NEURAL MACHINE TRANSLATION USING DEEP LEARNING TECHNIQUES

AUTHOR: FAHEEM MOHAMMED ABDUL

DEGREE: MASTER OF DATA SCIENCE AND ANALYTICS

DEPATMENT: MASTER OF SCIENCE (MSc)

UNIVERSITY: RYERSON UNIVERSITY

Year: 2022

ABSTRACT

Deep Learning has grown a lot over the years and showed much significant achievement, which is tough when humans do. It has changed machine translation from a statistical and traditional approach to a neural network-based system. In this project, we will explore and build such neural network-based state-of-the-art machine translation models. We developed a deep learning pipeline using various new techniques research and extracted from the literature review. We push the performance of the model by decreasing the logarithmic loss via an iterative approach. We maximize the BLEU score, a benchmark for deciding the efficiency of machine translations between any two languages. These methods will overcome many challenges the traditional Machine Translation models face, such as keeping the semantic meaning, sarcasm, and orientation of sentences.

ACKNOWLEDGEMENTS

A Special thanks to my Professor Davood Pirayesh Neghab for his inspiration, guidance, and advice during the Deep Learning Course, which helped in project development. I am also grateful to all my family members, friends who support me during the past 2021 - 22.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Overview

The growth in technology has become a super factor in connecting the world more deeply. The physical distance between two places is no longer a blockage as it once was, allowing people from different ethnic groups, regions, country to connect freely and frequently. Many social media platforms have made this process very easy, and people are now more interested to learn about other cultures. It is creating much scope for urban and developed people to connect with rural and undeveloped areas of the world and raise voice on behalf of them. Many viral videos from different corners of the world create many buzzes about inequality, social injustice, and crime. The growth in technology has also brought many joys to many faces as people can now explore their future career choices, traveling plans, and many more. However, while distance may not divide us like it once did, the vast number of languages spoken worldwide still creating universal communication difficult. There are over 6500 significant languages spoken globally by billions of people, creating a considerable communication gap. Several ethnic, state, national, and traditional languages are expressed even inside a country, state, or district. Sometimes they are not even connected in any sense and are utterly different from each other. There are numerous families of languages spoken worldwide where all the dialects and pronunciations are very different. Some of them are Sino-Tibetan, Indo-European, Afro-Asiatic, Oto-Manguean, Austronesian, and many more. Translators have been a pivotal member to reduce this communication gap between multi-language communication gap throughout the centuries. From the era of ancient rulers to this modern world, translators and scholars have made this language translation very easy from time to time. Nevertheless, as technology has grown a lot, the demand to reduce this communication gap has been increased a lot [1].

Machine Translation has become a revolution in recent years to translate different languages. It is a subfield of computational linguistics that investigates software/technologies to translate text or speech from one language to another language. It has been in use since the early '80s when there was a massive revolution in technology. This method uses various methods such as statistical and rule-based techniques to translate one language to another language, which

seldom produces perfect translation keeping the semantic meaning and ideas. It has its own set of limitations.

Despite its use cases, there is a scope of exploration for machine translation using various new approaches. One of the approaches is Neural Machine Translation (NMT) [16]. *Neural Machine Translation* is a state-of-the-art machine translation approach that uses neural network-based algorithms to predict the likelihood of a sequence of words. It is suitable for small sentences, phrases, text fragments, or the entire document. NMT is a different approach to solve the machine translation problems more effectively and robustly. It uses deep learning-based algorithms such as Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), GRU. It trains models based on prior knowledge of data (like humans learn based on prior knowledge), then uses the learning to predict the output efficiently. In its last few years, it becomes the benchmark for machine translation [14].

The project's main objective is to provide a highly effective algorithm that handles machine translation effectively within the given time frame. Since the state-of-the-art algorithms have codes that are not publicly available completely (partial availability), development on prior research will make this solution feasible, based on the proposal given in the paper.

For the training purpose, the model will be trained on one/many sources language (French, Italian, German) to English (target language) language on sentences which can have word range from 1 to 30. The baseline model will be a Recurrent Neural Network based Long Short-term memory (LSTM) based model. A many-to-many LSTM model is going to be used, which at input time step takes source sentence as input, and an output time step will produce target sentence tokens. The other algorithms will be the state-of-the-art attention network [12], as it displays the best performance for translation so far and is used as the standard for additional research papers.

The models will be implemented using the "tatoeba" project dataset. The data has many source languages (input) to English language (output), sentence to sentence translation which can be used for multilingual machine translation.

The models will be validated using log loss scores for both training and validation data. Common cases like overfitting and underfitting will also be taken care of during the training process. The performance metric used for most algorithms is the BLUE score (Bilingual evaluation understudy) to measure machine translation quality based on human translation. The

BLUE score provides the correct form of translation between the source and target sentence, but it does not measure overall quality. As the sentence length will be well within the range, BLUE will be perfect for validation accuracy.

A prediction pipeline will be generated, taking the source sentence as input, automatically detecting the language, and providing an English sentence (target). Latency will be a significant factor to consider as the translation process should be fast and agile.

1.2 Problem Statement

NMT has become a benchmark in machine translation, and even Google Translate is using NMT as its backbone of translation. Though it has achieved a remarkable feat in recent years, there are still many scopes to improve the models using various new approaches. The problem statement is building a neural machine translation model using various deep learning techniques that can effectively and robustly predict the sentences from different languages. Attaining high accuracy in machine translation is a difficult task; language is more than a collection of words. Numerous things need to be considered: grammar, rules, cultural references, irony, personification, and sarcasm.

1.3 Objective

The project aims to build a deep learning neural network-based neural machine translation model, which allows for a sentence of a particular language to be given to the model and return a prediction of the same sentence in the English language with a confidence score of the result. The below are the objectives that will be taken care of during the project.

Objective 1: Study the existing techniques and methodologies already been published in the public forum

Objective 2: Review the literature of the published papers within the score of the project

Objective 3: Build a neural machine translation pipeline end to end using open-source deep learning frameworks such as Keras and TensorFlow

Objective 4: Apply an approach and method to validate the created model and generate a scope of improvement in the performance. Conduct testing to evaluate the output using appropriate performance metrics

2. Literature Review

2.1 Overview

Machine Translation (NMT) has become an essential part of Natural Language Processing (NLP) field due to some breakthrough research has been done in recent years, benefiting computer scientists, linguists, and sociologist in an effective manner. Many research papers have proposed numerous methods and approaches to make machine translation more effective and robust in the last few decades. It is being used in many fields, such as multilinguistic apps, speech detectors, and healthcare systems. Despite its use cases, there is a scope of exploration for machine translation using various new approaches. One of the approaches is Neural Machine Translation (NMT). This project will provide an NMT model using multiple deep learning techniques such as long short-term memory (LSTM) and state-of-the-art models such as the encoder-decoder and attention models. The TensorFlow framework will implement such a state-of-the-art model, and an improvement scope will also be generated. A machine learning approach will also be included for a baseline model.

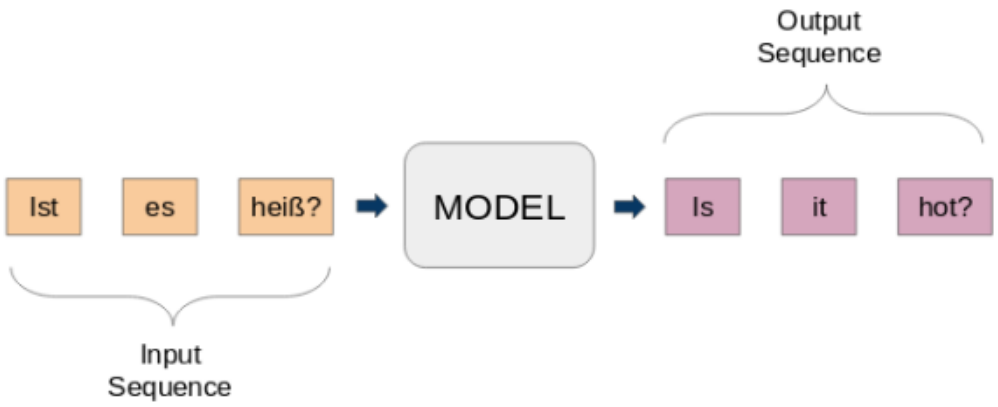


Figure 2.1: Figure showing the traditional machine learning process where the translation is done based on the word without keeping semantic meaning and combination

Before the advancement of deep learning and neural networks, less sophisticated machine translation methods were used. This approach used a typical step called feature extraction from

raw data to understand the relationship and semantic meaning between the words and sentences. The visualization of this specific step has shown below in figure 2.1.

The first documented use of quantitative and numerical techniques for machine translation can date back to 1947, during the second world war in Great Britain, deciphering encrypted messages sent by Germany. In 1954 USA managed to translate about 254 words from Russian to English, demonstrating the first usage of language translation using a dictionary-based model [1]. Being a dictionary-based model, it could not capture the description due to the problem of word arrangement. Seeing this as 1966, the ALPAC standing for Automatic Language Translation Processing Advisory Committee, reported that human translators do a better job than machine translation, and such accuracy cannot be achieved with existing methods [2]. An experiment was designed to lay the foundations for a standard procedure for measuring quality translations, but it was too laborious, too out of standards, and unreliable. The measurement procedure focuses on two significant characteristics of translation: its eligibility and its fidelity to the sense of the original text.

2.2 Evolution

The primary evolution stage in machine translation was known as Rule-Based Machine Translation. As opposed to the current state-of-the-art techniques based on mathematics, this was a system based on manually written rules. It was based mainly on linguistic representations and theories. A solution was developed by the company Systran that signed contracts with the US Air Force and Xerox, eventually making their software available on Windows for users [12]. Apertium was another platform that was a leader in RBMT made available on various platforms [13]. This had multiple drawbacks, the most important being in-depth knowledge and understanding of both languages. The paper addresses the problem of machine translation (MT) of domain-specific texts for which extensive data were not available at that time. They focused on the IT domain and translation from English to Portuguese translation and performed various strategies to improve system performance. There were two baseline models set up, the first one used the large dataset of scope of domain data, and the second used only a small dataset of actual domain data. Adding a domain-specific bilingual lexicon to the training dataset significantly improved the performance of both a hybrid MT system and a PBSMT system.

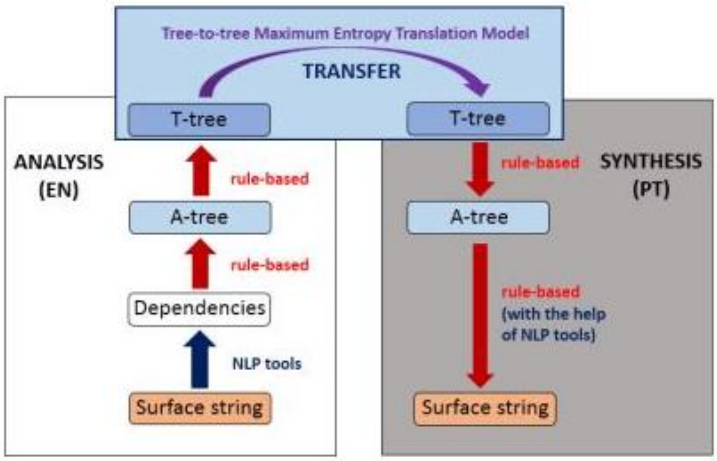


Figure 2.2: Model schema of TectoMT

It further led to the secondary evolution stage, which is known as Statistical Machine Translation. These methods work on the concept of assigning weights to every pair of strings based on the probability that a human will give a specific line of the target language, given a series of the source language. Hence every possible pair of source and target string is defined and assigned weights. It was a significant drawback as the computational resources required for multiple permutations as such are huge. It did automate the process but reduced the accuracy of translation in the process. It could not do complex translations and displayed the need for an advanced method for automated translation [14]. This led to the birth of Neural based Machine Translation techniques.

2.3 Neural Machine Translation (NMT)

The initial use of neural networks in translation dates to continuous space models or connectionist models [3][4]. When the use of the GPU to train models using neural networks was realized, and the evolutionary concept of deep learning was discovered, it acted as a gamechanger for machine translation.

The RNN (Recurrent Neural Network) is a widely used network for translation. It was proved that LSTM, a model that uses the RNN architecture, led to greater accuracy and reduces time complexity. It also solved the artificial long-time-lag tasks that didn't have a solution using other RNN [8]. Further research led to the discovery of the RNN Encoder and Decoder model, which consists of two parts trained simultaneously so that the conditional probability of the target sequence given the source sequence is maximized. The first part is responsible for encoding the symbols to a fixed-length vector by mapping, whereas the second part decodes

the vector into a series of characters. This solves the problem of linguistic phrases in translation [9].

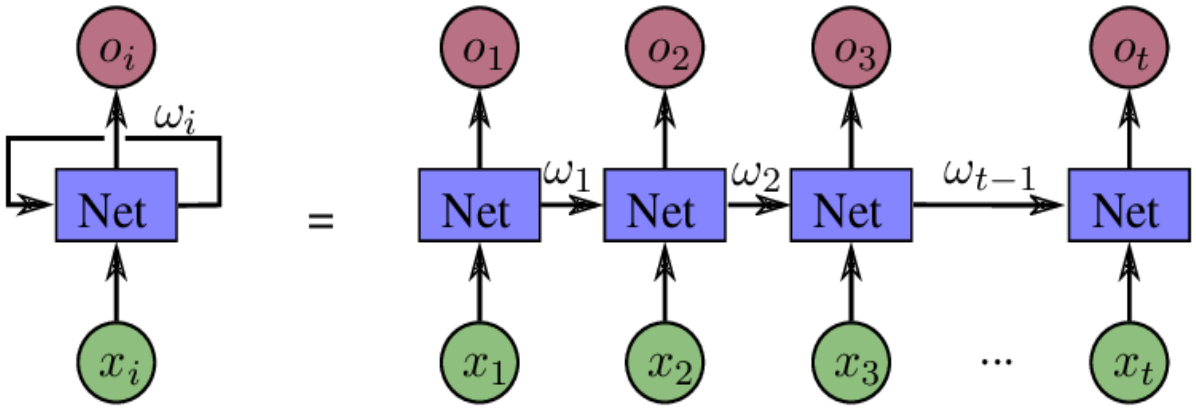


Figure 2.3: A general architecture of Recurrent Neural Network (RNN)

CNN (Convolution Neural Network) is another network used for translation. It was discovered to be an efficient alternative to the RNN where computations are parallelized, and the non-linearities are independent of the input length and fixed. They use an attention module and gated linear units that help outperform the LSTM algorithm that proved to have given the best performance out of all that were compared [10].

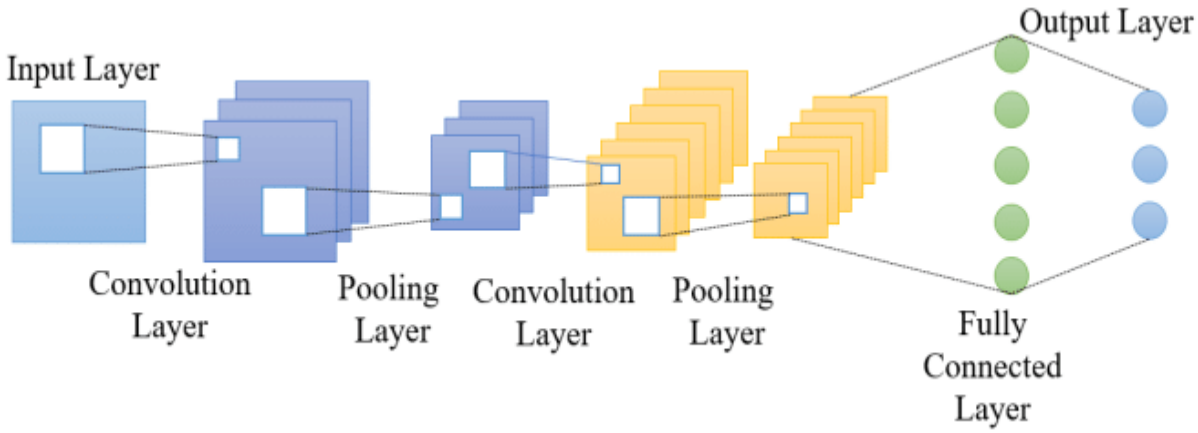


Figure 2.4: A Basic Convolutional Neural Network architecture

Further research was conducted, resulting in improved performance on existing models. RCTM, also known as the Recurrent Continuous Translation Models, is based on continuous translation using probability, showing a drastic improvement in sensitivity to syntax, meaning, and word order [5]. LSTM, also known as the Long Short-Term Memory, was used in neural

networks so that the sentence for translation would be mapped to a vector of fixed dimensions to be encoded, and then another LSTM would decode the sequence from the vector. This method would solve the problem of active and passive voice in machine translation, making it sensitive to word order [6]. It was found that using these techniques of encoding the text into a vector of fixed length is a reason for the reduced translation performance. Hence a solution was developed to detect those words in the sentence that would accurately predict the target word without forming fixed-length vectors [7].

The drawback in the RNN architecture was that the context vector had a size limitation. To tackle this problem, the concept of Attention was introduced to the traditional RNN based encoder-decoder architecture, which was used in a CNN-based model [10]. As the name suggests, Attention is a technique that mimics cognitive Attention, as it lays more stress on essential parts of the text, fading the rest out so that more computing power is used on the critical part of the data. This led to further research and discovery of the Transformer Neural Networks [11]. An evolutionary concept in encoder-decoder architecture by Google. It eliminates the use of RNN and CNN with proven faster and better performance in machine translation tasks. It also considers position encoding, which is a solution to a long-lasting problem in NLP. The improvisation is made so by reducing sequence computation by implementing parallelization. It eliminates most of the issues in language translation. It is also used as a performance standard for further research and developments.

Proving to be a state-of-the-art algorithm, this can be a good algorithm for the first half of the research. The latest developments that were made were trained upon mainly two datasets (WMT and IWSLT). The WMT dataset is a corpus file consisting of 4M sentences from various sources. The IWSLT is a dataset created by the automatic translation of various TEDX and TED talks consisting of 130K sentences. The dataset that can be used to train the transformer model can be WMT or something more specific that relates to the image synthesis dataset example: training on a book which is a guide to the flowers in the UK if our image dataset is the 102 flowers in the UK by Oxford leading to better results. The performance metric used for most algorithms is the BLUE score (Bilingual evaluation understudy) to measure machine translation quality based on human translation.

2.4 Attention All You Need

Advanced papers have been written regarding translation, the most famous being "Attention is all you need" [11] by researchers from Google, which eliminates the use of a Recurrent Neural Network architecture or a Convolutional Neural Network architecture and uses parallelization. This paper presents a new approach called the encoder-decoder model for sequence-to-sequence machine translation based on an attention-based mechanism. The advantage of the phenomenal architecture is,

- Reduces training time and complexity of the architecture
- Generate a higher BLEU score (1 score high) compared to other models becoming a state-of-the-art model
- It entirely proposes a novel architecture that worked without convolution or recurrent neural networks

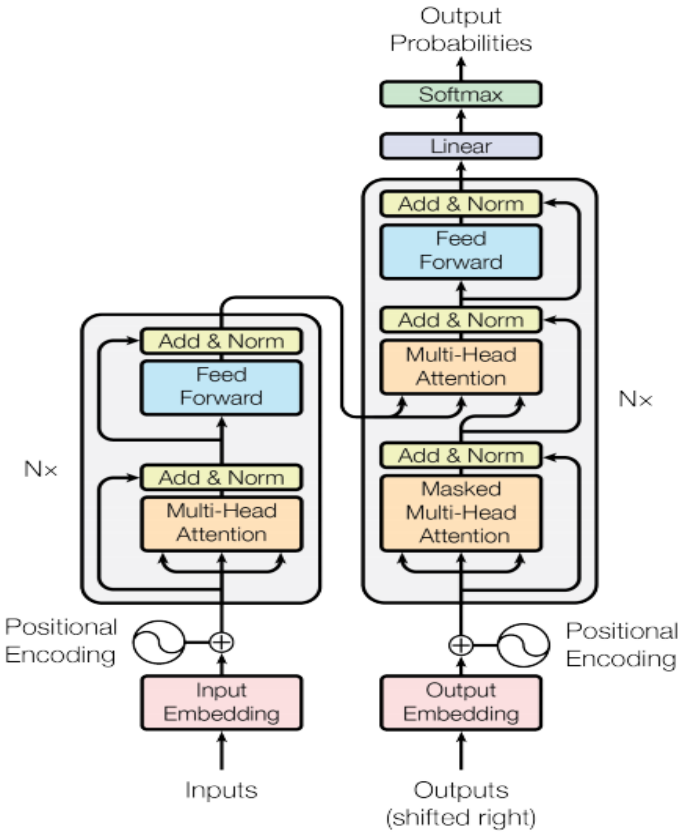


Figure 2.5: Transformer Architecture

2.4.1 Components

Below we are going to discuss the components on which transformer architecture is based upon.

2.4.1.1 Encoder

The encoder is composed of a stack of N=6 identical layers. Each layer has another two sub-layers. The sub-layers are a multi-head self-attention mechanism, a position-wise fully connected feed-forward neural network. A residual connection has been applied around each of the two sub-layers, followed by a layer normalization. The output of each sub-layer is $LayerNorm(x + sublayer(x))$, where $sublayer(x)$ is a function implemented by the sub-layer. All the embedding layers, as well as the sub-layers, produce output dimension $d_{model} = 512$ to facilitate these residual connections.

2.4.1.2 Decoder

The encoder is composed of a stack of N=6 identical layers. It has three sub-layers, including two as described in the encoder. The 3rd sub-layer performs multi-head Attention over the output of the encoder stack. Here also a residual connection is applied around each sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent posts. This masking, combined with the fact that the output embeddings are offset by one part, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

2.4.1.3 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the question, keys, values, and production are all vectors. A weighted sum of values, where each value is assigned, is computed by a compatibility function with the corresponding key as an output.

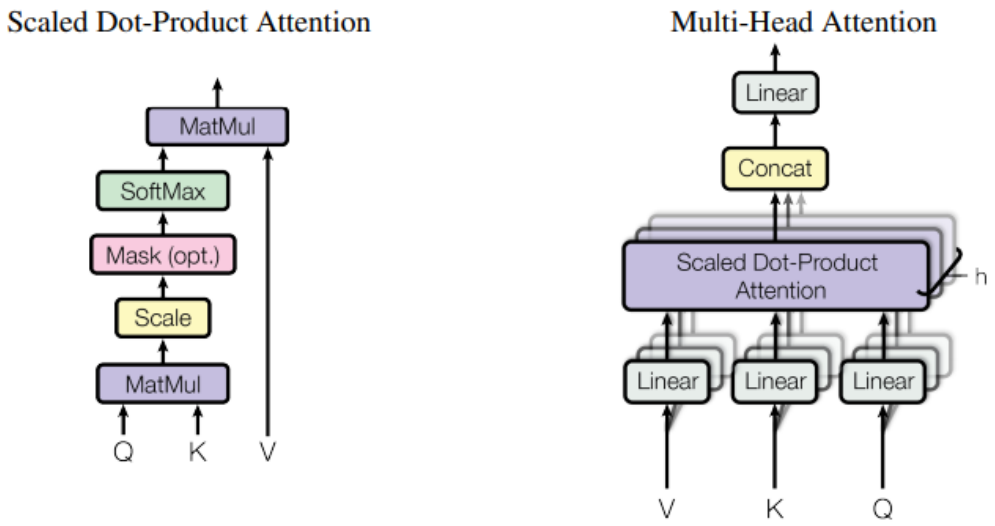


Figure 2.6: (left) Scaled Dot Product attention (Right) Multi-Head Attention consists of several attention layers running in parallel

2.4.1.4 Result

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Figure 2.7: Detail architecture variables for the WMT 2014 English to German task

On the WMT 2014 English to German translation task, the transformer model outperforms all previous models by a 2.0 BLEU score, with a new high score of 28.4. The model configuration is shown below.

3. Methods

3.1 Overview

To implement the proposed idea, there are some objectives to be met. This project is divided into several requirements which must be achieved during the model building process. These requirements have been gathered based on the MoSCoW rule. This rule sets conditions by stating the project must, should, or could achieve the detailed requirements by the end of the implementation stage. If the need is not satisfied with the priority, then the developed product will not fit the purpose. There are several high-priority goals which the project must achieve.

3.2 Dataset

The dataset we will use is taken from ManyThings.org, an open-source platform to use any multilingual dataset for machine translation. The text present on the website is Tab-delimited Bilingual Sentence Pairs. There are several bilingual sentence pairs present on the website. We will work with French to English translation as it consists of "190206" samples present in the dataset, which is relatively higher than other current datasets. The sample structure present in the file is shown in Figure 3.1.

```
Go.     Va !    CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #1158250 (Wittydev)
Go.     Marche. CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #8090732 (Micsmithel)
Go.     Bouge ! CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #9022935 (Micsmithel)
Hi.     Salut ! CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #509819 (Aiji)
Hi.     Salut.  CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #4320462 (gillux)
Run!    Cours ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #906331 (sacredceltic)
Run!    Courez !        CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #906332 (sacredceltic)
Run!    Prenez vos jambes à vos cous !  CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #2077449 (sacredceltic
Run!    File !  CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #2077454 (sacredceltic)
Run!    Filez ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #2077455 (sacredceltic)
Run!    Cours ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #4580779 (franlexcois)
Run!    Fuyez ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #7957917 (Micsmithel)
Run!    Fuyons !        CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #7957918 (Micsmithel)
```

Figure 3.1: A sample of the input machine translation text data

As we can see, the dataset has many junk texts other than the translations. To create a dataset to feed as an input to the model, we must pre-process the data and bifurcate the English and French sentences into two different sets.

3.2.1 Boxplot Analysis

The dataset has a "190206" number of sentences. These sentences vary from having just one word to several words present there. As deep learning models are highly prone to overfitting, the length of the sentences before feeding into the model should be the same. As this is impossible to achieve in the raw data, we encode and pad the data discussed later. But one thing can be done in the raw data: we can analyse the distribution of words present in the sentences and represent using a box plot.

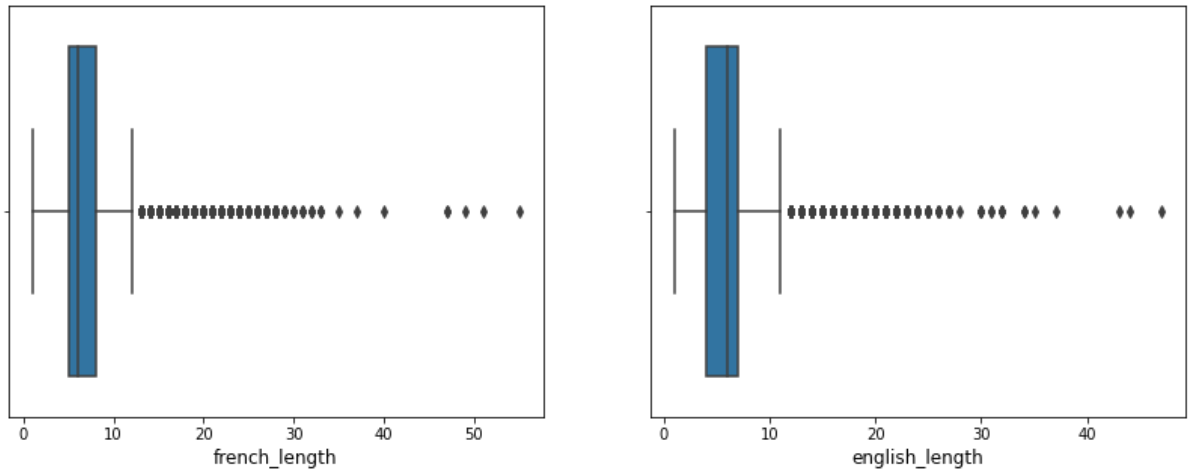


Figure 3.2: Box plot analysis for both French and English word counts present in each sentence

As we can see above, the upper threshold for the box plot for both sentences are between 12 to 14, which is generally an average number of words present in a sentence. But there are some sentences present that have as high as 60 words present in a single sentence.

3.2.2 Percentile Analysis

As it was unclear what should be the upper threshold for words for each language of sentences, we must analyze the data using percentile analysis. Percentile analysis can give us a clear picture of the word distribution present in the dataset.

	count	mean	std	min	25%	50%	75%	90%	95%	99%	99.9%	max
french_length	190206.0	6.637157	2.828913	1.0	5.0	6.0	8.0	10.0	12.0	16.0	22.0	55.0
english_length	190206.0	6.085413	2.527687	1.0	4.0	6.0	7.0	9.0	11.0	14.0	20.0	47.0

Figure 3.3: Percentile Analysis for each language word distribution

As we can see, there is a massive gap between 99.9th and 100th percentile for word distribution for both French and English sentences we must keep till 99.9th as the threshold for the number of words present in a sentence.

3.2.3 Teacher Forcing

Teacher forcing is a common technique used in Neural Machine Translation. It is a method for quickly and efficiently training recurrent neural network models that use the ground truth for a previous time step as input. There are sequence prediction models that use the output from the last time step $y(t-1)$ as input for the model at the current time step $x(t)$.

Let's consider the example in the data point –

```
In French: Nous nous sommes accordés sur un prix.
English Input to the model:    <start>   We agreed on a price
                                         |     |    | |   |
English Input from the model:            We agreed on a price    <end>
```

Figure 3.4: Teacher Forcing Method

If we observe the sentences, the start talking is getting mapped to the first word in your English sentence, the first word is mapped to the second word, the second-word map to the 3rd word, and so on, finally, the last one will be mapped to end a token. It means the word will be mapped to $i+1$th word. As we will be passing "english input" as an input to the decoder, and the outputs(predictions) of the decoder will be compared against the "english output." With this way of mapping, we can ensure that the model will predict the next word and calculate the loss accordingly.

3.3 Explaining LSTM

LSTM, also known as long short-term memory, is a type of recurrent neural network used for building. Deep learning models based on sequence-to-sequence modelling and machine translation.

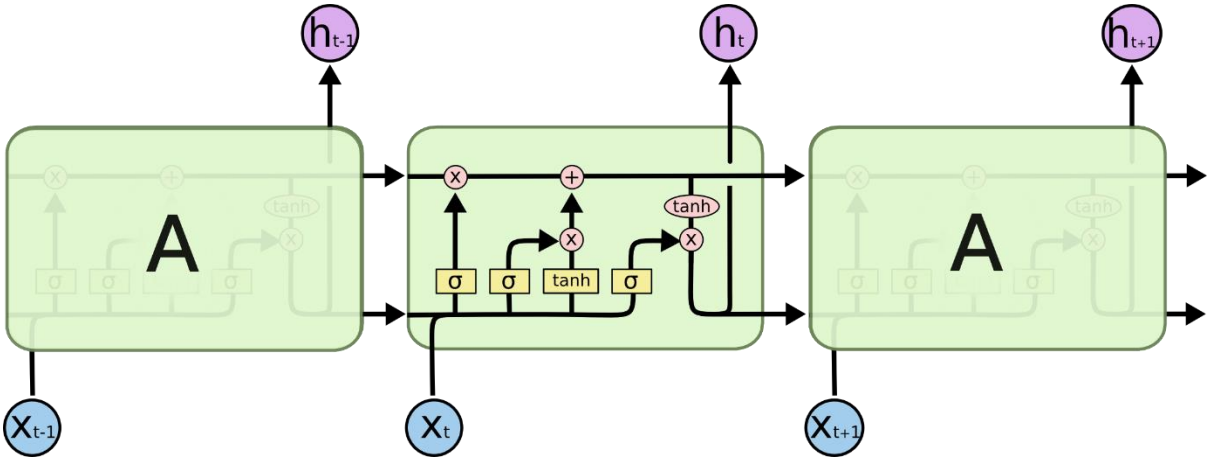


Figure 3.5: An LSTM core architecture

We risk vanishing gradient problems when we tend to work with a vast data set and multiple RNN layers. When training, an intense neural network gradient or derivative decreases exponentially as it propagates down the layer, known as vanishing gradient problems. The gradients are used to update the weights of a neural network; when the gradient vanishes, these weights will not get updated. In the worst-case scenario, it will completely stop the neural network from training. This vanishing gradient problem is a common issue in deep neural networks. So, to overcome this vanishing gradient problem in RNN, LSTM was introduced. LSTM is a modification to RNN hidden layer. LSTM can remember RNN's weights and the inputs over a very long period. In addition to the hidden state, cell state is passed down to the next block. The way LSTM works is that it can capture long dependencies. It can have memories of previous inputs for a very extended time duration.

3.3.1 Architecture

The key idea to the LSTM architecture is the cell state which is an addition from RNN structures. It looks like a conveyor belt and runs straight throughout the chain of architecture.

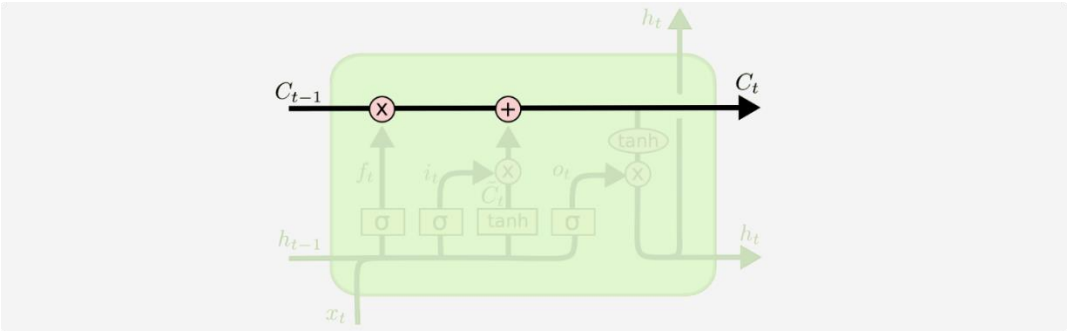


Figure 3.6: LSTM cell state

LSTM can only add and remove information from the previous state to the current state using this cell state. It is carefully regularized by some structures called Gates. Gates are a way which lets the information. These are composed of sigmoid activation in LSTM.

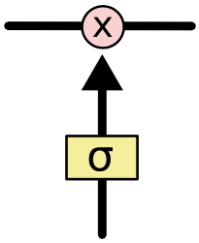
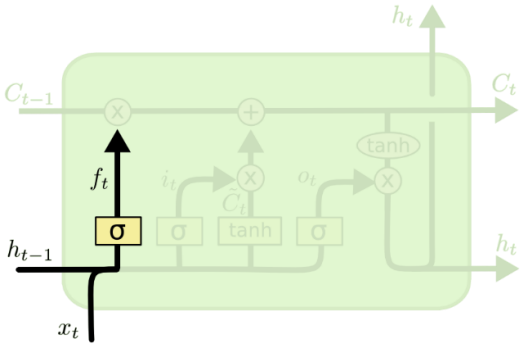


Figure 3.7: LSTM sigmoid activation

An LSTM has three of these gates to protect and control the cell state.

3.3.2 Forget Gate

The first thing an LSTM layer performs is to decide which information it should pass to the cell state before going into other operations. This decision is made by the forget gate, which consists of a sigmoid activation. As we know sigmoid function reduces any input between 0 to 1. If the gate output is one, then all the information from the previous layer will be kept, and if the gate output is 0, all the information will be thrown out and will not be added to the cell state.

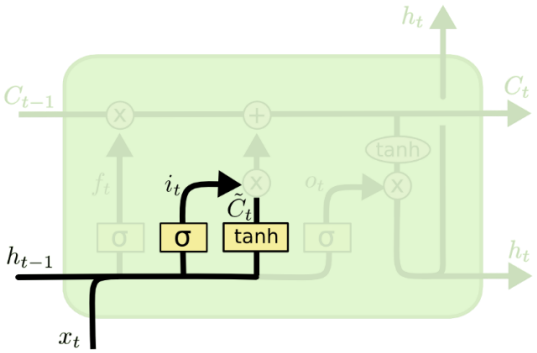


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 3.8: LSTM forget gate architecture

3.3.3 Input Gate

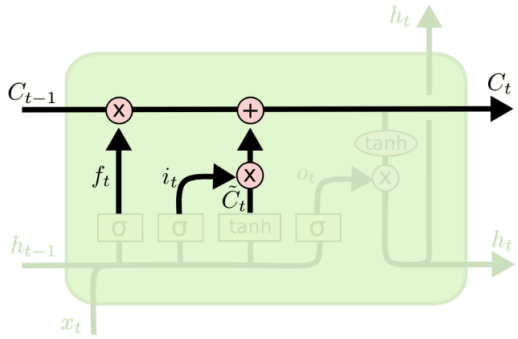
The next part is to decide what new information is going to be stored in the cell state. It consists of two parts; a sigmoid layer called the "input gate layer." It decides the values which will be updated in the cell state. Then there is another layer called the activation. It creates a new vector and gets added with the input gate to the cell state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 3.9: LSTM input gate architecture

In the language model, before adding into the cell state, we drop the old information like the gender of the person in the below-shown part of the model.

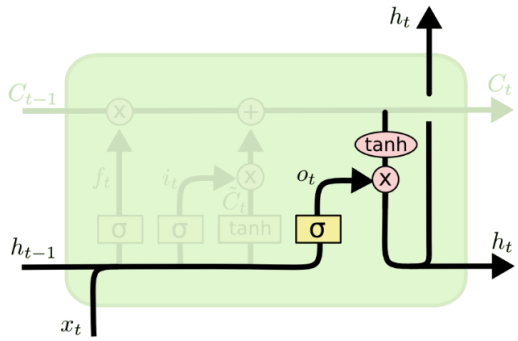


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.10: LSTM input gate with cell state addition

3.3.4 Output Gate

Finally, we decide what information we are giving to the output gate, which will ultimately be the hidden state for another LSTM layer. The output will be based on cell state but will be a filtered version. The information gets filtered out via a sigmoid layer which outputs the communication between 0 to 1. Then we put the output value to the cell state through a tanh activation which outputs the weights between -1 to 1. The process is shown below.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.11: LSTM output gate architecture

3.3.5 Need of LSTM in machine translation

The model used for neural machine translation, LSTM, is the backbone for those models. Those models are based on the working techniques of LSTM. LSTM can keep the extended information in a sentence; it is the best implementation method in any sequence-to-sequence machine translation model.

3.4 Sequence-to-Sequence Machine Translation

Before going into deep neural machine translation models, we must understand what sequence to sequence machine translation is. It is a method via which a model converts one line of text input from one domain into another sequence of text output from another domain. It may be applicable for language translation, text summarization, speech recognition, video captioning, and many other tasks.

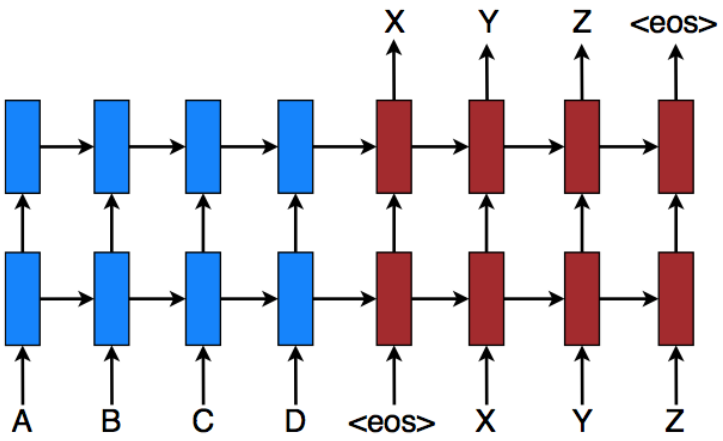


Figure 3.12: Neural Machine Translation basic architecture

Let's take an example using the above-shown figure. Here are two parts presents, which is called encoder and decoder. Both the part consists of RNN layer (in many of the cases it is LSTM). The words or token in an input sentence or sequence is passed to the encoder layer one by one. These recurrent layers are interconnected to each other.

At the end of the encoder, it generates outputs in terms of hidden state and cell state. These states are responsible for remembering the information and sequence present in the sentence keeping all other information.

The hidden state becomes the input to the LSTM of the decoder cell along with the decode input sequence. This sequence of words is being used with the teacher forcing method. Teacher forcing, as discussed earlier, adds start and end tokens to the output sentence or sequence. It will convey the model with the start and endpoint to finish the training and prediction.

3.4.1 Vanilla Sequence to Sequence model

The Seq2Seq framework relies on the encoder-decoder paradigm. The encoder encodes the input sequence, while the decoder produces the target sequence. We will use a vanilla sequence to sequence the model based on encoder decoder architecture for our model.

3.4.1.1 Encoder

Suppose we have an input sequence called "how are you." Each word from the input sequence is associated with a vector $w \in R^d$. Here the w is our embedding vector. As we have three words, so our output is going to be transferred into $[w_0, w_1, w_2] \in R^{d \times 3}$. Each of the embedding vectors is of d dimension. Then we run an LSTM over this sequence of vectors and store the last hidden state outputted by LSTM: this will be our encoder representation.

The hidden state equations will be $[e_0, e_1, e_2]$.

As we can see in figure xx, there are 3 LSTM units are present in the encoder along with three input sequences how, are, and you with the associated embedding w_0, w_1, and w_2 respectively. At 0^{th} timestep, I am passing "how" to the LSTM layer, where I am getting e_0 as the output. At 1^{st} timestep, "are" given to the input LSTM getting e_1 as the output, and the last timestep input is "you" is input and e_2 is the output state. All the LSTM cell states are connected.

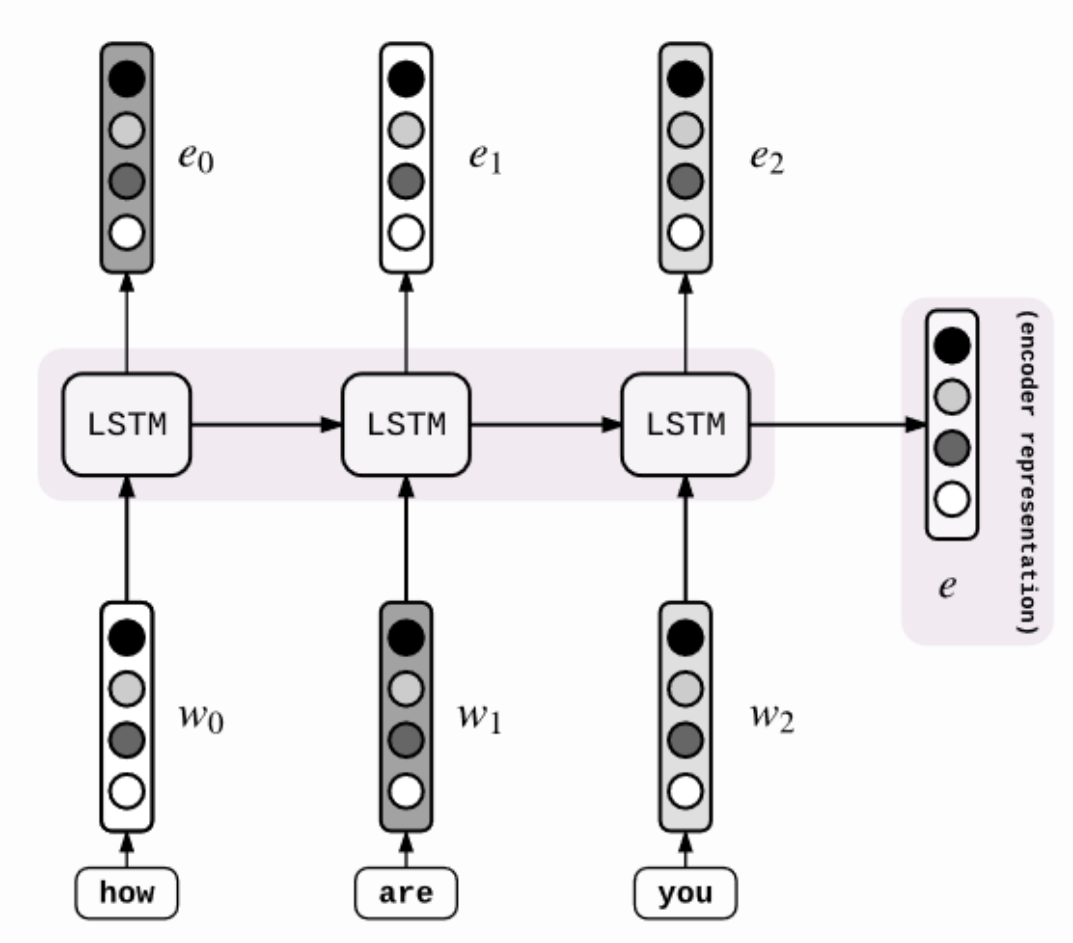


Figure 3.13: Encoder state for NMT

3.4.1.2 Decoder

Now that we have a vector e that captures the meaning of the input sequence, we'll use it to generate the target sequence word by word. Feed to another LSTM cell: e as hidden state and a special start of sentence vector W_{eos} as input. The LSTM computes the next hidden state $h_0 \in R^h$. Then, we apply the function $g : R^h \rightarrow R^V$ so that $s_0 \coloneqq g(h_0) \in R^V$ is a vector of the same size as the vocabulary.

$$h_0 = LSTM(e, W_{eos}) \ldots\ldots (i)$$

$$s_0 = g(h_0) \ \ldots\ldots (ii)$$

$$p_0 = softmax(s_0) \ \ldots\ldots (iii)$$

$$i_0 = argmax(p_0) \ldots\ldots (iv)$$

In the decoder's first timestep, the output hidden state of the encoder is given with an embedding vector of the start of the sentence token. The output will be the hidden state. On the top of hidden state h_0. A dense layer is being applied. On top of s_0 a softmax function is being applied to get the likelihood of each word present in the vocabulary.

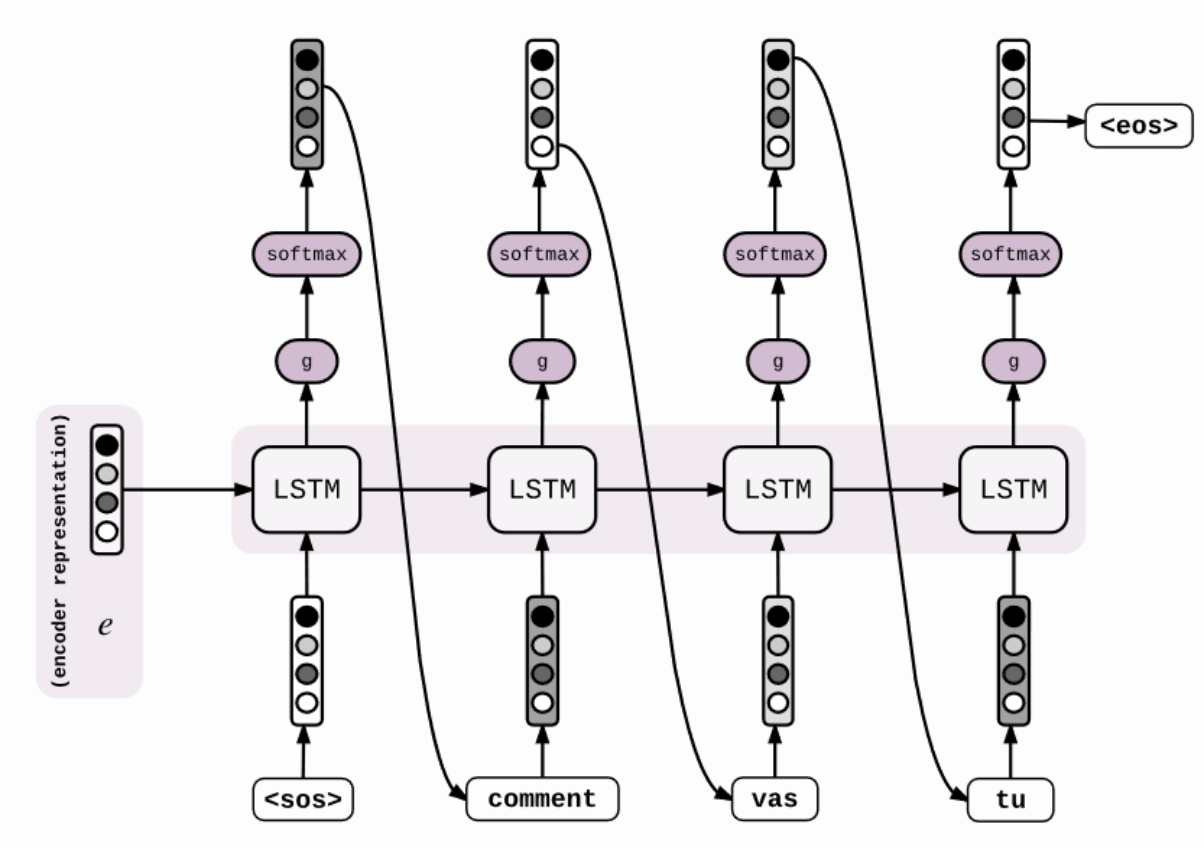


Figure 3.14: Decoder state for NMT

Now, each entry of p_0 will measure how likely is each word in the vocabulary. Let's say that the word *"comment"* has the highest probability (and thus $i_0 = argmax(p_0)$) corresponds to the index of *"comment"*). Get a corresponding vector $w_{io} = w_{comment}$ and repeat the procedure: the LSTM will take h_0 as hidden state and $w_{comment}$ as input and will output a probability vector p_1 over the second word, etc.

$$h_1 = LSTM(h_0, W_{io}) \ldots\ldots (v)$$

$$s_1 = g(h_1) \ \ldots\ldots (vi)$$

$$p_1 = softmax(s_1) \ \ldots\ldots (vii)$$

$$i_1 = argmax(p_1) \ \ldots\ldots (viii)$$

The decoding stops when the predicted word is a particular end of sentence token.

In the simple vanilla sequence-sequence models, we will pass the last time step hidden and cell states to the decoder; instead, we can do average pooling or max-pooling of all the hidden forms of enc and then pass the results as the inputs to the decoder.

3.4.2 Attention Mechanism

The attention mechanism has revolutionized neural machine translation; it produced better results than the previous models by a more significant margin. The word "attention" itself refers to focus on something and taking more considerable notice. Similarly, it is being used in machine translation with a modification in the sequence-to-sequence machine translation. Attention is one component of a network's architecture and oversees managing and quantifying the interdependence:

- Between the input and output elements (General Attention)
- Within the input elements (Self-Attention)

3.4.2.1 Concept and mechanism

Assume we are working on a machine translation problem where we must translate language L_1 to L_2 using previously discussed encoder-decoder architecture. Here L_1will be sent as inputs to the encoder, whereas L2 will be sent to the decoder. The final state of the encoder will be used to initialize the decoder starting states. But if we follow this approach, our model will not understand and mistranslate the more significant sequences. The problem with this approach is that the whole translation will only depend on the last state of the vector. So, translating more significant sequences might be worse. In the attention mechanism, the decoder will get the information from all the hidden states of the encoder, removing the limitation of the encoder-decoder model.

In the decoder model, at time step "t," we will find the weighted average of all the encoder hidden states.

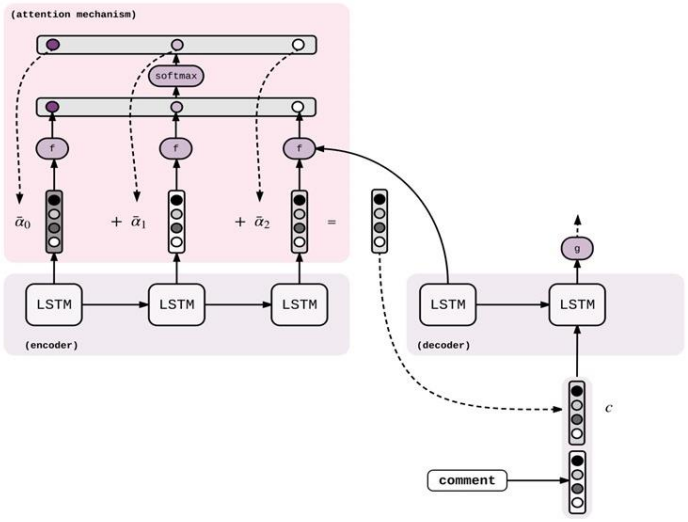


Figure 3.15: Attention model architecture for NMT

- Assume $h_{d(t-1)}$ is the decoder hidden vector at time step "t-1", assume its dimension is 64.
- Assume $H_e = [h_{e(t=0)}, h_{e(t=1)}, h_{e(t=2)}, \ldots, h_{e(t=n)}$ is the matrix of encoder hidden states
- If we consider each $h_{e(t)}$ of 128 dimensions, the matrix H_e will be of $n \times 128$ dimensions
- We will compute the weighted average like

$$\alpha_0 \times h_{e(t=0)} + \alpha_1 \times h_{e(t=1)} + \alpha_2 \times h_{e(t=2)} + \cdots + \alpha_n \times h_{e(t=n)}$$

- If we observe the above formulation, the resultant will be a 128d vector, like the hidden state of **the** encoder.

  The resultant vector is

$$128d\ vector = \alpha_0 \times h_{e(t=0)} + \alpha_1 \times h_{e(t=1)} + \alpha_2 \times h_{e(t=2)} + \cdots + \alpha_n \times h_{e(t=n)}$$

Now how can we find the values of α ?

The step to calculate the α is shown below.

Step 1: (Dot Method same dimension)

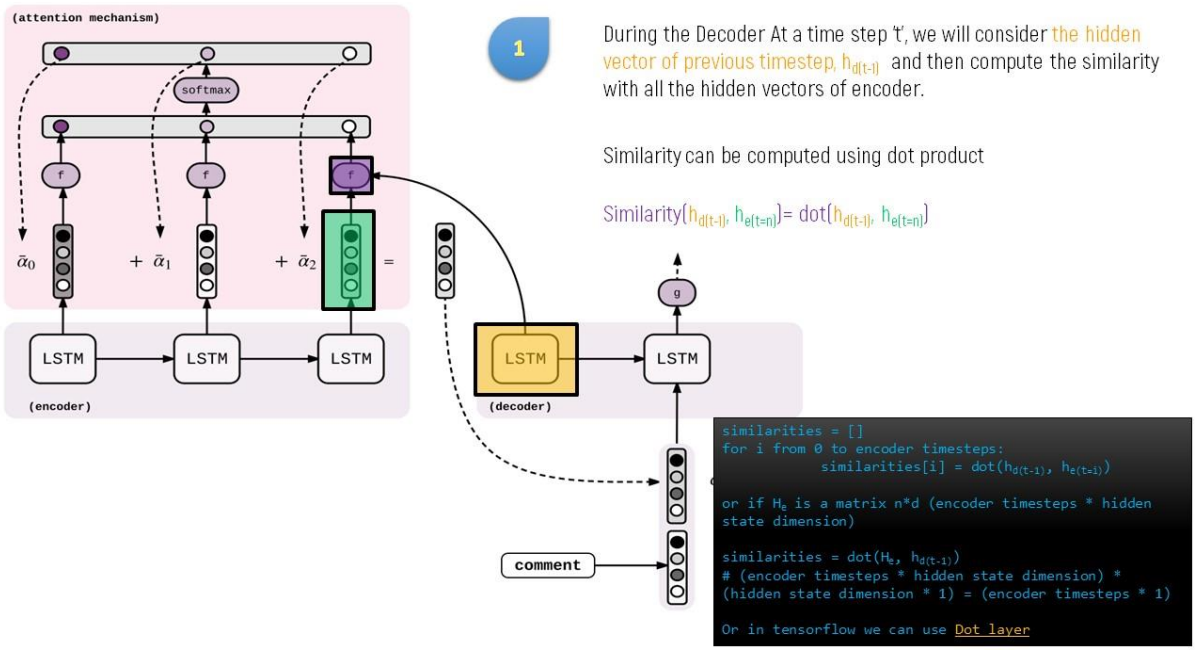


Figure 3.16: Similarity score using dot product for the exact dimension of the hidden state

Figure 3.15 explains the way we are going to calculate the similarity scores. We have the hidden state for all the timestep. The similarity score will be calculated using the dot product of a particular timestep with all timesteps present in the matrix.

The similarity score is

$$Similarity[h_{d(t-1)}, h_{e(t=0)}] = dot[\, h_{d(t-1)}, h_{e(t=0)}]$$

$$h_{d(t-1)} \times h_{e(t=0)} + \; h_{d(t-1)} \times h_{e(t=1)} + \; h_{d(t-1)} \times h_{e(t=2)} + \cdots + \; h_{d(t-1)} \times h_{e(t=n)}$$

Suppose our hidden state matrix size is $n \times d$, and the hidden state size is $d \times 1$. The final shape will be,

$$n \times 1 = (n \times d).(d \times 1)$$

Step 1: (Dot Method of different dimension)

Suppose the hidden state of the encoder and hidden state of the decoder is of different dimension. In that, the similarity score calculation will be different. Here we must add a weight matrix which will be learned using backpropagation.

Let say $h_{d(t-1)}$ has dimension of $d' \times 1$, whereas $h_{e(t=n)}$ has a dimension of $n \times d$. In this case, a weight matrix will be introduced in between them, which will have a size of $d \times d'$.

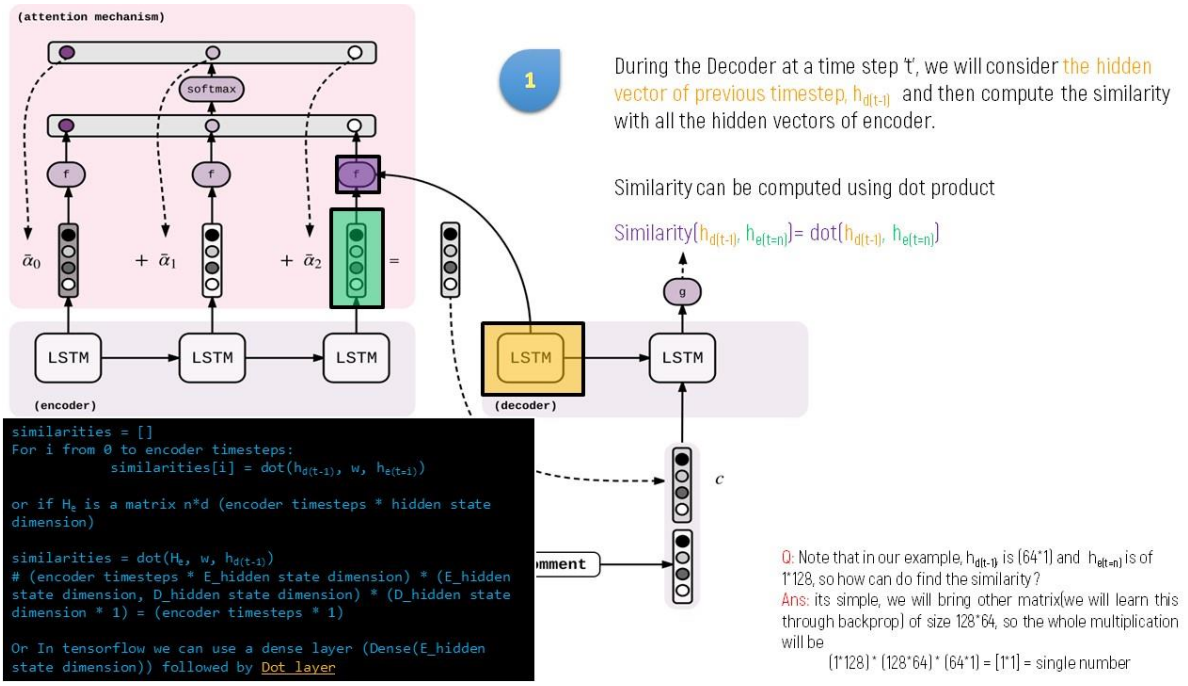


Figure 3.17: Similarity score using dot product for different dimensions of the hidden state

Step 1: (Concatenation Method)

There is another approach to calculate the similarity score, which is going to be explained here.

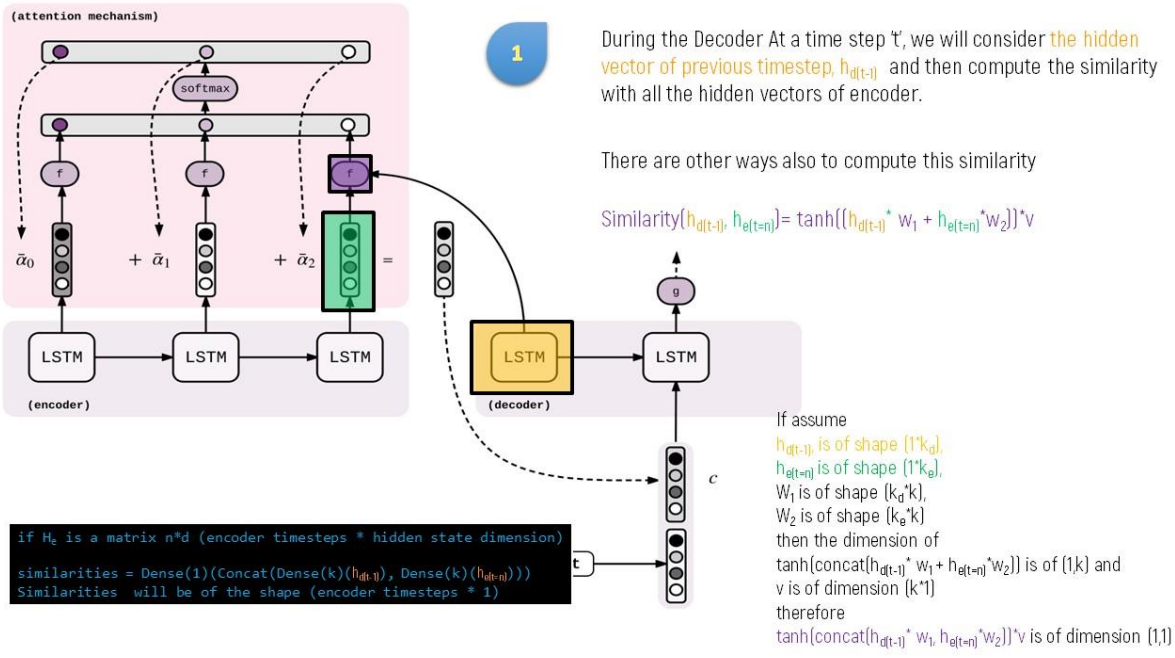


Figure 3.18: Similarity score using concatenation method

As seen in Figure 3.17, instead of calculating the dot product between the hidden states, we concatenate these two vectors. Before that, we are multiplying both the hidden states with a

weight matrix separately. On top of it, we are applying tanh activation function, further normalizing the output.

$$Similarity\left[h_{d(t-1)}, h_{e(t=0)}\right] = tanh\left[\ (h_{d(t-1)} * w_1) + (h_{e(t=n)} * w_2)\ \right] * v$$

The shape calculation is shown in figure 3.17.

Step 2:

In step 1, the similarities are of shape (encoder timesteps * 1) now; in this step, we will apply the softmax function to the similarity scores to get a better probabilistic interpretation. This will give us a range of scores for all weights, whose sum will be 1. These values are the likelihood of each time step.

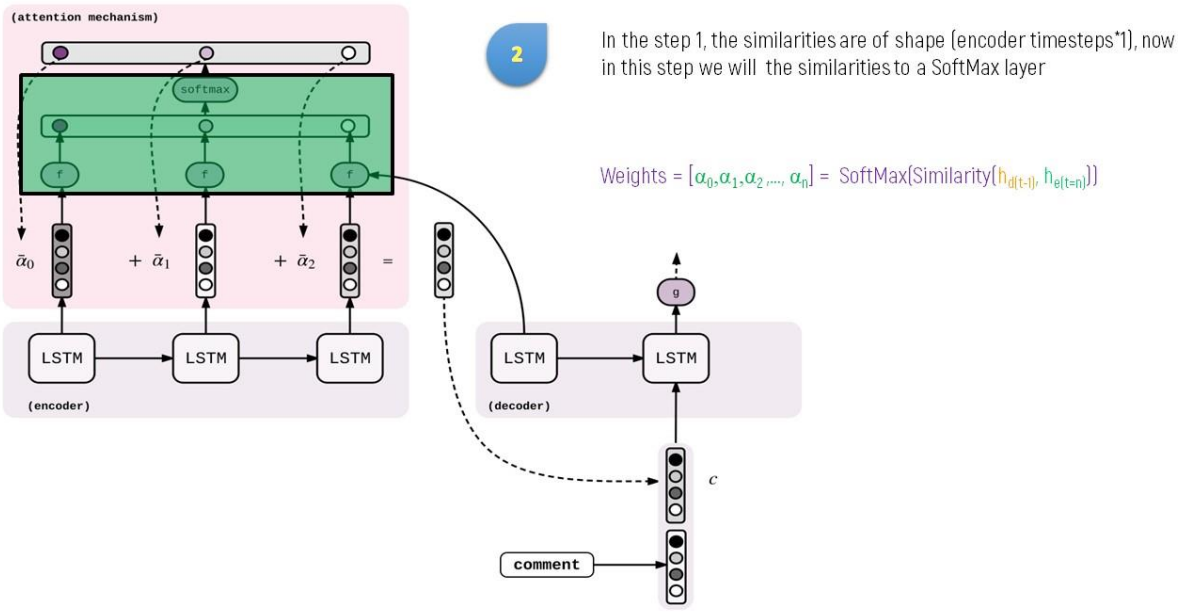


Figure 3.19: weights calculation from similarity scores

$$weights = [\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n] = SoftMax[Similairy\left[h_{d(t-1)}, h_{e(t=0)}\right]]$$

These weights are in the range from 0 to 1, and the total sum will be 1. Based on these weights, we are going to calculate the weighted average. The more the weight is, the more focus will be put on the group.

Step 3:

Here we are going to calculate the weighted average, which is also called context vectors, using extracted α values.

$$context = \alpha_0 \times h_{e(t=0)} + \alpha_1 \times h_{e(t=1)} + \alpha_2 \times h_{e(t=2)} + \cdots + \alpha_n \times h_{e(t=n)}$$
$$= weighted\ avg.$$

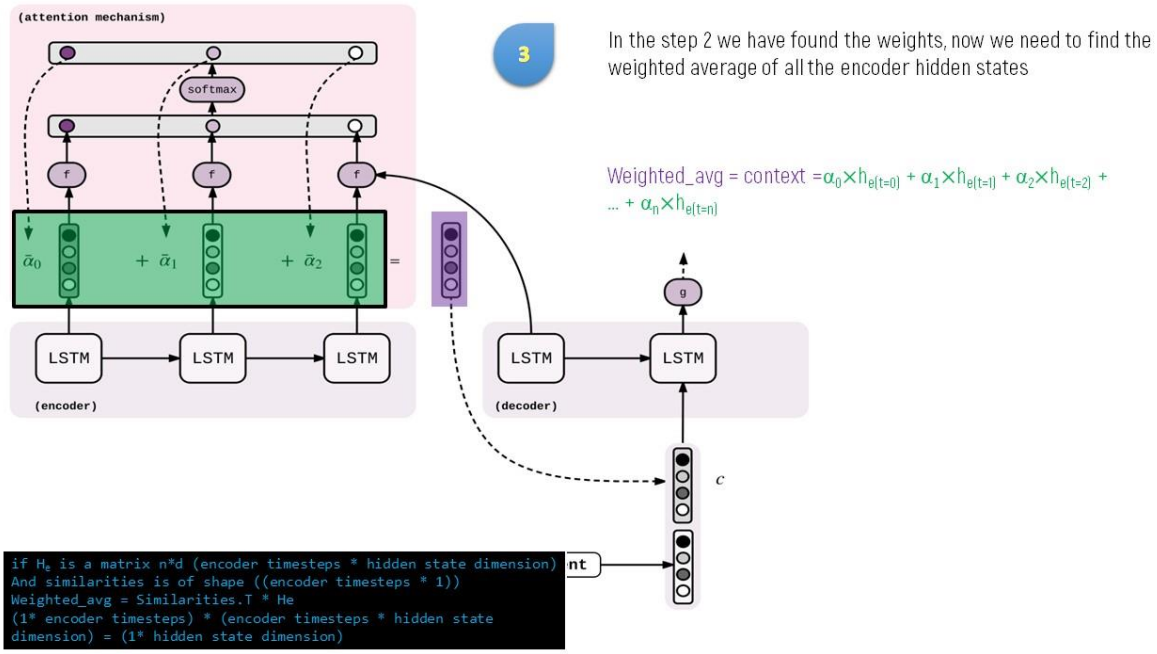


Figure 3.20: context vector calculation

Figure 3.19 shows the code as well as the method to calculate the context vector.

Step 4:

In step 3, we found the context vector that holds the weighted information from the encoder stage. We'll concatenate the context vector with the decoder input and then pass it to the decoder network.

$$decoder\ input = concat[context\ vector, l2\ language\ sequence]$$

So, for a particular time step, we calculate the decoder input by concatenating the context vector calculated using step1, 2, and 3 with the language sequence and passing it to the decoder network. It will be repeated till we get the end of sentence token from the embedding. Step 4 process is shown in figure 3.20.

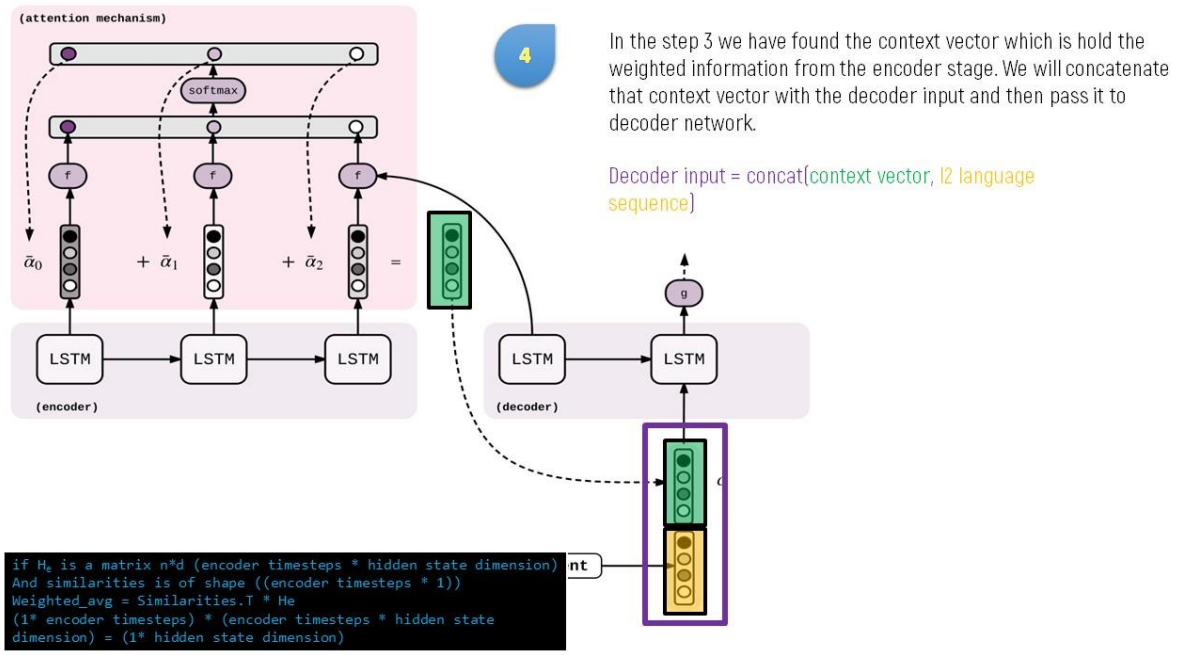


Figure 3.21: attention decoder input

Step 5:

Step 5 is where we are going to get output from the learned embedding layer. The decoding stops when the predicted word is a particular end of sentence token.

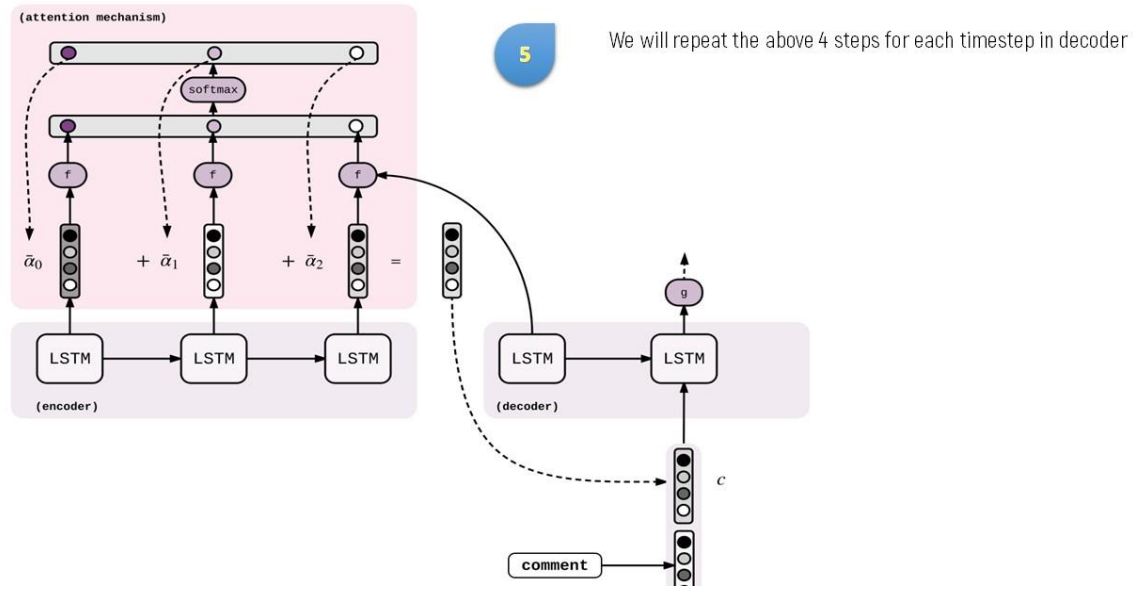


Figure 3.22: attention decoder output

3.5 Log Loss

- Log loss measures the performance of a model whose output is a probability value between 0 to 1.
- The goal of an ml/dl model is always to minimize this value.
- A perfect model would have a log loss of 0. The value of log loss can be anything from 0 to infinity.
- It increases as the predicted probability diverges from the actual label.
- Predicting a probability of 0.10 when the actual observation label is one would be wrong and result from a high log loss.

3.6 BLEU Score

The BLEU score is a string-matching algorithm that provides basic quality metrics for MT researchers and developers.

To conduct a BLEU measurement, the following is necessary:

- One or more human reference translations. It should be data that has not been used in building the system (training data) and ideally should be unknown to the MT system developer.
- It is generally recommended that 1,000 or more sentences be used to get a meaningful measurement. Too small a sample set can sway the score significantly with just a few sentences that match or do not match well.
- Automated translation output of the same source data set.
- A measurement utility that performs the comparison and score calculation

The BLEU method scores a translation on a scale of 0 to 1 to measure the adequacy and fluency of the MT output. The closer to 1 the test sentences score, the more overlap there is with their human reference translations, and thus, the better the system is deemed to be. BLEU scores are often stated on a scale of 1 to 100 to simplify communication, but this should not be confused with the accuracy percentage.

4. Experimental setup

This section explains the experimental setup that will be used to implement the discussed method in other areas. Here the hardware and software setup will be discussed in brief.

4.1 Hardware Implementation

A system with the specification shown in the table 3.1, will be used to carry out the experiment.

Operating System	Windows 10 OS
Processor	Intel(R) Core (TM) i7-10510U
CPU	@1.80GHz 2.30GHz
GPU	NVIDIA Tesla T4
GPU Memory	128GB HBM2 (8x 16GB)

Table 4.1: Computer System Specification

4.2 Software Implementation

The setup for the experiments involves Python 3. The solution is provided using the open-source neural network framework TensorFlow and Keras, written in Python. It is designed to enable fast experimentation with deep neural networks. It contains all the deep learning functionalities such as adding layers, creating custom functions, adding loss and metrics, regularization, and many more. Its multiple Graphics Processing Units - GPUs. This process is achieved by CUDA parallel computing platform and API (Application Programming Interface). NVIDIA creates this processor.

5. Results

5.1 Overview

In this section, we will discuss all the experiments carried out during the duration of the project. It will include a detailed discussion about the network and its architecture, its flaws, and the challenges we faced during the implementation. It will consist of the architecture of the encoder-decoder, Attention, and variants used for experiments. All the codes have been implemented using TensorFlow, and Keras framework. We have also used model subclassing, a researcher way to implement models when we don't have direction implementation using Keras.

5.2 Experiment 1

5.2.1 Model Architecture

For experiment 1, I used the encoder-decoder model. The model is built from scratch. The input to the encoder model is French sentences. I have initialized an embedding matrix that doesn't use any pre-trained matrix. It is initialized with random values (normally distributed). The embedding layer follows up the LSTM layer, which returns both hidden state and cell state as output. For each timestep, this feature is being generated. All the LSTM layers' hidden state and cell state are interconnected. The decoder network takes the hidden state as an input along with the English sentences token with the teacher forcing method. At the decoder, we initialize the embedding layer where the initial weights are initialized with random values (normally distributed). These embedding layers in both encoder and decoder will be trained, and the weights will get updated. The output layer is added with a softmax layer to get the likelihood of the words during training and prediction.

The architecture is implemented using model subclassing in TensorFlow, where we have created an object for encoder and decoder class. Using the inheritance method, the final encoder-decoder model is created.

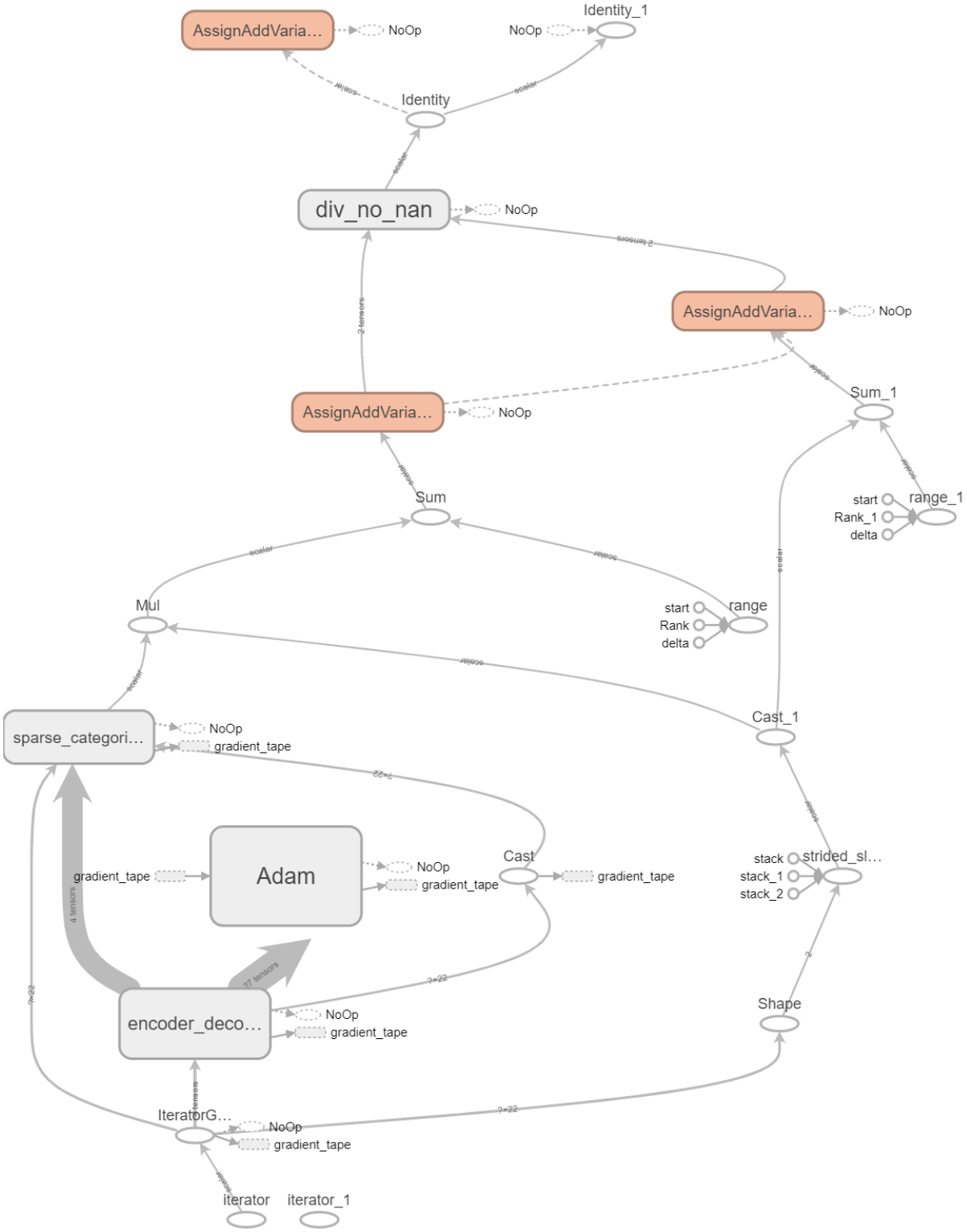


Figure 5.1: Experiment 1 – Encoder-Decoder tensorboard Model graph

The above encoder-decoder model graph taken from tensorboard shows the complexity of the model where it is connected to various inputs and outputs.

5.2.2 Training Parameters

Figure 5.2 shows the total number of parameters model is trained upon.

```
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Encoder)            multiple                  4319744
_________________________________________________________________
decoder (Decoder)            multiple                  3174400
_________________________________________________________________
dense (Dense)                multiple                  7461072
=================================================================
Total params: 14,955,216
Trainable params: 14,955,216
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 5.2: Encoder Decoder model total training parameter

There is almost 15M of training parameters, or weights is present in the model. The encoder part has more weights/parameters than the decoder part.

5.2.3 Hyperparameters

- Encoding threshold = 22
- Epoch to train model = 100
- Embedding size of both encoder and decoder = 128
- Batch size for training = 1024
- Number of parallel LSTM units for both encoder and decoder = 512

As we can see, the encoding threshold for word encoding is taken 22. That means the model will only accept those sentences with less or equal 22 words present in them. If the words are less than 22, it will be padded with 0, and for those with more than 22 words, the rest will be truncated.

The embedding size is relatively small compared to the pre-train models, where we generally use 300 dimensions. Here we are using 128, which is enough in the case of this model.

The number of LSTM units is also a bit higher. We are using the dataset with more than 100k examples with a great variety of translations; it's good to use more LSTM layers parallelly until it is not overfitting.

5.2.4 Loss

The loss method we use to increase the model capacity is cross entropy, which is nothing but log loss.

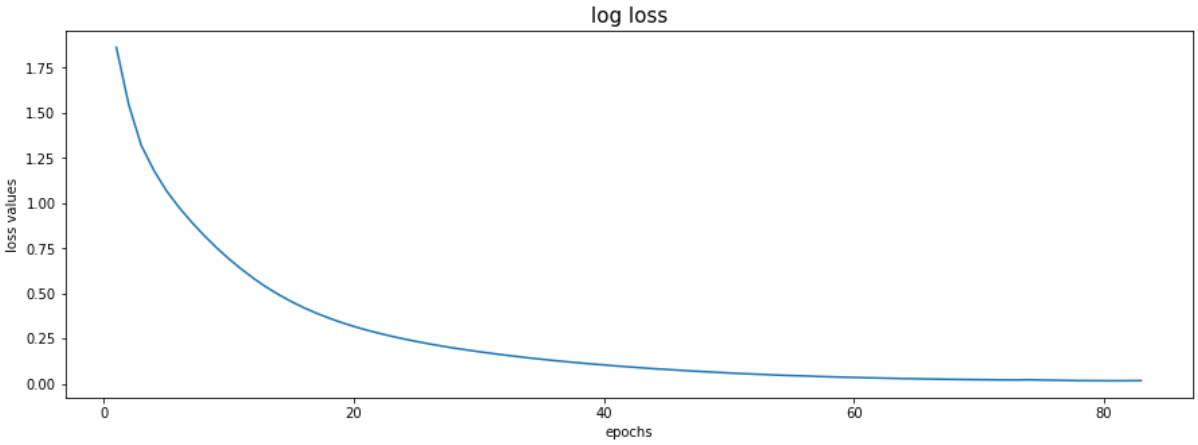


Figure 5.3: Encoder-Decoder model loss vs. epoch graph

From figure 5.3, we can clearly say that after 100 epochs, the model training loss has decreased by a considerable amount. The model is working very well if we say. The model has been appropriately trained. But at the end of the epochs, the loss is not reducing, stating it has reached its limits.

5.2.5 BLEU Score

The BLEU score we generated on the prediction dataset is shown in figure 4.4.

```
import random
import nltk.translate.bleu_score as bleu

score = sum([bleu.sentence_bleu(xval_english[val].lower(), eng_sent_out[val]) for val in range(0, len(xval_english))])
print('BLEU score for Encoder Decoder Model is {:.4f}.'.format(score/len(xval_english)))
```

```
BLEU score for Encoder Decoder Model is 0.82.
```

Figure 5.4: Encoder-Decoder model with BLEU score

The score is excellent as compared to other papers and models. The value 0.82 represents that the model is 82% accurate in translating sentences from French to English.

5.3 Experiment 2

5.3.1 Model Architecture

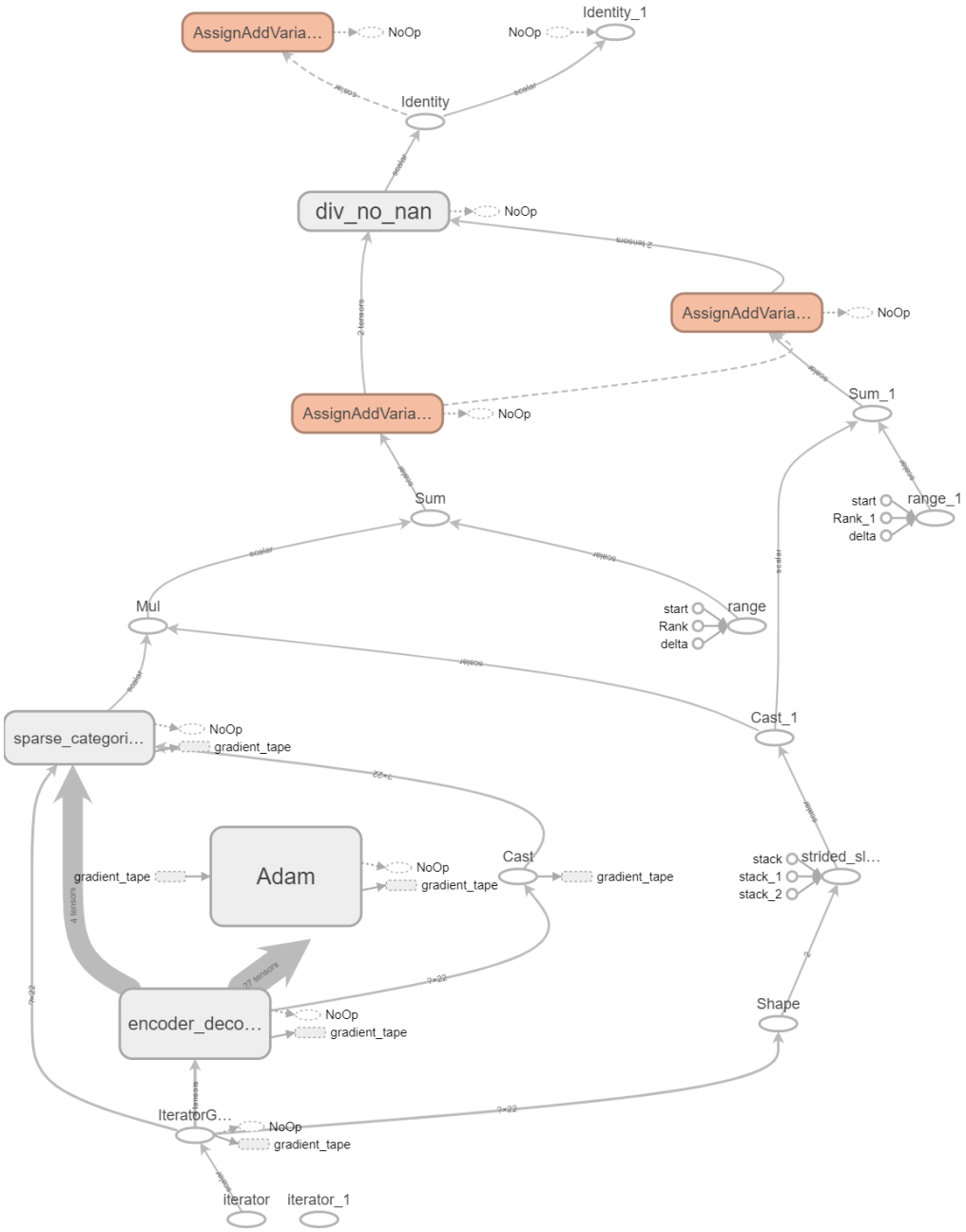


Figure 5.5: Attention model (Dot Product) with model graph

Experiment 2 is carried out with the attention encoder-decoder model using dot product to calculate the context vector. This model also has an encoder and decoder architecture with the

attention model in between them to calculate the context vector. The attention model keeps the hidden state of each timestep. The dot product method multiplies each timesteps' confidential form with all the timestep to generate n timestep weights. For that in the model, we have added an architecture that is called a one-step decoder. This block is responsible for the calculation of the context vector and all alpha values.

5.3.2 Training Parameters

```
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Encoder)            multiple                  12175616
_________________________________________________________________
decoder (Decoder)            multiple                  8555984
=================================================================
Total params: 20,731,600
Trainable params: 20,731,600
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 5.6: Attention model (Dot Product) total training parameter

The training parameter present in the attention model with the dot product method is 20M, which is very high than the previously discussed encoder-decoder model.

5.3.3 Hyperparameters

- Encoding threshold = 22
- Epoch to train model = 100
- Embedding size of both encoder and decoder = 512
- Batch size for training = 1024
- Total attention units = 1024
- Total number of parallel encoder LSTM units = 64
- Total number of parallel decoder LSTM units = 64

The embedding size used for the model is relatively high. Due to the tall model complexity, it is needed to use the high matrix of embedding size.

The attention units used in the model are also very high, which is 1024 units. To find out the context vector, we use 1024 parallel units to capture the idea and concept behind the translation during the training process.

As the attention units are very high compared to the model complexity, we have used comparatively fewer LSTM units in parallel, only 64. It is optimal for this case.

5.3.4 Loss

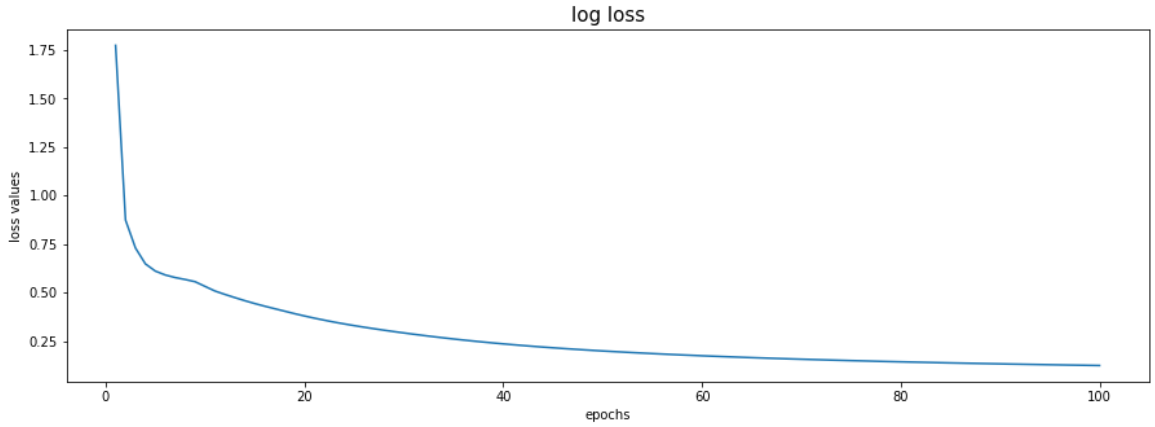


Figure 5.7: Attention model with Dot Product loss vs. epoch graph

As we can see from figure 5.7, during the training process, the loss suddenly dropped to a significant amount during the start of the training. After some epoch, the learning process became slow, but it is excellent as the model didn't overfit and continue to reduce the model training.

5.3.5 BLEU Score

```
import random
import nltk.translate.bleu_score as bleu

score = sum([bleu.sentence_bleu(english_validation[val].lower(), eng_sent_out[val].strip()) for val in range(0, len(english_validation))])
print('BLEU score for Encoder Decoder Model is {:.4f}.'.format(score/len(english_validation)))
```

```
BLEU score for Encoder Decoder Model is 0.7906.
```

Figure 5.8: Attention model with Dot Product with BLEU score

The BLEU score for the attention model with Dot Product has a BLEU score of almost 80%. It doesn't mean that the model performs a bit worse compared to the encoder-decoder model. After many hyperparameters tuning, we achieved this score; still, there is a scope of improvement needed.

5.4 Experiment 3

5.4.1 Model Architecture

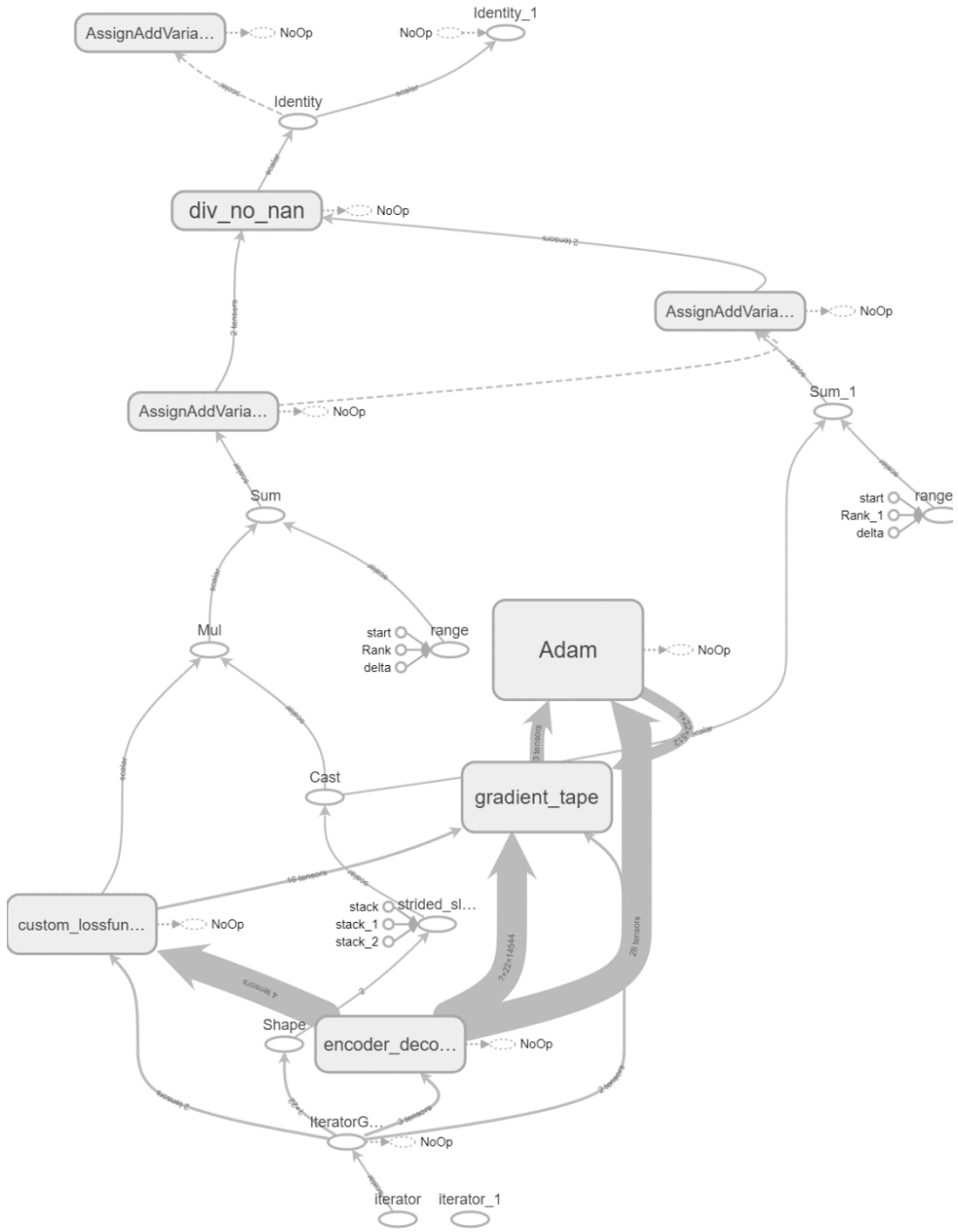


Figure 5.9: Attention model (General method) with model graph

The model architecture for experiment 3 is an attention model with a general method where we have added a dense layer with an extra attention unit that will yield a better context vector than the dot product model. Other than the context vector calculation change, the model has the same architecture as experiment 2. It has one encoder part along with attention units. We have

added a one-step decoder to incorporate the attention calculation. Further, it relates to the decoder layer.

5.4.2 Training Parameters

```
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Encoder)            multiple                  12175616
_________________________________________________________________
decoder (Decoder)            multiple                  8560144
=================================================================
Total params: 20,735,760
Trainable params: 20,735,760
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 5.10: Attention model (General method) total training parameter

The entire parameter to be updated during training of experiment 3 is more than 20M. It is almost like experiment 2 as they both have the same architecture with slight modification.

5.4.3 Hyperparameters

- Encoding threshold = 22
- Epoch to train model = 100
- Embedding size of both encoder and decoder = 512
- Batch size for training = 1024
- Total attention units = 1024
- Total number of parallel encoder LSTM units = 64
- Total number of parallel decoder LSTM units = 64

The hyperparameter combination is the same as experiment 2. The embedding size is the same as 512, which is good as compared to model complexity. The attention units along with LSTM parallel units are also the same.

5.4.4 Loss

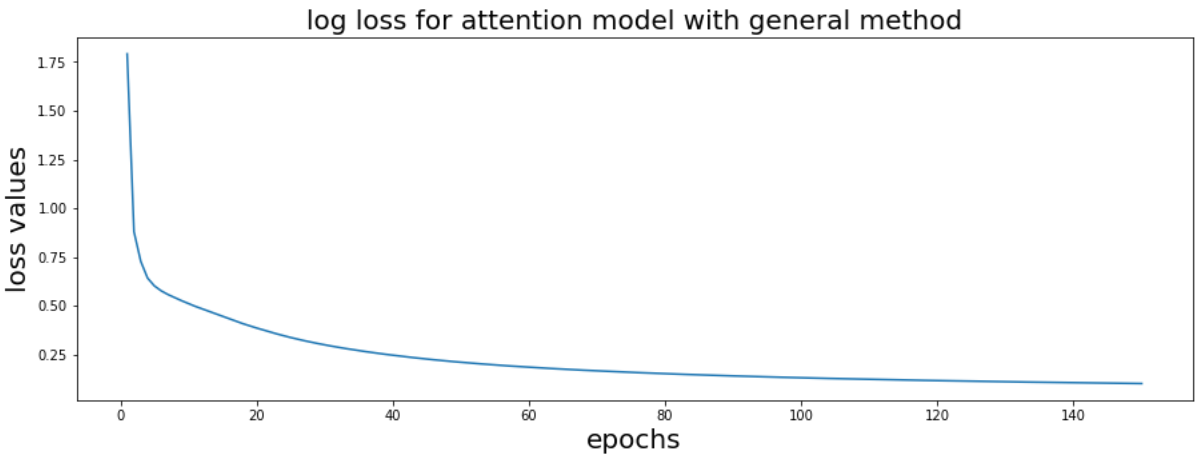


Figure 5.11: Attention model (general method) loss vs. epoch graph

As we can see from figure 5.11, during the training process, the loss suddenly dropped to a significant amount during the start of the training. The model didn't overfit till the 150^{h} epoch, and the loss is kept decreasing till and didn't stop.

5.4.5 BLEU Score

```
import random
import nltk.translate.bleu_score as bleu

score = sum([bleu.sentence_bleu(english_validation[val].lower(), eng_sent_out[val].strip())
             for val in range(0, len(english_validation))])
print('BLEU score for Encoder Decoder Model is {:.4f}.'.format(score/len(english_validation)))
```

```
BLEU score for Encoder Decoder Model is 0.7890.
```

Figure 5.12: Attention model (general method) with BLEU score

The BLEU score is slightly lower than the experiment 1 and 2, which is acceptable, but the score can be further improved using more tuning.

5.5 Experiment 4

5.5.1 Model Architecture

The model architecture used in experiment 4 is almost the same as experiment 2. But the calculation for the context vector is not more rigid, and we are now applying tanh function to get the vectors with a range between -1 to 1.

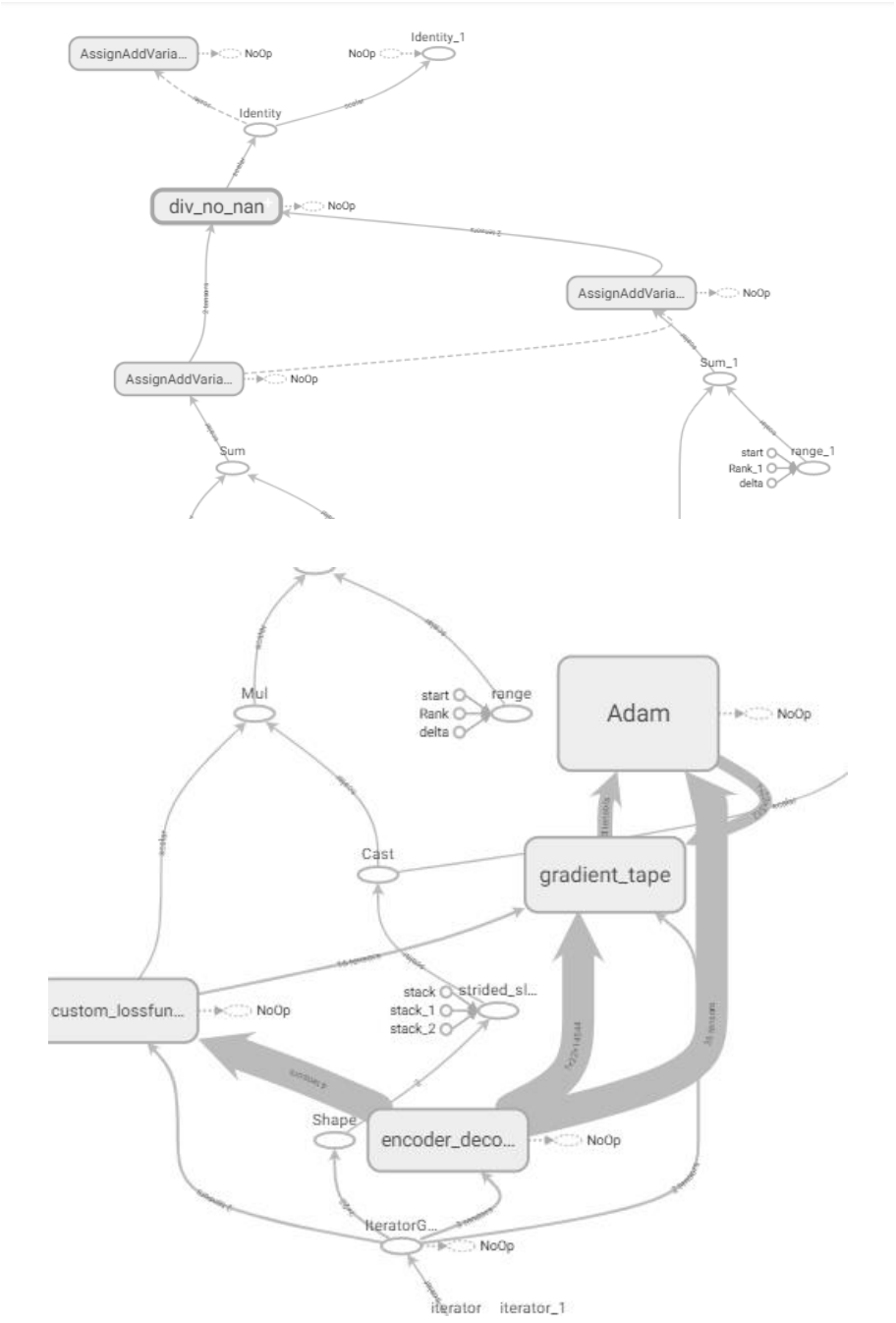


Figure 5.13: Attention model (concat method) with model graph

5.5.2 Training Parameters

```
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Encoder)            multiple                  12175616
_________________________________________________________________
decoder (Decoder)            multiple                  8589521
=================================================================
Total params: 20,765,137
Trainable params: 20,765,137
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 5.14: Attention model (concat method) total training parameter

5.5.3 Hyperparameters

- Encoding threshold = 22
- Epoch to train model = 100
- Embedding size of both encoder and decoder = 512
- Batch size for training = 1024
- Total attention units = 256
- Total number of parallel encoder LSTM units = 64
- Total number of parallel decoder LSTM units = 64

For experiment 4, all parameters are the same except the total attention units. As the model complexity had been increased, we must decrease the attention units. The calculation way for context vector has changed now, so reducing total attention units should not affect the model much. Other than that, we are still using the same embedding size, which is 512.

5.5.4 Loss

The loss function shows that model has a sudden drop in the loss value during the initial epochs, but it increased a bit after that. At the end, the model is not overfitting, and the loss has decreased to a good extend.

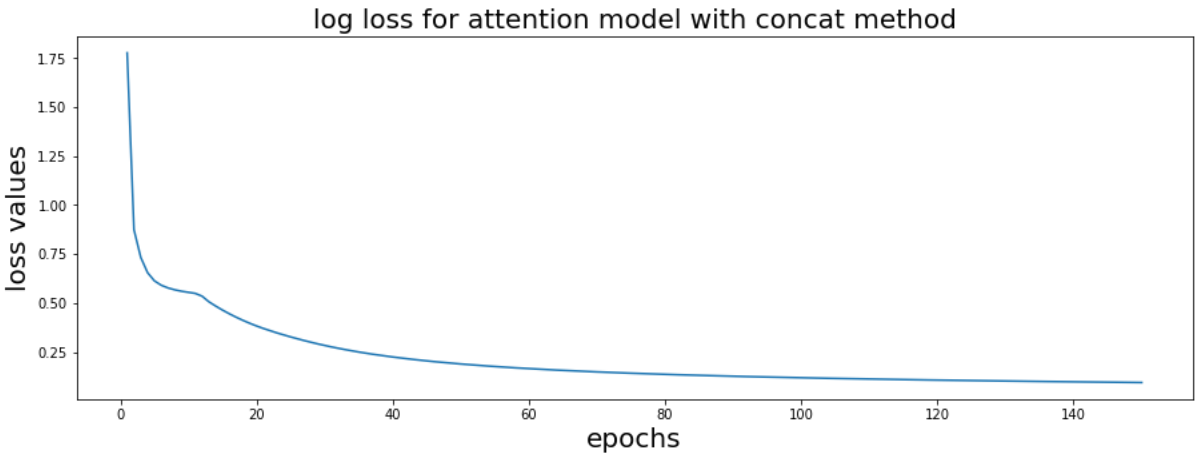


Figure 5.15: Attention model (concat method) loss vs. epoch graph

5.5.5 BLEU Score

```
import random
import nltk.translate.bleu_score as bleu

score = sum([bleu.sentence_bleu(english_validation[val].lower(), eng_sent_out[val].strip())
             for val in range(0, len(english_validation))])
print('BLEU score for Encoder Decoder Model is {:.4f}.'.format(score/len(english_validation)))
```

```
BLEU score for Encoder Decoder Model is 0.7847.
```

Figure 5.16: Attention model (concat method) with BLEU score

The BLEU score for experiment 4 is not high compared to experiment 1, still model performing well, and the score is acceptable.

5.6 Summary

After performing four experiments, we can say that purpose of our implementation has been fulfilled to a greater extend. All the experiments were carried out very well, and after a log of hyperparameter tuning is performed well.

5.7 Analysis and Synthesis

In this section of the project, we will discuss the result and the performance of each successful experiment. 4 successful experiments are being carried out for French to English translation.

All the experiments have been tuned to get the best hyperparameter, and the scores are impressive.

We used the BLEU score as a model validation parameter and log loss as the loss method to train the model. All the successful experiments provided an excellent log loss which is below 0.010. For some experiments, it is shallow, which is relatively good to work on. The BLEU score benchmark was 75% which is cleared by all investigations.

5.8 Summary of Result

Sl. No.	Experiment Type	Epochs	Log Loss	BLEU
1	Encoder Decoder	100	0.0181	0.8200
2	Attention dot method	100	0.1246	0.7906
3	Attention general method	150	0.1024	0.7890
4	Attention concat method	150	0.0938	0.7847

Table 5.1: Result Summary

After four experiments and a lot of hyperparameters tuning, the results are tabulated above. As we can see, the best-performing model above is encoder-decoder. If we see all the models performed well, the losses are also the same for the attention model. But the best performing model is encoder-decoder. The encoder-decoder model has taken fewer epochs and fewer hyperparameters.

Compared to the encoder decoder model, the attention model has taken many hyperparameters but performed a little less. The above results show how a deep learning model can create elegant models to make the best neural machine translation.

A prediction pipeline is also implemented to get the output for the input French sentences. Some of the results are shown below.

```
Please enter a french sentence which can have maximum 22 words.

J'aime jouer au football
i like playing football .
```

Figure 5.17: Prediction Example 1

```
Please enter a french sentence which can have maximum 22 words.

C'est un bon garçon. Il a besoin d'en apprendre davantage.
he is a good writer and think he should try it .
```

Figure 5.18: Prediction Example 2

```
Please enter a french sentence which can have maximum 22 words.

Parlons d'autre chose.
let s talk about something else .
```

Figure 5.19: Prediction Example 3

The above images show the input French sentence and output English sentence. For figure 5.17 and figure 5.19, the translation is perfect, and there is no change in the output. But for figure 5.18, the model has predicted a partially correct answer. The possible reason behind this is that the model is not trained on a large domain of data and can be improved by adding more words and meanings.

6. Conclusion

6.1 Overview

In this part, we conclude the project by considering all the reviews that have been done extensively. We tried to implement the models the way it is in some papers and developed our model using model subclassing TensorFlow. All the models performed well and performed almost similarly. However, the encoder-decoder model has outperformed all the models performed during the experiment.

All the models went through extensive hyper-parameter tuning, which yielded the best in the models.

6.2 Future Work

Although the achievements done during the project are a good start for the neural machine translation, there is a considerable scope of improvement. Some other methods can be implemented to make the NMT model more robust and best.

6.2.1 Transformers

The transformer models, which we have already discussed in the research paper attention all you need, is the best benchmark model for any neutral machine translation models. All the information can be extracted from the research paper [16]. This method is more robust and best to implement as it uses a pre-trained model which is already being trained on big platforms.

Q1. What are the weaknesses of the study and your project (i.e., is there any strong modelling assumptions they might not be valid in practice)? What are the threats to the validity?

For the experiments discussed in the project doesn't have any strong assumptions related to the model building. Though the dataset used for the modelling is weak and doesn't have a huge a corpus of translation. This may be major threat for the model validation.

Appendix

The project will be in Python with the help of existing libraries. The main objective is to provide a highly effective algorithm that handles the multi-language translation within the given time frame. Since it is a state-of-the-art algorithm, the codes are publicly available, development on prior research will make this solution feasible. It will have multiple applications in the field of engineering, design, and education. It will eliminate the language barrier making the process of information transfer and communication easier.

Most of the NLP-related libraries are going to be used as text pre-processing will be involved. Some example libraries are NLTK (Natural Language Tool Kit) and genism. The frameworks for deep learning model development provided in Python are TensorFlow, Keras, and PyTorch. For the project, a stable version of TensorFlow, which is V2.5, will be used. Model Subclassing technique will be implemented to develop the state-of-art models. For the baseline model, Keras framework-based LSTM and GRU API will be used for implementation.

As deep learning models are highly complex, considering GPU for training models quickly, a high computational resource might be needed. To overcome this challenge, Google Colab is a free source platform provided by Google, can be used to train, and implement models.

References

[1] Trujillo, A., 2012. Translation Engines: techniques for Machine Translation. First Edition. Place of publication: Springer Science & Business Media.

[2] ALPAC (Automatic Language Processing Advisory Committee). 1966. Language and machines: computers in translation and linguistics. Washington, DC. National Academy of Sciences.

[3] Mikel L. Forcada and Ramón P. Ñeco. 1997. Recursive hetero-associative memories for translation. In Biological and Artificial Computation: From Neuroscience to Technology, pages 453–462.

[4] A. Waibel, A. N. Jain, A. E. McNair, H. Saito, A.G. Hauptmann, and J. Tebelskis. 1991 JANUS: A Speech-to-Speech Translation System using Connectionist and Symbolic Processing Strategies. In Proceedings of ICASSP 1991 (International Conference on Acoustics, Speech and Signal Processing). Pages 793–796.

[5] Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. Association for Computational Linguistics. In Proceedings of the 2013 Conference on Empirical Methods in NLP. Pages 1700–1709.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Advances in neural information processing systems. Pages 3104–3112.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. CoRR, arXiv/1409.0473

[8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8, 1735–1780. DOI:https://doi.org/10.1162/neco.1997.9.8.1735

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In Proceedings of the 2014

EMNLP (Conference on Empirical Methods in Natural Language Processing). Association for Computational Linguistics. Pages 1724–1734.

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. CoRR, arXiv/1705.03122.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Advances in Neural Information Processing Systems 30. pages 6000–6010.

[12] Peter Toma. 1977. Systran as a multilingual machine translation system. In Proceedings of the Third European Congress on Information Systems and Networks, Overcoming the language barrier, pages 569–581.

[13] Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio OrtizRojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M. Tyers. Jun 2011. Apertium: a free/open-source platform for rule-based machine translation. Machine Translation, 25(2):pages 127–144. ISSN 1573-0573. doi: 10.1007/s10590-011-9090-0.

[14] All Languages Ltd, 2016. Machine Translation Basics: Technology has improved productivity but hasn't overcome the complexities of human language. Online: All Languages Ltd.

[15] Diederik P. Kingma and Max Welling. 2013. Autoencoding variational bayes. CoRR, abs/1312.6114.

[16] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. 2015. Draw: A recurrent neural network for image generation. In ICML.

[17] Elman Mansimov, Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. 2015. Generating images from captions with attention. CoRR, abs/1511.02793.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680.