UNIVERSITY OF SOUTHAMPTON

Faculty of Physical Sciences and Engineering

School of Electronics and Computer Science

Machine Translation Using Deep Neural Networks

by

Richa Ranjan(rr2n17)

6th of September, 2018

Project supervisor: Professor Mahesan Niranjan
Second examiner: Professor James Pilgrim

A dissertation submitted in partial fulfilment of the degree of MSc Data Science

UNIVERSITY OF SOUTHAMPTON

Faculty of Physical Sciences and Engineering

School of Electronics and Computer Science

Machine Translation Using Deep Neural Networks

by

Richa Ranjan(rr2n17)

6th of September, 2018

Project supervisor: Professor Mahesan Niranjan
Second examiner: Professor James Pilgrim

A dissertation submitted in partial fulfilment of the degree of MSc Data Science

Abstract

*Deep Neural Networks (DNNs) are extremely powerful machine learning models that have showcased promising outcomes in a number of practical applications like Text processing, Computer Vision, Speech Recognition, and other similar areas. The main idea behind this project was to efficiently demonstrate training methodologies for Deep Neural Networks. Machine Translation task was chosen to dive deeper in this area. Neural Machine Translation (NMT) is one of the most sought-after applications in Natural Language Processing(NLP). It has shown promising results by addressing the drawbacks of traditional phrase-based translation systems. NMT follows a direct, end-to-end learning approach, in contrast to the more statistical approach used by traditional translation systems. Instead of dealing with the predictive algorithms using statistical analysis, Neural machine translators learn from a single network, by performing direct mapping between input and output text. Globalization has given rise to a number of communication needs. Be it business expansion on a global scale, or international education, cultural studies, or to a large extent, social media, all of these have made communication across cultures very common. Translation between languages is thus, one of the most widespread applications in machine learning. The leading names in the market include the likes of translators by Google and Microsoft, which have provided platforms for practically all the languages that can be scripted worldwide. How the Neural Network models can be trained to perform this task efficiently, was an insightful question, and in my view, being on the other side of the technology always gives a better idea of how things happen the way they do. One of the most commonly used approaches for training machines on text data is the Recurrent Neural Network (**RNN**). More specifically, and more recently, Long Short-Term Memory (**LSTM**s) have provided significant results in Sequence to Sequence and Encoder-Decoder approaches by Neural Translators. The idea of this research is to build an LSTM-based model and to execute the same on the vectorized language data set. This will be the baseline model for this project. Efficient prediction approach using **Beam Search** and the evaluation metric for calculating translation accuracy i.e. the **BLEU** score is discussed.*

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Professor Mahesan Niranjan for his constant guidance and valuable feedback throughout the project. I consider myself fortunate enough to have worked under his supervision. He has been an incredible mentor and his suggestions have been really helpful in moving the project in the right direction.

Secondly, I would like to thank my second examiner, Professor James Pilgrim. His evaluation provided a different perspective to the project, which was equally important.

I am very thankful to the Electronics and Computer Science Department of the University of Southampton, for providing high performance GPU-based machines and requisite institutional facilities. I would also like to thank my fellow MSc students at the University of Southampton, for having healthy group discussions about new researches and ideas about Machine Learning.

Finally, I would like to thank my family and friends for believing in me and encouraging me to pursue my goals. A special mention to my husband, for constantly supporting me and inspiring me to be better every day!

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

I have acknowledged all sources, and identified any content taken from elsewhere.

I have not used any resources produced by anyone else.

I did all the work myself, or with my allocated group, and have not helped anyone else.

The material in the report is genuine, and I have included all my data/code/designs.

I have not submitted any part of this work for another assessment.

My work did not involve human participants, their cells or data, or animals.

'A baby learns to crawl, walk and then run. We are in the crawling stage when it comes to applying Machine Learning.'

Dave Waters

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine Translation(MT) is an automated task in Natural Language Processing(NLP), where computers are used to translate the text in one language to another. The concept of MT is not new. It dates way back to 1949, when Warren Weaver penned a memorandum, titled "Translation".[1] He is credited for coming up with the prospect of using computers for translation tasks. Ever since, MT community has been evolving, especially in the last two decades. Initial researches in this area were dedicated to developing *rule-based* (based on semantic, morphological or syntactic analysis) or *example-based* (by analogy) translation systems using bilingual text corpora. While the rule-based MT (RBMT) follows an "Analyze-Transfer-Generate" paradigm, an example-based system (EBMT) is about "Decompose-Translate-Recombine" approach.[3] One of the major breakthroughs of RBMT was the SYSTRAN system, powered by 80 language pairs and was used by all major platforms like Google, Yahoo, Apple[4] and so on. The late 1980s or early 90s saw a rise in Statistical Machine Translation(SMT), which is by far the most widely accepted translation paradigm.[5] This achieved commercial-level access with IBM's Watson Research on SMT, a different approach. More recently, Neural Network-based training has significantly over-shadowed previous approaches.

[1] Written in July 1949 at Carlsbad, New Mexico, often called 'Weaver 1949', available at: http://www.hutchinsweb.me.uk/MTNI-22-1999.pdf

1.1 Neural Machine Translation

Neural Machine Translation(NMT) is one of the most recent approaches to machine translation. NMT addresses one of the main limitations of SMT systems, which is, that an SMT-based system comprises of complex and multiple, independently trained machines. This requires a lot of human feature engineering and is computationally tedious as well. NMT alters this approach by using only one machine for training (in most of the baseline models). A Deep Neural Network is basically an end-to-end model, which allows training of models on the same machine. Translation systems deal with sequential data i.e. a sentence or phrase, which comprise of sequences of words. Training Neural Networks for sequential data follows a different approach than the traditional methods. Unlike the human mind, machines process each unit of the data individually. Humans, on the other hand, while reading a piece of text, do not process each word separately, throwing away the earlier words/phrases. However, in case of sequential data, like a sentence, every word is processed based on the understanding of its context i.e. previous words. Training machines for such level of processing needs more than a Traditional Neural Network like *Multi-Layer Perceptrons*. Feed-forward Networks have limitations when it comes to interpreting arbitrary length time series or, text sentence, in general. To handle the issue of persistence in machines, *Recurrent Neural Networks* (RNNs) were developed. These networks contain a *feedback loop*, delayed in time, so that information can be passed from one step of the network to another. RNNs, particularly LSTMs have been used in a number of researches involving sequential data, and have achieved remarkable results.

1.2 Project Scope

The scope of this research was to demonstrate the empirical process of training a Deep Neural Network to obtain maximum efficiency with the said combination of inputs and parameters. Specifically, the project is designed to perform a Machine Translation task on the *Hindi* and English languages using Deep Neural Networks. A range of hyper parameters like batches, epochs, gradient descent optimization algorithms etc. were tested to optimize the results and

accuracies were observed to find the closest possible translation. One of the important aspects of this research was to analyze the results and to showcase how optimized implementation of learning algorithms shows improvement in results. The baseline model is an end-to-end network, containing LSTM layers, which translates sentences from Hindi to English language. As this work deals in Machine Translation, accuracies from previously built models were tested on the same data set and a comparison of techniques was made. Once the baseline accuracy is available, there can be multiple methods to tune the parameters or hyper parameters and to improve the accuracies. To measure the accuracy of the translations, BLEU (**Bi**-**L**ingual **E**valuation **U**nderstudy) score is used.

1.3 Background and Motivation

Of the two languages chosen for this project, English is a global language, and Hindi, although spoken only in India, and in some parts of UAE, is widely gaining popularity owing to globalization. Personally, I chose *Hindi* as one of the languages, as I come from India. With the Indian population spreading across the globe for education, work and other reasons, translators are facilitating communication, helping people to understand better. As a postgraduate student who comes from India, attaining a higher degree in the UK, I found language translators very useful when I communicated with people from across the globe. With different languages making their place in the global dictionaries, Hindi is one of the most promising one, considering a large section of the Indian population has spread worldwide.

In terms of the popular evaluation metric *BLEU* score(to be defined later in section 7.2), NMT achieved an improvement ($\approx$ 5.3 BLEU points) over traditional phrase based SMT(PBSMT) approach[6]. This result was first presented in the IWSLT 2015 evaluation campaign[7] by the MT system described by Luong and Manning[8]. Promising state-of-the-art results and intuitive training approaches were enough motivation to take up this challenge (I would term this as a 'challenge' because the data used for this project is, to the best of my knowledge, currently the largest, freely available parallel corpus for Hindi-English translation).

Chapter 2

Literature Review

Neural Networks date way back to the 1950s, but it was not until the major breakthrough in 2006 (Hinton *et al.*)[9] that brought about the 'renaissance' of Deep Learning. Training deep networks on sequential data was one of the bigger challenges, as translation systems required models to be trained on massive data sets. Neural Network-based training for language models came into light only after the work of Bengio *et al.*[10] in 2003, where the '*curse of dimensionality*' issue was addressed, by training the model to learn a distributed word representation. The Neural model proposed using "*distributed feature vectors* associated with all words in the vocabulary"[10]. The probability function over words is a function that maps input sequence of feature vectors to conditional probability distribution of words in that vocabulary, for the next word in the sequence. The Neural network calculated the following probability, with a *softmax* output layer:

$$\hat{P}(w_t|w_{t-1},\dots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{2.1}$$

This was one of the initial Neural network based approach for translation tasks. Another research on training continuous translation models was by Kalchbrenner and Blunsom [11] where they had trained probabilistic models purely based on continuous representations for words or phrases. The sequential alignment of words was not considered in this case. They discussed Recurrent Language Models (RLM) that undergo three transformations, "an input vocabulary

transformation $\mathbf{I} \in \mathbb{R}^{q \times |V|}$, recurrent transformation $\mathbf{R} \in \mathbb{R}^{q \times q}$ and an output vocabulary transformation $\mathbf{O} \in \mathbb{R}^{|V| \times q}$ where q represents the parameter and V, the vocabulary"[11]. In the previous SMT approaches, models estimated probabilities on source-target sentence pairs. For distinct phrases, statistical weights were not shared and this caused sparse/skewed results. These issues were first addressed by continuous representations in Bengio *et al.*[10].

With Neural Networks succeeding statistical approaches, more researches have come up in the last decade. A new approach was introduced by Cho *et al.*[1], where an Encoder-Decoder based model was trained on source-target pairs. Two different RNNs were used, one for encoding the whole input sequence to a fixed dimension-vector, and the other for decoding it back to the target sequence. An illustration of the model is shown below:

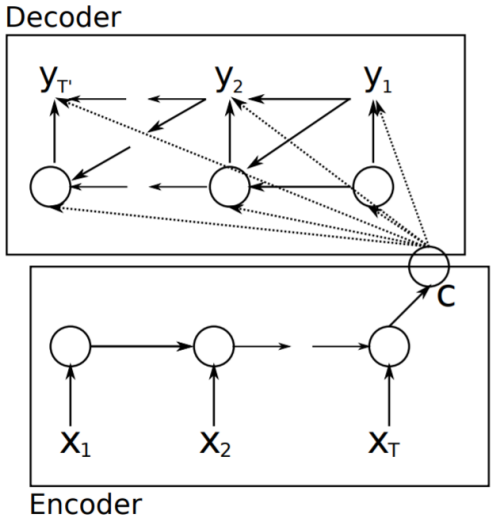

Figure 2.1: Illustration of the RNN encoder-decoder model proposed by Cho *et al.*, Image taken with thanks from Cho *et al.* [1]

Both networks were trained sequentially, and the maximum conditional probabilities of the translated sentences were chosen as the target sequences. This architecture largely inspired subsequent NMT researches. Another variant of this training method was proposed by Cho *et al.*[12] where instead of using an RNN-based encoder, a *gated recursive convolutional neural network* (grConv) was used. A binary CNN (Convolutional Neural Network) allowed "the weights to be recursively applied to the input and eventually obtain a vector representation. The traditional sigmoid activation function was used in the form of two gating units called reset and update gates"[12]. This enabled the network to learn the input structures dynamically.

This gating mechanism was later used in many translation-based applications, and hence the popularity of LSTM-based translation models.[1]

Learning word vectors for huge data sets was also a big challenge, which was addressed by Mikolov *et al.*[13]. With the hypothesis that similar words will be close to each other in the vector space, *skip-gram* models were introduced. Also, it was claimed that words can have multiple degrees of similarity. Their model could learn high quality vector representations for huge, unstructured text data. This was a major contribution to NLP, and aided several researches thereafter. An extension to this research was the "Distributed representations of words and phrases" by Mikolov *et al.*[14] where training speed was increased by negative sampling instead of using hierarchical softmax. With distributed representation of words in vector space, it was easier to group similar words. The model was found to be more efficient in predicting nearby words. This became the basis for word embeddings.

Following Cho *et al.*[1], Bahdanau *et al.*[15] claimed that the fixed dimension-vector in the RNN encoder-decoder model, produces a bottleneck. The idea put forward by them was to search for only those parts of the source sequence, which are relevant in predicting the target. A bidirectional RNN was used as encoder and the decoder performs search on the source sentence while performing the reverse mapping from decoded data to translation sequence. This claim was stronger as the model "outperformed" the conventional RNN encoder-decoder. [15]

Several similar researches inspired the 'Sequence to Sequence' model, [16] where two deep LSTMs are used as encoder and decoder respectively, and the input sequences are fed in reverse order. Given an input sequence $(x_1, \ldots x_T)$, an RNN model predicts the output sequence $(y_1, \ldots y_T)$ using the following equations, later applied using LSTMs:

$$h_t = sigm(W^{hx} x_t + W^{hh} h_{t-1}) \tag{2.2}$$

$$y_t = W^{yh} h_t \tag{2.3}$$

[16] where h is the state value. This project is largely inspired by sequence to sequence model.

[1]LSTM comprises of gates to forget/retain the information. The details are discussed in the upcoming sections.

Chapter 3

Recurrent Neural Networks

3.1 Why RNN?

Let us consider an example of a *Named-Entity Recognition* problem, where we are trying to build a model, that takes a sentence as input, and identify where the names of people are, in that sentence. For example, in a sentence like: "*Jack and Jill went up the hill*", the model should be able to recognize *Jack* and *Jill* to be names of people. While this is a simple sentence, the NE-Recognition problems have bigger applications like being used by search engines to index the people mentioned in the news in a definite period in time. In addition to people's names, NE-Recognition problems have also helped in identifying names of Countries, organizations etc. In this example, the input is a sequence of 7 words, and thus, there will be 7 sets of features to represent these words. If $(X = x^{\langle 1\rangle}, x^{\langle 2\rangle}, \ldots x^{\langle 7\rangle})$ is the input sequence, and $(\hat{y} = y^{\langle 1\rangle}, y^{\langle 2\rangle}, \ldots y^{\langle 7\rangle})$ is the output, then the element $y^{\langle i\rangle} = 1$ if $x^{\langle i\rangle}$ is a name or $y^{\langle i\rangle} = 0$ otherwise. If a standard Neural Network is used for this problem, the model looks like the one shown in figure 3.1.

The problem with this implementation is that, unlike this case, the input and output sequences can have different lengths when handling different problems. Also, this type of naïve Neural Network architecture does not share the learned features across different positions of text, i.e. once the Network encounters '*Jack*' in the first position, and identifies it to be a

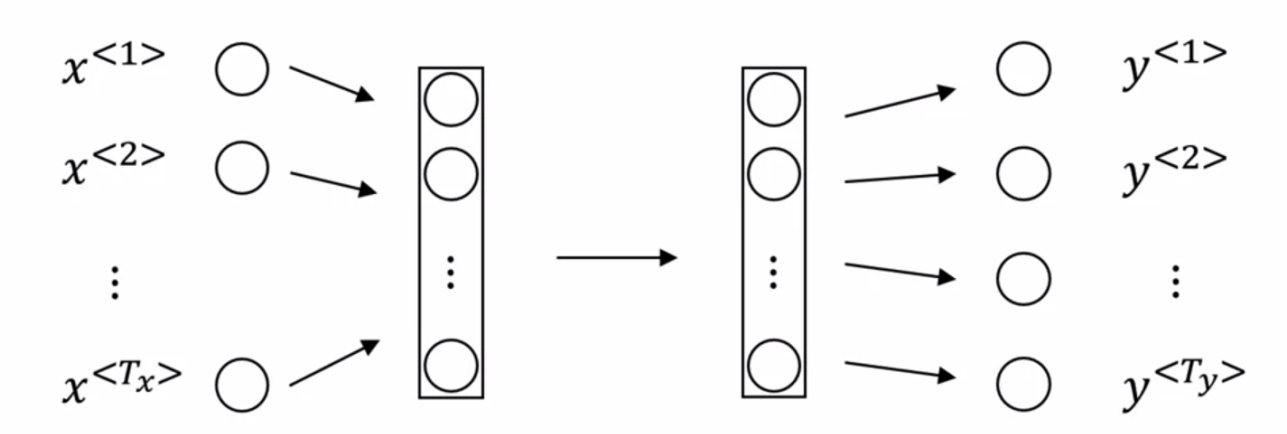

Figure 3.1: Traditional Neural Network used for NE-Recognition problem, T_x and T_y being the input and output sequence lengths respectively, Image taken with thanks from Deep Learning course on *Coursera*

name, it does not affect the learning speed if the same name is encountered later in the text. In this representation, $x^{\langle t \rangle}$ for each value of $'t'$ in the sentence is a one-hot vector, defined on the whole vocabulary size. In such a scenario, weight matrix of the first network layer has a huge number of parameters. To overcome these limitations, Recurrent Neural Networks were used on sequential data.

3.2 Training RNNs

While processing sequential data, a *feedback* loop is required to preserve the information throughout the processing, so that the information about the past is retained as and when required to interpret the remaining portion of the sequence. This feedback loop, when unrolled, makes the Network representation as shown in figure 3.2.

The architecture above works like multiple copies of the same network stacked together horizontally, each passing a message to the successor. In this case, the activation values from the previous time-step are passed on to the next. These activation functions $a^{\langle i \rangle}$ are essentially the feedback loops, or sometimes called the '*memory*' that is getting passed on to the next steps. At each time step, we keep on feeding new inputs (words, in this case), and the network produces outputs and activation functions(memory). This memory is fed to the next time-

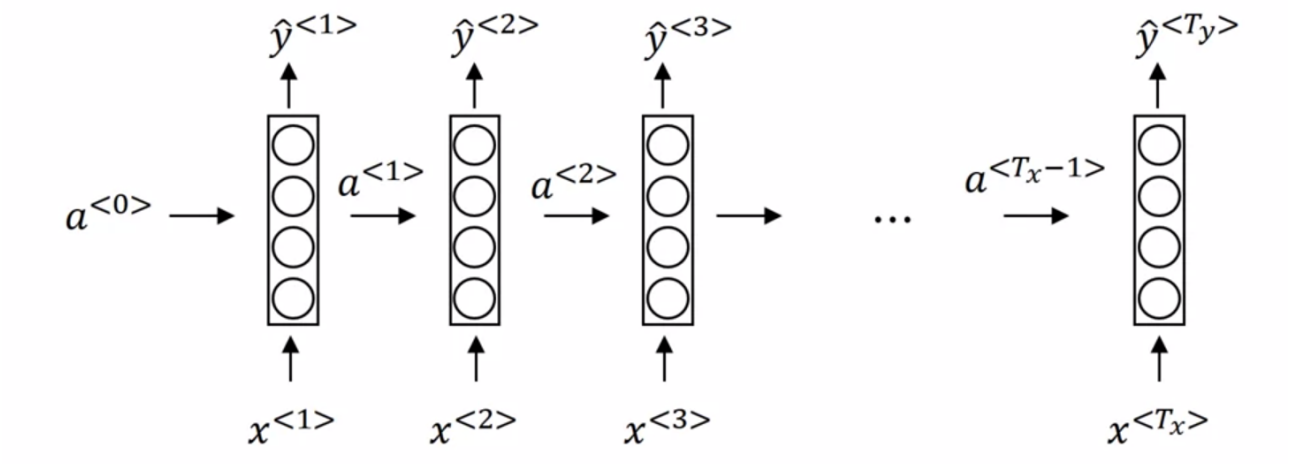

Figure 3.2: An unrolled RNN where $a^{\langle i \rangle}$ defines the activation functions, Image taken with thanks from Deep Learning course on *Coursera*

step, and that's how the network is unrolled in time. Also, if we look at the parameters, say, 'W_{ax}' for the input, 'W_{aa}' for the activations and 'W_{ay}' for the outputs, it is evident that these parameters are being shared across the network, thereby addressing the issues encountered in standard networks. Thus, given a set of inputs: $D = ((x^{\langle t \rangle}, y^{\langle t \rangle})|t = 1, 2, \dots T)$, the RNN can be trained to minimize the Error function as:

$$E(W) = \sum_{t=1}^{T} ||y^{\langle t \rangle} - f(x^{\langle t \rangle}, a^{\langle t-1 \rangle})|W||^2 \tag{3.1}$$

This process of learning is known as "*backpropagation through time*".

3.3 Vanishing and Exploding gradients in RNNs

In the example discussed in the previous section about RNN, we can say that:

$$a^{<t>} = f_1(x^{<t-1>}, a^{<t-1>}|W) \tag{3.2}$$

If the output $y^{\langle t \rangle}$ depends upon the input value $x^{\langle t-3 \rangle}$, the prediction will necessarily be a chain of values affecting the next one, like:

$$f(x^{<t>}, f_1(x^{<t-1>}, f_1(x^{<t-2>}, f_1(x^{<t-3>}|W), W), W), W) \tag{3.3}$$

During back-propagation, the error will involve f_1 getting applied multiple times, and each time, the error gets multiplied by some factor 'τ'. The back-propagated signal is thus proportional to $a^{\langle \tau-1 \rangle}$. As the value of τ changes, $a^{\langle \tau-1 \rangle}$ vanishes or explodes accordingly. The exploding gradients would result in `NaN`[1] values (in python).

In language modeling problems, where the job is to predict the next word, and the sequence is "*The clouds are in the ...*", no further context is required, and if the model has been trained well, the next word predicted will be "*sky*". RNNs perform well when the gap between the relevant information and the point where it is needed, is small. On the other hand, if we have a longer sentence, such as "*Holly gave me a book to read, which was a classic, based on the religious wars in the middle ages, and as a return gift, I gave her my favorite book*". As we can see, this is a long sentence, and towards the end, we need the information i.e. the name '*Holly*', to predict the gender pronoun '*her*' correctly. Thus, the gender information needs to be carried until the end of the sentence, where the pronoun '*her*' is needed. This is a classic case of long term dependency. As the gap between relevant information and prediction step increases, RNNs become unable to propagate the information. Thus, they are not considered appropriate for sequences with long-term dependencies. As we've seen in the earlier example, it becomes difficult for the error to propagate to the beginning, if the sequences are long. To handle such long-term dependencies, LSTMs were designed. The next section explains about LSTMs and how they are implemented.

3.4 LSTM

In 1997, Hochreiter and Schmidhuber[17] introduced a special type of RNN, called the LSTM (Long Short-term Memory), with an added advantage, i.e. its capability to learn long term dependencies, and to retain this memory for a long period of time. However, it doesn't have to remember all the information for longer durations. In some cases, we have to forget a memory completely, and change the memory in some cases. Gates are used for this purpose. LSTMs are composed of four layers in each repeating Neural Network. This internal structure is what

[1] "not a number": https://docs.python.org/3/library/math.html

differentiates LSTMs from RNNs, on an architectural level. In RNNs, every repeating network has a simple, tanh structure, where LSTMs are composed of four different layers. A detailed architecture of LSTM with the gates is shown in figure 3.1.[2] The notations are slightly different

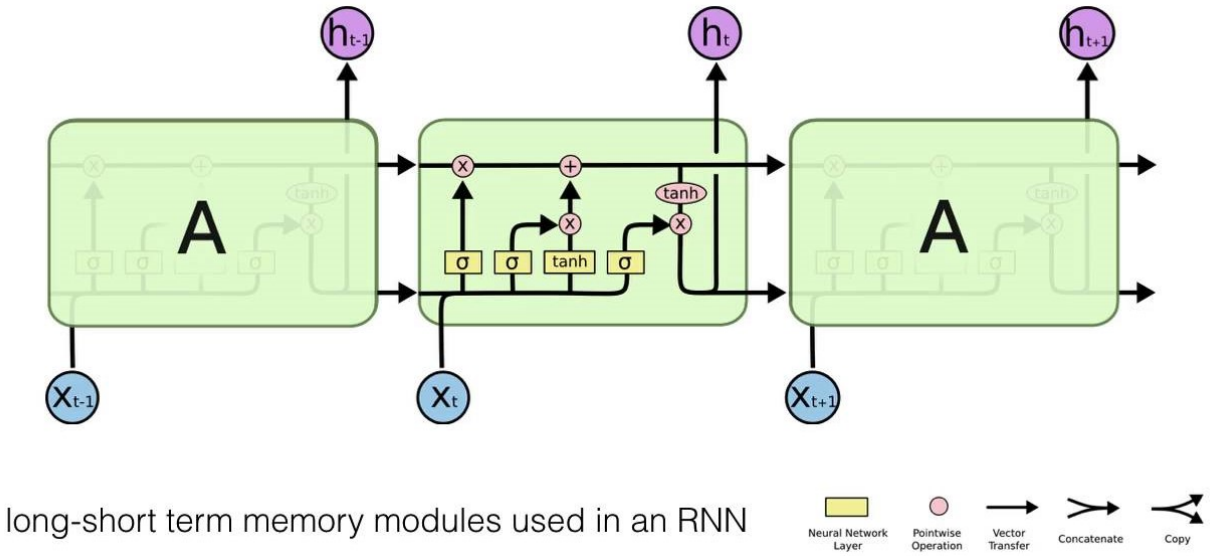

Figure 3.3: LSTM architecture (Image taken with thanks from *colah's blog*)[2]

than the ones used previously. Here, the input sequence is $(x_{t-1}, x_t, x_{t+1} \dots)$ [3] and the output sequence is $(h_{t-1}, h_t, h_{t+1} \dots)$. The 'A' stands for repeating Neural Networks, which handles the sequence one step at a time.

3.4.1 LSTM Architecture

The architecture of an LSTM differs from RNN in terms of the number of internal layers. Every recurrent network has a chain of repeating neural network modules. These modules can be a simple *tanh* layer in standard RNNs. In an LSTM, these modules consist of four dense layers, three *Sigmoids* and one *tanh*[2].

Cell state

Considering the example of text sequences, as we read the sentence from left to right, the LSTM unit will have a new variable, called the memory cell. This cell state is denoted in fig. 3.1 by

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] previously, we have used angular brackets to denote values at a particular time-step in the sequence, such as $x^{\langle t \rangle}$, we can use either this or simply x_t, they both mean the same in context

the horizontal line on top, inside the network module. The cell provides memory to remember information like the gender, or singular/plural etc. as needed for the prediction. LSTMs use *Gates* to control what information is to remembered, and what is to be forgotten. A 0 value means "forget completely", and a value of 1 means, "remember everything"[2].

Gates

The LSTM mechanism is governed by three gates, namely, forget, input and output gates.

1. **Forget gate:** The first step in any LSTM-based task, is to decide what information is required later, and what is not useful, so that it can be forgotten. This is decided by a Sigmoid layer. In the fig. 3.1, the first sigmoid layer performs this task. It takes as input, x_t (the input at current time-step) and y_{t-1} (the output from previous time-step),[4] and for each value in the previous cell state C_{t-1}, outputs a value between 0 and 1, the implications of 0 and 1 being the same as mentioned in the last paragraph. If the input is denoted by $z_t = (x_t, y_{t-1})$, the forget gate equation will be:

$$f_t = \sigma(W_f z_t + b_f) \tag{3.4}$$

   where W_f and b_f are the weight and bias parameters respectively. The reason for using sigmoid in this layer is that it is easy to saturate the function completely (values between 0 and 1) for memory maintenance.

2. **Input gate:** Once we have decided what values are to be forgotten, the cell state has to be updated with new values. This is decided by the Input gate, which is a combination of two layers, a sigmoid and a *tanh* layer. First, the sigmoid layer decides which values are to be updated. The functionality is similar to the forget gate sigmoid i.e. based on the values between 0 and 1, it is decided which information has to be updated. In the previously mentioned example sentence (section 3.1), the gender information has to be carried on through the end of the sequence. But later, if that information needs to be updated, say when we mention some other person, belonging to a different gender, that's

[4] the output notation in fig 3.1 uses 'h', but we will continue to use 'y', as we formulated our previous equations using 'y' as the output variable

when the input gate comes into picture. For the same reason, it is sometimes also called the '*update*' gate. For the same set of inputs, the update gate equation will be:

$$g_t = \sigma(W_g z_t + b_g) \tag{3.5}$$

Once we decide what values are to be updated, we need new values with which we are going to update them. These are called '*candidate values*', and are created by the *tanh* layer. This layer creates a vector of the new candidate values, which is added to the memory later. These values are in the range -1 and 1. The candidate values are generated as:

$$h_t = tanh(W_h z_t + b_h) \tag{3.6}$$

Now, to add these values to the cell state, we use element-wise multiplication first: $(g_t \otimes h_t)$ Also, from the earlier step, we have the values which are to be forgotten. So, we multiply the old state by f_t, the forget gate output: $(f_t \otimes C_{t-1})$ Combining these two by an element-wise addition operation, we have the Long term memory update equation:

$$C_t = f_t \otimes C_{t-1} \oplus g_t \otimes h_t \tag{3.7}$$

3. **Output gate:** Once we have forgotten and updated the values, we can now output them. The output gate, which is a combination of a sigmoid layer and a *tanh* operation, allows us to produce a filtered version of the output. First, we decide what part of the cell state we want to output. This is done by the sigmoid layer:

$$o_t = \sigma(W_o z_t + b_o) \tag{3.8}$$

Next, we put the current cell state through a *tanh*, so that the values are normalized to be between -1 and 1. These values, when multiplied by the sigmoid output, gives us the final output:

$$y_t = o_t \otimes tanh(C_t) \tag{3.9}$$

The implementation of LSTMs in machine translations is explained in the later sections. A more detailed representation of LSTM gates can be found in this figure:

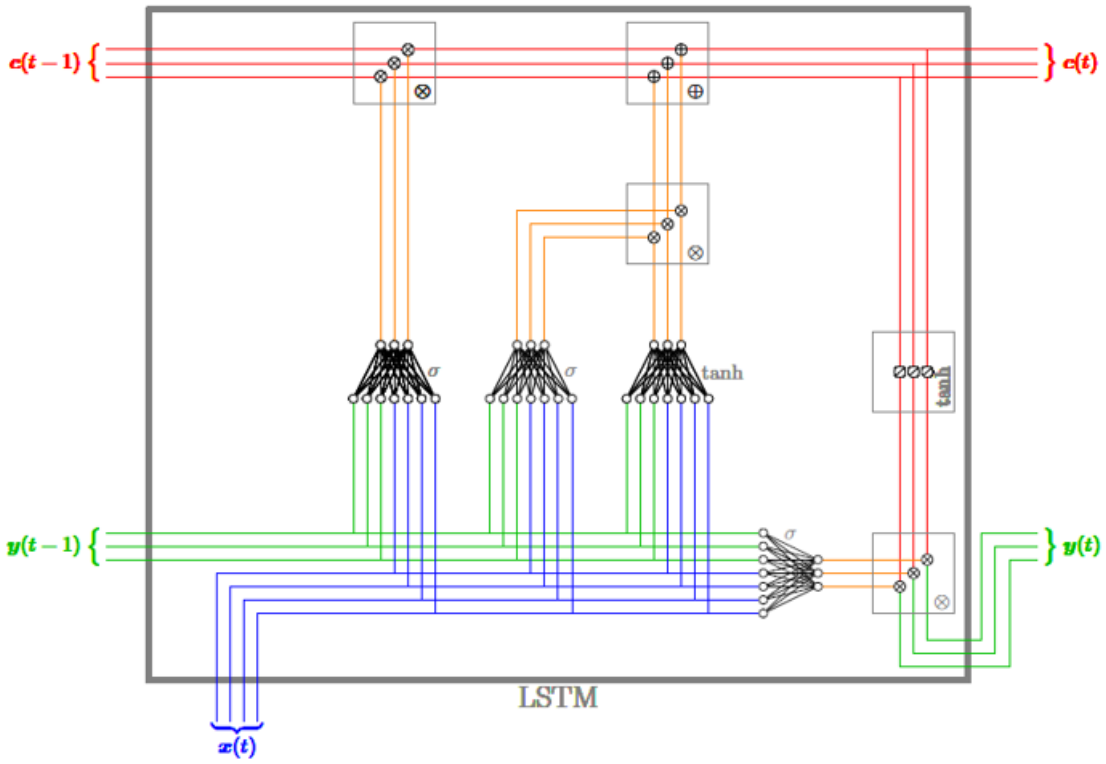

Figure 3.4: LSTM gates, Image taken with thanks from *Adam Prügel-Bennett*'s lecture slide on *COMP6208 Advanced Machine Learning*

The blue lines in the figure represents the input, the red lines are for cell state values and the green lines are for output values (both from previous and current time-step). The vector operations, *sigmoid* and *tanh*, all operations are clearly shown in the above picture. How the LSTMs are trained on the Hindi-English parallel corpus, and how the predictions are made, are discussed in the upcoming sections.

Chapter 4

Corpus

Translation tasks between any two languages require a rich data set, and the parallel corpus for this project was obtained from the '*IIT Bombay Hindi-English Parallel Corpus*' [18]. This was a Hindi-English parallel corpus containing 1,492,827 pairs of sentences. It was obtained from multiple resources, like *GNOME(Opus), HindEnCorp, TED talks, Wiki Headlines, Book Translations (Gyaan-Nidhi Corpus)* and some Indian Government websites, all collated into one, huge, corpus, developed at the Center for Indian Language Technology, IIT Bombay. This is, to the best of my knowledge, the largest freely available Hindi-English corpus.

4.1 Linguistic Analysis

The two languages used for this translation task, are linguistically very different from each other. To start with, the English alphabet has 26 letters, comprising of 5 vowels and 21 consonants as per the Roman script. The Hindi alphabet, which follows the **Devanagari** script, has 50 symbols which constitute the alphabet, known as the *"Varnamala"*, consisting of 12 vowels, 33 consonants and 5 compound letters. These letters are characterized by a horizontal bar on top. English sentences are based on the *Subject-Verb-Object*(SVO) word order, while Hindi follows the *Subject-Object-Verb* (SOV) order. I, being fluent in both these languages, can say that Hindi is a morphologically richer language than English, and is highly phonetic. When

building Machine Translation systems, *language divergence* should always be considered. Two main concerns with the data were **text alignment** and **word sense disambiguation**. In computational linguistics, these are the two major challenges faced in all NLP applications. As mentioned, English and Hindi follow SVO and SOV word orders respectively. This makes the basis for several text alignment cases. Also, English has several cases, where the same word could carry multiple meanings i.e. *heteronyms*. This gives rise to word sense disambiguation. One of the examples could be the word *'art'* used in the following two scenarios:

Scenario 1: `Where art thou?`

Scenario 2: `This painting is a great piece of art.`

In Hindi, the word 'art' has only one meaning (the second scenario), which is 'कला' (pronounced as *'kala'*) and so, the scenario 1 translations can be meaningless. The first scenario may be an archaic English and rarely used, but if encountered, *'art'* still would get translated into 'कला' irrespective of the context. This corpus has many such instances, and these limitations are due to the morphology of both the languages. Such scenarios, their results and limitations are discussed in further sections.

4.2 Pre-processing

To begin with the sentence translations, it was necessary to first obtain a workable corpus. The data, being collected from multiple resources like TED talks and chat transcripts, was a heterogeneous mix of sentences, and thus, cleaning the text was mandatory. Apart from the basic clean-up such as removing special characters, converting to lower case, removing digits and punctuations, there were other discrepancies to be dealt with.

1. **Bad data in files:** A number of sentences in the Hindi text file were non-Unicode characters. It was necessary to find these out, before deleting them, so as to remove the corresponding English sentence as well. Also, there were other language characters, like Chinese and Russian, in the English file. The corresponding sentences from Hindi had to be checked and removed. There were some English characters included in the Hindi

file, which needed no translation. On calculation, it was found that only about 1.8% of the text in Hindi file was written in English, which is a small portion, and therefore, was removed from both files.

2. **Copyright statements and E-mails:** These kind of text items were present only in English language, for obvious reasons. For instance, if an e-mail address is *'abc@gmail.com'*, there is no way to represent it in Hindi. As a result, the corresponding Hindi text was blank, and thus they were found and removed because no translation was needed.

3. **Symbols used as part of Hindi characters:** Hindi, being a linguistically different language compared to English, commonly uses two symbols i.e. '़' and ':' as part of the letters. These symbols are used with the characters and are a crucial part of the combination. While removing punctuations, care was taken not to remove them from the Hindi text, as they are very much a part of the text. The python library `punctuation` from `string` was used, but the Hindi words had to be recognized as one entity (using '`UTF-8`' encoding) so as to avoid getting accidentally removed and distort the word. The important thing to note here is that the symbol '़' or ':' carry meaning only when they are clubbed with the Hindi alphabets. So, if, while splitting the sentences, any of these symbols existed independently, it was safe to remove them, as they would then carry no meaning.

4. **Dealing with sentence lengths:** The corpus, as mentioned above, was a heterogeneous mix and thus had sentences of varying ranges. The average lengths of English and Hindi sentences were calculated to be about 14 and 15 respectively. But, the maximum length of sentences in both languages were 2178 and 2068. While this is bizarre in terms of linguistic rules, it was possible in this corpus, because several sentences formed a single line, in some cases. We can take a look at the sentence-lengths graphically in figures 4.1 and 4.2. Roughly, about 4 sentences are completely out of the range, and these could be considered as outliers. These four sentences were removed from both the sets, as the outliers would insanely increase the embedding matrix size and would result in computational complexities. As the average lengths were 14 and 15, sentences with a

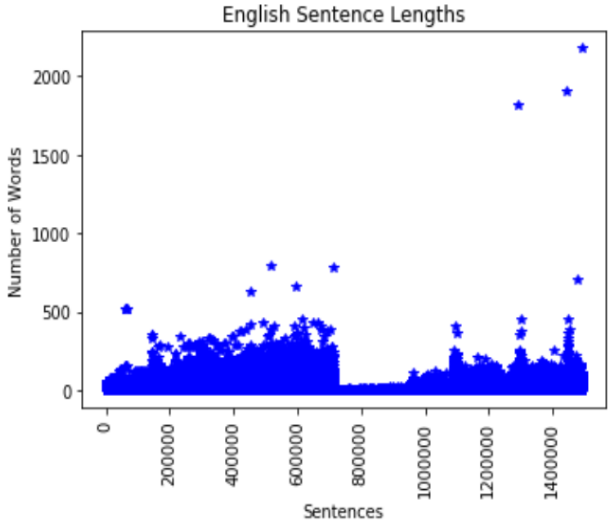

Figure 4.1: Length of English Sentences

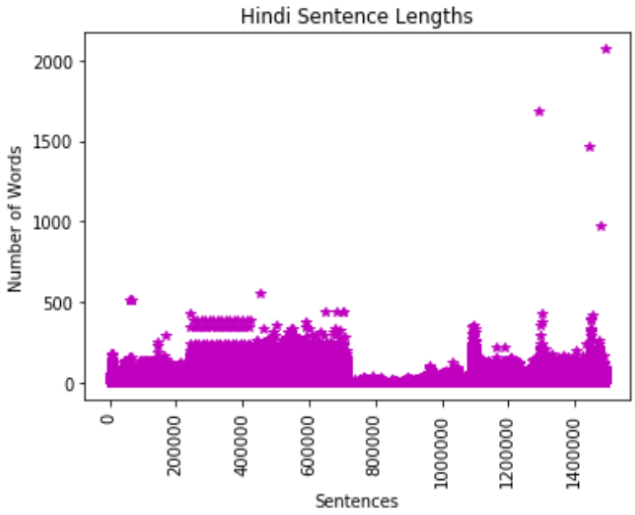

Figure 4.2: Length of Hindi Sentences

maximum length of 30 were considered for translation, as this would give a fair dimension to the encoder and decoder matrices, explained further.

After the above pre-processing, the remaining pairs were estimated to be 1,311,772, which was still a significant-sized corpus to proceed with.

4.3 Tokenization and Indexing

Tokens are the smallest units of any language, capable of independent existence and more importantly, are semantically meaningful. In NLP terms, tokenization is the process of breaking down a given sequence(sentence) into smaller units, i.e. words. Tokenization also includes removing special characters like punctuations, and the resulting tokens are the basic units for processing. A machine learning model is incapable of processing text sequences directly. So, the sequences are broken down into tokens (split by spaces), and a *vocabulary* is created, which contains all the words only once. This gives the vocabulary for that particular language, which means, any word prediction going forward, will be picked from this set. Language Models are also designed to deal with any word which is not present in the vocabulary of that language. These are called '*Out of Vocabulary*' tokens. The easier way to handle this is to have a <OOV>

token as part of the vocabulary. In this model, any OOV words will be replaced with a blank space while decoding the translations. Following this, a dictionary was created, where each word was mapped to an integer index. This will be explained in a later section for 'Embedding'.

The data statistics are shown below:

	Language	Data
No. of sentences		1,492,827
Vocabulary size	eng hindi	154,661 253,673
Maximum Sentence length used	eng hindi	30 30
Train-test split	eng hindi	80-20 % 80-20 %

Table 4.1: Data set statistics.

4.4 Zipf's Law

To understand the distribution of words in both languages, Zipf's law was implemented. The law states that the frequency of a word in any corpus is inversely proportional to the rank of that word. The idea is that, only a small portion of words in the entire vocabulary, are used more frequently, and a large number of words fall in the less-used category. For any language, if the words are ranked according to their frequencies in a table, the actual frequency of any word is inversely proportional to the rank of that word in the frequency table. The Mathematical representation of Zipf's law states that it has the *Pareto distribution*:

$$f(r) \propto \frac{1}{r_\alpha} \tag{4.1}$$

where $f(r)$ is the frequency of the r^{th} rank-word, and the exponent of the law, $\alpha \approx 1$ (or 0.5 sometimes). [1] This is also called the *Power law*[19]. While there is no concrete justification

[1] https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-3-zipfs-law-data-visualisation-fc9eadda71e7

for using the value of α as 1, one of the examples used on the '*Brown*'[2] corpus from python's `nltk` library shows the power law in the graph shown in figure 4.3.

By using the Maximum Likelihood Estimation, the negative slope of the above graph is

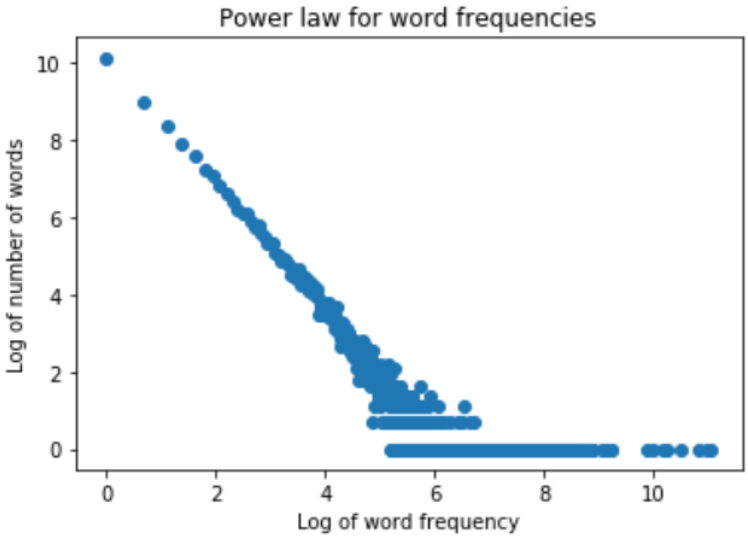

Figure 4.3: Power Law for word frequencies, plotted on the `nltk` *Brown* corpus

found to be 0.54, which accounts for the value of α parameter in Zipf's law. The frequency distributions for English and Hindi words are plotted on log scales[20], and are shown in the graphs in figures 4.4 and 4.5.

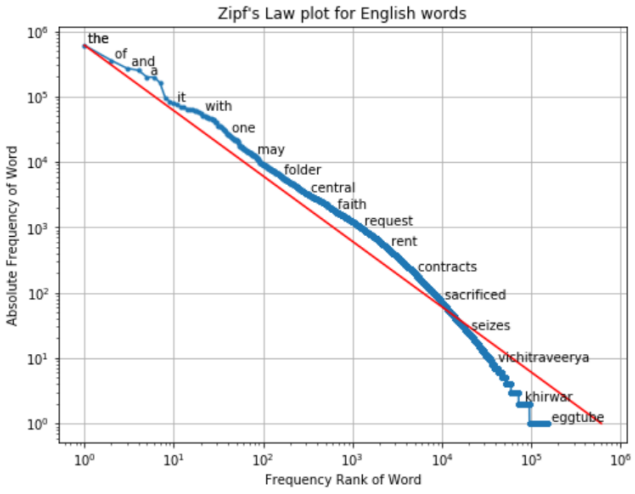

Figure 4.4: Word Distribution for English Corpus

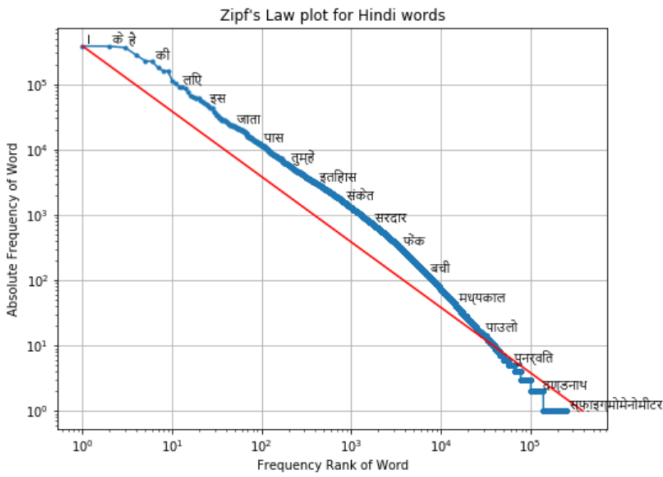

Figure 4.5: Word Distribution for Hindi Corpus

It was observed that the 20 most frequent words, when plotted on the graph, ranked according

[2]https://www.nltk.org/api/nltk.corpus.html

to their frequencies, shows a curve having an almost linear behaviour. The most frequent words in both the graphs lie above the expected lines. These are the highest ranked words. As the word frequency decreases, the lower ranked words are found to be below the linear plot. This kind of behaviour is often termed as the *"near-Zipfian"* distribution of words. Thus, if the most frequent word occurs x times, the second most frequent word would have a frequency reduced by almost half of the first word (x/2), the third word will occur one-third times the first (x/3), and so on. This can be verified in the above graphs as well. In the plot for English language, the word *'the'* is used most frequently in the corpus. 'The' being a stop-word, is obviously the most frequent word in almost all English corpora. In this data set, 'the' occurs about 608,484 times, and the next frequent word as per the Zipf plot is *'of'*, and it occurs about 356,282. As we see, the second word's occurrence is roughly 1/2 times the first word. This establishes the Zipf's Law hypothesis. The frequency trend continues further with the following words. A similar behaviour is observed with the Hindi corpus as well. This "emergence of scaling laws" has also been discussed in Thurner *et al.*[20]. To sum up, as the word rank increases, the frequency decreases rapidly, and hence the expression in equation 4.1. This analysis can also be done after removing the stop-words from both the languages. The "Zipfian" behaviour holds true for both corpora.

Chapter 5

The Model

5.1 Neural Machine Translation

The Statistical MT systems were largely overtaken by Neural Machine Translation(NMT) systems because of their ability to get trained as a whole system. NMT systems do not require multiple, complex machines pipelined as the SMT systems. Instead, NMT systems are end-to-end models, designed to be trained in a supervised manner, where the source and target text are used for training a single system. A basic encoder-decoder network is required for such an approach. Here, a conditional language model is built, which allows us to estimate the probability of target sentence. A Machine Translation model has an encoder network that figures out some representation for the input sentence, and then initiates the decoder network with this representation of the input. This is unlike generic language models, where initially, the vectors contain only 0s. Machine Translation Models are basically Conditional Language Models because instead of modeling the probabilities of any random sentence, it models the probability of the output (translation). If $(X = x_1, x_2, \ldots x_{T_x})$ is a source sequence, and $(Y = y_1, y_2, \ldots y_{T_y})$ is the target, then the encoder network converts the source sentence into vectors of fixed dimensions, and, the decoder network produces the output one word at a time, using the conditional probability:

$$P(Y|X) = P(Y|X_1, X_2, X_3, \ldots X_M) \tag{5.1}$$

where $X_1, X_2, X_3, \ldots X_M$ are the fixed-size vectors encoded by the encoder. The decoder network looks at the input, and previously translated words, to predict the next word. Thus the conditional probability can be re-written as:

$$P(Y|X) = \prod_{i=1}^{N} P(y_i|y_0, y_1, y_2, \ldots y_{i-1}; X_1, X_2, X_3, \ldots X_M) \tag{5.2}$$

[21] The probability of a translated word y_i is conditioned on previously translated words (y_0 to y_{i-1}) and the hidden state outputs $X_1, X_2, X_3, \ldots X_M$ from the encoder LSTM.

5.2 Sequence to Sequence Model

Sequence to sequence models are used when the output is to be predicted after reading the whole sequence. A basic Sequence to sequence model has an encoder-decoder architecture. These encoders and decoders are two separate Neural Networks, combined together to form a bigger Network[1]. This model has two LSTMs, one each for encoder and decoder[16]. In this approach, a translation task is framed as a sequence prediction task, and hence the name. The general working of a sequence to sequence model is outlined below:

1. Input the source sentence to the Network, one word at a time.
2. *Encode* the input sentences into fixed dimension vectors.
3. Take the encoder results and train them to predict probabilities of target words.
4. *Decode* and output the translated sentence, one word at a time.

This is only an outline, the detailed explanation follows. The basic sequence to sequence architecture is shown in the following figure:[1]

[1] adapted from https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

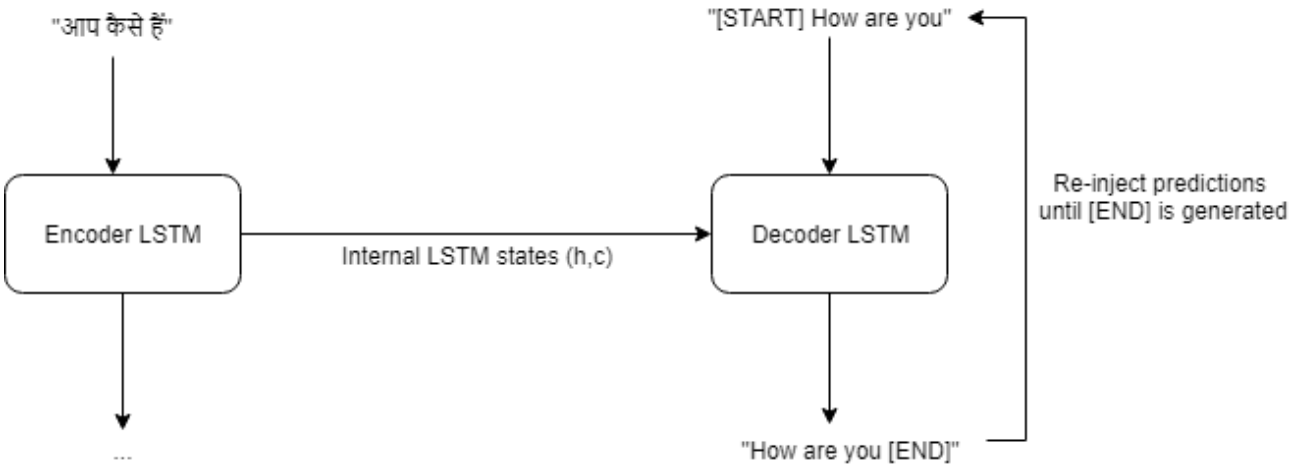

Figure 5.1: Encoder-Decoder structure, adapted from `Keras` blog on Sequence-to-sequence

5.2.1 Word Embeddings

The first, and one of the most important aspects of handling text data is to build word representations. If we consider the generic approach of language modeling, where the training is done for a certain word, and the model is expected to predict the same word in some other sequence, this could be difficult considering the *one-hot encoding* of the words will be same, and thus, an inner product between them will result in a zero. This would result in a huge sparse matrix. Same is the case with Euclidean distance between vectors for same words. A better approach for word representations is to have a **featurized representation** or what is also called a **word embedding**. The idea is to learn a set of features and their values, so that we have dense vector representations for words.

The most common example to explain word embeddings is the '*Man-Woman-King-Queen*' example. If we have a vocabulary of size 10,000 and the positions of these words in the corpus are: [Man-5545, Woman-9678, King-4426, Queen-7523]. We start by creating **one-hot encoded** vectors for these words. A one-hot vector is an n-dimensional vector (n is the vocabulary size of this language), which contains all 0s, except for the word in reference. In this example, if 'Man' is at the position 5545 in a vocabulary of size 10,000, the one-hot vector encoding for 'Man' is a 10,000-dimensional vector of 0s, with only one entry as '1' i.e. at the position 5545, denoted as O_{5545}. Now, if we define these words by some features, say 50, we

can define a matrix of the values of all 50 features, for each word in the corpus. An informal representation of the '**Embedding matrix**' is given below:

$$\mathbf{E_{50\times 10K}} = \begin{array}{c} \begin{matrix} \text{Man} & \text{Woman} & \text{King} & \text{Queen} \end{matrix} \\ \begin{bmatrix} -1 & 1 & -0.99 & 0.98 \\ 0.02 & 0.01 & 0.94 & 0.97 \\ \vdots & \vdots & \vdots & \vdots \\ 0.78 & 0.84 & 0.91 & 0.92 \end{bmatrix} \end{array} \begin{matrix} \\ \text{Gender} \\ \text{Royal} \\ \\ \text{Kind} \end{matrix}$$

This embedding matrix is initialized randomly. 'Gender', 'Royal' and 'Kind' are 3 out of 50 features. So, each word from this matrix will be a 50-dimensional vector. For example, if the male gender is considered as -1 and female as 1 (only for differentiating in this example, this is completely an unbiased and random assumption), in the embedding matrix, the word 'Man' has a value -1 for the 'Gender' feature and 'King' has -0.99 for the same feature. This states that 'man' and 'king' are very similar in gender. So, when these word vectors will be placed in a 50-dimensional vector space, there are high probabilities of a greater *cosine similarity* in between vectors, if all other features are similar too. Now, to find the word embedding of 'Man', the embedding matrix(E) is multiplied with the one-hot encoded matrix (O_{5545}) for 'Man':

$$\begin{array}{c} \begin{matrix} \text{Man} & \text{Woman} & \text{(All words)} & \text{King} & \text{Queen} \end{matrix} \\ \begin{bmatrix} -1 & 1 & \dots & -0.99 & 0.98 \\ 0.02 & 0.01 & \dots & 0.94 & 0.97 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0.78 & 0.84 & \dots & 0.91 & 0.92 \end{bmatrix}_{50\times 10K} \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \boxed{1} \\ \vdots \\ 0 \end{bmatrix}_{10K\times 1} = \begin{bmatrix} -1 \\ 0.02 \\ \vdots \\ 0.78 \end{bmatrix}_{50\times 1}$$

The result is a 50-dimensional vector representation of 'Man'. This is called the 'embedding' for the word 'Man'. For the same reason, the number of features i.e. 50, is also called the embedding size. Similarly, all the words in the corpus are represented as 50-dimensional vectors. Some

well-known pre-trained word embeddings are *Word2Vec*[2], *Glove* [3], *fasttext*[4].

`Keras` provides a slightly different approach than the ones mentioned above. In the pre-defined class `Embedding`, keras pulls out from the embedding matrix (E), only the column which corresponds to the input word, and directly produces that as the word embedding. This avoids multiplication of the huge matrix E (in case the vocabulary size is huge) and the one-hot encoded vector. This is logically right as well. Each input integer is treated as an index to pick the relevant column from the embedding matrix. Only the relevant column from the randomly initialized matrix is picked, which is dimensionally 50×1 in this case (same as previous approach). This makes the embedding process faster. This stage is defined as the "Embedding layer" in the network. The word embedding weights are 50-dimensional and so, in order to visualize them, **t-SNE** (*t-Distributed Stochastic Neighbour Embedding*)[5] can be used. Given below is a representation of about 2000 words from the corpus, in a 2-dimensional space, with one of the word coordinates highlighted:

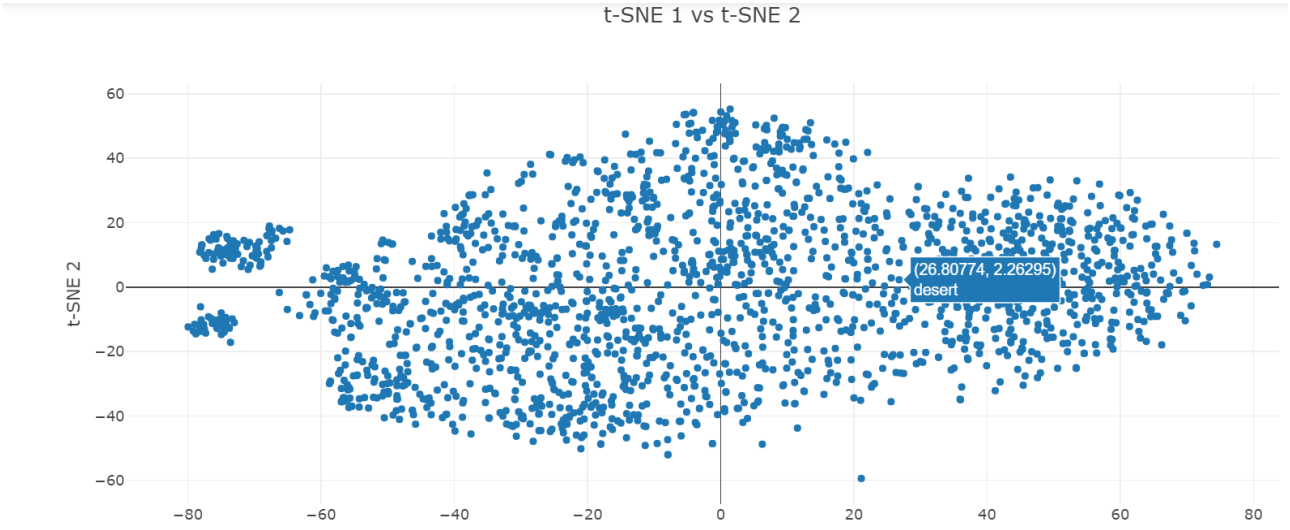

Figure 5.2: Word embeddings on a scatter plot of first two components of t-SNE

These embeddings are obtained from trained models, which are discussed in further sections. Each of these 50-dimensional word embedding vector is further fed into the LSTM, and the embedding matrix parameters are learned through back-propagation to maximize the likelihood

[2]https://radimrehurek.com/gensim/models/word2vec.html
[3]https://nlp.stanford.edu/projects/glove/
[4]https://fasttext.cc/
[5]http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

of training set to predict errors.

5.2.2 Encoder LSTM

In Sequence to Sequence learning methodology, the input sequence is fed to the encoder LSTM, one word at a time. These words are the embeddings generated in the previous step. Structurally, the LSTM layer sits right after the Embedding layer, thus taking the embedding vectors as inputs. The important thing to note here is that the input sequences will be of different lengths. So, to maintain a constant length of inputs, the maximum length of sentences in calculated, and the input matrix dimension was created accordingly. Dimensionality of input and output sequences is explained in a separate section. Now, the Encoder LSTM processes the input sequence and returns the internal states. As described in previous sections, LSTMs have internal gates to forget/remember values, and also the cell states. The actual output from the encoder is discarded, as only the *context* [6] is required going ahead. For this reason, the `return_state` argument is set to `True` in the `LSTM()` function call. This returns the hidden and cell state for all the cells in this LSTM layer. To sum up, the Encoder model will have a structure that takes Input into the Embedding layer, followed by an LSTM layer. At this stage, the input sequence is mapped to a fixed dimension state vector, which is further fed to the decoder LSTM.

5.2.3 Decoder LSTM

The decoder LSTM takes as input the vector mapped by the encoder, and it is then trained to output the translation, one word at a time. This LSTM predicts the next words of the target sequence, given the previously translated words from the sequence. It basically generates the target sequence, offset by one time-step in future, using the state vectors from encoder as the initial state. This method of learning is also called the "teacher forcing" [7] method.

[6] A context in NLP, is a situation, or condition under which the models are trained to make predictions accordingly.

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

Structurally, the Inputs of a decoder LSTM goes through the Embedding layer, where word embeddings are created for the target words (English in this case). The embedding dimension chosen for both encoder and decoder networks is 50. This means we are specifying the number of features for each word in source and target vocabulary. The first argument to the `LSTM()` call is the number of units, which is the dimensionality of output space. In this case, the number of units is chosen as 50 (same as embedding size). Within the LSTM layer, each of these units will have an internal cell state (c), and a hidden state(h). These state values are recorded along with decoder output, so that the state values for next time-step are updated using these current values. This process is explained under "predictions". If we want to access the hidden state values for each input time step, the `return_sequences` argument should be `True`. Also, the `return_state` argument, if set to `True`, allows access to the hidden state output of the previous time step, and the cell state for the last input time step. Once the encoder and decoder LSTMs are defined using appropriate Embedding dimensions, the conditional probability discussed in previous equations are estimated using *softmax*. Each probability distribution is represented using a softmax over the entire vocabulary. Thus, finally the LSTM outputs are wrapped using the densely connected network layer, `Dense`. The following operation is implemented by Dense layer:

$$output = softmax(dot(input, weight) + bias) \tag{5.3}$$

This softmax activation is carried out over all the probabilities of each word in target vocabulary. The Model structure is shown in figure 5.3: [8]

5.2.4 Dimensionality

Before building the model, as part of the pre-processing, "START_" and "_END" were appended to the beginning and the end of the target sentences (in this case, English sentences only). This is done so that the decoder has a stopping condition which is either and "_END" being encountered, or maximum word limit is reached.

[8] https://towardsdatascience.com/sequence-to-sequence-using-encoder-decoder-15e579c10a94

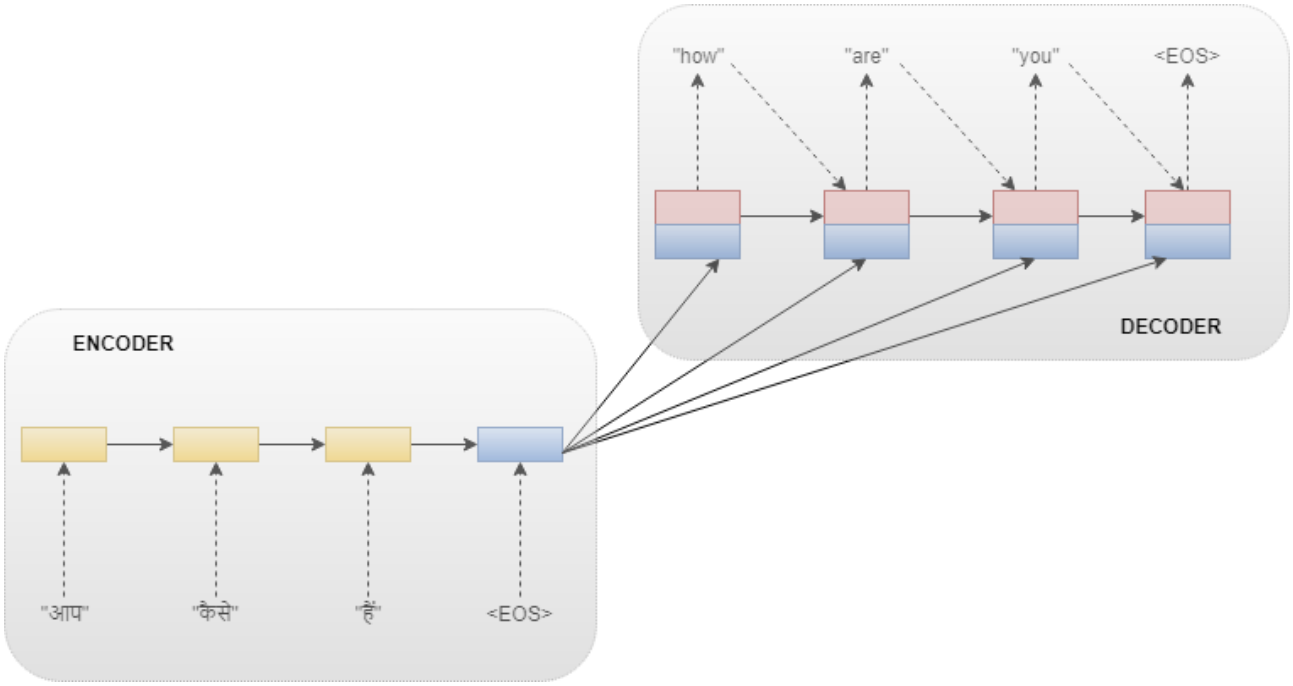

Figure 5.3: Encoder-Decoder structure, adapted from *towardsdatascience* blog on Sequence-to-sequence, originally inspired by Cho *et al.* [1]

Vocabulary size for Hindi: 253673

Vocabulary size for English: 154661

This was a huge vocabulary size, resulting in `resource exhausted` errors. Thus, sentences were first split according to lengths and multiple models were trained. Due to machine restrictions, only 1 Million sentence pairs were trained successfully.

Once the unique words of both the languages are found, and separate vocabulary is created, the encoder and decoder dimensions can be defined. Each word is mapped to an integer and that integer is used to create the input data matrix for encoder and decoder. Now the encoder input data will have the index of words in Hindi vocabulary, and the decoder input will have the index of words from English vocabulary. The third matrix i.e. the decoder target data will have as the third dimension, all words in the English vocabulary, initialized by 1, and later, the softmax over all these words, would predict the probabilities of the word being predicted next. The input and output sequences are defined as `numpy` [9] arrays, the dimensions of which are given below:

[9] https://docs.scipy.org/doc/numpy/user/quickstart.html

1. Encoder input: (No. of sentence pairs, Maximum Hindi sentence length)

2. Decoder input: (No. of sentence pairs, Maximum English sentence length)

3. Decoder target: (No. of sentence pairs, Maximum English sentence length, No. of words in English vocabulary)

   Dimensionality is similar to the decoder input, only offset by one time-step in future. Since the decoder target is ahead of the decoder input, the first character i.e. "START_" will be ignored by the decoder. This is why it was appended to the beginning of each target sentence, so that the decoder starts making predictions from the actual first word in the sentence.

With all the above properties, the overall model summary looks something like this:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, None)	0	
input_2 (InputLayer)	(None, None)	0	
embedding_1 (Embedding)	(None, None, 50)	12683650	input_1[0][0]
embedding_2 (Embedding)	(None, None, 50)	7733050	input_2[0][0]
lstm_1 (LSTM)	[(None, 50), (None,	20200	embedding_1[0][0]
lstm_2 (LSTM)	[(None, None, 50), (	20200	embedding_2[0][0] lstm_1[0][1] lstm_1[0][2]
dense_1 (Dense)	(None, None, 154661)	7887711	lstm_2[0][0]

Total params: 28,344,811
Trainable params: 28,344,811
Non-trainable params: 0

Figure 5.4: Model summary captured after compilation

Chapter 6

Experiments

6.1 Source Sentences Reversed

One of the most important features of this model is that the source sentences were fed to the encoder in reverse order. This key idea was inspired by Google's research on 'Sequence to Sequence Learning'. [16] Although LSTMs usually handle long term dependencies efficiently, this reversal of source sentences retained a uniform distance between corresponding source and target words. The idea was to keep the initial words of the source sentence in close proximity to the initial words of the target sequence. Due to this, the target sentences were not reversed. As the distance between some of the words in each source-target pair was less due to reversal, there was less time-lag and hence, easier source-target communication during back propagation. This model was trained under both conditions, i.e. with and without reversing source sentences. The **BLEU** score showed a change from 0.24 to 0.29 when the source sentences were reversed. On a huge data set like this, this change was significant, and it supports the claim made in the Sequence-to-Sequence paper that "due to many new short-term dependencies being introduced, optimization was easier".[16] Although, this data set has a large number of short sentences, which contributed to the advantage. Contradictory to the claim in [16], the longer sentences did not fair well on this model. Some example translations are analyzed under the 'Analysis' section.

6.2 Training Parameters

Deep Learning is essentially an empirical process. There is never a well-defined number of epochs, or perfect batch size and so on. Consequently, there were different configurations each time the model was trained. A generic parameter description is given below:

- Number of features: 50 (The embedding size, described in previous sections)
- Optimizer: Both `RMSprop` and `adam` were tested, `adam` being better in terms of results. As *Adam* is an adaptive moment estimate algorithm, it was found to work well with "sparse gradients and moving averages"[22]. It was more suitable for optimizing the cost function, as high-dimensional parameter spaces were encountered. Also, it combines the advantages of *RMSprop* and *AdaGrad*[22]. The default learning rate of 0.001 was used.
- Train-test split: This was mostly an 80-20 ratio. However, in some cases, the distribution differed slightly. The final model had 800K sentence pairs for training, and 200K pairs for test.
- Number of epochs: For smaller data sets, such as 150K pairs of sentences, regular `model.fit()` function from `Keras` was used, with about 50-60 epochs. For the entire data set i.e. 1M sentence pairs, generators(explained in the next section) were used, and hence the number of epochs were significantly reduced to 20, or even up to 10!
- Batch size: As stated, two different training methodologies were attempted for different data sizes, thus making batch-size explanation tougher. For normal training on 150k sentence pairs, a batch size of 1024 was used. For generator-based training, both training and test generators had 32 as the batch size (with different step-sizes).
- As the data size is large, a note on the hardware: A machine that contains a graphics processing unit (GPU) that supports the CUDA® Deep Neural Network library (cuDNN) designed for Nvidia GPUs.

With several permutations of the above training parameters, experiments were performed, and the best learning curves obtained are shown in figures 6.1 and 6.2.

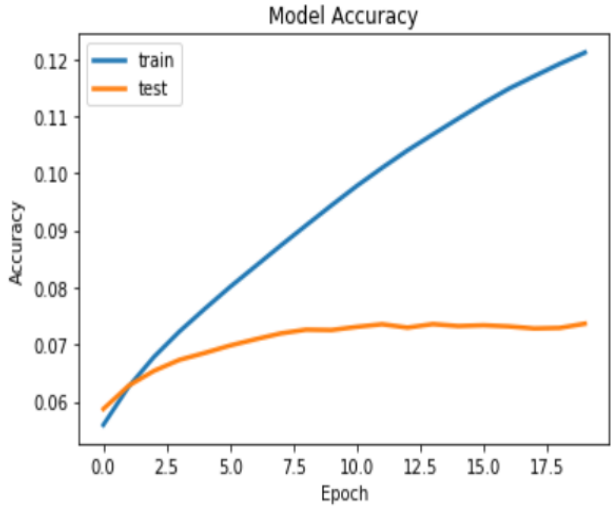

Figure 6.1: Model Accuracy on 50% data

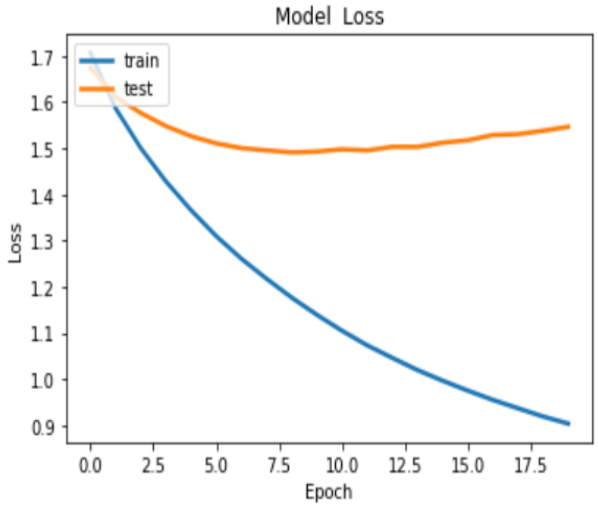

Figure 6.2: Model Loss on 50% data

6.3 Generators

Large data sets consume a significant amount of memory. The basic training configurations are not enough to train around 1.4 million sentence pairs. The training time for 150,000 pairs of sentences was nearly 4 to 5 hours, and this accounted for only 10% of the data. To train large data sets, `Keras` provides the concept of multi-processing and batch training. In this configuration, the real-time data is generated on multiple cores and fed to the deep learning model in a parallel fashion. The `generator` [1] instance is run in parallel to the model. The output of a generator is a tuple containing (`input, target`). While training, a batch of data is used (on GPU) and the generator runs on the other batch of data(on CPU) to augment it. This batch training method makes training a model on a very large data set possible. Different training configurations were tried with generators. Some of them are mentioned below:

- Train-test data: both were generator objects with batch size 32 each
- steps per epoch: This was roughly between 25000-35000, which means, at every epoch, 25000 steps of 32-batch data were trained. This is calculated as: (Number of training samples / batch size). Validation steps were similarly calculated to be 6250.

[1] https://keras.io/models/sequential/

Chapter 7

Results

After training the model, predictions were sampled using **Beam Search** and the quality of translations were calculated using **BLEU** scores. For any given source sentence, if there are more than one "perfect" translations, which is a reasonable expectation in linguistics, the accuracy of the translation system should be calculated. The next sections explains Beam Search and BLEU score calculations.

7.1 Predictions: Beam Search

Given an input Hindi sentence, the model estimates the probability of different corresponding English translations i.e. $P(Y|X)$ where $(Y = y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \dots y^{\langle T_y \rangle})$. To pick the most likely translation, if the outputs are sampled randomly from this distribution, for the first couple of times, good translations may be obtained. But the translation accuracy could go worse with more and more sampling. So, to maximize the conditional probability:

$$\hat{Y} = \arg\max_{Y} P(Y|X) \tag{7.1}$$

beam search algorithm is used. Beam search encoders were introduced in 2003, initially for a trigram SMT-based language model. Inspired by the traveling-salesman problem, the main

aim of this algorithm was "to restrict the possible word orderings between source and target language for an efficient search".[23] A basic left-to-right beam search works in the following way:

1. If we choose Beam Width = B = 3, in the first step, the model evaluates the probability of the first word, given only input X i.e. $P(y^{\langle 1 \rangle}|X)$. In this step, the input Hindi sentence is run through the encoder LSTM and the first step of the decoder LSTM will be a softmax output over all the possibilities in the English vocabulary. Now for the first word, of all the likely translations, top three are picked. The algorithm stores these three choices for the next steps.

2. In this step, for each of the three options picked, the next choice is estimated i.e. $P(y^{\langle 2 \rangle}|X, y^{\langle 1 \rangle})$. The network hard-wires the first word $y^{\langle 1 \rangle}$. After this step, we have the conditional probability calculated as:

$$P(y^{\langle 1 \rangle}, y^{\langle 2 \rangle}|X) = P(y^{\langle 1 \rangle}|X).P(y^{\langle 2 \rangle}|X, y^{\langle 1 \rangle}) \tag{7.2}$$

3. Similarly, in the third step, the network fragment will hard-wire the first two words calculated in the previous steps. And if $B > 3$, the same process is followed by the model to output the most likely sentence where it goes until $< EOS >$ is encountered, or the maximum word limit is reached.

In this model, a Beam Width of 2 is used. A Beam Width of 1 is basically a greedy search where an `argmax` over the probabilities would be sufficient. The results with beam width of 2 provided better results, along with the final probability. The conditional probabilities were multiplied after each step and a final probability was obtained in the last step of beam search. This probability was obtained too, for the top two possible translations. This helped in ensuring that the algorithm is working according to the conditional probability values.

Also, to make the results more stable numerically, 'Length Normalization' was used. The

probability was earlier defined as:

$$\arg\max_{Y} \prod_{t=1}^{T_y} P(y^{\langle t \rangle} | y^{\langle 1 \rangle}, y^{\langle 2 \rangle} \dots y^{\langle t-1 \rangle}; X) \tag{7.3}$$

To avoid numerical underflow, log values were used and then, normalized by the number of words used in translation:

$$1/T_y^{\alpha} \{ \sum_{t=1}^{T_y} logP(y^{\langle t \rangle} | y^{\langle 1 \rangle}, y^{\langle 2 \rangle} \dots y^{\langle t-1 \rangle}; X) \} \tag{7.4}$$

where $0 < \alpha < 1$ is one of the hyper-parameters. [1] An example of how the predictions are displayed using beam search is given below:

Source: कालबैक फोन

Target: `callback phone`

Translation with Greedy search: `callback phone`

Translation with Beam search (B=2): `[(' callback phone', 0.19577222737129762), (' phone contrast', 0.0032021621079655893)]`

So, we can see that Beam search gives the advantage of displaying top two probabilities with the predictions, and then verifying if the translation makes sense accordingly.

7.2 Evaluation: BLEU Score

BLEU (*Bilingual Evaluation Understudy*) is an automatic evaluation metric used to compute the accuracy of a Machine Translation (MT) system. The idea was originally proposed by Papineni *et al.* [24] as an alternative to human evaluation of translation systems. The BLEU score is currently one of the most important evaluation metrics used by the MT community, owing to it's inexpensive calculations, and language independence. In case of sentence-based translations, given a source sentence and it's human translation (called *reference*), we compare all the machine translations (called *candidates* or *hypotheses*), and the BLEU score evaluates

[1] The Beam Search hypothesis was inspired by Andrew Ng's Deep Learning course on Coursera.

which of the multiple translations is closest to the reference. The closest one gets the highest BLEU score, and the score keeps decreasing for the less closer translations, essentially remaining between 0 and 1. Here, the `corpus_bleu` [2] method from python's `nltk` package is used to calculate the BLEU score for the entire set of translated sentences. This score is calculated on a system level, and is for all the candidates against their respective references. For supervised learning as this, the references are already given in the form of target data (English text, in this case).

The raw BLEU score calculation tends to produce misleading results in many cases. The main idea behind this calculation is to compare *n-grams* of the candidates to that of the reference and give a count of the matching n-grams. The problem with this approach is that the matching words are position independent. Let us consider the following example, translated from a Hindi sentence:

Reference: `I like music.`

Candidate 1: `I like music.`

Candidate 2: `I enjoy music.`

Candidate 3: `I love music.`

While all the above candidates are fairly acceptable translations, there may be a possibility where the machine translation could go wrong due to over-fitting or under-fitting like:

Candidate: `like like like`

The precision calculated in this case will be 3/3 which means a perfect BLEU score of 1. Logically, all words of this candidate appear in the reference sentence and hence the incorrect score. To deal with this, we give each word credit only up to the maximum number of times it appears in the reference. This is called the $Count_{clip}$. [3] [24]. In this case, the word 'like' appears only once in the reference, so the correct BLEU score should be 1/3.

Result: The best BLEU score achieved using this approach was 0.55, which was much higher than the state-of-the-art score. So, to handle this, n-grams were used.

[2] https://www.nltk.org/_modules/nltk/translate/bleu_score.html

[3] $Count_{clip} = min(Count, Max\_Ref\_Count)$

7.2.1 *n*-gram precision

To make a more sensible calculation, it was necessary to take into account, groups of words for comparing. The *unigram* BLEU score was not up to the mark. For n-gram precision scores on a translated group of sentences, say $\hat{y}$, the BLEU score is calculated as:

$$P_n = \frac{\sum_{n-gram \epsilon \hat{y}} Count_{clip}(n-gram)}{\sum_{n-gram' \epsilon y} Count(n-gram')} \tag{7.5}$$

where p_n is the BLEU score on n-grams. This gives the number of times each n-gram appears on the reference. This model uses 4-gram precisions.

7.2.2 Brevity Penalty

In cases where candidate translations are longer than the references, the scores are penalized by the n-gram precision measure. With short translations, getting a higher precision is easier. Thus, *Brevity Penalty* is used as an adjustment factor, to penalize the system for short translations. It is used to modify the overall BLEU score according to the sentence length. First, the average of modified n-gram precisions is calculated i.e. p_n with N as the n-gram length and w_n as the weights. So the combined BLEU score is calculated as:

$$BLEU = BP.exp\left(\sum_{n=1}^{N} w_n log p_n\right) \tag{7.6}$$

where BP = Brevity Penalty =

$$\begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

c and r being the respective lengths of candidates and reference corpus. [24] In this model, N=4 and $w_n = 1/4$ values are used for calculating the corpus BLEU scores. The best results

obtained are **0.35** on a 4-gram precision system. This result was obtained on a small data set i.e. 150K sentence pairs. Different samples of training and tests were used for predictions, and the BLEU scores obtained are presented below:

n-grams	0 to 150K	150K to 300K	300K to 450K	450K to 600K
uni-grams	0.52	0.41	0.42	0.51
bi-grams	0.44	0.41	0.41	0.47
tri-grams	0.37	0.37	0.37	0.41
4-grams	0.32	0.32	0.33	0.35

Table 7.1: Training BLEU scores on different data samples

Separately, a model was trained and tested on 600K sentence pairs together, and the training BLEU score for 4-gram precision was found to be 0.33, which is also the average scores obtained from above four samples. Similarly, the test BLEU scores were also calculated for the same four data samples:

n-grams	0 to 150K	150K to 300K	300K to 450K	450K to 600K
uni-grams	0.50	0.49	0.49	0.51
bi-grams	0.41	0.33	0.35	0.42
tri-grams	0.34	0.28	0.29	0.36
4-grams	0.30	0.25	0.25	0.31

Table 7.2: Test BLEU scores on different data samples

The overall test BLEU score on 600K sentence pairs was 0.32. This score was different from the average of the above four scores, which was 0.28.

Additionally, 800K sentence pairs were trained and tested for quality. The final BLEU score on this data size was 0.32 and 0.29 for training and test respectively. And finally, when the model was trained on 1 million sentence pairs, the training and test BLEU scores were found to be **0.33** and **0.30** respectively. This may be regarded as the final outcome of this project.

Chapter 8

Error Analysis

With the model configurations and training parameters discussed in previous sections, fairly good translations were obtained for a large number of sentences. Some sample translations are shown below:

Source: वर्तनी सुझाव

Target: `spelling suggestions`

Translation: `spelling suggestions`

Source: कैमरा से लिया गया छवि की ऊंचाई पिक्सेल में

Target: `the height of the image captured from the camera in pixels`

Translation: `the height of the image captured from the camera in pixels`

Source: समूह अनुसार होम स्क्रीन नाम

Target: `groupwise home screen name`

Translation: `groupwise home screen name`

Source: कैमरा से लिया गया तस्वीर का संतृप्ति समायोजित करें

Target: `adjusts the saturation of the picture coming from the camera`

Translation: `adjusts the saturation of the image coming from the camera`

8.1 Synonymous Translations

The last example is an interesting case of the machine being able to learn and predict in a more human-like way. As we can see, the prediction is very close to target, except only one word i.e. *picture* getting predicted as *image*. Rest of the sentence is same as the expected translation. The translation is also equally valid. The reason being, the same Hindi word is translated to two different words in the target set. Few source-target pairs are shown below:

Source	*Target*
तस्वीर चमकीलापन	**Image** brightness
कैमरा से लिया गया **तस्वीर** का चमकीलापन समायोजित करें	Adjusts the brightness of the **image** coming from the camera
कैमरा से लिया गया **तस्वीर** का कंट्रास्ट समायोजित करें	Adjusts the contrast of the **image** coming from the camera
कैमरा से लिया गया **तस्वीर** का संतृप्ति समायोजित करें	Adjusts the saturation of the **image** coming from the camera
कैमरा से लिया गया **तस्वीर** का वर्ण समायोजित करें	Adjusts the hue (color tint) of the **image** coming from the camera
तस्वीर चमकीलापन	**Picture** brightness
वेब कैम से लिए जाने वाले **तस्वीर** का चमकीलापन का स्तर समायोजित करें	Adjusts brightness level of the **picture** coming from the camera
वेब कैम से लिए जाने वाले **तस्वीर** का विरोध स्तर समायोजित करें	Adjusts contrast level of the **picture** coming from the camera
वेब कैम से लिए जाने वाले **तस्वीर** का संतृप्ति स्तर समायोजित करें	Adjusts saturation level of the **picture** coming from the camera
वेब कैम से लिए जाने वाले **तस्वीर** का संतृप्ति स्तर समायोजित करें	Adjusts hue level of the **picture** coming from the camera

Figure 8.1: Examples of one Hindi word translated to two different English words with same meaning

Thus, the model was trained to map the Hindi word तस्वीर to *picture* as well as *image*. This is why at the time of making predictions, the model was 'clever' enough to pick either of the synonymous target words! Another example of deviating from the target data, and yet getting perfect translation is shown below:

Source: अग्नि फ़्लैश

Target: `fireflash`

Translation: `flash of fire`

This is another example of the 'intelligence' used by the model. As we can see, the target is a single word *fireflash*. The model was not only able to understand that it is a combination of two words *fire* and *flash*, but also the preposition '*of*' was inserted at the right place, making it a perfect translation. The reason *flash* was paired with *of* is because the target sequences contains the phrase '*flash of*' 50 times in the entire corpus of sentences. Thus during training, the model learns these two words together for 50 cases and treats them as a single unit while making predictions.

8.2 Learning weights during training

In previous sections based on 'Embedding', it was stated that weights are randomly assigned to the embedding matrix, and during the course of training, these weights are learned by the model. Here are a few snapshots of the weights before and after the training, to understand how these weights are affected during the training process. Only a part of the matrices are displayed, which are weights corresponding to a single word.

```
array([[[-2.12743524e-02,  2.70604007e-02,  1.39794834e-02,
         -8.60427693e-03, -4.53822128e-02, -1.65938959e-02,
         -3.00812367e-02, -3.71700302e-02,  2.00772770e-02,
         -2.93174982e-02,  1.53748281e-02,  3.75246666e-02,
         -1.30359530e-02,  5.55950403e-03,  4.20277826e-02,
         -5.73606417e-03,  4.03365232e-02, -4.26580310e-02,
          5.70797920e-03,  3.47492434e-02,  2.10695751e-02,
         -4.78031412e-02,  3.27713788e-05,  2.89356224e-02,
         -1.78753249e-02, -4.34534550e-02,  3.79843153e-02,
         -1.53545290e-03, -2.41463427e-02, -3.99859436e-02,
          1.96794383e-02,  4.25958298e-02, -4.78396192e-02,
          3.64323594e-02,  3.82750519e-02, -3.04806978e-04,
         -1.25782713e-02,  2.96005867e-02, -5.55708259e-03,
          4.32652347e-02,  3.01399343e-02,  3.08627002e-02,
         -4.00831588e-02,  3.69814374e-02,  3.46665420e-02,
          4.56570052e-02, -2.57054102e-02,  3.70939262e-02,
          4.31931876e-02, -3.98536436e-02],
```

Figure 8.2: Sample Encoder weights before training

```
array([[[-2.39873101e-04,  4.68372479e-02,  3.92991826e-02,
         -1.49542049e-01, -1.66344479e-01, -1.06822208e-01,
         -5.81612624e-02, -2.99421042e-01,  3.18810046e-01,
         -9.06844065e-02,  2.35995933e-01,  2.67584957e-02,
          3.28982264e-01,  1.19509595e-02,  8.32481086e-02,
         -1.55106619e-01,  1.27734646e-01, -4.24802490e-03,
         -1.18083403e-01,  1.92940518e-01,  2.04273984e-01,
         -5.12237027e-02,  3.80488820e-02,  9.43991244e-02,
         -1.39788672e-01, -1.82414159e-01,  1.62410960e-01,
          7.99430236e-02, -5.09389453e-02, -2.36108974e-01,
          2.84239829e-01,  1.15832888e-01, -3.40746641e-01,
          1.19470879e-01,  9.60653350e-02,  2.55730841e-02,
         -2.15238243e-01,  1.78756282e-01, -1.22981509e-02,
          1.72097459e-01, -1.25346370e-02,  1.82670951e-01,
         -1.59429640e-01,  6.49007931e-02,  6.45693839e-02,
          7.37967193e-02, -1.82873249e-01,  1.34815201e-02,
          8.73813480e-02, -1.19421013e-01],
```

Figure 8.3: Sample Encoder weights after training

The matrices show that the weights are different before the training, and once the model learns these weights, different values are loaded in the matrices. Although these values do not provide a great deal of information, as these are random numbers, initialized in the beginning, when

the encoder and decoder matrices are created. Every word is a 50-dimensional vector, and each of these 50 values correspond to a separate feature describing the word. These values further help in creating word embeddings. The only piece on information from these snapshots is that the model learns these weights throughout the training process.

8.3 Effects of Reversing Source sentences

LSTMs work well with long term dependencies, but as an additional measure, reversing the source sentences showed improvements in BLEU scores, and significant improvements in the accuracy of translations. As claimed by Sutskever *et al.* [16] reversing the source sentences made LSTMs learn better. As explained in section 6.1, reversing the source sentences retains the average distances between the corresponding source and target words, thus making the optimization easier. However, the claim that source reversal improves accuracy of longer sentences, did not hold good for this data set. Here are a few examples of longer sentence translations:

Source: विंडो की आरंभिक चौड़ाई यह मान अद्यतनीकृत होता है जब उपयोक्ता इस विंडो का क्षैतिज रूप से आकार बदलता है
Target: `initial width of the send and receive mail window the value updates as the user resizes the window horizontally`
Translation: `initial height of the select video window in the network`

Source: किस मापदंड को फ़ाइलों को व्यवस्थित करने के लिए प्रयोग किया जाना चाहिए संभावित मान हैं: नाम आकार प्रकार समय पथ
Target: `what criteria must be used to arrange files possible values name size type time path`
Translation: `drag or list of work address images if exit the default`

Using reversed sentences as an input to the encoder LSTM did not do well here. A possible explanation could be because of the proportion of long length sentences in the entire data

set. Almost 50% of the data consists of short length sentences i.e. less than 10 words. This dominance by short sentences causes the model to be trained on fewer short-term dependencies. Thus, with longer sentences, during back-propagation, the model still does not find enough epochs to establish communication between source and target words. This is contrary to what was claimed Sutskever *et al.* [16], the reason being the proportion of long sentences in WMT'14 English to French translation task (used in Sutskever *et al.* [16]) was greater than in this data set. On the other hand, some researches have reported poor performances of similar models, when tested on long sentences[1, 15]. Cho *et al.*[25] also suggested that neural network models perform poorly with lengths $>\approx 20$ words in similar settings.

8.4 Effects of data size on Gradient Descent

The idea of training different models on different data sizes and sets of parameters gave a clearer picture of how the results change with larger data. Varying data sizes helped gain a better understanding of the training parameters. Each time the model was trained on a bigger data set than the previous one, and the results were recorded. Beyond 200K sentence pairs, `generators` were used for batch-wise training. Models were trained separately on 600K, 800K and 1M sentence pairs. The learning trends of the largest data set is shown in the following graphs:

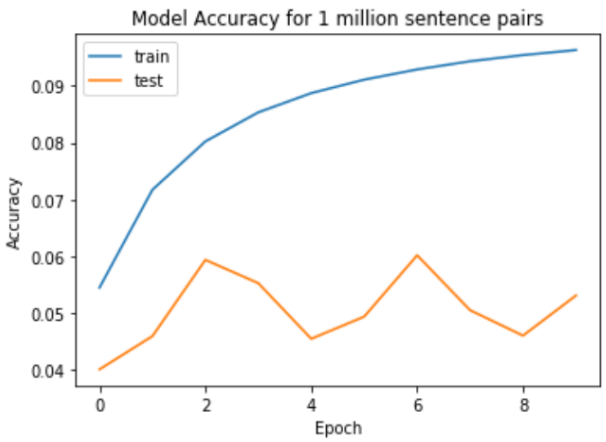

Figure 8.4: Model Accuracy on 1M sentence pairs using generators

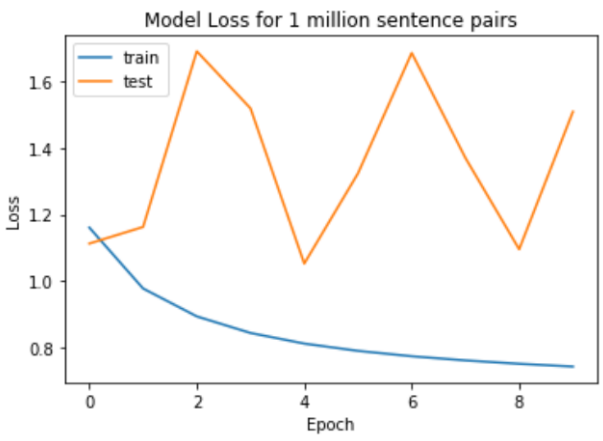

Figure 8.5: Model Loss on 1M sentence pairs using generators

These graphs show different loss and accuracy trends when compared to previously presented graphs (using `fit()` function to train, described in section 6.2). With a reasonable batch size, the previous training method took large steps towards convergence, and hence the steps showed no noise in the learning graphs. On the other hand, the above graphs are obtained while training on a much bigger data set i.e. 1M sentence pairs. For bigger data, generators were used to fit the model, so as to avoid memory errors. The learning curves for training set in the above graphs show advancements throughout the training process. The curve nature of both loss and accuracy are as expected. But, the test set learning rate is of nearly stochastic nature. This is due to the value assigned to `validation_steps` argument of the `fit_generator()` function while training. The generator object `validation_data` was divided into equal batches of size 32, which made the gradient descent behave like a mini-batch SGD, making the steps noisy.

8.5 Effect of Sentence length on BLEU score

The model, when trained on 1M sentence pairs, provided a decent BLEU score of 0.33. However, another aspect to think about would be the length of sentences. As mentioned before, the data set consists of roughly 50% sentences with length less than 10. Given below is a detailed statistics according to sentence lengths:

Group	Sentence length	No. of sentences	Training BLEU score	Test BLEU score
1	0-10	817,780	0.35	0.34
2	10-20	260,292	0.33	0.32
3	20-30	102,960	0.33	0.33
4	>30	123,532	-	-

Table 8.1: BLEU scores of sentences grouped according to lengths

Group 4 BLEU scores are blank because the model was trained on a maximum length of 30 words. The results show a slightly higher score for Group 1 sentences. This group contains a large number of short sentences, and thus, the n-gram precision is basically just a *unigram* precision for most of the sentences. So, group 1 sentences get the advantage of several 1 or 2 word-sentences. Although, the other groups have very close scores, which means the BLEU score is not largely affected by increasing sentence length. The thing to note here is, these

BLEU scores are calculated on a 4-gram precision rule. And when groups of four words are considered for calculations, the effects are visible if the number of sentences is less. In the above table, all groups can be seen having more than 100K sentences. In such a large data size, small changes like 0.35 to 0.33 is significant.

Additionally, the unigram scores were calculated for the same groups of sentences. These scores for the first three training groups are **0.61**, **0.55** and **0.36**. This shows how group 1 has the advantage of a large number of short sentences and thus end up in a high unigram BLEU score. This score shows a clear decreasing trend as the sentence length increases, which is logically correct as well. The n-gram BLEU score variations for each group is shown below:

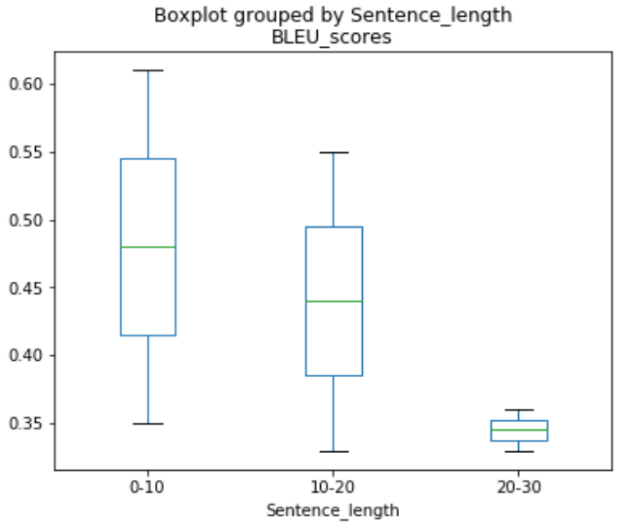

Figure 8.6: BLEU score variations for Training set(unigram to 4-gram)

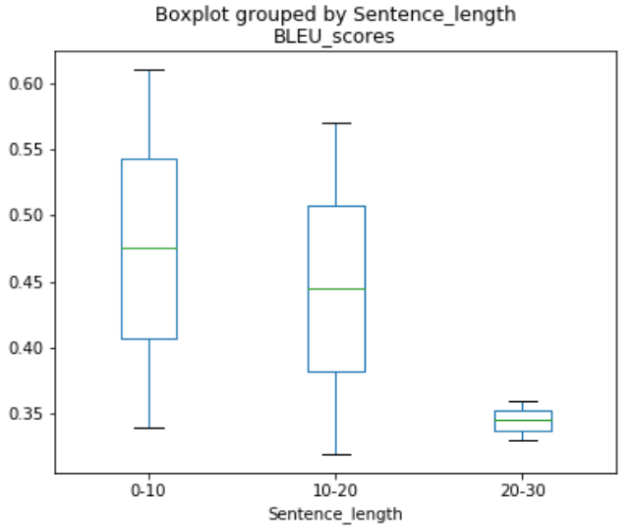

Figure 8.7: BLEU score variations for Test set(unigram to 4-gram)

BLEU scores are shown on the y-axis. It is evident from the above graphs that BLEU scores vary largely between 1-gram and 4-gram tests. But when the sentence length keeps increasing, after a certain point, the 4-gram precision shows less variation. Thus, it can be inferred that sentence length is the most important factor in determining the n-gram precision BLEU scores.

8.6 Comparison with Google Translations

As part of a preliminary evaluation of the model, the translation accuracies were checked on few shorter sentences. The group of sentences with less than 10 words, were further broken

down into categories of single word, two words, three, four, five and so on. For each of these categories, a sample of about 15-20 sentences were picked, and the translations from the model were compared with the translations from Google. Some of the examples are presented below:

Source	**Target**	**Model Translation**	**Google Translation**
सारणी सूचना	table information	table information	table notification
अंतराफलक दर्शक	interface viewer	interface viewer	interface viewer
प्रवाहित करनेयोग्य विषयवस्तु	streamable content	streamable content	streaming content
चिड़ी की तिक्की	three of clubs	the three of clubs	Bird's tikki
एक्सेर्साइसर पहुंचनीयता अन्वेषक	acceciser accessibility explorer	acceciser accessibility explorer	Excursor Accessibility Investigator
एक खाली फाउन्डेशन स्लाट	an empty foundation slot	an empty foundation slot	a blank foundation slot
नज़रअंदाज़ करें फ़ाइल अनुशंसित	ignore cvsrc file recommended	ignore cvsrc file recommended	ignore file recommended
कनेक्ट को रिमोट लक्ष्य	connect to remote target	connect to a remote target	remote target to connect
हाइलाइट किया गया भराई का रंग और पारदर्शिता	the color and opacity of the highlight fill	the color and opacity of the highlight as	highlighted color color and transparency
इस समय चुने गए एक्सेसेबेल से काम लेने के लिए अंतर्क्रियात्मक कन्सोल	interactive console for manipulating currently selected accessible	interactive console for manipulating currently selected	Interactive Console To Work With Accessible At This Time
फोन गुणवत्ता में सुधार लाने के लिए प्रतिध्वनि रद्दीकरण का उपयोग करें	use echo cancellation to improve call quality	use echo cancellation to improve call quality	use the echo cancellation to improve phone quality
क्या पॉप अप अधिसूचना दिखानी है जब संपर्क ऑफलाइन होता है	whether to show a popup notification when a contact goes offline	whether to show a popup notification when a contact goes	whether popup notifications are displayed when the contact is offline
तुम्हारा पिता इनकार करते हैं और तेरे नाम से मना	deny thy father and refuse thy name	deny thy father and refuse thy name	your father denies and refuses you by your name

Table 8.2: Sample Model Translations compared to Google Translations

The above examples show fairly good translations in both the cases. For a better understanding of the translation accuracies, BLEU scores were calculated on the samples of sentences in each

category, i.e. 1 word, 2 words... up to 5 and more words. However, the sample set for Google translations consisted of only about 15-20 sentences. The BLEU scores were calculated using target sentences as the reference, and the predictions from Google and the model were the hypotheses. The scores are shown the graphs below:

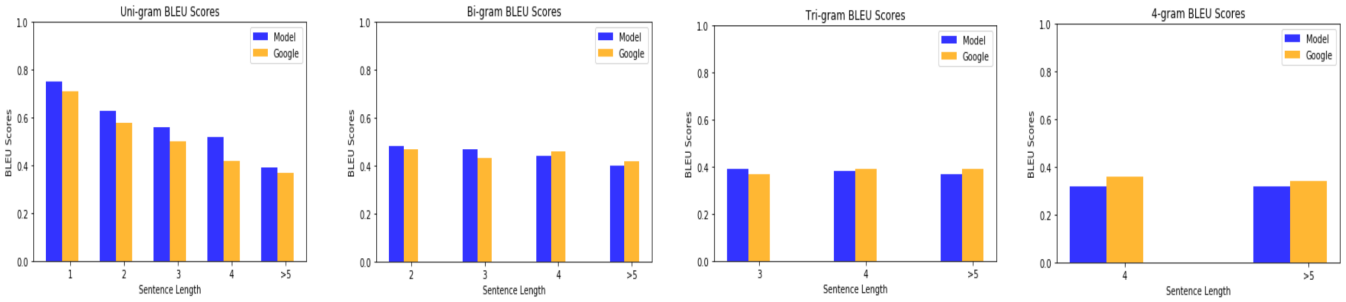

Figure 8.8: n-gram BLEU scores comparison between Model translations and Google Translate results

We see a clear decreasing trend in the first graph, which shows the unigram BLEU scores. For the group of sentences with only one word, the unigram score for model translations is as high as 0.75! As the sentence length increases, there's a decrease in the n-gram precision scores, both in case of model translations, as well as translations from Google. The above graphs show that the model has a performance comparable to that of "Google Translate" in several cases, and in some cases, even better!

8.7 Informal Evaluation

Once the model was trained and tested on the available data, an additional test on a different data set was carried out. A bunch of sentence pairs were collected from a portal called *Many-things.org*[1] and the model was tested on this new, unseen data. Some example translations from this data set are shown below:

Source: वह अभिनेता है

Target: `he is an actor`

Translation: `he is an actor`

[1] http://www.manythings.org/bilingual/hin/

Source: ऊपर तक चढ़ जाओ

Target: `climb to the top`

Translation: `move up`

Source: मैं पुराने मंदिरों को देखने जाने का शौकीन हूँ

Target: `my hobby is visiting old temples`

Translation: `old I see this`

Source: उसने अपने दुख को अपनी मुस्कुराहट के पीछे छिपा लिया

Target: `he hid his sadness behind a smile`

Translation: `he always his back`

Source: ताज महल विश्व के सात आश्चर्यों में से एक है

Target: `Taj Mahal is one of the seven wonders of the world`

Translation: `taj mahal has the seven`

As we can see, some words in the translations are remotely related to those in the target, but the translations are not correct. A below average performance was observed with the new data, specially with longer sentences, as most of the words were unseen for the model, and hence, were tagged as '*out-of-vocabulary*'(OOV) terms. The OOV terms were displayed as a blank space, which is why most of the translations shown above are incorrect. This step was necessary to understand the limitations of the model. It can be inferred that if the training corpus contained words with a wider range of contexts (such as, entertainment, social media etc.), there would be more correlation in the input words and the training vocabulary, thus reducing the OOV tags and generating better translations. More details about this issue are discussed under 'Future Enhancements' section.

Chapter 9

Conclusion

9.1 Summary of Achievements

The focus of this project was to demonstrate efficient training methodologies for Deep Neural Networks. While this is a vast subject area, fairly applicable to a number of fields, text data, specifically translation tasks have gained immense popularity in the last decade. This project showcases various deep learning techniques, their implementation and fairly good results. Previously, NMT systems have been built on Hindi and English languages, and some great results in terms of BLEU scores have been achieved [18, 26, 27, 28]. What makes this project different from the previously developed systems is that the initial pre-processing step uses no built-in libraries. Tokenizing, cleaning, indexing, creating input and output matrices, all these steps were done manually, thereby observing changes at each step. The usual approach for doing this is to use `Tokenizers` already available as libraries in `python`. Also, for embedding, pre-trained vectors like *Word2Vec, Glove* etc. are used in many researches. But in this project, pre-trained weights were not used. Instead the random-initialization and learning process was followed.

In terms of results, the BLEU scores achieved are at par with the previously built NMT systems. The best results were observed in the *IIT Bombay* research [18], which was further sub-tasked and discussed in '*XMU NMT* for WAT 2017'[29]. Comparable results were obtained by this model. However, previous results cannot be directly challenged, as there were some data

reductions(sentence length limit, outliers removal etc.) and that make the data sizes different from previously mentioned researches. Also, while sampling with roughly 25 sentences, it was found that the model results are comparable to that of Google Translate, which is a significant achievement. Of course the data contains a large number of short sentences (less than 5 words), which contributes to a higher BLEU score. At this stage, the model may not have a great performance on long sentences, but the baseline model is enough to understand the technicalities, and with some amount of tuning, better performances can be achieved with longer sentences too. Also, the model presents a clean implementation of Beam search algorithm, and there is a scope of obtaining the top 5 translations (if available). This is helpful in cases where the beam search runs on medium length sentences (roughly 10 words). If the probabilities have errors, the second or third probable translation could be actually better than the first one. If there is human intervention in recording these observations, a more accurate translation can be picked from the list. An example is shown below:

Source: मैं आपकी मदद कर सकता हूँ

Target: `I can help you`

Translation: `(' can help i', 0.01404), (' can can help you', 0.00501), (' can help you can get', 0.00361), (' i can help you', 0.00356)`

As we see, the output is a list of possible translations, along with the likelihoods. In this case, the fourth translation is the best. This is an advantage only if there is a human involvement in recording these results, someone who has a knowledge of both languages. To sum up, this model is a simple and straightforward implementation of the Sequence to sequence architecture, which works well for a baseline model. Going forward, there is always scope for tuning and making improvements in terms of implementation strategies as well as translation accuracies.

9.2 Challenges

Building an NMT system for two morphologically different languages requires extensive pre-processing and training intuitions. Hindi and English are structurally different and training the model to understand those differences and make accurate predictions, was indeed a challenge.

9.2.1 Pre-processing

One of the bigger challenges in this project was cleaning the data. As this happens to be a **huge corpus** of about 1.49 million sentence pairs, studying the data to gain a better understanding was tedious. The data, collected from multiple sources, was heterogeneous. For example, text from **different languages** (other than Hindi and English) were present as part of the data. This was difficult to identify initially, and resulted in several inconsistencies while encoding. **Distorted data** was also a considerable part of the corpus. For example, in the corpus, the word *'Clean'* was represented as `(C _ lean)`. First of all, without a human intervention, particularly a person who has no idea of the language, it would be impossible to understand that this is a single word (there were spaces in between characters). It was easy to just get rid of the special characters, but this was only one of the distortions, which was observed and addressed. The data size being huge, there could be possibilities of multiple such scenarios, and these kind of issues needed customized *regular expressions* because just by removing the '_', the word would still not be meaningful, and more importantly, different from the corresponding Hindi word. It was hence a challenge, to remove distortions, and yet preserve the meaning of the word. This is similar to the **Wordpiece Model**(WPM) mentioned in Schuster *et al.*[30]. Another issue with the data was, the maximum **sentence lengths** were found to be 2178 for English, and 2068 for Hindi. This logically seems incorrect. Also, the average lengths for English and Hindi sentences were found to be around 14 and 15 respectively. While creating encoder and decoder matrices, the second dimension was the 'maximum length of sentences' in that language corpus. When all sentences were considered, matrices with a dimension of about 2000, and with almost all sentences with lengths in the range 14-15, were largely sparse. Consequently, this wasted a lot of computational resources as well as time. Later, these sentences were found to be only 4 in the entire corpus. They were then tagged as *'outliers'* and removed.

9.2.2 Resource restrictions

High performance GPU machines were made available for such deep learning tasks. Text data usually requires more time for training. This model in particular, required around 44

hours to train completely on 1 million sentence pairs. Although, when such resource-intensive applications are executed on these machines, in some cases, if there is any connection error at some point during the training, the entire program fails and the process gets killed. Also, the resources were limited to a usage time of only 60 hours. While this should be sufficient, in some cases, the training may need more time than this. An alternative could be to parallelize the training, but it is not advisable to do so when the model has a simple architecture as this one. Unless the model is structurally very deep, parallelization would result in unnecessary resource consumption. Handling this trade-off was difficult because the training parameters had to be tuned to wrap up the training within 60 hours.

9.3 Limitations

The model performs fairly well on average length sentences i.e. less than 15 words because the maximum length considered for training is 30, in both languages. With the increase in sentence lengths, the translation accuracies deteriorate. This happens because about 50% of training corpus contains short sentences (less than 10 words). Thus, the model shows near-perfect translations for similar sentence lengths, but greater than that, it goes downhill, due to lack of training on longer sentences. Consequently, the BLEU score also shows this deterioration as sentences become longer.

Another limitation of this model is related to sampling the probabilities. As mentioned earlier, Beam search algorithm was used to sample probabilities from the model predictions. The sampling technique is designed for a beam width of 2. This value could be increased, but beam search is computationally expensive, particularly in terms of duration. Each step of the algorithm requires recursive calls to the search algorithm. This is a time taking process and hence, the beam width was finalized to be 2.

The model is trained on a large vocabulary, but most of the words in the corpus have similar contexts. For example, a number of sentences are system generated messages, which makes words like *'files', 'folder', 'delete', 'window', 'save'* etc. occur quite frequently. If an

input sentence has entirely different subject area and the words in use are totally unknown, the model would treat them as *'out-of-vocabulary'* words, and would not translate them. This is a limitation because the corpus used here, does not cover a wide range of areas from where words can be expected in a user input.

9.4 Future Enhancements

This model is a simple baseline implementation of an NMT system. There is still scope for improvements. Some of the probable enhancements are discussed below:

1. The **maximum length of sentences** considered for training, can be more than 30. This would probably increase the performance in case of longer sentence inputs.

2. Secondly, for sampling predictions, a **beam width greater** than 2 can be used for beam search. This would be computationally expensive, but might produce better results for longer sentences. However, that still is a trade-off and decisions like these are entirely based on programmer's discretion.

3. A very crucial step could be to **extend the data set** from a more common source like *Twitter*, or other social media platforms. The reason this could increase performance is that social media platforms contain the most commonly used words by people currently. If the model can be trained with those enhancements to the vocabulary, the most commonly used words will not be tagged as "*out-of-vocabulary*", unlike this data set, which contains all conventional words. These words are grammatically correct, but are less commonly used in day-to-day communications. Therefore, social media data could train the model for more realistic translations.

4. **Pre-trained word embeddings** like *Word2Vec* and *Glove* can be used to train the model. The results can be observed, and if there are significant improvements in BLEU scores, it can be claimed that these pre-trained embeddings are better for training NMT models than using random weight initializations.

5. **NE recognition** and **POS tagging** could be another crucial enhancement to the model. When humans read a sentence, they tend to emphasize the verbs and other important entities like proper nouns, to make translations more sensible. The model could be trained to do so using *Part-Of-Speech tagging* or *Named Entity Recognition*. This is a significant limitation of the model, and the example below shows why this could be crucial:
   Source: जार्ज वाशिंगटन संयुक्त राज्य अमेरिका के पहले राष्ट्रपति थे
   Target: `George Washington was the first president of the Unites States of America`
   Translation: `north american president of capital`
   The model does not recognize the person's name and makes incorrect translation. This sentence has several proper nouns, and they would be less frequent in the corpus. If NE recognition could be done for Hindi words, the model would be able to recognize the person's name and make proper predictions. Thus, POS tagging and NE recognition could be valuable enhancements.

6. An **Attention** based NMT system, as suggested in Manning *et al.*[31], can be built. By combining the information from target state and the source context vector, an *attention* hidden state can be produced, which can further be passed on to the softmax layer. The model suggested in [31] has performed better with long sentences, and thus can be a significant improvement.

7. An important enhancement to this project could be building an **interactive web application**, that could take a Hindi sentence as input from the user, and display the corresponding English translation. This is more on the lines of how "Google Translate" behaves, or even few other language translators, currently available in the market.

9.5 Personal Reflection

This project has been an incredible learning experience for me. Although I have been fluent in both these languages, and also had an idea of the linguistic differences, I thoroughly enjoyed

digging deeper and exploring the technical aspects of these languages. When it comes to machine learning, it is imperative to understand how to process the text data so that it can be fed to a machine, make it learn various features, and then be able to make predictions. Training a model on sequential data involved a great deal of understanding about how LSTMs work. It was indeed a black box for me at the start of this project. I now have a better understanding of how the LSTM states are used, and how they enable better learning for sequential data. As an initial test, I also trained a model to translate from German to English language. It was easier to program a model on two similar languages, and to understand the training specific challenges. This provided some great insights and I was able to tune the training parameters in a better way for diving into two completely different languages, both in terms of scripts, and structure. Also, exploring strategies like Beam Search algorithm, calculating BLEU scores, and to be able to successfully implement all of them in a single application, was indeed a great learning experience.

For any project, the ground work done before starting is one of the most important steps. I came across several amazing research papers and journals while doing the 'Literature Review' for this project. Some related literary works such as *The IIT Bombay English-Hindi Parallel Corpus*[18] and Agrawal *et al.* [32] provided great motivation and ideas on working with similar data. Reading research articles and journals helped a great deal in expanding the Deep learning knowledge.

To sum up, I would like to appreciate how incredibly challenging the field of Machine Learning and Artificial Intelligence is, and I feel fortunate to have experienced a drop in this vast ocean! 'Training machines to behave like humans' always sounds fascinating, but to have actually experienced taking the first step towards it, has been incredible. I hope and wish to continue learning and contributing as much as I can, to the Machine Learning community.

Bibliography

[1] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[2] "Understanding lstm networks." http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[3] J. Hutchins, "Example-based machine translation: a review and commentary," *Machine Translation*, vol. 19, pp. 197–211, Dec 2005.

[4] L. Dugast, J. Senellart, and P. Koehn, "Statistical post-editing on systran's rule-based translation system," in *Proceedings of the Second Workshop on Statistical Machine Translation*, pp. 220–223, Association for Computational Linguistics, 2007.

[5] Y. Al-Onaizan, J. Curin, M. Jahr, K. Knight, J. Lafferty, D. Melamed, F.-J. Och, D. Purdy, N. A. Smith, and D. Yarowsky, "Statistical machine translation," in *Final Report, JHU Summer Workshop*, vol. 30, 1999.

[6] L. Bentivogli, A. Bisazza, M. Cettolo, and M. Federico, "Neural versus phrase-based machine translation quality: a case study," *arXiv preprint arXiv:1608.04631*, 2016.

[7] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, R. Cattoni, and M. Federico, "The iwslt 2015 evaluation campaign," in *IWSLT 2015, International Workshop on Spoken Language Translation*, 2015.

[8] M.-T. Luong and C. D. Manning, "Stanford neural machine translation systems for spoken language domains," in *Proceedings of the International Workshop on Spoken Language Translation*, pp. 76–79, 2015.

[9] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006. PMID: 16764513.

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[11] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709, 2013.

[12] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[15] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] A. Kunchukuttan, P. Mehta, and P. Bhattacharyya, "The iit bombay english-hindi parallel corpus," *arXiv preprint arXiv:1710.02855*, 2017.

[19] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

[20] S. Thurner, R. Hanel, B. Liu, and B. Corominas-Murtra, "Understanding zipf's law of word frequencies through sample-space collapse in sentence formation," *Journal of the Royal Society Interface*, vol. 12, no. 108, p. 20150330, 2015.

[21] K. Revanuru, K. Turlapaty, and S. Rao, "Neural machine translation of indian languages," in *Proceedings of the 10th Annual ACM India Compute Conference on ZZZ*, pp. 11–20, ACM, 2017.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] C. Tillmann and H. Ney, "Word reordering and a dynamic programming beam search algorithm for statistical machine translation," *Computational linguistics*, vol. 29, no. 1, pp. 97–133, 2003.

[24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

[25] J. Pouget-Abadie, D. Bahdanau, B. Van Merrienboer, K. Cho, and Y. Bengio, "Overcoming the curse of sentence length for neural machine translation using automatic segmentation," *arXiv preprint arXiv:1409.1257*, 2014.

[26] S. Singh, R. Panjwani, A. Kunchukuttan, and P. Bhattacharyya, "Comparing recurrent and convolutional architectures for english-hindi neural machine translation," in *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pp. 167–170, 2017.

[27] S. Parida and O. Bojar, "Translating short segments with nmt: A case study in english-to-hindi," 2018.

[28] T. Banerjee and P. Bhattacharyya, "Meaningless yet meaningful: Morphology grounded subword-level nmt," in *Proceedings of the Second Workshop on Subword/Character LEvel Models*, pp. 55–60, 2018.

[29] B. Wang, Z. Tan, J. Hu, Y. Chen, *et al.*, "Xmu neural machine translation systems for wat 2017," in *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pp. 95–98, 2017.

[30] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[31] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[32] R. Agrawal and D. M. Sharma, "Experiments on different recurrent neural networks for english-hindi machine translation,"