracy of the predicted y^. Now, the third step of the training process is to repeat steps (1) and (2) on x_i for every $i \in [1, n]$ and take a summation over l_i to obtain a total loss L for a single loop through the entire dataset. This process is depicted by the following equation:

$$L = \sum_{i=1}^{n} l_i \tag{4}$$

A complete run through every training point in the dataset is referred to as a single epoch. Now, we can use this L to update W, the final step of the training process. L is differentiated with respect to W to obtain $\partial L/\partial W$. Since L is being differentiated by W, $\partial L/\partial W$ will retrieve a different gradient for each value in W, thus, $\partial L/\partial W$ will be a matrix the same size as W with each index corresponding to the gradient of the loss with respect to the weight in W at that particular index. Therefore, we can use $\partial L/\partial W$ to update W as follows:

$$W = W - lr(\partial L/\partial W) \tag{5}$$

lr is a hyperparameter referred to as the learning rate and determines by how much to update W.

Once complete, these four steps are performed iteratively for a specified number of epochs to train the model. These four steps which make up the process of training the network are depicted in Figure 1.

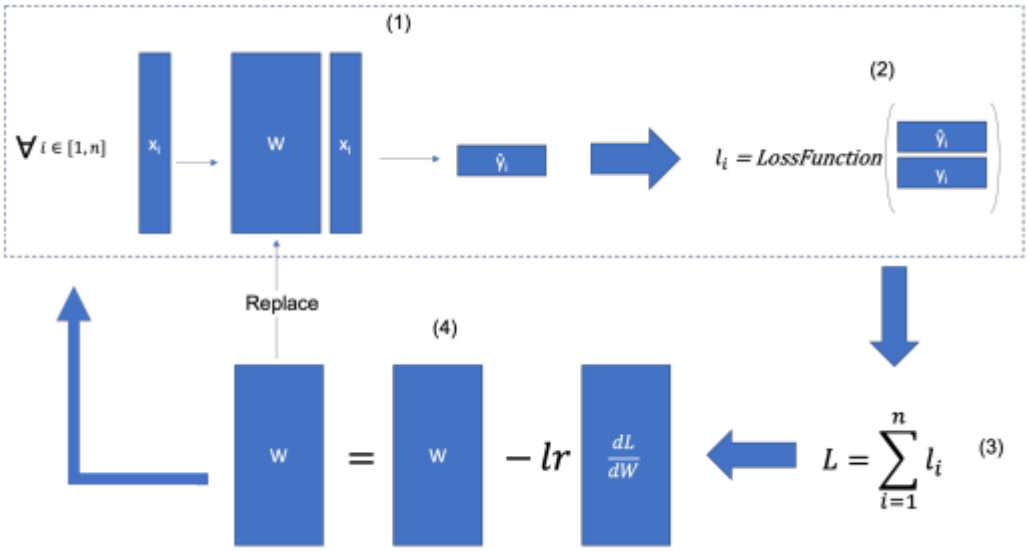


Fig 1 : The training process for one-layer neural network

In order to track the progress of the training process, the accuracy of the model on the validation set is computed at predetermined points in the training (ex. every 5 training loops). This is done by computing the loss on the validation set, L_{val} exactly like in the training steps 1- 3. Thus, the only difference between this process and the training process performed on the train set is that step 4 of the training process, where the parameters of the weight matrices are updated, is not performed. By calculating this loss, L_{val}, at points throughout training, the ability for the model to accurately make predictions on data beyond the train set can be tracked. This step is crucial in avoiding issues such as overfitting. Finally, once the training process is complete, the loss is computed on the test set, L_{test}, as another way of determining the effectiveness of the model. While the loss on the validation set, Lval, should represent the model's ability to generalize to data outside of the training set, the test set is used at the completion of the entire training process as a way of exposing the model to data it has never before seen and thus double checking this characteristic.

While it is generally good practice to use both a validation set and a test set, oftentimes a validation set is not used. This allows for more data to be used in the train set (for example an 80%:20% train set:test set split is common). If this method is employed, the accuracy of the model is just computed on the test set, rather than the validation set, at predetermined points in training. And while this does slightly decrease the ability to determine how the model generalizes to outside data, it oftentimes is the superior method due to the increased training set size.

As an important note, in the above example, since there was only a single weight matrix W, this type of network is referred to as a one-layer neural network. However, in the application of deep learning, several of these weight matrices are applied to the input data, with non-linear functions between them, before a prediction vector is calculated. While this paper will not cover this topic in detail, it is important to mention as it is the standard in neural networks today.

Recurrent Neural Network

In the task of neural machine translation, the input data comes in the form of time-series data. Recurrent Neural Networks are a particular type of neural network designed specifically to deal with the intricacies of temporal data. In the case of time-series data,1 the input of each training point, xi , consists of a number of sequential data points $xi_{t=1}$, $xi_{t=2}$, ..., $x_{it=Lx}$ of equal size, where Lx is the length of the time-series. Just like a standard neural network, the aim of a RNN is to predict the label y_i for each x_i . Training a RNN is very similar to training a standard neural network, with just slight modifications to steps 1 and 4 outlined in the Neural Network section above.2 In step 1 of training, a RNN analyzes each data point $xi_{t=1}$, $xi_{t=2}$, ..., $xi_{t=Lx}$ in consecutive order and stores information in a hidden vector, ht , to ultimately make a prediction, y^i , at the end of the sequence. $h_{t=0}$ is initialized to a zero vector, and updated at each time step t. This update process is defined by the following equation:

$$h_t = \sigma(Rh_{t-1} + Wx_{i_t}) \tag{6}$$

In (6), R and W are weight matrices and σ is a non-linear function such as tanh or sigmoid. Using this architecture, the hidden vector h_t captures the important information from the time series data by being updating at each time step t. Thus, at any time step s < Lx, the hidden state h_t=s contains information about the data points $xi_{t=1}$ up to $xi_{t=s}$. In this way, at any time step s, the hidden vector, $h_{t=s}$, can be used to create a prediction, y^i . This prediction vector is created just as in the first step of training a standard neural network, and is shown below:

$$\hat{y}_i = \text{softmax}(Uh_{t=s}) \tag{7}$$

Where, in (7), U is another weight matrix which converts the size of h_s to the size of the possible outcome classes. The modified first training step for a RNN is shown in Figure 2. Now, using this y^i , steps 2 and 3 of training a RNN are exactly the same as training a standard neural network in that (2) the loss is computed for y^i and (3) steps 1 and 2 are repeated for each training example and the losses are summed together to obtain a total loss L. With this L, a slightly modified step 4 is conducted to update all of the weight matrices W, R, and U. In particular, L is differentiated with respect to each weight matrix and used to update each matrix as depicted in the following set of equations:

$$W = W - lr(\partial L/\partial W)$$
$$R = R - lr(\partial L/\partial R)$$
$$U = U - lr(\partial L/\partial U) \quad (8)$$

Just as in any neural network, this process is then repeated for multiple epochs in order to train the RNN model. With an understanding of the structure of a RNN, let's turn our attention towards the topic of neural machine translation. However, we must first address how we transform text data into computationally friendly data.

One Hot Encoding

In machine translation, we are given a sentence in one language and tasked with translating the sentence to another language. The most obvious roadblock with machine translation is the issue of performing machine learning with textual data. The answer to this issue is one hot encoding, which involves converting each word in the dataset into a vector with a 0 at every index with the exception of a single 1 at the index corresponding to that particular word.3 To determine the size of these one hot encoding vectors, a separate vocabulary is created for both the input and output languages. Ideally, these vocabularies would be every unique word in each of these respective languages. However, given that a single language can have millions of unique words, vocabularies often consist of a subset of the most common words in each language. Typically, these vocabularies are allowed to be in the hundreds of thousands (if not millions) of words for each language. But, for the sake of example, let's let the vocabulary for our input language (English) consists of the words in Table 1. We are then able to use this vocabulary to make one-hot encoding vectors for each of the word in the input language where each vector will be the same size as the input vocabulary. For example, if one of our

input sentences is “the blue whale ate the red fish” the one hot encoding vectors for this sentence are shown in and Figure 3

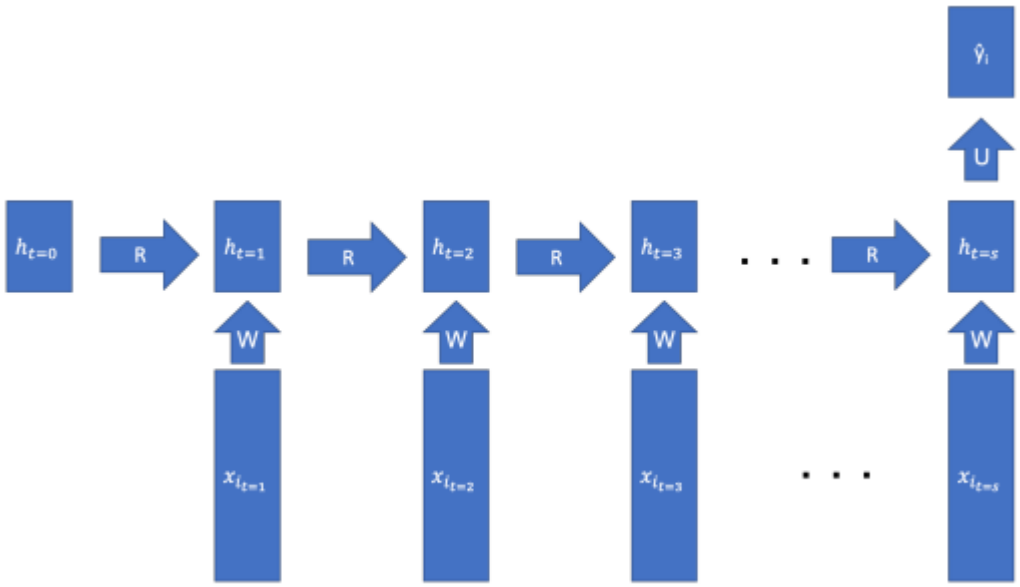


Fig 2 : Predicting a label using a RNN

The start of sentence, <SOS> and end of sentence,<EOS> , tags are added to the vocabulary to denote the start and end-points of sentences. These tags become important during the training process, which will become apparent in the next section. And, while not shown here, this same process must be done to the output sentences of the dataset (i.e. the y_s).

a	0
the	1
red	2
orange	3
blue	4
black	5
fish	6
whale	7
beaver	8
ate	9
drank	10
<SOS>	11
<EOS>	12

$$the = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \quad blue = \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \quad whale = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix} \quad ate = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix} \quad the = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \quad red = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \quad fish = \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Fig 3 : One hot encoding of “the blue whale ate the red fish”

Encoder-Decoder

With a way to transform textual data into numerical vectors, we can now begin the task of NMT. The training set for an RNN translating from language A to language B is composed of pairs of an input sentence in language A (the xi) and target sentence in language B (the yi). The structure of a single training pair is shown below:

$$x_i = x_{i_{t=1}}, x_{i_{t=2}}, ..., x_{i_{t=L_x}} \mid L_x = length(input\ sentence)$$
$$y_i = y_{i_{t=1}}, y_{i_{t=2}}, ..., y_{i_{t=L_y}} \mid L_y = length(output\ sentence) \quad (9)$$

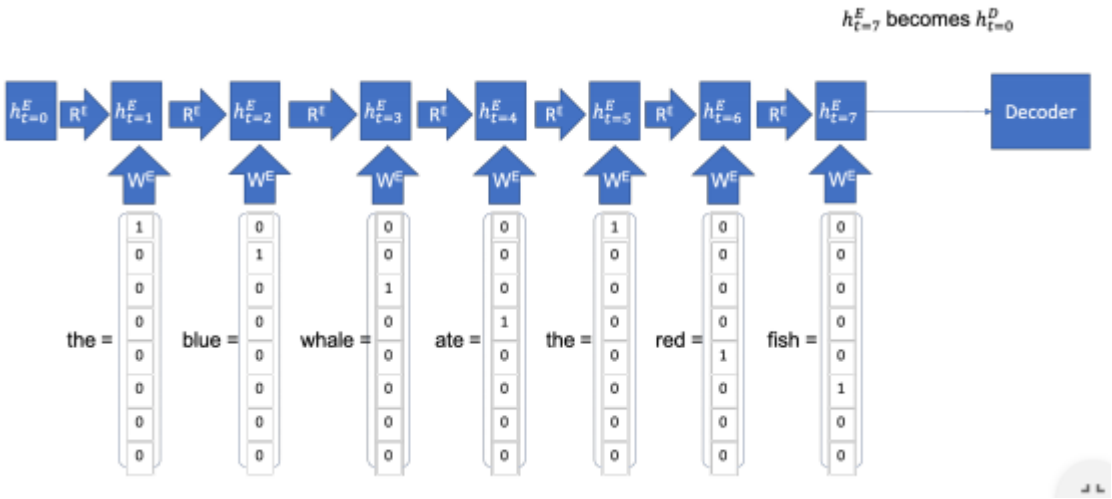


Fig 4 : Encoder portion of NMT

The obvious difference between this dataset and the type used in the RNN section is the fact that the target, y_i , is a variable length vector rather than a single label. In order to combat this issue, NMT utilizes the Encoder-Decoder architecture. In this structure, a first RNN (the encoder) analyzes the input sentence and passes its final hidden state, $h^E_{t=Lx}$, onto a second RNN (the decoder) to use as its first hidden state, $h^D_{t=0}$. For example, if we wanted to translate the English sentence ”the blue whale ate the red fish” the encoder portion of this translation would take “the blue whale ate the red fish” as input, and feed its final hidden vector, h E t=7 (t = 7 because the sentence is 7 words long), to the decoder to use as its first hidden vector h D $_{t=0}$ (Figure 4). On the decoder portion of the Encoder-Decoder, a separate RNN predicts words for the variable length output statement y^i . In the first time step, the $h^D_{t=0}$, retrieved from the encoder, and the token, which is used as the first input vector $xi_{t=1}$, are used to compute h D t=1. However, unlike the vanilla RNN described in the RNN section, the decoder outputs a

predicted word for each time step t up to Ly. Thus, the Decoder uses the hidden state h D t=1 to compute a predicted word y^it=1 in the same manner as the final step of a vanilla RNN. This process is depicted in the following equation and in Figure 5.

$$\hat{y}_{i_{t=1}} = \text{softmax}(U^D h^D_{t=1}) \tag{10}$$

With this prediction vector, y^i$_{t=1}$, a loss is computed for the first word, where yi$_{t=1}$ is the correct word for that output sentence at t = 1. The equation to compute the loss is shown below:

$$l_i = LossFunction(\hat{y}_{i_{t=1}}, y_{i_{t=1}}) \tag{11}$$

When testing the model, the index with the highest value in the prediction vector, y^i$_{t=1}$, determines what word becomes the next input vector, xi$_{t=2}$. However, during training, a more

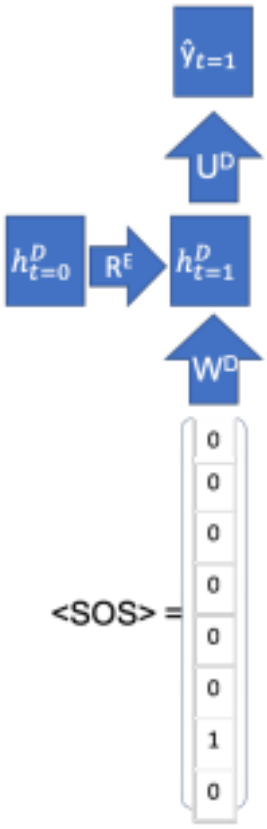


Fig 5 : First step of Decoder

common procedure, referred to as Teacher Forcing, is used to speed up the training process. In teacher forcing, rather than using the predicted word, the Decoder simply inputs the correct

word, $yi_{t=1}$, as the next input vector, $xi_{t=2}$. This process is then carried out Ly times until a prediction vector has been created for each word in the output sentence (Figure 6).

The losses for each word are summed to obtain a loss for the whole sentence, and this process is repeated for each word in the training set to obtain a total loss L. From here, L is differentiated with respect to each weight matrix and used to update both the encoder's and the decoder's weights, as shown in following set of equations:

$$\begin{aligned} W^E &= W^E - lr(\partial L/\partial W^E) \\ R^E &= R^E - lr(\partial L/\partial R^E) \\ W^D &= W^D - lr(\partial L/\partial W^D) \\ R^D &= R^D - lr(\partial L/\partial R^D) \\ U^D &= U^D - lr(\partial L/\partial U^D) \end{aligned} \tag{12}$$

Now, with an understanding of NMT, let's take a look at a more sophisticated RNN architecture which will go a long way in improving the results of machine translation.

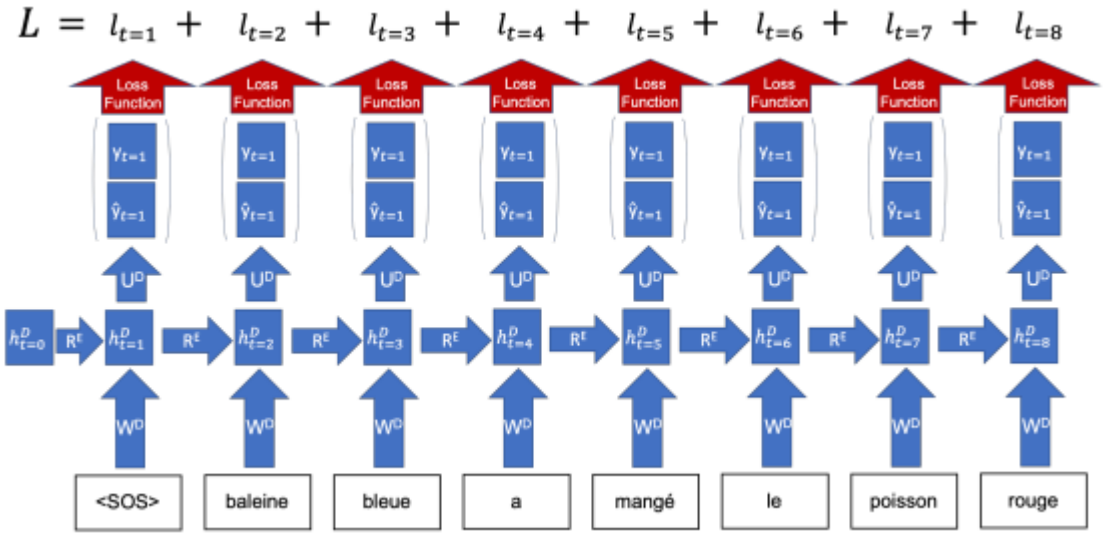


Fig 6 : Decoder portion of NMT

Long Short-Term Memory

The Long Short-Term Memory (LSTM) architecture is perhaps the most widely used RNN structure in large part due to its effectiveness at dealing with the issue of vanishing gradients. LSTM is modeled after the traditional RNN format, but incorporates a number of "gates," which work together with a memory cell, to smooth the back-propagation procedure and create a seamless flow through the network. In particular, at any time step t = s the LSTM structure

updates the hidden state, ht=s, by incorporating information from the current input vector, xt=s, the previous hidden state, ht=s−1, and an additional memory cell, ct=s. In order to conduct this update, three separate gates, known as the input gate (i), forget gate (f), and the output gate (o), are used to determine how much information to draw from the current input versus how much information to draw from the memory cell. All three of these gates are calculated using information from the current input vector, xt=s, and the previous hidden state, ht=s−1. To use information from each of these vectors, they are concatenated together [ht=s−1, xt=s]; meaning the xt=s vector of size Xx1 is added to the end of the ht=s−1 vector of size Hx1 to create a vector of size (H + X)x1. A separate weight matrix for each gate is then used to transform this concatenated vector, [ht=s−1, xt=s], into a vector the same size as the memory cell, and each of these transformed vectors are then passed through a σ function to produce a vector with

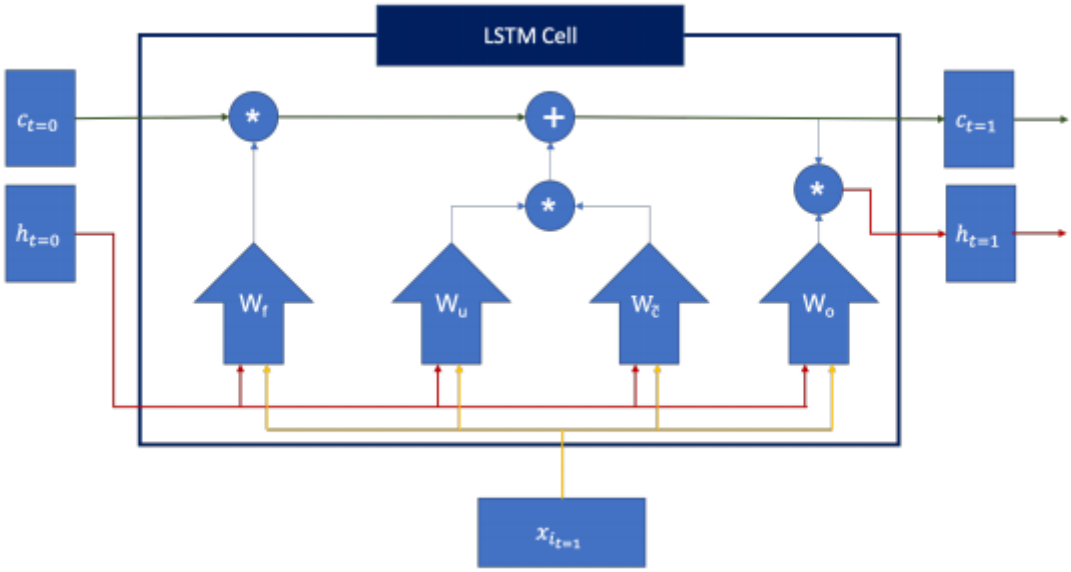


Fig 7 : First step of LSTM

weighted indices that sum to one. The equations that define each of these gates are below:

$$\begin{aligned} i &= \sigma(W_i[h_{t=s-1}, x_{t=s}]) \\ f &= \sigma(W_f[h_{t=s-1}, x_{t=s}]) \\ o &= \sigma(W_o[h_{t=s-1}, x_{t=s}]) \end{aligned} \tag{13}$$

Before these gates can be used, the current input vector xt=s and the previous hidden state, $h_{t=s-1}$ are used to generate $\tilde{c}_{t=s}$, which is defined by the following equation:

$$\tilde{c}_{t=s} = tanh(W_{\tilde{c}}[h_{t=s-1}, x_{t=s}]) \tag{14}$$

Now, the gates and c˜ are used to calculate a new c which is subsequently used with $c_{t=s-1}$ to generate a hidden state $h_{t=s}$. This process is depicted below:

$$c_{t=s} = (i * \tilde{c}_{t=s}) + (f * c_{t=s-1})$$
$$h_{t=s} = o * tanh(c_{t=s}) \tag{15}$$

In (15), the $*$ symbol signifies element wise multiplication; meaning the corresponding indices of each of the vectors are multiplied together. Figure 7 puts all these steps together into a visualization of the first step of an LSTM.

Replacing this LSTM architecture with the vanilla RNN in the NMT section above, the full Encoder can be visualized as in Figure 8. This same substitution is performed for the Decoder RNN to convert an NMT model into a full LSTM NMT architecture. The other steps of training remain exactly the same, with predictions computed from the hidden vector, ht , and the total loss, L, determined from these predictions. The total loss is then differentiated with respect

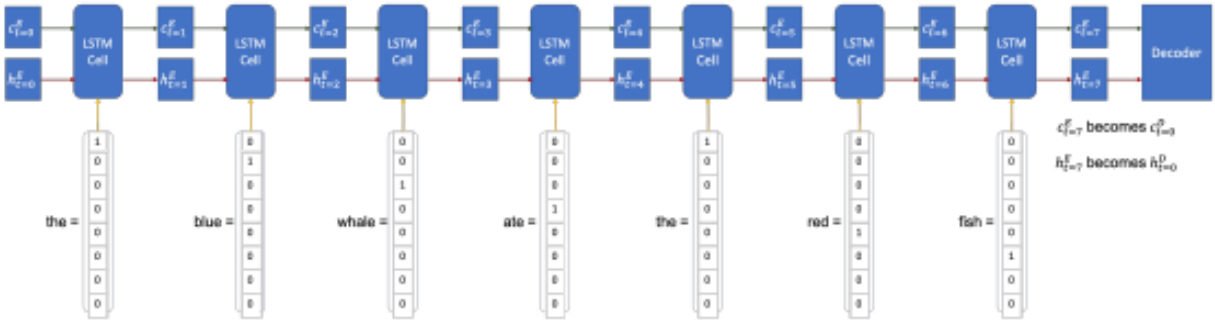


Fig 8 : Encoder of NMT with LSTM

to each of the weight matrices in the encoder and the decoder and updated as shown in the following set of equations:

$$W_i^E = W_i^E - lr(\partial L/\partial W_i^E)$$
$$W_f^E = W_f^E - lr(\partial L/\partial W_f^E)$$
$$W_o^E = W_o^E - lr(\partial L/\partial W_o^E)$$
$$W_{\tilde{c}}^E = W_{\tilde{c}}^E - lr(\partial L/\partial W_{\tilde{c}}^E)$$
$$W_i^D = W_i^D - lr(\partial L/\partial W_i^D)$$
$$W_f^D = W_f^D - lr(\partial L/\partial W_f^D)$$
$$W_o^D = W_o^D - lr(\partial L/\partial W_o^D)$$
$$W_{\tilde{c}}^D = W_{\tilde{c}}^D - lr(\partial L/\partial W_{\tilde{c}}^D)$$
$$U^D = U^D - lr(\partial L/\partial U^D) \tag{16}$$

Along with creating a more seamless flow back through the network for back-propagation, by creating the ht at each time-step t using both information from the input vector, xt , and the memory cell state from the previous time step, ct−1, LSTMs also helps preserve long term dependencies that often exist in time series data. This is a highly desirable trait of a MT model given that oftentimes in the task of machine translation a word towards the end of the sentence may be highly dependent on a word towards the beginning of the sentence. It is because of these numerous advantages that the LSTM architecture was chosen to perform the task of NMT in this research.

Attention Mechanism

when the sentence becomes longer, it may be difficult for the encoder to preserve information from the beginning of the sentence in the thought vector it produces and passes onto the decoder. Subsequently, the decoder is provided with little information on how it should start its outputted sentence. In order to combat this issue, it had previously been found beneficial to reverse the order of the input sentence in order to move the words thats correspond to the first words of the outputted sentence closer to the end of the encoder; ensuring that information on these first few words becomes encoded in the thought vector and passed onto the decoder.6 However, while this method has minimal success with translating between some languages, it does not generalize well to translation between all languages (given that the sentence structure of languages varies greatly). Furthermore, this method sacrifices quite a bit of the information from the end of the input sentence, as that information is inputted first into the encoder and is oftentimes not accessed again until the end of the output sentence, creating several steps over which this information often becomes lost.

A more full-proof fix to the issue of translating long pieces of text, the attention mechanism proposed by Bahdanau in 2014, has become the standard in machine translation attention architecture. In general, Luong's attention mechanism utilizes the hidden states outputted by the encoder at each time step, rather than just the final thought vector. In doing so, the decoder can "refer back" to the source sentence at each step to help determine which parts of the inputted sentence are most helpful in determining what the next outputted word should be. In general, the attention mechanism gives heavier weights to encoder hidden states which more closely resemble the current hidden state. This is done by computing a context vector dt , at each time step t in the decoder, as a weighted average over a selection of the hidden states outputted from the encoder. This weighted average is computed by first creating an alignment

vector, at , whose length corresponds to the number of source hidden states the weighted average is being taken over. This at is computed by comparing the similarity between the current hidden state, h D t , and each separate encoder hidden state, h E t 0 , being considered. In this way, at is defined by the following equation:

$$a_t[s] = \frac{exp(score(h_t^D, h_{t'=s}^E))}{\sum_{\tau=0}^{L_x} exp(score(h_t^D, h_{t'=\tau}^E))} \quad \forall s \in [0, L_x] \tag{17}$$

Where h^D_t is the current hidden state being inputted into the decoder, $h^E_{t'=\tau}$ is an encoder hidden state for some time-step τ, Lx is the number of encoder hidden states which are being weighted over, and exp is the exponential function. By computing an element at [s] ∀s ∈ [0, Lx], at becomes a vector of size L_xx1. Finally, in (17), score is the function which computes the similarity between the hidden states, and can be one of a variety of different forms:

$$\begin{aligned} score(h_t^D, h_{t*}^E) &= (h_t^D)^T(h_\tau^E) && (dot) \\ score(h_t^D, h_{t*}^E) &= (h_t^D)^T(W_a h_\tau^E) && (general) \\ score(h_t^D, h_{t*}^E) &= V_a^T tanh(W_a[h_t^D : h_\tau^E]) && (concatenation) \end{aligned} \tag{18}$$

In the general and concatenation score functions, Wa and Va are learnable weight matrices. Using these attention scores, Luong proposed two attention mechanisms; global and local. The global approach is the simpler form of attention and the one which will be covered in this paper. For details on local attention refer to (Luong et al., 2015).8 Global attention considers all parts of the input sequence, using information from each of the hidden states in the encoder to help predict each of the target words. In order to use information from every encoder hidden state, the softmax function is used to take a weighted average over all of the indices in at to create a vector, a^0_t , whose indices sum to one. In this way, each index at [s] where s ∈ [0, Lx] corresponds to the weight of that encoder hidden state, $h^E_{t=s}$. Thus, the context vector, dt , is simply the weighted average over all the source hidden state's h E t 0=s ∀s ∈ [0, Lx] and is computed as follows:

$$\begin{aligned} a'_t &= \mathbf{softmax}(a_t) \\ d_t &= \sum_{s=0}^{L_x} a'_t[s] * h_{t'=s}^E \end{aligned} \tag{19}$$

This context vector d_t is then used along with the the decoder hidden state, h^D_t , to compute $\tilde{h}^D_t$, which takes the place of h D t as the outputted hidden state. This process is depicted by the following equation:

$$\tilde{h}_t^D = tanh(W_{\tilde{h}}[d_t : h_t^D]) \tag{20}$$

From here the prediction vector, yˆt , is computed by simply replacing the h^D_t with the new $\tilde{h}^D_t$, as shown below:

$$\hat{y}_i = \text{softmax}(U\tilde{h}_t^D) \tag{21}$$

$\tilde{h}^D_t$ also replaces h^D_t as the hidden state fed to the next step step t + 1 in the RNN. The attention mechanism in the first step of the decoder is depicted in Figure 9.

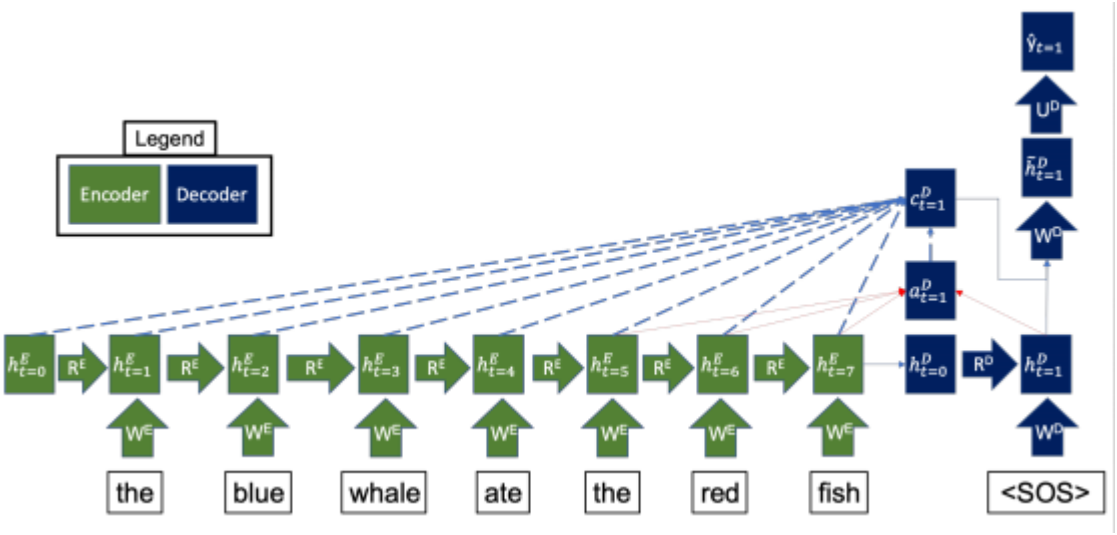
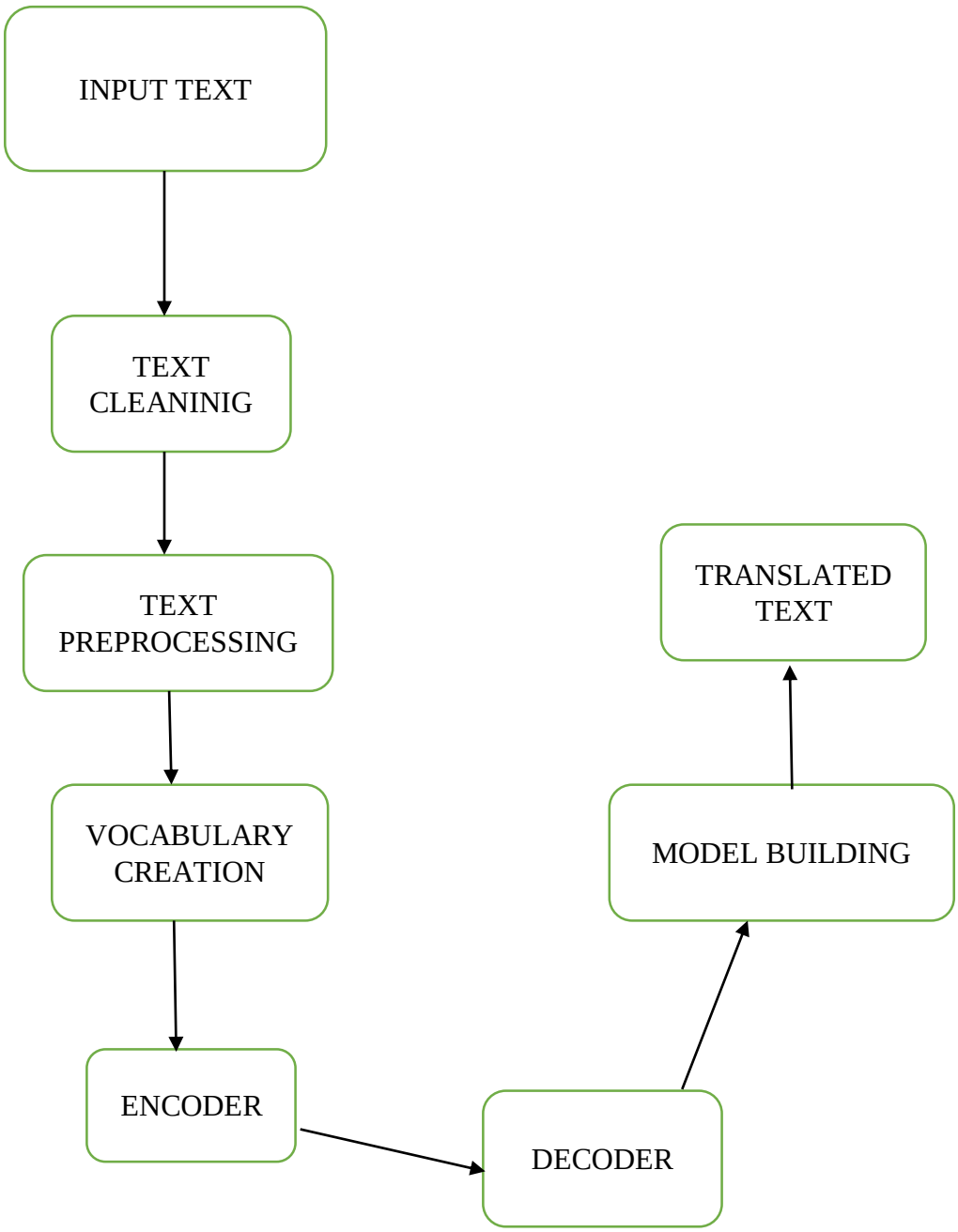


Fig 9 : Step 1 of Decoder using Global Attention

CHAPTER IV: MODELLING AND IMPLEMENTATION DETAILS

DESIGN DIAGRAMS



IMPLEMENTATION DETAILS AND ISSUES

Importing Libraries

```
import re
import math
import psutil
import time
import datetime
from io import open
import random
from random import shuffle
import argparse
import numpy as np
import matplotlib.pyplot as plt

import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
from torch import optim
import torch.cuda
import sys; sys.argv=['']; del sys
```

Preparing Vocabulary

```

SOS_token = 0
EOS_token = 1
UNK_token = 2

class Lang:
    def __init__(self, language):
        self.language_name = language
        self.word_to_index = {"SOS":0, "EOS":1, "<UNK>":2}
        self.word_to_count = {}
        self.index_to_word = {0: "SOS", 1: "EOS", 2: "<UNK>"}
        self.vocab_size = 3
        self.cutoff_point = -1


    def countSentence(self, sentence):
        for word in sentence.split(' '):
            self.countWords(word)


    def countWords(self, word):
        if word not in self.word_to_count:
            self.word_to_count[word] = 1
        else:
            self.word_to_count[word] += 1

```

```
    def createCutoff(self, max_vocab_size):
        word_freqs = list(self.word_to_count.values())
        word_freqs.sort(reverse=True)
        if len(word_freqs) > max_vocab_size:
            self.cutoff_point = word_freqs[max_vocab_size]


    def addSentence(self, sentence):
        new_sentence = ''
        for word in sentence.split(' '):
            unk_word = self.addWord(word)
            if not new_sentence:
                new_sentence =unk_word
            else:
                new_sentence = new_sentence + ' ' + unk_word
        return new_sentence

    def addWord(self, word):
        if self.word_to_count[word] > self.cutoff_point:
            if word not in self.word_to_index:
                self.word_to_index[word] = self.vocab_size
                self.index_to_word[self.vocab_size] = word
                self.vocab_size += 1
            return word
        else:
            return self.index_to_word[2]
```

Preparing Train and Test Pairs

```

def prepareData(lang1, lang2, file_path, max_vocab_size,trim, perc_train_set):

    input_lang, output_lang, pairs = prepareLangs(lang1, lang2,
                                                  file_path)

    print("Total %s sentence pairs" % len(pairs))


    if trim != 0:
        pairs = filterPairs(pairs, trim)
        print("Trimmed to %s sentence pairs" % len(pairs))

    print("Counting word frequency...")
    for pair in pairs:
        input_lang.countSentence(pair[0])
        output_lang.countSentence(pair[1])


    input_lang.createCutoff(max_vocab_size)
    output_lang.createCutoff(max_vocab_size)

    pairs = [(input_lang.addSentence(pair[0]),output_lang.addSentence(pair[1]))
             for pair in pairs]

    shuffle(pairs)

    train_pairs = pairs[:math.ceil(perc_train_set*len(pairs))]
    test_pairs = pairs[math.ceil(perc_train_set*len(pairs)):]

    print("Train pairs: %s" % (len(train_pairs)))
    print("Test pairs: %s" % (len(test_pairs)))
    print("%s, %s -> %s" % (input_lang.language_name, len(input_lang.word_to_count),
                            input_lang.vocab_size,))
    print("%s, %s -> %s" % (output_lang.language_name, len(output_lang.word_to_count),
                            output_lang.vocab_size))
    print()
    print(pairs)

    return input_lang, output_lang, train_pairs, test_pairs
```

```
Total 2867 sentence pairs
Trimmed to 2396 sentence pairs
Counting word frequency...
Train pairs: 1678
Test pairs: 718
en, 2945 -> 2948
hi, 2653 -> 2656


[('Cheers!', 'वाह-वाह!'), ('Keep an eye on the baby for a while.', 'बच्चे पर ध्यान रखना थोड़ी देर के लिए।'), ('He tends to be arrogan
t.', 'उसको अहंकारी होने की आदत है।'), ("You didn't have to come so early.", 'तुम्हे इतना जल्दी आने की ज़रूरत नहीं थी।'), ('I will pay
my debt as soon as possible.', 'मैं अपना कर्ज़ जल्द-से-जल्द चुकताऊँगा।'), ("What're you two doing?", 'तुम दोनो क्या कर रहे थे?'), ('I am
who I am.', 'मैं हूँ जो हूँ।'), ('He is tall and strong.', 'वह लम्बा और ताकतवर है।'), ("I can't put up with his arrogance.", 'मुझसे उ
सका अक्खड़पन झेला नहीं जाता।'), ("Didn't you go out?", 'आप बाहर नहीं गए थें क्या?'), ('He makes fun of everybody.', 'वह सबका मज़ाक
उड़ाता है।'), ('I know what that is.', 'मुझे पता है कि वह चीज़ क्या है।'), ('The house was ablaze.', 'घर आग में लिपटा हुआ था।'), ("I do
n't want there to be any misunderstanding.", 'मैं नहीं चाहता कि कोई भी ग़लतफ़ैमी हो।'), ("Let's play baseball!", 'चलो बेसबॉल खेलतें
हैं।'), ('We cannot do the work in a day.', 'हम यह काम एक दिन में नहीं कर सकते।'), ("Don't drink any alcohol.", 'शराब की एक बूंद
भी मत पियो।'), ('I was in London last month.', 'मैं पिछले महीने लंदन में था।'), ('You should take care of your sick mother.', 'तुम्हें अ
पनी बीमार माँ का ख़याल रखना चाहिए।'), ("I'm full.", 'मेरा पेट भर गया है।'), ('They hated each other.', 'वे एक-दूसरे से नफ़रत करते थें।'),
('He kept us waiting for a long time.', 'उसने हमसे बहुत देर तक इंतेज़ार करवाया।'), ('I have to answer his letter.', 'मुझे उसकी चिट्ठी
```

Creating Encoder Class

```python
class EncoderRNN(nn.Module):
    def __init__(self,input_size,hidden_size,layers=1,dropout=0.1,
                 bidirectional=True):
        super(EncoderRNN, self).__init__()

        if bidirectional:
            self.directions = 2
        else:
            self.directions = 1
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = layers
        self.dropout = dropout
        self.embedder = nn.Embedding(input_size,hidden_size)
        self.dropout = nn.Dropout(dropout)
        self.lstm = nn.LSTM(input_size=hidden_size,hidden_size=hidden_size,
                            num_layers=layers,dropout=dropout,
                            bidirectional=bidirectional,batch_first=False)
        self.fc = nn.Linear(hidden_size*self.directions, hidden_size)

    def forward(self, input_data, h_hidden, c_hidden):
        embedded_data = self.embedder(input_data)
        embedded_data = self.dropout(embedded_data)
        hiddens, outputs = self.lstm(embedded_data, (h_hidden, c_hidden))

        return hiddens, outputs

    def create_init_hiddens(self, batch_size):
        h_hidden = Variable(torch.zeros(self.num_layers*self.directions,
                                        batch_size, self.hidden_size))
        c_hidden = Variable(torch.zeros(self.num_layers*self.directions,
                                        batch_size, self.hidden_size))
        if torch.cuda.is_available():
            return h_hidden.cuda(), c_hidden.cuda()
        else:
            return h_hidden, c_hidden
```

Creating Decoder Class

```
class DecoderAttn(nn.Module):
    def __init__(self, hidden_size, output_size, layers=1, dropout=0.1, bidirectional=True):
        super(DecoderAttn, self).__init__()

        if bidirectional:
            self.directions = 2
        else:
            self.directions = 1
        self.output_size = output_size
        self.hidden_size = hidden_size
        self.num_layers = layers
        self.dropout = dropout
        self.embedder = nn.Embedding(output_size,hidden_size)
        self.dropout = nn.Dropout(dropout)
        self.score_learner = nn.Linear(hidden_size*self.directions,
                                   hidden_size*self.directions)
        self.lstm = nn.LSTM(input_size=hidden_size,hidden_size=hidden_size,
                        num_layers=layers,dropout=dropout,
                        bidirectional=bidirectional,batch_first=False)
        self.context_combiner = nn.Linear((hidden_size*self.directions)
                                      +(hidden_size*self.directions), hidden_size)
        self.tanh = nn.Tanh()
        self.output = nn.Linear(hidden_size, output_size)
        self.soft = nn.Softmax(dim=1)
        self.log_soft = nn.LogSoftmax(dim=1)


```

```
    def forward(self, input_data, h_hidden, c_hidden, encoder_hiddens):

        embedded_data = self.embedder(input_data)
        embedded_data = self.dropout(embedded_data)
        batch_size = embedded_data.shape[1]
        hiddens, outputs = self.lstm(embedded_data, (h_hidden, c_hidden))
        top_hidden = outputs[0].view(self.num_layers,self.directions,
                                 hiddens.shape[1],
                                 self.hidden_size)[self.num_layers-1]
        top_hidden = top_hidden.permute(1,2,0).contiguous().view(batch_size,-1, 1)

        prep_scores = self.score_learner(encoder_hiddens.permute(1,0,2))
        scores = torch.bmm(prep_scores, top_hidden)
        attn_scores = self.soft(scores)
        con_mat = torch.bmm(encoder_hiddens.permute(1,2,0),attn_scores)
        h_tilde = self.tanh(self.context_combiner(torch.cat((con_mat,
                                                     top_hidden),dim=1)
                                              .view(batch_size,-1)))
        pred = self.output(h_tilde)
        pred = self.log_soft(pred)


        return pred, outputs
```

Creating the Model

```
def train_and_test(epochs, test_eval_every, plot_every, learning_rate,
                   lr_schedule, train_pairs, test_pairs, input_lang,
                   output_lang, batch_size, test_batch_size, encoder, decoder,
                   loss_criterion, trim, save_weights):

    times = []
    losses = {'train set':[], 'test set': []}

    test_batches, longest_seq, n_o_b = batchify(test_pairs, input_lang,
                                            output_lang, test_batch_size,
                                            shuffle_data=False)

    start = time.time()
    for i in range(1,epochs+1):

        if i in lr_schedule.keys():
            learning_rate /= lr_schedule.get(i)


        encoder.train()
        decoder.train()

        encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
        decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)

```

```
        batches, longest_seq, n_o_b = batchify(train_pairs, input_lang,
                                        output_lang, batch_size,
                                        shuffle_data=True)
        train_loss = train(batches, encoder, decoder, encoder_optimizer,
                       decoder_optimizer, loss_criterion)

        now = time.time()
        print("Iter: %s \nLearning Rate: %s \nTime: %s \nTrain Loss: %s \n"
          % (i, learning_rate, asHours(now-start), train_loss))


        if i % test_eval_every == 0:
            if test_pairs:
                test_loss = test(test_batches, encoder, decoder, criterion)
                print("Test set loss: %s" % (test_loss))

        if i % plot_every == 0:
            times.append((time.time()-start)/60)
            losses['train set'].append(train_loss)
            if test_pairs:
                losses['test set'].append(test_loss)
            showPlot(times, losses, output_file_name)
            if save_weights:
                torch.save(encoder.state_dict(), output_file_name+'_enc_weights.pt')
                torch.save(decoder.state_dict(), output_file_name+'_dec_weights.pt')

```

Setting The Parameters

```
input_lang_name = 'en'
output_lang_name = 'hi'

raw_data_file_path = ('DATASET.txt',)
dataset = 'orig'

trim = 10

max_vocab_size= 20000

reverse=False

start_filter = False

perc_train_set = 0.7
```

```
test_eval_every = 1

plot_every = 1

create_txt = True

save_weights = True
```

```
bidirectional = True
if bidirectional:
    directions = 2
else:
    directions = 1

layers = 2

hidden_size = 440

dropout = 0.2

batch_size = 32

test_batch_size = 32

epochs = 100

learning_rate= 1

lr_schedule = {}

criterion = nn.NLLLoss()
```

Training and Testing

```
use_cuda = torch.cuda.is_available()

plt.switch_backend('agg')

output_file_name = "weights"
if create_txt:
    print_to = output_file_name+'.txt'
    with open(print_to, 'w', encoding="utf-8") as f:
        f.write("Starting Training \n")
else:
    print_to = None

input_lang, output_lang, train_pairs, test_pairs = prepareData(
    input_lang_name, output_lang_name, raw_data_file_path,
    max_vocab_size=max_vocab_size, trim=trim, perc_train_set=perc_train_set) #changes made by ishan
print('Train Pairs #')
print(len(train_pairs))

parser = argparse.ArgumentParser(description='PyTorch Wikitext-2 RNN/LSTM Language Model')
parser.add_argument('--clip', type=float, default=0.25,
                    help='gradient clipping')
args = parser.parse_args()

mem()

```

```
encoder = EncoderRNN(input_lang.vocab_size, hidden_size, layers=layers,
                     dropout=dropout, bidirectional=bidirectional)

decoder = DecoderAttn(hidden_size, output_lang.vocab_size, layers=layers,
                      dropout=dropout, bidirectional=bidirectional)

print('Encoder and Decoder Created')
mem()

if use_cuda:
    print('Cuda being used')
    encoder = encoder.cuda()
    decoder = decoder.cuda()

print('Number of epochs: '+str(epochs))

if create_txt:
    with open(print_to, 'a') as f:
        f.write('Encoder and Decoder Created\n')
        f.write(mem())
        f.write("Number of epochs %s \n" % (epochs))
'''
train_and_test(epochs, test_eval_every, plot_every, learning_rate, lr_schedule,
               train_pairs, test_pairs, input_lang, output_lang, batch_size,
                test_batch_size, encoder, decoder, criterion, trim, save_weights)
'''
```

CHAPTER V: TESTING

Test Case 1

```
enc = torch.load("weights_enc_weights.pt")
dec = torch.load('weights_dec_weights.pt')
encoder.load_state_dict(enc)
decoder.load_state_dict(dec)
outside_sent = "How are you?"
evaluate(encoder, decoder, outside_sent, cutoff_length=10)
```

'तुम कैसी हो? <EOS>'

Test Case 2

```
outside_sent = "Have fun."
evaluate(encoder, decoder, outside_sent, cutoff_length=10)
```

'मज़े करो। <EOS>'

Test Case 3

```
outside_sent = "What are you doing?"
evaluate(encoder, decoder, outside_sent, cutoff_length=10)
```

'तुम क्या कर रहे हो? <EOS>'

Test Case 4

```
outside_sent = "I am studying."
evaluate(encoder, decoder, outside_sent, cutoff_length=10)
```

'मैं पढ़ रही हूँ। <EOS>'

Test Case 5

```
outside_sent = "Congratulations"
evaluate(encoder, decoder, outside_sent, cutoff_length=10)
```

'मुबारक हो! <EOS>'

LIMITATIONS

1. Slower training and inference speed.
2. Ineffectiveness in dealing with rare words.
3. Sometimes failure to translate all words in the source sentence.
4. Translation works only for English to Hindi not for other languages.

CHAPTER VI: FINDINGS, CONCLUSIONS AND FUTURE WORK

FINDINGS

```
Test set loss: 4.673015966319075
Iter: 26
Learning Rate: 1
Time: 0h 43m 14s
Train Loss: 2.7547529795230967

Test set loss: 4.871987375586925
Iter: 27
Learning Rate: 1
Time: 0h 44m 50s
Train Loss: 2.7050761475522287

Test set loss: 4.870112881515966
Iter: 28
Learning Rate: 1
Time: 0h 46m 29s
Train Loss: 2.6588438673916035

Test set loss: 4.791593202918467
```

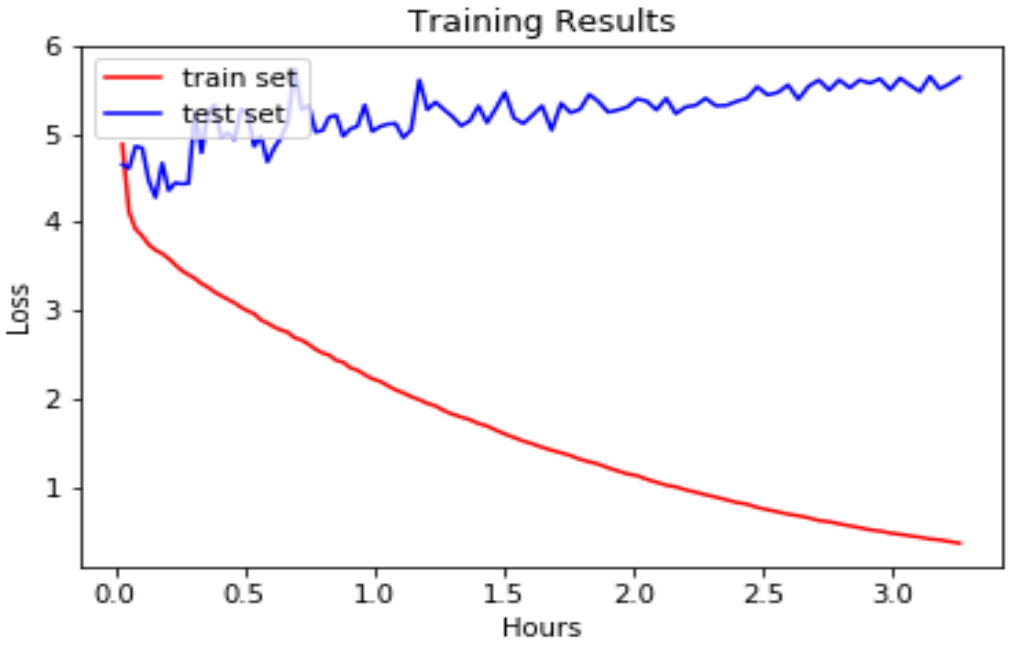


Fig 10 : Training Results

CONCLUSION

With increased computational power, NMT has emerged as an effective method for machine translation. By utilizing the Encoder-Decoder structure, NMT is able to translate sentences with incredible accuracy. Furthermore, with the added power of attention mechanisms, machine translation is becoming even more powerful. The Python machine learning package, PyTorch, provides all of the necessary tools to create a NMT model. In fact, the PyTorch website itself has a tutorial on creating such a model. However, the PyTorch model created is lagging in areas that are crucial to creating a model that fully utilizes the computational power of GPUs. Thus, this research aimed to enhance this NMT model, while keeping it at a tutorial scale. By increasing the size of the train set, adding a test set, batchifying the data, and tweaking a number of other model parameters, a much more effective machine translation model was ultimately created.

FUTURE WORK

This research focused primarily on understanding the Encoder Decoder structure and making improvements to the PyTorch tutorial on machine translation. Thus, the aim of this research was to create a model that was slightly more advanced and sophisticated than the current PyTorch tutorial model, but still capable of being easily reproduced by an individual looking to learn about neural machine translation. Moving forward, however, there are still several areas of improvements that can be made to the machine translation model outlined above. The easiest enhancement would be to simply train the existing model on a significantly larger dataset. Due to time constraints this was not performed, but would theoretically create an even more capable NMT model. More related to the architecture of the model, a more advanced attention mechanism could be implemented. In particular, global attention was used in this model, but research has found various local attention mechanisms to be much more effective. Finally, while the cross entropy loss is one way of understanding a model's accuracy, there are several more sophisticated manners of quantifying a machine translation model's accuracy. In particular, a BLEU score algorithm could be implemented to more effectively understand the accuracy of the model's translations.

REFERENCES

[1]Dhariya, Omkar, Shrikant Malviya, and Uma Shanker Tiwary. “A Hybrid Approach for Hindi-English Machine Translation.” 2017 International Conference on Information Networking (ICOIN) (2017): n. pag. Crossref. Web.

[2]Agrawal, Ruchit and Dipti Misra Sharma. “Building an Effective MT System for English-Hindi Using RNN's.” (2017).

[3]Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). Attention Is All You Need.

[4]Saini, Sandeep & Sahula, Vineet. (2018). Neural Machine Translation for English to Hindi. 1-6. 10.1109/INFRKM.2018.8464781.

[5]X. Wang, Z. Tu and M. Zhang, "Incorporating Statistical Machine Translation Word Knowledge Into Neural Machine Translation," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 26, no. 12, pp. 2255-2266, Dec. 2018.

[6]Mahata, Sainik Kumar & Das, Dipankar & Bandyopadhyay, Sivaji. (2018). MTIL2017: Machine Translation Using Recurrent Neural Network on Statistical Machine Translation. Journal of Intelligent Systems. 10.1515/jisys-2018-0016.

[7]Song, Linfeng et al. “Semantic Neural Machine Translation Using AMR.” Transactions of the Association for Computational Linguistics 7 (2019): 19–31. Crossref. Web.

[8]Johnson, Melvin & Schuster, Mike & Le, Quoc & Krikun, Maxim & Wu, Yonghui & Chen, Zhifeng & Thorat, Nikhil & Viégas, Fernanda & Wattenberg, Martin & Corrado, G.s & Hughes, Macduff & Dean, Jeffrey. (2016). Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. Transactions of the Association for Computational Linguistics. 5. 10.1162/tacl_a_00065.

[9]Axelrod, Amittai et al. “Domain Adaptation via Pseudo In-Domain Data Selection.” EMNLP (2011).

[10]Peter, Jan-Thorsten & Toutounchi, Farzad & Wuebker, Joern & Ney, Hermann. (2015). The RWTH Aachen German-English Machine Translation System for WMT 2015. 158-163. 10.18653/v1/W15-3018.

[11]Folajimi, Yetunde & Isaac, Omonayin. (2012). Using Statistical Machine Translation (SMT) as a Language Translation Tool for Understanding Yoruba Language. 10.13140/2.1.3522.8485.

[12]He, Wei et al. "Improved Neural Machine Translation with SMT Features." AAAI (2016).

[13]Thu, Ye Kyaw et al. "A Large-scale Study of Statistical Machine Translation Methods for Khmer Language." PACLIC (2015).

[14]Wu, Haiyang & Dong, Daxiang & Hu, Xiaoguang & Yu, Dianhai & He, Wei & Wu, Hua & Wang, Haifeng & Liu, Ting. (2014). Improve Statistical Machine Translation with Context-Sensitive Bilingual Semantic Embedding Model. 142-146. 10.3115/v1/D14-1015.

[15]Bhatt, Rajesh & Narasimhan, Bhuvana & Palmer, Martha & Rambow, Owen & Sharma, Dipti & Xia, Fei. (2009). A Multi-Representational and Multi-Layered Treebank for Hindi/Urdu.. 186-189. 10.3115/1698381.1698417.

[16]Chiang, David. (2005). A Hierarchical Phrase-Based Model for Statistical Machine Translation.. 10.3115/1219840.1219873.

[17]Koehn, Philipp et al. "Statistical Phrase-Based Translation." HLT-NAACL (2003).

[18]Och, Franz. (2003). Minimum Error Rate Training in Statistical Machine Translation. Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics. 10.3115/1075096.1075117.

[19]Marcu, Daniel & Wong, William. (2004). A Phrase-Based, Joint Probability Model for Statistical Machine Translation Daniel Marcu. 10.3115/1118693.1118711.

[20]Fuji Ren and Hongchi Shi, "Parallel machine translation: principles and practice," Proceedings Seventh IEEE International Conference on Engineering of Complex Computer Systems, Skovde, Sweden, 2001, pp. 249-259

[21]Sutskever, Ilya & Vinyals, Oriol & Le, Quoc. (2014). Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems. 4.

[22]Su, Jinsong & Zeng, Jiali & Xiong, Deyi & Liu, Yang & Wang, Mingxuan & Xie, Jun. (2018). A Hierarchy-to-Sequence Attentional Neural Machine Translation Model. IEEE/ACM Transactions on Audio, Speech, and Language Processing. PP. 1-1. 10.1109/TASLP.2018.2789721.

[23]Deep Architectures for Neural Machine Translation. / Miceli Barone, Antonio Valerio; Helcl, Jindrich; Sennrich, Rico; Haddow, Barry; Birch, Alexandra.
Proceedings of the Second Conference on Machine Translation, Volume 1: Research Papers. Copenhagen, Denmark : Association for Computational Linguistics, 2017. p. 99-107.

[24]Choudhary, Himanshu & Pathak, Aditya & Saha, Rajiv & Kumaraguru, Ponnurangam. (2018). Neural Machine Translation for English-Tamil. 770-775. 10.18653/v1/W18-6459.

[25]Nguyen, Phuoc & vo, anh-dung & Shin, Joon-Choul & Tran, Phuoc & Ock, Cheol-Young. (2019). Korean-Vietnamese Neural Machine Translation System With Korean Morphological Analysis and Word Sense Disambiguation. IEEE Access. PP. 10.1109/ACCESS.2019.2902270.

[26]Singh, Shashi & Kumar, Ajai & Darbari, Hemant & Singh, Lenali & Rastogi, Anshika & Jain, Shikha. (2017). Machine translation using deep learning: An overview. 162-167. 10.1109/COMPTELIX.2017.8003957.

[27]Sutskever, Ilya & Vinyals, Oriol & Le, Quoc. (2014). Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems. 4.

[28]Agrawal, Ruchit & Sharma, Dipti. (2017). Experiments on Different Recurrent Neural Networks for English-Hindi Machine Translation. 63-74. 10.5121/csit.2017.71006.

[29]Mahata, Sainik Kumar & Das, Dipankar & Bandyopadhyay, Sivaji. (2018). MTIL2017: Machine Translation Using Recurrent Neural Network on Statistical Machine Translation. Journal of Intelligent Systems. 10.1515/jisys-2018-0016.

[30]Sin, Yi & Soe, Khin. (2019). Attention-Based Syllable Level Neural Machine Translation System for Myanmar to English Language Pair. International Journal on Natural Language Computing. 8. 01-11. 10.5121/ijnlc.2019.8201.

[31]Goyal, Vikrant & Sharma, Dipti. (2019). LTRC-MT Simple \& Effective Hindi-English Neural Machine Translation Systems at WAT 2019. 137-140. 10.18653/v1/D19-5216.

[32]Saini, Sandeep & Sahula, Vineet. (2018). Neural Machine Translation for English to Hindi. 1-6. 10.1109/INFRKM.2018.8464781.

[33]Simard, Michel & Ueffing, Nicola & Isabelle, Pierre & Kuhn, Roland. (2007). Rule-based Translation With Statistical Phrase-based Post-editing. Proceedings of the Second Workshop on Statistical Machine Translation. 10.3115/1626355.1626383.

[34]Su, Jinsong & Xiong, Deyi & Huang, Shujian & Han, Xianpei & Yao, Junfeng. (2015). Graph-Based Collective Lexical Selection for Statistical Machine Translation. 1238-1247. 10.18653/v1/D15-1145.

[35]Tu, Zhaopeng & Hu, Baotian & Lu, Zhengdong & Li, Hang. (2015). Context-Dependent Translation Selection Using Convolutional Neural Network. 10.3115/v1/P15-2088.