UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Department of Electronic and Computer Engineering

Final Year Project Report

Project Title: Machine Translation using Deep Learning Neural Networks

Student Name: Yashwardhan Kaul

Student I.D: 14097095

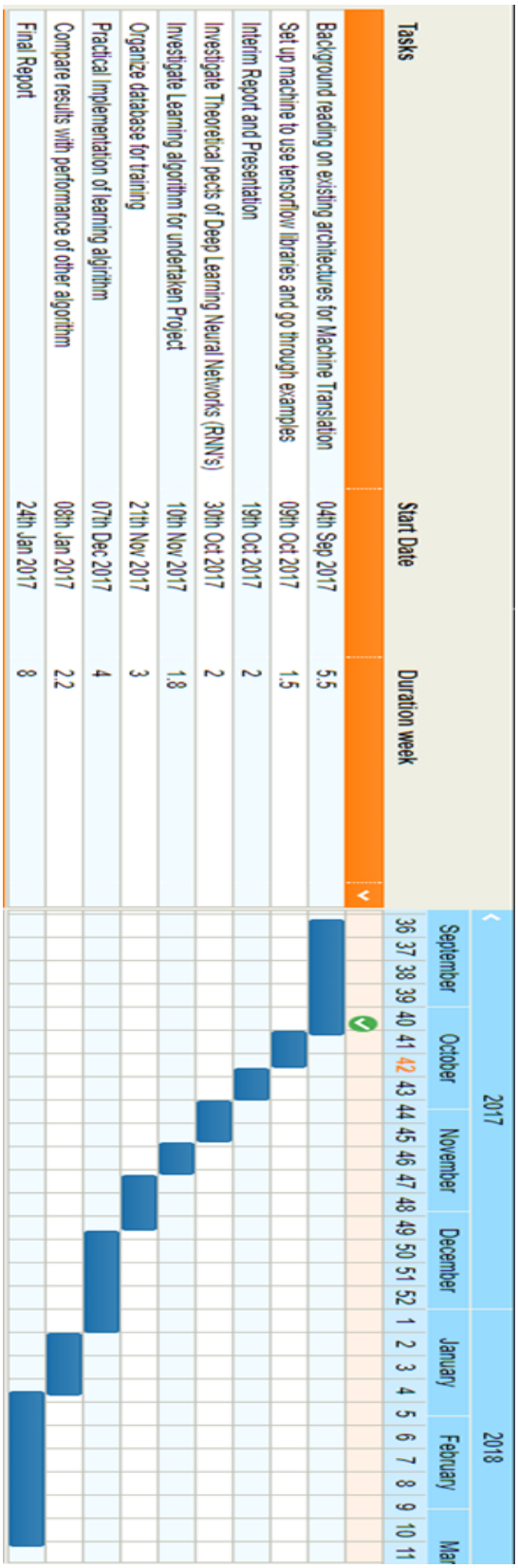
Supervisor: Dr Colin Flanagan

Course: BEng In Electronic and Computer Engineering

Course Code: LM118

Academic Year: 2017/2018

PLAN OF ACTION



ABSTRACT

This report details the background, working of various models, and implementation of deep learning techniques, in order to achieve machine translation between two languages. The project uses a specific type of Recurrent Neural Network (RNN) called Long-short-term Memory (LSTM) for encoding and decoding with an attention mechanism to achieve better accuracy.

The training data for this project is a parallel corpora of translated movie subtitles available on the internet and is open-source. In this project the student has used tensorflow libraries at build the encoder-decoder model. It uses the tensorflow built in data utils class to process the data before training. The attention mechanism works by querying on the output of the encoder LSTM. Each encoder output gets a relevancy score and is converted to a probability score by applying a softmax activation to it.

The results achieved were fairly-accurate, keeping in mind the limited processing power in a standard computer. More accurate translations could have been achieved with the use of multiple GPUs and by using Bidirectional RNNs with more layers. Also, changing the hyperparameters and using a bigger training set could possibly result in increased accuracy. Further development of this project is also proposed and recommended.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to my supervisor, Dr. Colin Flanagan, for his support, insight and guidance which played a vital role in bringing this project to fruition. The knowledge that I gained through the year my taking the subjects related to this project helped me to complete the project on time.

I would also like to thank the entire staff of the Department of Electronic and Computer Engineering, especially Dr. Colin Fitzpatrick, Dr. John Nelson and Emily Spencer for answering my questions and giving me their time to assist me in any way they could.

Lastly, many people have contributed in my learning experience in University of Limerick. The skills that I acquired here were vital to realise this project. I would like to thank my fellow classmates, students and machine learning peers that shared their knowledge on the Internet, making it available to the public. This project was a great learning experience, for which I am grateful.

Table of Contents

PLAN OF ACTION 3

ABSTRACT 5

ACKNOWLEDGEMENTS 7

LIST OF FIGURES 12

1. INTRODUCTION 15

1.1 OBJECTIVES 17

1.2 ASSUMPTIONS 17

1.3 RATIONALE 17

2. EARLIER TECHNIQUES AND THEIR COMPARISON 18

2.1 RULE-BASED MACHINE TRANSLATION (RBMT) 18

2.1.1 DIRECT MACHINE TRANSLATION 18

2.1.2 TRANSFER-BASED MACHINE TRANSLATION 19

2.1.3 Interlingual Machine Translation 20

2.2 EXAMPLE BASED MACHINE TRANSLATION 20

2.3 STATISTICAL MACHINE LEARNING 21

2.3.1 WORD-BASED SMT 21

2.3.2 SYNTAX-BASED SMT 21

2.3.3 PHRASE-BASED SMT 22

3. DEEP NEURAL MACHINE TRANSLATION 23

3.1 FEED-FORWARD NETWORKS 24

3.2 Recurrent Neural Networks (RNNs) 25

3.2.1 ENCODINGS 27

3.2.2 Vanishing (and Exploding) Gradients 28

3.2.3 Backpropagation Through Time (BPTT) 29

3.2.4 Truncated BPTT 29

3.3 Long-Short Term Memory (LSTM) 30

4. SOFTWARE SETUP 33
4.1 ANACONDA (PYTHON) 33
4.2 ATOM 34
4.3 PYTHON LIBRARIES USED 34
4.3.1 TENSORFLOW 34
4.3.2 NUMPY 35
4.3.3 SCIKIT-LEARN 35
4.3.4 DATA-UTILS 35
4.3.5 MATPLOTLIB.PYPLOT 35
4.3.6 nltk.translate.bleu_score 36
5. OPEN-SOURCE SUBTITLES FOR TRAINING AND TESTING 36
6. IMPLEMENTATION 40
7. RESULTS 45
7.1 ENGLISH TO GERMAN 45
7.1.1 TRAINING 45
7.1.2 PLOT FOR LOSS 52
7.1.3 TRANSLATIONS 53
7.2 ENGLISH TO SPANISH 54
TRANSLATIONS 54
7.3 ENGLISH TO ARABIC 55
8. CONCLUSION 57
9. FUTURE WORK AND RECOMMENDATIONS 58
REFERENCES 59
APPENDICES 61
APPENDIX A 61
APPENDIX B 66
COPY OF POSTER 68

LIST OF FIGURES

Figure 1: Natural Language Processing Levels

Figure 2: History of Machine Translation [3]

Figure 3: Word by word translation from English to Spanish [4]

Figure 4: Example of bilingual corpus

Figure 5: Example of Syntax-based Machine Translation [6]

Figure 6: A feed forward neural network

Figure 7: Simple Recurrent Network proposed by Elman [15]

Figure 8: Example of encoding [3]

Figure 9: Effects of applying a sigmoid function [14]

Figure 10: Data flow through a memory cell [14]

Figure 11: Detailed schematic of a RNN unit (left) and a LSTM block (right) as used in the hidden layers of a recurrent neural network [14]

Figure 12: Working of the gates

Figure 13: Snippet of Anaconda Python on Student's machine

Figure 14: Front page of open subtitles website

Figure 15: First 35 sentences in Training set for english-german (German)

Figure 16: First 35 sentences in Training set for english-german (English)

Figure 17: First 35 sentences in Training set for english-spanish (Spanish)

Figure 18: First 35 sentences in Training set for english-spanish (English)

Figure 19: First 35 sentences in Training set for english-arabic (Arabic)

Figure 20: First 35 sentences in Training set for english-arabic (English)

Figure 21: Screenshot of import block in the translator.py code

Figure 22: screenshot of vocab size in code

Figure 23: screenshot of usage of data-utils class

Figure 24: Code snippet for data padding and data splitting

Figure 25: Code snippet for tensorflow placeholders

Figure 26: Representation of an LSTM encoder-decoder model

Figure 27: Code snippent for the attention mechanism implementation

Figure 28: Example of an attention-based NMT system as described in (Luong et al., 2015) [25]

Figure 29: Equations for attention mechanism

Figure 30: Code Snippet for softmax implementation

Figure 31: Hyperparameters

Figure 32: Code for decoding output sequence

Figure 33: Code for training data and plotting losses

Figure 34: Code for training and translation i.e. decoding

Figure 35: Training log pg 1

Figure 36: Training log pg 3

Figure 37: Training log pg 2

Figure 38: Training log pg 5

Figure 39: Training log pg 4

Figure 40: Training log pg 6

Figure 41: Training log pg 7

Figure 42: Training log pg 8

Figure 43: Training log pg 9

Figure 44: Training log pg 10

Figure 45: Training log pg 11

Figure 46: Training log pg 12

Figure 47: Loss Plot

Figure 48: Snapshot of the translated outputs 1

Figure 49: Snapshot of the translated output 2

Figure 50: Loss plot for Eng-Sp

Figure 51: Snapshot of translated output

Figure 52: Loss plot for Engilsh to Arabic

Figure 53: Snapshot of Translation output for en- ar

1. INTRODUCTION

One of the earliest goals for computers was the automatic translation of text from one language to another. Automatic or machine translation is perhaps one of the most challenging artificial intelligence tasks given the fluidity of human language. Classically, rule-based systems were used for this task, which were replaced in the 1990s with statistical methods. More recently, deep neural network models achieve state-of-the-art results in a field that is aptly named neural machine translation. Machine translation is the task of automatically converting source text in one language to text in another language. [1]

In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language. [2]

Natural language processing is a field at the intersection of computer science, artificial intelligence and linguistics. The goal for language processing is to "Understand" natural language in order to perform tasks that are useful like, making appointments, buying things or question answering like Siri, Cortana etc.

There is no one single best translation of a sentence to another language. This is because of the flexibility and complexity of the human languages. This makes automatic machine translation a very big challenge, perhaps the most difficult challenge in the sphere of artificial intelligence.

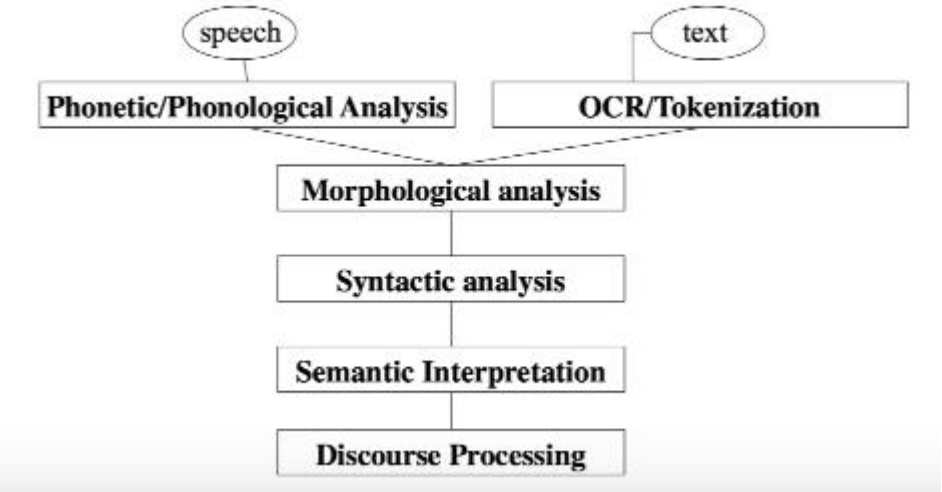


Figure 1: Natural Language Processing Levels

The key limitations of the classical machine translation approaches are both the expertise required to develop the rules, and the vast number of rules and exceptions required.

For this project the student developed a LSTM encoder-decoder model with an attention mechanism and softmax activa to realise machine translation.

1.1 OBJECTIVES

Following are the main objectives of this project:

- To obtain an understanding of Deep Learning and Recurrent Neural Networks (RNNs), and how they are used to carry out translation between languages.
- Build a machine translation system by developing and training a deep learning RNN architecture, and see how it performs on a machine with standard processing power.
- Play around with the hyperparameters, training data size, number of training steps and compare the accuracy of the results for different combinations.

1.2 ASSUMPTIONS

The project relied heavily on the hardware and software support provided by the computer he student had access to. The computer used for running the project had the following specifications:

- Running Ubuntu 16.04 with the latest drivers and packages to provide development environment support for the implementation.
- NVIDIA GTX 970 GPU and 6GB RAM to provide hardware support for the implementation.
- Access to internet to download publicly available open source movie subtitle files for training and cross-validation process.
- Anaconda Distribution Version 5.1 for building the framework for the model and ease of adding modules in python.

1.3 RATIONALE

A significant reason why the student chose this project was because of his past experience with machine learning during his co-op semester. Working on deep learning models during the co-op experience gave him an exposure to deep learning. This project gave the student the opportunity to go deeper into this field and get to know more about technologies like RNNs and attention mechanisms. The machine translation system was designed using RNNs which requires a knowledge of variety of engineering disciplines with a heavy focus on AI, software, and mathematics. Being keen on this project the student took up modules relating to these subject to use the theory learned in his Aritficial Intelligenge modules.

Deep learning today provides a direction to implement ‘intellectual’ learning, the student considers it a small step towards AI.

2. EARLIER TECHNIQUES AND THEIR COMPARISON

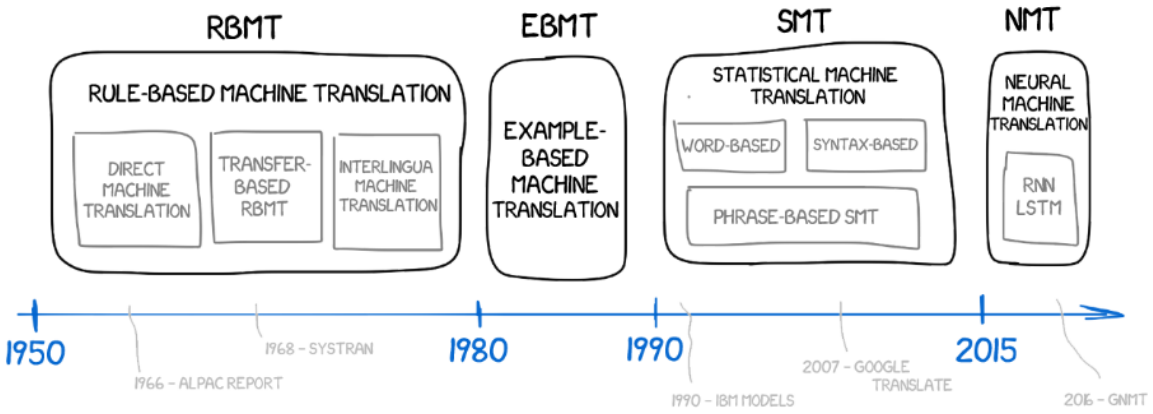


Figure 2: History of Machine Translation [3]

2.1 RULE-BASED MACHINE TRANSLATION (RBMT)

The first ideas surrounding rule-based machine translation appeared in the 70s. The systems used for this kind of machine translation used:

- Bilingual dictionary
- A set of linguistic rues for each language

If needed, the systems were supplemented with hacks, such as lists of names, spelling correctors, and transliterators. PROMPT and Systran are the most famous examples of RBMT systems. [3]

RBMT systems are capable of producing translations with reasonable quality, but constructing the system is very time-consuming and labor-intensive because such linguistic resources need to be hand-crafted, frequently referred to as knowledge acquisition problem. Moreover, it is of great difficulty to correct the input or add new rules to the system to generate a translation.

2.1.1 DIRECT MACHINE TRANSLATION

The simplest approach to machine translation is to replace every word in a sentence with the translated word in the target language. This is easy to implement because it uses a dictionary to look up each word and its translation in the target language, however, the results are poor because it ignores grammar and context. [4]

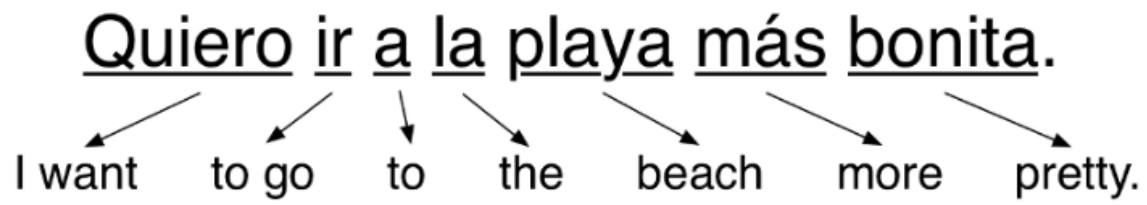


Figure 3: Word by word translation from English to Spanish [4]

Words of the source language are translated without passing through an additional/intermediary representation. The analysis of source language texts is oriented to only one target language. Direct translation systems are basically bilingual and uni-directional. Direct translation approach needs only a little syntactic and semantic analysis. Source language analysis is oriented specifically to the production of representations appropriate for one particular-target language. Direct machine translation is a word-by-word translation approach with some simple grammatical adjustments. [5]

2.1.2 TRANSFER-BASED MACHINE TRANSLATION

Transfer-based machine translation creates a translation from an intermediate representation that simulates the meaning of the original sentence. It depends partially on the language pair involved in the translation. On the basis, of the structural differences between the source and target language, a transfer system can be broken down into three different stages:

i) Analysis,
ii) Transfer, and
iii) Generation.

In the first stage, the source language parser is used to produce the syntactic representation of a sentence. In the next stage, the result of the first stage is converted into equivalent target language oriented representations. In the final step of this translation approach, a morphological analyzer is used to generate the final output texts. It is possible with this translation approach to obtain fairly high-quality translations, with accuracy in the region of 90%. [5]

In practice, it still resulted in verbatim translation and exhausted linguists. On the one hand, it brought simplified general grammar rules. But on the other, it became more complicated because of the increased number of word constructions in comparison with single words. [4]

2.1.3 Interlingual Machine Translation

In this method, the source text is transformed to the intermediate representation, and is unified for all the world's languages (interlingua). The target language is then generated out of the interlingua. One of the major advantages of this system is that the interlingua becomes more valuable as the amount of target languages it can be turned into increases. However, the only interlingual machine translation system that has been made operational at the commercial level is the KANT system (Nyberg and Mitamura, 1992), which is designed to translate Caterpillar Technical English (CTE) into other languages. The interlingua approach is clearly most attractive for multilingual systems. Each analysis module can be independent, both of all other analysis modules and of all generation modules. [4-5]

It is extremely hard to create such universal interlingua. Even if anyone were to succeed in creating an ideal RBMT, and linguists enhanced it with all the spelling rules, there would always be some exceptions: all the irregular verbs in English, separable prefixes in German, suffixes in Russian, and situations when people just say it differently. Any attempt, to take-into account all the nuances would waste millions of man hours.

2.2 EXAMPLE BASED MACHINE TRANSLATION

Example-based machine translation (EBMT) is characterized by its use of bilingual corpus with parallel texts as its main knowledge, in which translation by analogy is the main idea. An EBMT system is given a set of sentences in the source language (from which one is translating) and corresponding translations of each sentence in the target language with point to point mapping. These examples are used to translate similar types of sentences of source language to the target language. There are four tasks in EBMT: example acquisition, example base and management, example application and synthesis. At the foundation of example-based machine translation is the idea of translation by analogy. The principle of translation by analogy is encoded to example-based machine translation through the example translations that are used to train such a system. [5]

English	Japanese
How much is that **red umbrella**?	Ano **akai kasa** wa ikura desu ka.
How much is that **small camera**?	Ano **chiisai kamera** wa ikura desu ka.

Figure 4: Example of bilingual corpus

2.3 STATISTICAL MACHINE LEARNING

In early 1990, at the IBM Research Centre, a machine translation system was first shown which knew nothing about rules and linguistics as a whole. It analysed similar texts in two languages and tried to understand the patterns.

Statistical machine translation (SMT) is generated on the basis of statistical models whose parameters are derived from the analysis of bilingual text corpora. The initial model of SMT, based on Bayes Theorem, takes the view that every sentence in one language is a possible translation of any sentence in the other and the most appropriate is the translation that is assigned the highest probability by the system. The idea behind SMT comes from information theory. This method was much more efficient and accurate than all the previous machine translation methods. And no linguists were needed. [6]

2.3.1 WORD-BASED SMT

The first statistical translation system worked by splitting the sentence into words. The model translates one word into multiple words and counts stats. It memorizes the usual place the word takes at the output sentence and shuffles the word for the more natural sound at the intermediate step.

Despite their revolutionary nature, word-based systems still failed to deal with cases, gender, and homonymy. Every single word was translated in a single-true way, according to the machine. Such systems are not used anymore, they have been replaced by the more advanced phase-based methods. [7-8]

2.3.2 SYNTAX-BASED SMT

It’s necessary to do quite a precise syntax analysis of the sentence—to determine the subject, the predicate, and other parts of the sentence, and then to build a sentence tree. Syntax-based translation is based on the idea of translating syntactic units, rather than single words or strings of words (as in phrase-based MT), i.e. (partial) parse trees of sentences/utterances. Using it, the machine learns to convert syntactic units between languages and translates the rest by words or phrases. [7-8]

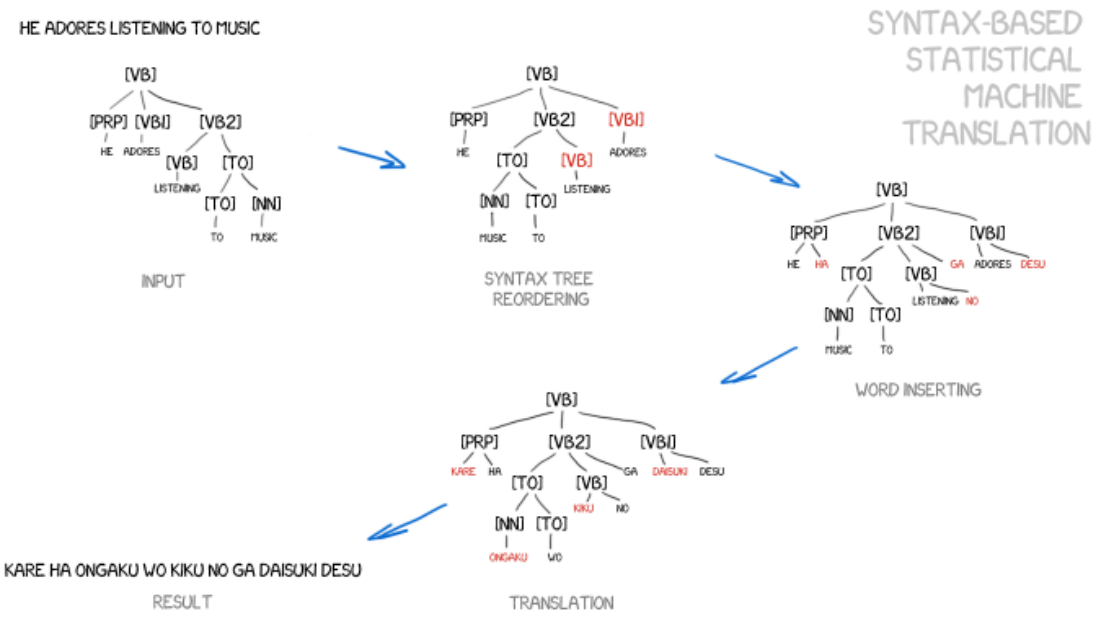


Figure 5: Example of Syntax-based Machine Translation [6]

2.3.3 PHRASE-BASED SMT

This method is based on all the word-based translation principles: statistics, reordering, and lexical hacks. Although, for the learning, it split the text not only into words but also phrases. These were the n-grams, to be precise, which were a contiguous sequence of n words in a row.

Thus, the machine learned to translate steady combinations of words, which noticeably improved accuracy.

Besides improving accuracy, the phrase-based translation provided more options in choosing the bilingual texts for learning. For the word-based translation, the exact match of the sources was critical, which excluded any literary or free translation. The phrase-based translation had no problem learning from them. To improve the translation, researchers even started to parse the news websites in different languages for that purpose. Starting in 2006, everyone began to use this approach. Google Translate, Yandex, Bing, and other high-profile online translators worked as phrase-based right up until 2016. [6-8]

3. DEEP NEURAL MACHINE TRANSLATION

Neural machine translation (NMT) is an approach to machine translation that uses a large artificial neural network to predict the likelihood of a sequence of words, typically modelling entire sentences in a single integrated model.

Deep learning applications appeared first in speech recognition in the 1990s. The first scientific paper on using neural networks in machine translation appeared in 2014, followed by a lot of advances in the following few years. (Large-vocabulary NMT, application to Image captioning, Subword-NMT, Multilingual NMT, Multi-Source NMT, Character-dec NMT, Zero-Resource NMT, Google, Fully Character-NMT, Zero-Shot NMT in 2017). [9]

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design where they have produced results comparable to and in some cases superior to human experts. [10]

A deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers between the input and output layers. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modelling complex data with fewer units than a similarly performing shallow network. Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains. It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets. DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back.

Recurrent neural networks (RNNs), in which data can flow in any direction, are used for applications such as language modelling. Long short-term memory (LSTM) is particularly effective for this use. Convolutional deep neural networks (CNNs) are used in computer vision. CNNs also have been applied to acoustic modelling for automatic speech recognition (ASR). [10-11]

Deep Neural Networks (DNNs) have provably enhanced the state-of-the-art Neural Machine Translation (NMT) with their capability in modelling complex functions and capturing complex linguistic structures. However, NMT systems with deep architecture in their encoder or decoder RNNs

often suffer from severe gradient diffusion due to the non-linear recurrent activations, which often make the optimization much more difficult. [12]

The solution is the use of an attention mechanism that allows the model to learn where to place attention on the input sequence as each word of the output sequence is decoded. The encoder-decoder recurrent neural network architecture with attention is currently the state-of-the-art on some benchmark problems for machine translation. And this architecture is used in the heart of the Google Neural Machine Translation system, or GNMT, used in their Google Translate service. Although effective, the neural machine translation systems still suffer some issues, such as scaling to larger vocabularies of words and the slow speed of training the models. There are the current areas of focus for large production neural translation systems, such as the Google system. NMT systems are known to be computationally expensive both in training and in translation inference. Also, most NMT systems have difficulty with rare words. These issues have hindered NMT's use in practical deployments and services, where both accuracy and speed are essential. [1]

Google's Neural Machine Translation system, attempts to address many of these issues. Their model consists of a deep LSTM network, with 8 encoder and 8 decoder layers using attention and residual connections. To improve parallelism and therefore decrease training time, their attention mechanism connects the bottom layer of the decoder to the top layer of the encoder. To accelerate the final translation speed, they employ low-precision arithmetic during inference computations. To improve handling of rare words, they divide words into a limited set of common sub-word units ("wordpieces") for both input and output. This method provides a good balance between the flexibility of "character"-delimited models and the efficiency of "word"-delimited models, naturally handles translation of rare words, and ultimately improves the overall accuracy of the system. Their beam search technique employs a length-normalization procedure and uses a coverage penalty, which encourages generation of an output sentence that is most likely to cover all the words in the source sentence. On the WMT'14 English-to-French and English-to-German benchmarks, GNMT achieves competitive results to state-of-the-art. Using a human side-by-side evaluation on a set of isolated simple sentences, it reduces translation errors by an average of 60% compared to Google's phrase-based production system. [13]

3.1 FEED-FORWARD NETWORKS

These networks are named after the way they channel information through a series of mathematical operations performed at the nodes of the network. They feed information straight through (never touching a given node twice). The input examples are fed to the network and transformed into an output; with supervised learning, the output would be a label, a name applied to the input. That is, they map raw data to categories, recognizing patterns that signal, for example, that an input image should be labeled “cat” or “elephant.”

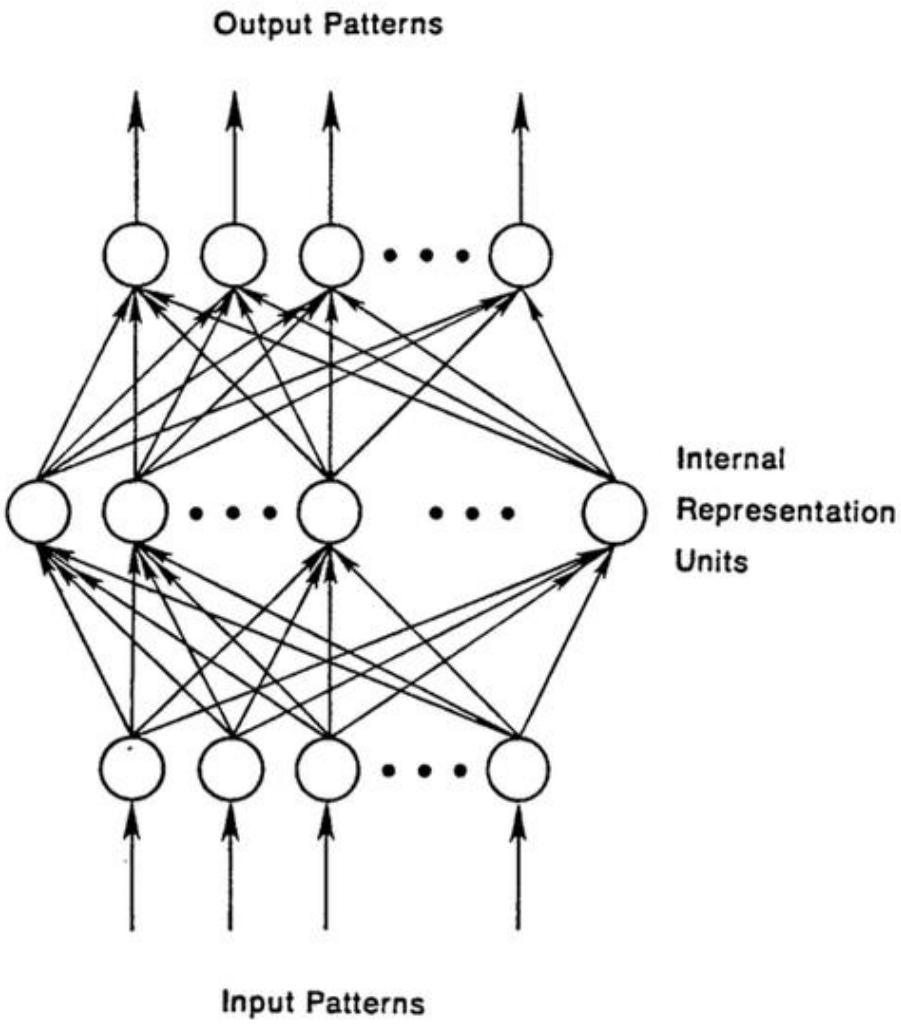


Figure 6: A feed forward neural network

A feedforward network is trained on labeled images until it minimizes the error it makes when guessing their categories. With the trained set of parameters (or weights, collectively known as a model), the network sallies forth to categorize data it has never seen. A trained feedforward network can be exposed to any random collection of photographs, and the first photograph it is exposed to will not necessarily alter how it classifies the second. Seeing photograph of a cat will not lead the net to perceive an elephant next.

That is, a feedforward network has no notion of order in time, and the only input it considers is the current example it has been exposed to. [14]

3.2 Recurrent Neural Networks (RNNs)

Recurrent networks, on the other hand, take as their input not just the current input example they see, but also what they have perceived previously in time.

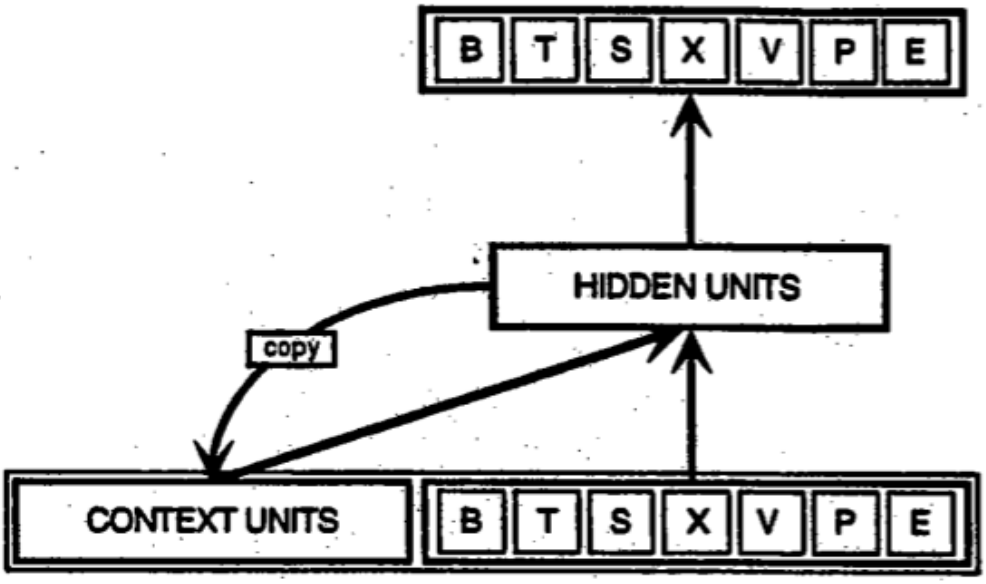


Figure 7: Simple Recurrent Network proposed by Elman [15]

Fig 7 is a diagram of an early, simple recurrent net proposed by Elman, where the BTSXPE at the bottom of the drawing represents the input example in the current moment, and CONTEXT UNIT represents the output of the previous moment. The decision a recurrent net reached at time step t-1 affects the decision it will reach one moment later at time step t. So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data, much as we do in life.

Recurrent networks are distinguished from feedforward networks by that feedback loop connected to their past decisions, ingesting their own outputs moment after moment as input. It is often said that recurrent networks have memory Adding memory to neural networks has a purpose: There is information in the sequence itself, and recurrent nets use it to perform tasks that feedforward networks cannot.

That sequential information is preserved in the recurrent network's hidden state, which manages to span many time steps as it cascades forward to affect the processing of each new example. It is finding correlations between events separated by many moments, and these correlations are called "long-term dependencies", because an event downstream in time depends upon, and is a function of, one or more events that came before. One way to think about RNNs is this: they are a way to share weights over time. [14]

The activation of the hidden states at timestep t is computed as a function f of the current input symbol xt and the previous hidden states

ht−1: ht =f (xt, ht−1)

It is common to use the state-to-state transition function f as the composition of an element-wise nonlinearity with an affine transformation of both xt and

ht−1: ht =φ (Wxt + Uht−1)….. (1)

where W is the input-to-hidden weight matrix, U is the state-to-state recurrent weight matrix, and φ is usually a logistic sigmoid function or a hyperbolic tangent function. We can factorize the probability of a sequence of arbitrary length into

p(x1, · · · , xT) = p(x1)p(x2 | x1)· · · p(xT | x1, · · · , xT−1)

Then, we can train an RNN to model this distribution by letting it predict the probability of the next symbol xt+1 given hidden states ht which is a function of all the previous symbols x1, · · · , xt−1 and current symbol xt: p(xt+1 | x1, · · · , xt) = g (ht).

This approach of using a neural network to model a probability distribution over sequences is widely used. [16]

3.2.1 ENCODINGS

When we are trying to tell two faces apart with a computer, we collect different measurements from each face and use those measurements to compare faces. For example, we might measure the size of each ear or the spacing between the eyes and compare those measurements from two pictures to see if they are the same person. The idea of turning a face into a list of measurements is an example of an encoding. We are taking raw data (a picture of a face) and turning it into a list of measurements that represent it (the encoding). we don't have to come up with a specific list of facial features to measure ourselves. Instead, we can use a neural network to generate measurements from a face. It lets us represent something very complicated (a picture of a face) with something simple (128 numbers). Now comparing two different faces is much easier because we only have to compare these 128 numbers for each face instead of comparing full images. [3]

We can do the same for sentences and use them for machine translation. To generate this encoding, we feed the sentence into the RNN, one word at time. The final result, after the last word is processed will be the values that represent the entire sentence. [14]

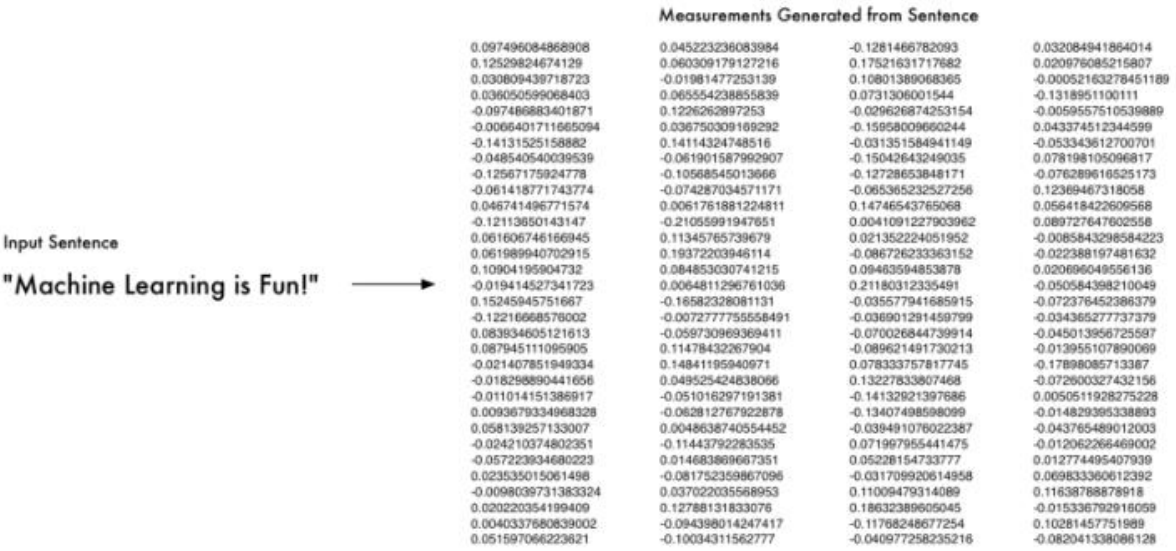


Figure 8: Example of encoding [3]

3.2.2 Vanishing (and Exploding) Gradients

By the early 1990s, the vanishing gradient problem emerged as a major obstacle to recurrent net performance.

Just as a straight line expresses a change in x alongside a change in y, the gradient expresses the change in all weights with regard to the change in error. If we do not know the gradient, we can't adjust the weights in a direction that will decrease error, and our network ceases to learn. Recurrent nets seeking to establish connections between a final output and events many time steps before were hobbled, because it is very difficult to know how much importance to accord to remote inputs.

This is partially because the information flowing through neural nets passes through many stages of multiplication. Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing or exploding.

Figure 9 represents the effects of applying the sigmoid function again and again. The data is flattened until, for large stretches, it has no detectable slope. This is analogous to a gradient vanishing as it passes through many layers. [14]

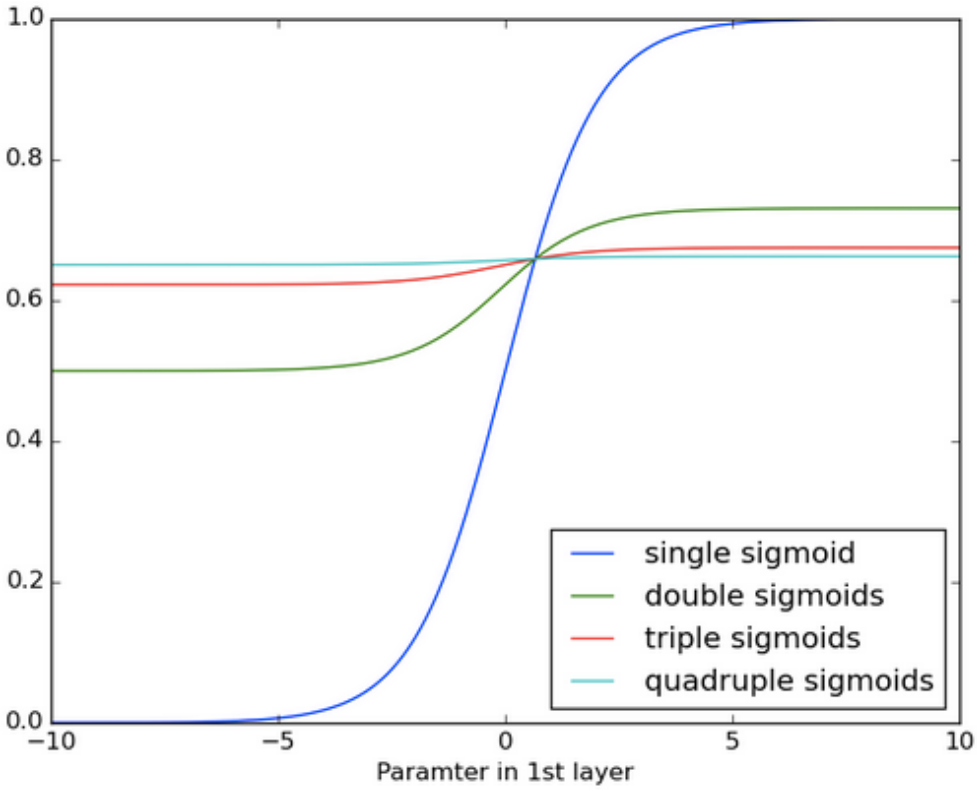


Figure 9: Effects of applying a sigmoid function [14]

3.2.3 Backpropagation Through Time (BPTT)

Backpropagation in feedforward networks moves backward from the final error through the outputs, weights and inputs of each hidden layer, assigning those weights responsibility for a portion of the error by calculating their partial derivatives – $\partial E/\partial w$, or the relationship between their rates of change. Those derivatives are then used by our learning rule, gradient descent, to adjust the weights up or down, whichever direction decreases error.

Recurrent networks rely on an extension of backpropagation called backpropagation through time, or BPTT. Time, in this case, is simply expressed by a well-defined, ordered series of calculations linking one time step to the next, which is all backpropagation needs to work.

Neural networks, whether they are recurrent or not, are simply nested composite functions like f(g(h(x))). Adding a time element only extends the series of functions for which we calculate derivatives with the chain rule. [14]

3.2.4 Truncated BPTT

Truncated BPTT, is an approximation of full BPTT that is preferred for long sequences, since full BPTT's forward/backward cost per parameter update becomes very high over many time

steps. The downside is that the gradient can only flow back so far due to that truncation, so the network can't learn dependencies that are as long as in full BPTT.

3.3 Long-Short Term Memory (LSTM)

In the mid-90s, a variation of recurrent net with so-called Long Short-Term Memory units, or LSTMs, was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem.

LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely.

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation.

Those gates act on the signals they receive, and similar to the neural network's nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent. [14]

There are several architectures of LSTM units. A common architecture is composed of a memory cell, an input gate, an output gate and a forget gate.

An LSTM (memory) cell stores a value (or state), for either long or short time periods. This is achieved by using an identity (or no) activation function for the memory cell. In this way, when an LSTM network (that is an RNN composed of LSTM units) is trained with backpropagation through time, the gradient does not tend to vanish.

The LSTM gates compute an activation, often using the logistic function. Intuitively, the input gate controls the extent to which a new value flows into the cell, the forget gate controls the

extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

There are connections into and out of these gates. A few connections are recurrent. The weights of these connections, which need to be learned during training, of an LSTM unit are used to direct the operation of the gates. Each of the gates has its own parameters, that is weights and biases, from possibly other units outside the LSTM unit. [17]

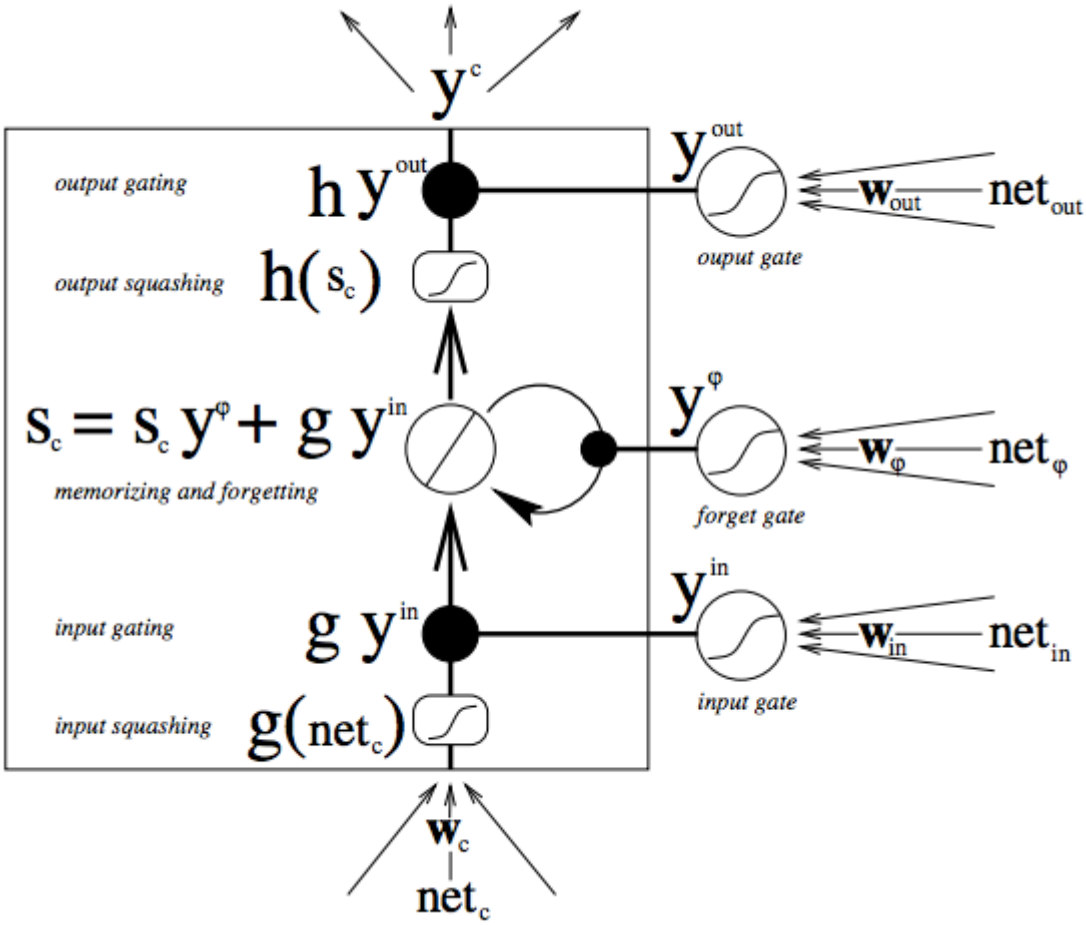


Figure 10: Data flow through a memory cell [14]

Starting from the bottom, the triple arrows show where information flows into the cell at multiple points. That combination of present input and past cell state is fed not only to the cell itself, but also to each of its three gates, which will decide how the input will be handled.

The black dots are the gates themselves, which determine respectively whether to let new input in, erase the present cell state, and/or let that state impact the network's output at the present time step. S_c is the current state of the memory cell, and g_y_in is the current input to it. Each gate can be open or shut, and they will recombine their open and shut states at each step. The cell can forget its state, or not; be written to, or not; and be read from, or not, at each time step,

and those flows are represented in figure 10. The large bold letters give us the result of each operation.

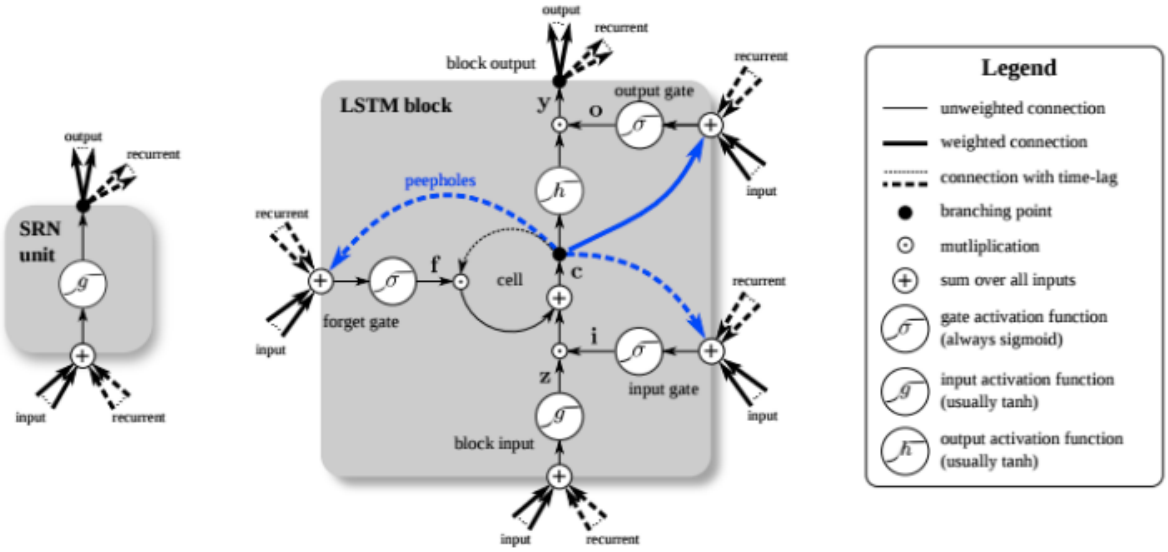


Figure 11: Detailed schematic of a RNN unit (left) and a LSTM block (right) as used in the hidden layers of a recurrent neural network [14]

LSTMs' memory cells give different roles to addition and multiplication in the transformation of input. The central plus sign in both diagrams is essentially the secret of LSTMs. This basic change helps them preserve a constant error when it must be backpropagated at depth. Instead of determining the subsequent cell state by multiplying its current state with new input, they add the two, and that quite literally makes the difference. (The forget gate still relies on multiplication)

Different sets of weights filter the input for input, output and forgetting. The forget gate is represented as a linear identity function, because if the gate is open, the current state of the memory cell is simply multiplied by one, to propagate forward one more time step.

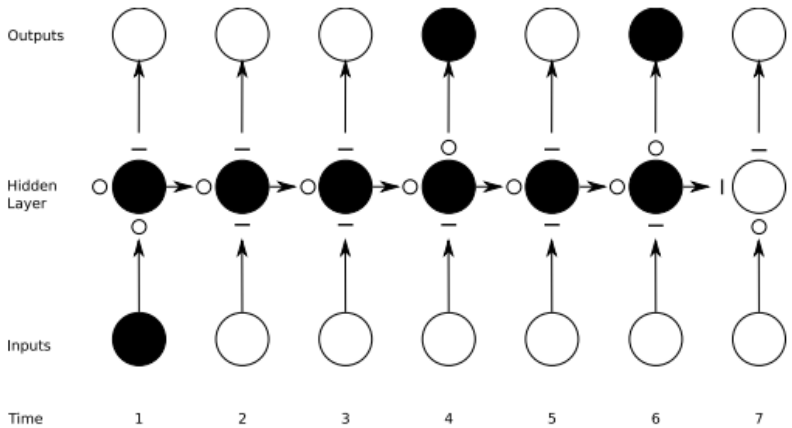


Figure 12: Working of the gates

In figure 12, we can see the gates at work, with straight lines representing closed gates, and blank circles representing open ones. The lines and circles running horizontal down the hidden layer are the forget gates It should be noted that while feedforward networks map one input to one output, recurrent nets can map one to many, as above (one image to many words in a caption), many to many (translation), or many to one (classifying a voice). [14]

Many applications use stacks of LSTM RNNs and train them by Connectionist Temporal Classification (CTC) to find an RNN weight matrix that maximizes the probability of the label sequences in a training set, given the corresponding input sequences. CTC achieves both alignment and recognition.

LSTM can learn to recognize context-sensitive languages unlike previous models based on hidden Markov models (HMM) and similar concepts. [18]

4. SOFTWARE SETUP

For information on hardware setup please look at the Assumptions in section 1.2 on the report. The student used a few softwares for the realisation of this project which made bug-fixing and handling easier for him. They are mentioned in the following sections of the report.

4.1 ANACONDA (PYTHON)

Anaconda is a freemium open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

The student set up an anaconda python 2.7 framework on his ubuntu system to kick start his project work.

The student likes Anaconda Python most for studying Data Science is because it comes with many powerful packages and data scientists in the open-source community seem to never stop contributing more to Python packages. While traditional Python provides just a basic platform where you have to install your desired packages manually (this even does not have NumPy and Pandas installed), Anaconda gives you just everything. It has the most useful packages for Mathematics, Science and Engineering already installed. [19-20]

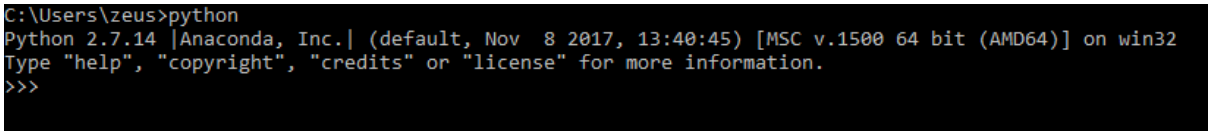


Figure 13: Snippet of Anaconda Python on Student's machine

4.2 ATOM

Atom is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in Node.js, and embedded Git Control, developed by GitHub. Atom is a desktop application built using web technologies. Most of the extending packages have free software licenses and are community-built and maintained. Atom is based on Electron (formerly known as Atom Shell), a framework that enables cross-platform desktop applications using Chromium and Node.js.

It can also be used as an integrated development environment (IDE).

Student used Atom as the preferred IDE due to its seamless integration services with github and rich python packages.

4.3 PYTHON LIBRARIES USED

4.3.1 TENSORFLOW

TensorFlow is an open source software library for numerical computation using data-flow graphs. It was originally developed by the Google Brain Team within Google's Machine Intelligence research organization for machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. TensorFlow is cross-platform. It runs on nearly everything: GPUs and CPUs—including mobile and embedded platforms—and even tensor processing units (TPUs), which are specialized hardware to do tensor math on.

The TensorFlow distributed execution engine abstracts away the many supported devices and provides a high performance-core implemented in C++ for the TensorFlow platform. On top of that sit the Python and C++ frontends. The Layers API provides a simpler interface for commonly used layers in deep learning models. On top of that sit higher-level APIs, including Keras and the Estimator API, which makes training and evaluating distributed models easier. [21]

The student used tensorflow libraries to design the deep learning model as well as to plot to test it using tensorboard.

4.3.2 NUMPY

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. [22]

4.3.3 SCIKIT-LEARN

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

It is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings. [23]

4.3.4 DATA-UTILS

Student used the tensorflow data-utils class to preprocess the training and testing sets for the project. The data-utils class takes in sentences in moses format and produces a .pkl file that was then be used to train the model.

4.3.5 MATPLOTLIB.PYPLOT

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be

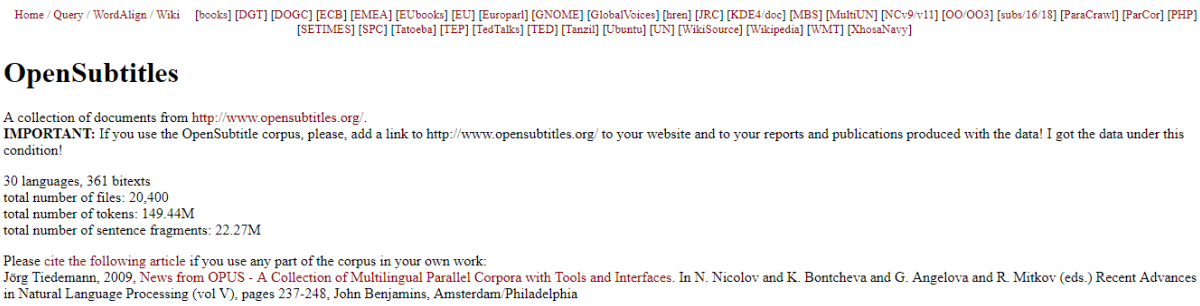
used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. [24]

The loss graphs were produced using this library.

4.3.6 nltk.translate.bleu_score

This library was imported to calculate the bleu score for the translated sentences. It compares the translated sentences to a set of reference sentences to generate and output the bleu scores.

5. OPEN-SOURCE SUBTITLES FOR TRAINING AND TESTING


Home / Query / WordAlign / Wiki [books] [DGT] [DOGC] [ECB] [EMEA] [EUbooks] [EU] [Europarl] [GNOME] [GlobalVoices] [hren] [JRC] [KDE4/doc] [MBS] [MultiUN] [NCv9/v11] [OO/OO3] [subs/16/18] [ParaCrawl] [ParCor] [PHP] [SETIMES] [SPC] [Tatoeba] [TEP] [TedTalks] [TED] [Tanzil] [Ubuntu] [UN] [WikiSource] [Wikipedia] [WMT] [XhosaNavy]

OpenSubtitles

A collection of documents from http://www.opensubtitles.org/.
IMPORTANT: If you use the OpenSubtitle corpus, please, add a link to http://www.opensubtitles.org/ to your website and to your reports and publications produced with the data! I got the data under this condition!

30 languages, 361 bitexts
total number of files: 20,400
total number of tokens: 149.44M
total number of sentence fragments: 22.27M

Please cite the following article if you use any part of the corpus in your own work:
Jörg Tiedemann, 2009, News from OPUS - A Collection of Multilingual Parallel Corpora with Tools and Interfaces. In N. Nicolov and K. Bontcheva and G. Angelova and R. Mitkov (eds.) Recent Advances in Natural Language Processing (vol V), pages 237-248, John Benjamins, Amsterdam/Philadelphia

Figure 14: Front page of open subtitles website

The student procured the testing and training datasets for the different languages from http://opus.nlpl.eu/OpenSubtitles.php

The English to German corpus has 67k sentence pairs in moses format.

The English to Spanish corpus has 506k sentence pairs in moses format.

The English to Arabic corpus has 35k sentence pairs in moses format.

```
Wenn Sie Fahrkarten kaufen wollen, müssen Sie zur Vorderseite des...
Na schön, ich denke, dieses Mal wird es gehen.
Was mache ich hier eigentlich?
Die kommen hier herein und...
Mal sehen.
Ich hoffe, ich habe...
Drei?
Das macht sieben Dollar...
.. und 50 Cent.
Nein.
Frank?
Frank hat uns geschickt.
Habt ihr ein Pferd für mich mitgebracht?
Sieht aus, als ob wir..... ein Pferd zu wenig hätten.
Es sind zwei Pferde zu viel.
Pa!
Sieh mal!
Das reicht.
Es wird spät.
Gehen wir heim.
Timmy.
Maureen, sieh mal.
Was machst du hier?
Geh schnell rein und wasch dich.
Geh nicht an den Apfelkuchen und den Braten.
Ist Patrick schon zum Bahnhof unterwegs?
Er fährt gleich los, Pa.
- Verdammt, Patrick!
- lch komm schon.
Nicht schlecht.
Größere Scheiben.
Was soll' s?
Wir geben schließlich ein Fest, oder?
Aber das sind die gleichen Scheiben wie immer.
Ja, stimmt.
```

Figure 15: First 35 sentences in Training set for english-german (German)

```
If you want any tickets, you' il have to go round to the front of...
Well, I suppose it' il be alright.
What the hell am I doing around here?
They walk in here and...
Let' s see.
I hope I got...
Three?
That' il be seven dollars...
.. and 50 cents.
No.
Frank?
Frank sent us.
Did you bring a horse for me?
Looks like we' re shy of one horse.
You brought two too many.
Pa!
Look!
That' s enough for now.
It' s getting late.
Come on home.
Timmy.
Maureen, look.
What you doing there?
Go inside, quick, and get washed.
And don' t touch the apple pie or the roast.
Patrick' s already left for the station.
He' s getting ready, Pa.
- Damn it, Patrick!
- Coming, Pa.
Not bad, I' d say.
Bigger, them slices.
What the hell?
We' re throwing a party, ain' t we?
But these are the same slices as usual.
Yeah, sure.
```

Figure 16: First 35 sentences in Training set for english-german (English)

```
Bien, entonces, adelante, a la oficina...
No pierdas el tiempo por el camino.
No te preocupes.
Te llamaré a la oficina para ver a qué hora llegas.
Oh, hola Fujio.
- Si
¿Por qué estás fuera?
No es divertido entrar ahí con mi madre.
¿Por qué no entramos juntos?
Prefiero estar aquí afuera.
Bien, espera afuera.
Dile a mamá que si ella no sale pronto..... me vuelvo a casa solo.
De acuerdo.
Ishino ha llegado.
Bien.
Beinvenida.
Por favor, pasa.
Espero que estés bien.
¿Estás aquí hace mucho?
Un rato.
Estás muy bonita hoy.
Estás estupenda hoy.
Hoy estás realmente hermosa.
¿Sólo "hoy"?
¿Crees que siempre estás hermosa?.
¡Estúpida!
- Es la´ultima moda.
¿Lo compraste en Mitsukoshi?
Si, es demasiado sencillo. ¿verdad?
¿Sobrio ésto?
¿No lo encuentras sobrio?
¿Qué edad tienes?
Siempre lo estropeas todo.
Si te ríes así...
- Adivina.
```

Figure 17: First 35 sentences in Training set for english-spanish (Spanish)

```
Go straight to the office...
Don' t dawdle on the way
Don' t worry
I' il call in 30 minutes to check
Oh, hello Fujio
Is your mother here, too?
Why are you outside?
It' s no fun listening to women' s talk
Well, why don' t we go in together?
I' m fine out here
Right, you wait out here
If she' s not out soon, I' m leaving!
I' il tell her
Mrs. Ishino has arrived
Good
Welcome
Please come in
I hope you' re well
Been here long?
A little while
You' re very pretty, today
You look stunning, today
Today you' re really beautiful
What' s with the '"today- today'"?
She always thinks she' s beautiful
Cheeky!
That' s a lovely wrap!
You bought it at Mitsukoshi?
No, is it too earthy?
Earthy?
You don' t feel it is?
How old are you?
Old enough
Terrible, terrible
Isn' t that it?
```

Figure 18: First 35 sentences in Training set for english-spanish (English)

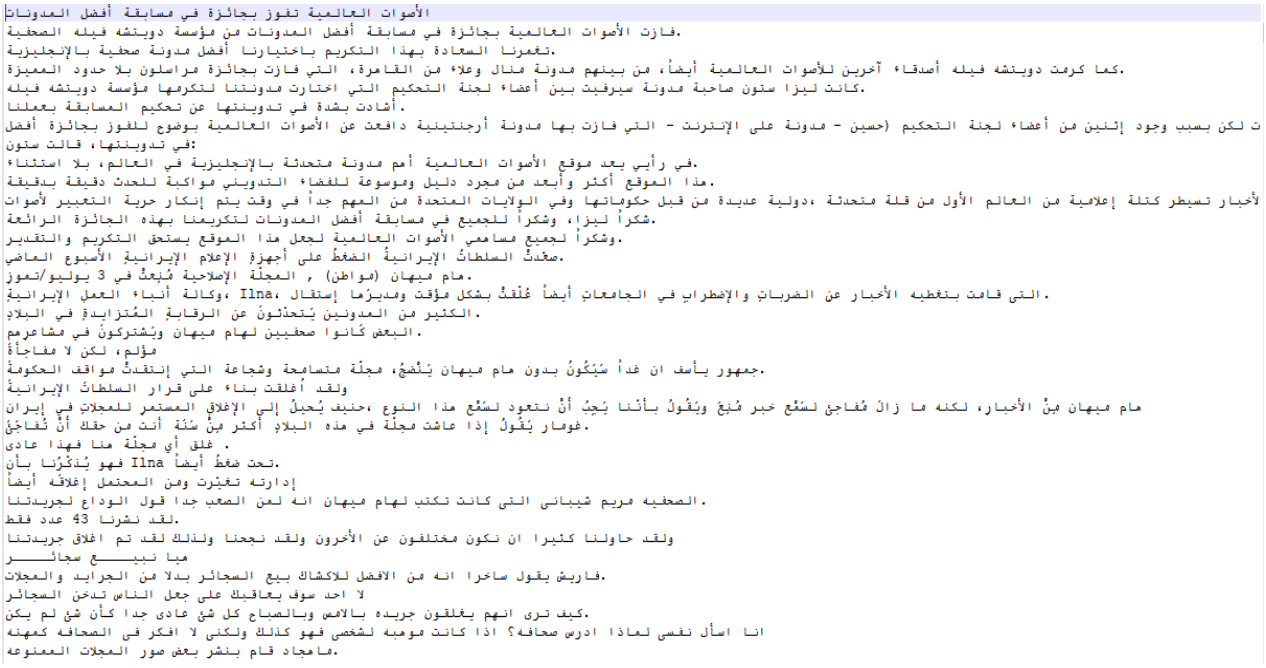


Figure 19: First 35 sentences in Training set for english-arabic (Arabic)

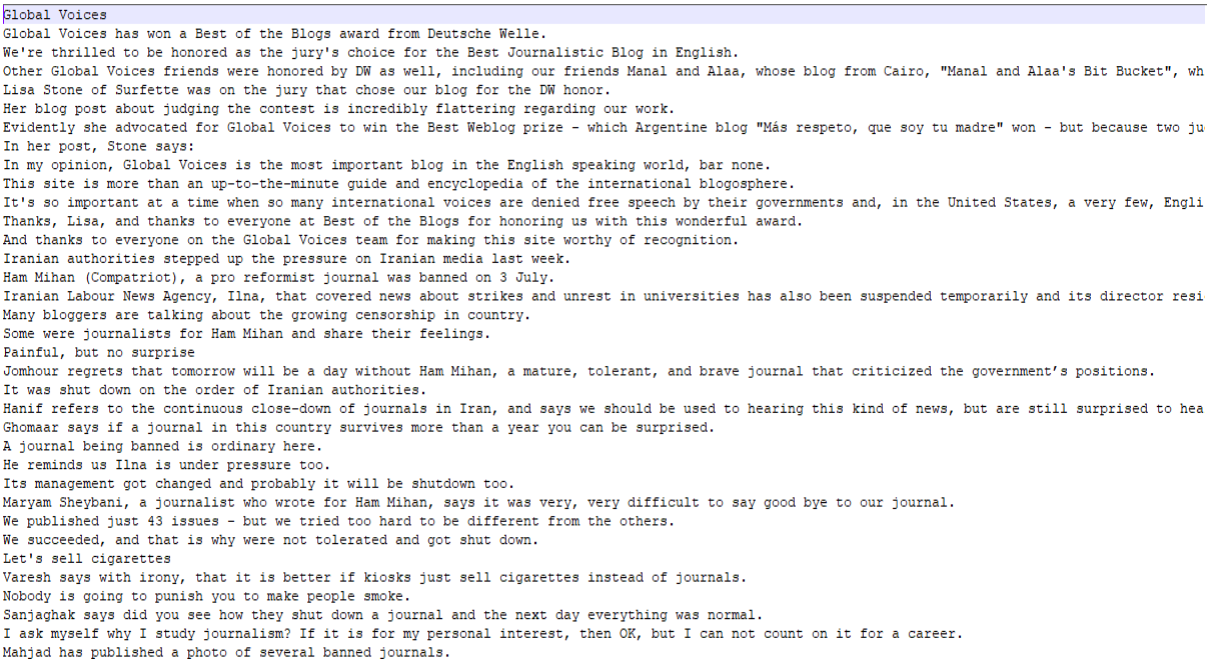


Figure 20: First 35 sentences in Training set for english-arabic (English)

6. IMPLEMENTATION

1. The student used python's cPickle module to write a data-utils.py script to process sentence pairs in English-German in moses format to produce a .pkl file which is then fed into the LSTM. It basically creates datasets from raw text files.

```
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
import time
import data_utils
import matplotlib.pyplot as plt
from nltk.translate.bleu_score import sentence_bleu
```

Figure 21: Screenshot of import block in the translator.py code

2. Then in the code the student defines the vocab size that is, the number of words that the model trains on from the dataset.

```
input_seq_len = 15
output_seq_len = 17
en_vocab_size = len(en_vocab) + 2 # + <pad>, <ukn>
de_vocab_size = len(de_vocab) + 4 # + <pad>, <ukn>, <eos>, <go>
```

Figure 22: screenshot of vocab size in code

3. Data-utils class to read data from data directory.

```
# read dataset
X, Y, en_word2idx, en_idx2word, en_vocab, de_word2idx, de_idx2word, de_vocab = data_utils.read_dataset('data.pkl')

# inspect data
print 'Sentence in English - encoded:', X[0]
print 'Sentence in German - encoded:', Y[0]
print 'Decoded:\n------------------------'
```

Figure 23: screenshot of usage of data-utils class

This returns formatted and tokenised words in both languages that is English and German.

4. Adding padding to the data before processing and splitting the data to training and testing sets.

```
def data_padding(x, y, length = 15):
    for i in range(len(x)):
        x[i] = x[i] + (length - len(x[i])) * [en_word2idx['<pad>']]
        y[i] = [de_word2idx['<go>']] + y[i] + [de_word2idx['<eos>']] + (length-len(y[i])) * [de_word2idx['<pad>']]

data_padding(X, Y)

# data splitting
X_train,  X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1)

del X
del Y
```

Figure 24: Code snippet for data padding and data splitting

5. Initialised tensorflow placeholders for encoder decoder inputs. Both are integer tensors which will are discrete values embedded into dense representation later.

```
# placeholders
encoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'encoder{}'.format(i)) for i in range(input_seq_len)]
decoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'decoder{}'.format(i)) for i in range(output_seq_len)]
```

Figure 25: Code snippet for tensorflow placeholders

6. Feed vocab into encoder and encoded representation into decoder
7. LSTM recurrent network to encode a language A. The RNN spits out a hidden state ‘s’ which represents the vectorised contents of the sentence.

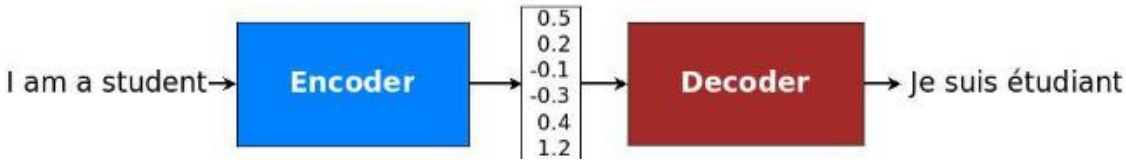


Figure 26: Representation of an LSTM encoder-decoder model

The code then feed ‘s’ to the decoder which will generate the translated sentence in language B, word by word. The drawback to this architecture is limited memory. The hidden state ‘s’ of the LSTM is where we’re trying to cram the whole sentence we want to translate. ‘s’ is usually only a few hundred floating points numbers long. The more we try to force our sentence into this fixed dimensionality vector, the more lossy the neural net becomes. Increasing the size of the hidden state in the LSTM increases the training time exponentially.

8. The student built an attention mechanism so that the neural net can store and refer to previous outputs of the LSTM. This changes the storage of the model without changing the functionality of the LSTM. This lets the LSTM look back at the source sentence to make sure all the details are getting captured, by iteratively paying attention to the relevant parts of the source sentence.

```
outputs, states = tf.contrib.legacy_seq2seq.embedding_attention_seq2seq(
                                                encoder_inputs,
                                                decoder_inputs,
                                                tf.contrib.rnn.BasicLSTMCell(size),
                                                num_encoder_symbols = en_vocab_size,
                                                num_decoder_symbols = de_vocab_size,
                                                embedding_size = 100,
                                                feed_previous = False,
                                                output_projection = output_projection,
                                                dtype = tf.float32)
```

Figure 277: Code snippent for the attention mechanism implementation

The student used a tensorflow embedding_attention wrapper to add the attention mechanism to his model.

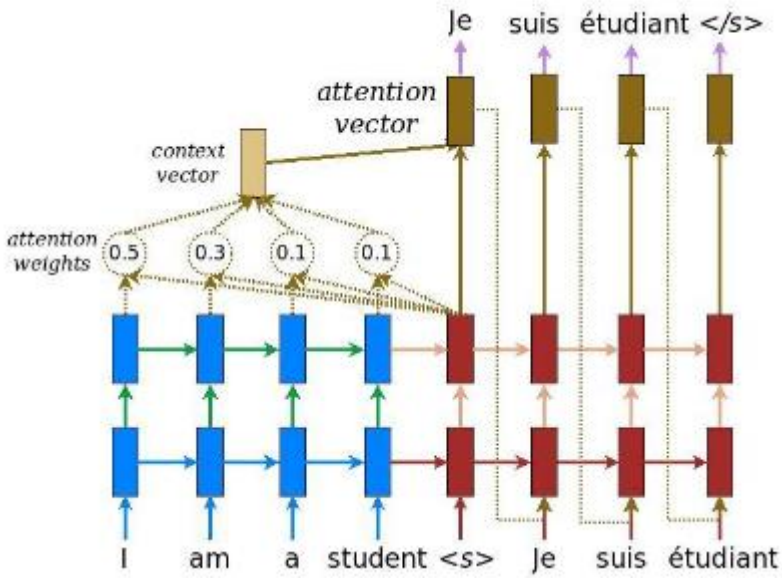


Figure 288: Example of an attention-based NMT system as described in (Luong et al., 2015) [25]

- The current target hidden state is compared with all source states to derive attention weights.
- Based on the attention weights we compute a context vector as the weighted average of the source states.
- Combine the context vector with the current target hidden state to yield the final attention vector
- The attention vector is fed as an input to the next time step (input feeding). The first three steps can be summarized by the equations below:

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

Figure 29: Equations for attention mechanism

Here, the function score is used to compare the target hidden state **ht** with each of the source hidden states **hs**, and the result is normalized to produced attention weights (a distribution over source positions). There are various choices of the scoring function; popular scoring functions include the multiplicative and additive forms. Once computed, the attention vector **at** is used to derive the softmax logit and loss. [25]

9. Once we have the outputs from the encoder LSTM stored we query each output asking how relevant they are to the computation happening in the decoder. Each encoder output gets a relevancy score which we can convert to a probability score by applying a softmax activation to it.

```
def sampled_loss(labels, logits):
    return tf.nn.sampled_softmax_loss(
                    weights = w_t,
                    biases = b,
                    labels = tf.reshape(labels, [-1, 1]),
                    inputs = logits,
                    num_sampled = 512,
                    num_classes = de_vocab_size)

# Weighted cross-entropy loss for a sequence of logits
loss = tf.contrib.legacy_seq2seq.sequence_loss(outputs, targets, target_weights, softmax_loss_function = sampled_loss)

# define helper functions

# simple softmax function
def softmax(x):
    n = np.max(x)
    e_x = np.exp(x - n)
    return e_x / e_x.sum()
```

Figure 30: Code Snippet for softmax implementation

10. Then we extract a context vector which is a weighted summation to the encoder outputs depending on how relevant they are.
11. Built the model using tensorflow built in embedding attention sequence to sequence function giving it our encoder and decoder inputs as well as a few hyper parameters like the number of layers.
12. Define the hyperparameters.

```
# ops and hyperparameters
learning_rate = 5e-3
batch_size = 64
steps = 2000
```

Figure 31: Hyperparameters

13. Decode the output sequence.

```
def decode_output(output_seq):
    words = []
    for i in range(output_seq_len):
        smax = softmax(output_seq[i])
        idx = np.argmax(smax)
        words.append(de_idx2word[idx])
    return words
```

Figure 32: Code for decoding output sequence

14. Training and plotting losses.

```
with tf.Session() as sess:
    sess.run(init)

    t = time.time()
    for step in range(steps):
        feed = feed_dict(X_train, Y_train)

        backward_step(sess, feed)

        if step % 5 == 4 or step == 0:
            loss_value = sess.run(loss, feed_dict = feed)
            print 'step: {}, loss: {}'.format(step, loss_value)
            losses.append(loss_value)

        if step % 20 == 19:
            saver.save(sess, 'checkpoints/', global_step=step)
            print 'Checkpoint is saved'

    print 'Training time for {} steps: {}s'.format(steps, time.time() - t)

# plot losses

with plt.style.context('fivethirtyeight'):
    plt.plot(losses, linewidth = 1)
    plt.xlabel('Steps')
    plt.ylabel('Losses')
    plt.ylim((0, 12))

plt.show()
```

Figure 33: Code for training data and plotting losses

15. Testing and translating the given sentences.

```
#Testing the model

with tf.Graph().as_default():

    # placeholders
    encoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'encoder{}'.format(i)) for i in range(input_seq_len)]
    decoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'decoder{}'.format(i)) for i in range(output_seq_len)]

    # output projection
    size = 512
    w_t = tf.get_variable('proj_w', [de_vocab_size, size], tf.float32)
    b = tf.get_variable('proj_b', [de_vocab_size], tf.float32)
    w = tf.transpose(w_t)
    output_projection = (w, b)

    # change the model so that output at time t can be fed as input at time t+1
    outputs, states = tf.contrib.legacy_seq2seq.embedding_attention_seq2seq(
                                            encoder_inputs,
                                            decoder_inputs,
                                            tf.contrib.rnn.BasicLSTMCell(size),
                                            num_encoder_symbols = en_vocab_size,
                                            num_decoder_symbols = de_vocab_size,
                                            embedding_size = 100,
                                            feed_previous = True, # <-----this is changed----->
                                            output_projection = output_projection,
                                            dtype = tf.float32)

    # ops for projecting outputs
    outputs_proj = [tf.matmul(outputs[i], output_projection[0]) + output_projection[1] for i in range(output_seq_len)]

    # Translate these sentences
    en_sentences = ["What' s your name", 'My name is', 'What are you doing', 'I am reading a book',\
                    'How are you','Please come to my house', 'I am good', 'Do you speak English', 'What time is it', 'Hi', 'Goodbye', 'Yes', 'No']
    en_sentences_encoded = [[en_word2idx.get(word, 0) for word in en_sentence.split()] for en_sentence in en_sentences]
```

Figure 34: Code for training and translation i.e. decoding

7. RESULTS

7.1 ENGLISH TO GERMAN

7.1.1 TRAINING

```
(tensorflow) zeus@zeus-Dell-System-XPS-L502X:~/Desktop/en-german$ python translat
Sentence in English - encoded: [109, 5, 869, 93, 38, 25, 2587]
Sentence in German - encoded: [168, 263, 8, 473, 267, 326, 68, 15, 132]
Decoded:
------------------------
They walk in here and

Die kommen hier herein und ------------------TRAINING-----------------
2018-03-20 01:42:32.445641: I tensorflow/core/platform/cpu_feature_guard.cc:137]
not compiled to use: SSE4.1 SSE4.2 AVX
step: 0, loss: 9.05614280701
step: 4, loss: 9.10534477234
step: 9, loss: 9.13997173309
step: 14, loss: 8.92261886597
step: 19, loss: 9.09201717377
Checkpoint is saved
step: 24, loss: 9.00710010529
step: 29, loss: 8.80250167847
step: 34, loss: 8.7556848526
step: 39, loss: 8.31876182556
Checkpoint is saved
step: 44, loss: 7.50344610214
step: 49, loss: 7.32426071167
step: 54, loss: 7.30140590668
step: 59, loss: 6.55706977844
Checkpoint is saved
step: 64, loss: 7.02557754517
step: 69, loss: 5.98360824585
step: 74, loss: 5.84747171402
step: 79, loss: 5.73983478546
Checkpoint is saved
step: 84, loss: 5.51460790634
step: 89, loss: 5.80535840988
step: 94, loss: 5.44519758224
```

Figure 35: Training log pg 1

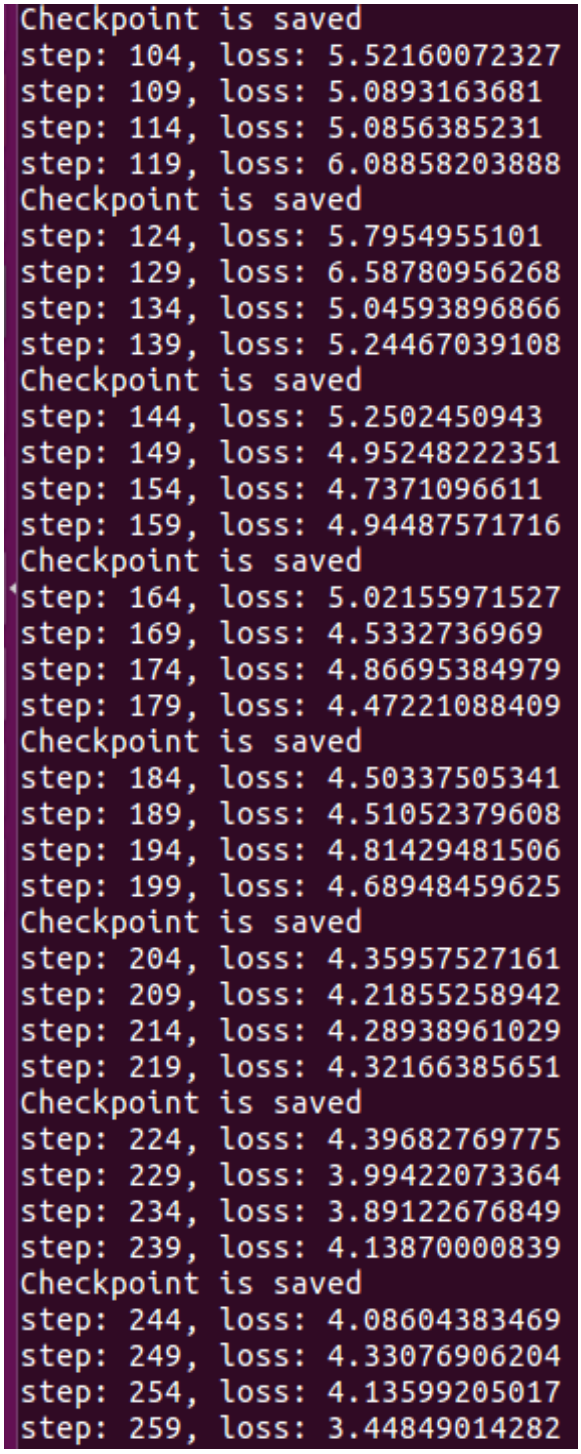


Figure 36: Training log pg 3

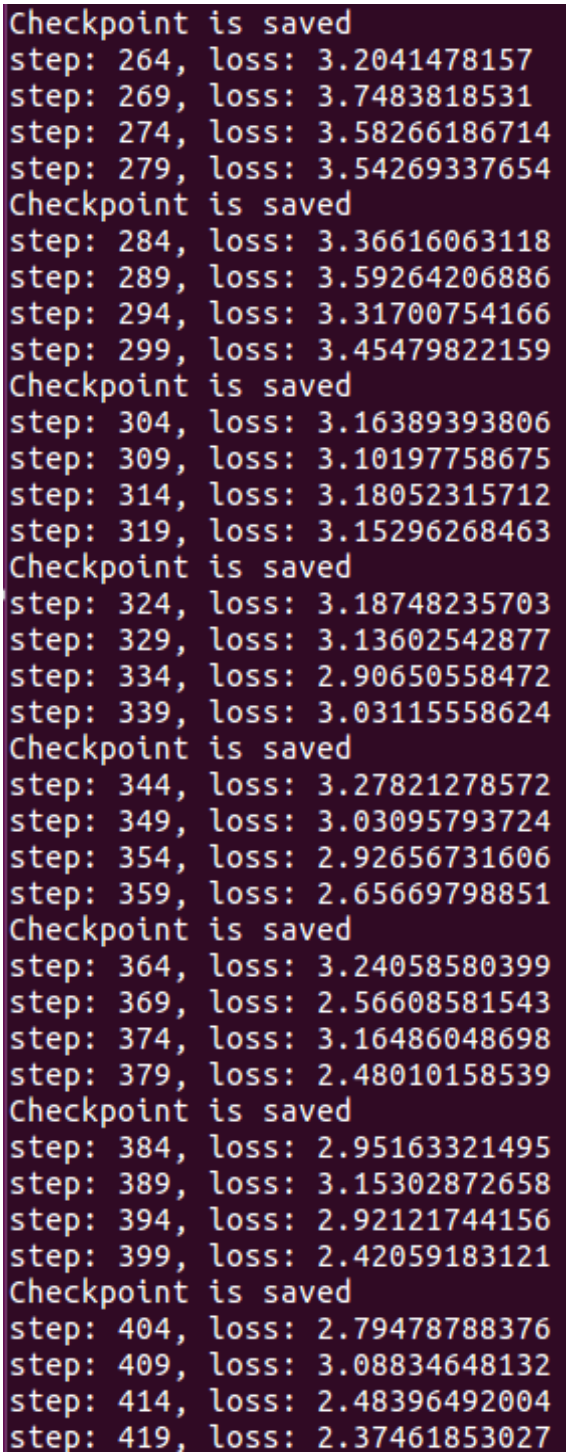


Figure 37: Training log pg 2

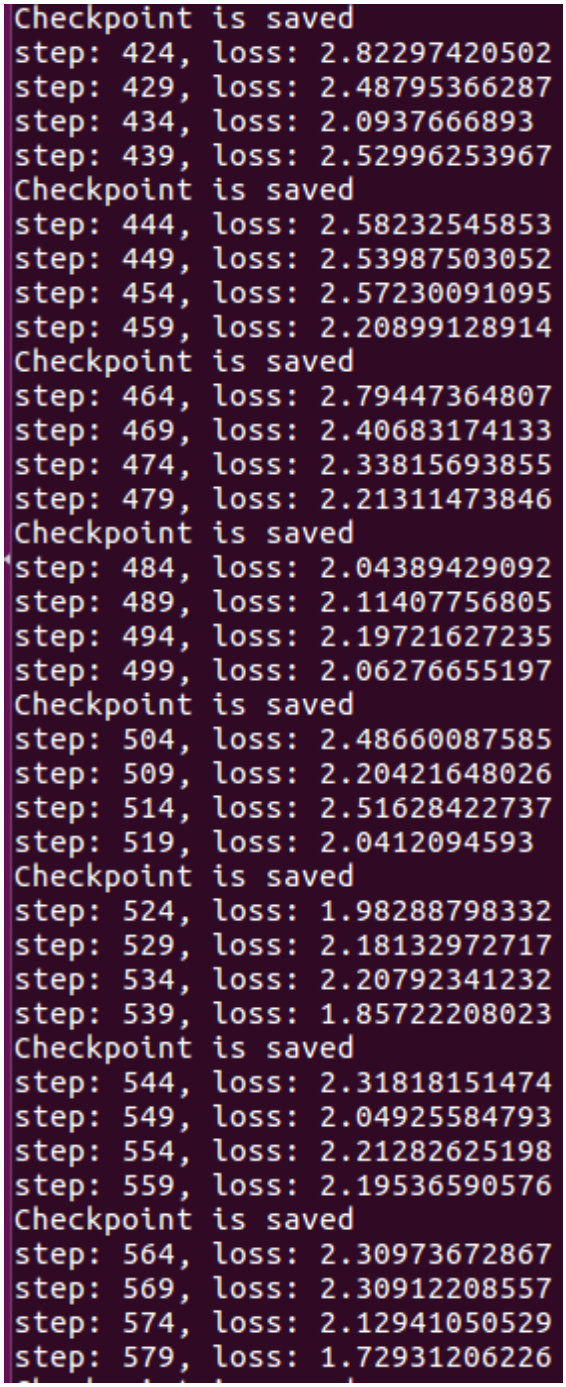


Figure 38: Training log pg 5

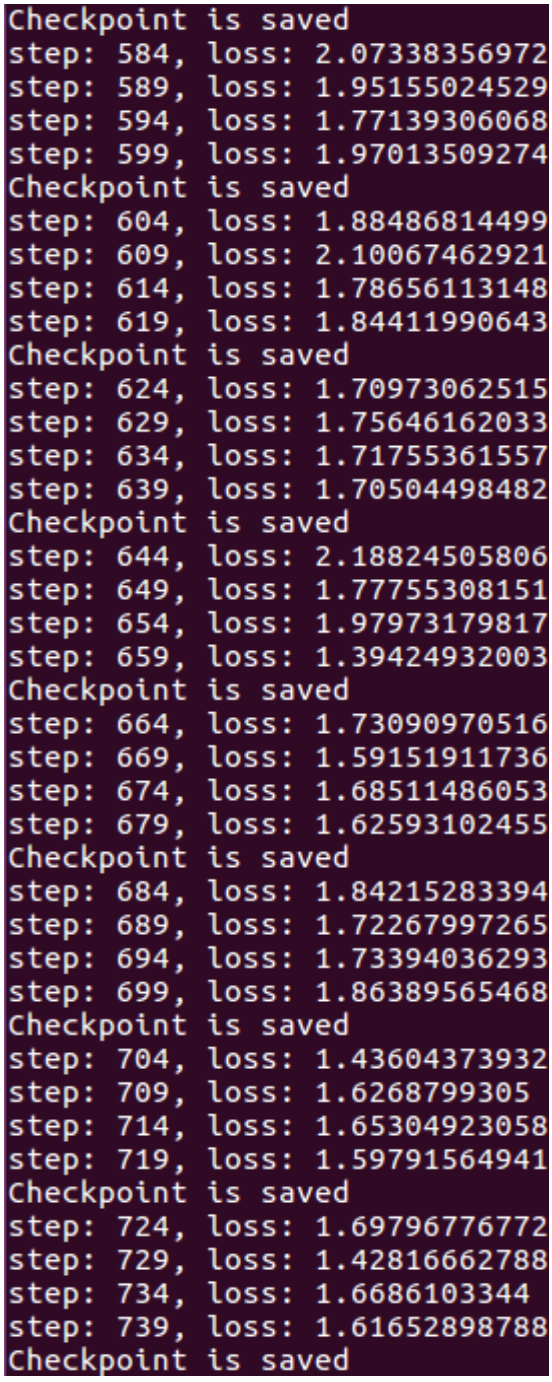


Figure 39: Training log pg 4

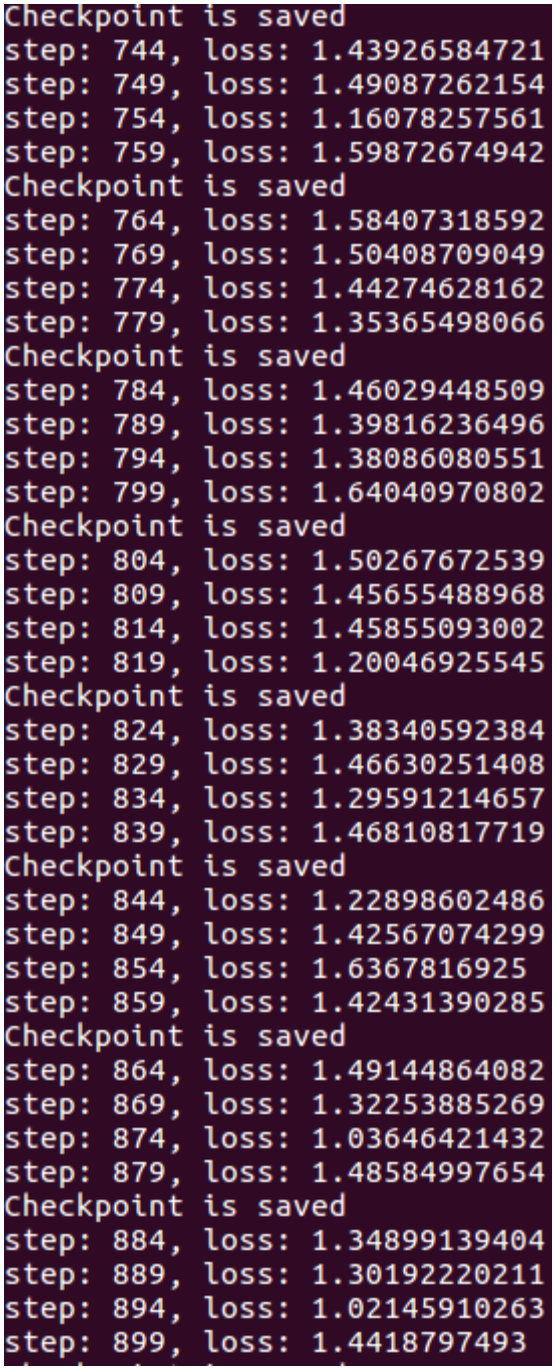


Figure 40: Training log pg 6

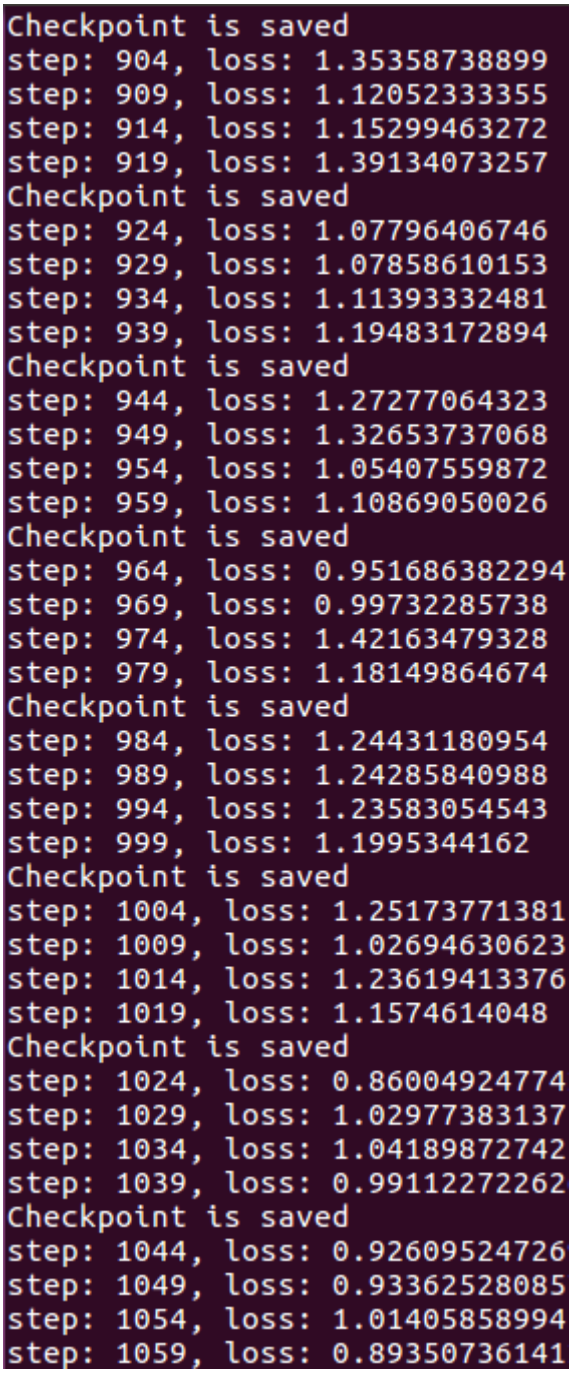


Figure 41: Training log pg 7

```
Checkpoint is saved
step: 1064, loss: 1.09271907806
step: 1069, loss: 1.03853297234
step: 1074, loss: 0.726714551449
step: 1079, loss: 0.871734380722
Checkpoint is saved
step: 1084, loss: 1.22062766552
step: 1089, loss: 1.03396689892
step: 1094, loss: 1.04566431046
step: 1099, loss: 0.928890347481
Checkpoint is saved
step: 1104, loss: 1.18727779388
step: 1109, loss: 1.01613557339
step: 1114, loss: 0.835236310959
step: 1119, loss: 0.97595000267
Checkpoint is saved
step: 1124, loss: 0.927545368671
step: 1129, loss: 0.904163241386
step: 1134, loss: 1.09931945801
step: 1139, loss: 1.0943390131
Checkpoint is saved
step: 1144, loss: 0.914264142513
step: 1149, loss: 1.19910144806
step: 1154, loss: 1.03148186207
step: 1159, loss: 0.990949451923
Checkpoint is saved
step: 1164, loss: 0.878303408623
step: 1169, loss: 0.923498570919
step: 1174, loss: 0.833215475082
step: 1179, loss: 1.03280317783
Checkpoint is saved
step: 1184, loss: 0.80378395319
step: 1189, loss: 0.853474497795
step: 1194, loss: 0.799584567547
step: 1199, loss: 0.869352936745
Checkpoint is saved
step: 1204, loss: 0.88744956255
step: 1209, loss: 0.836497426033
step: 1214, loss: 0.900201797485
step: 1219, loss: 0.794025957584
```

Figure 42: Training log pg 8

```
Checkpoint is saved
step: 1224, loss: 0.991951465607
step: 1229, loss: 0.757672727108
step: 1234, loss: 1.02250230312
step: 1239, loss: 0.874212384224
Checkpoint is saved
step: 1244, loss: 0.918414473534
step: 1249, loss: 0.805785119534
step: 1254, loss: 0.715442419052
step: 1259, loss: 0.635375380516
Checkpoint is saved
step: 1264, loss: 0.905603528023
step: 1269, loss: 0.790304303169
step: 1274, loss: 0.801654219627
step: 1279, loss: 0.799336373806
Checkpoint is saved
step: 1284, loss: 0.860827922821
step: 1289, loss: 0.86983704567
step: 1294, loss: 0.852125823498
step: 1299, loss: 0.824387907982
Checkpoint is saved
step: 1304, loss: 0.885291934013
step: 1309, loss: 0.987458825111
step: 1314, loss: 0.66711974144
step: 1319, loss: 0.722142279148
Checkpoint is saved
step: 1324, loss: 0.866540551186
step: 1329, loss: 0.684310793877
step: 1334, loss: 1.00592124462
step: 1339, loss: 0.739907205105
Checkpoint is saved
step: 1344, loss: 0.840182423592
step: 1349, loss: 0.753441810608
step: 1354, loss: 0.896600723267
step: 1359, loss: 0.825274884701
Checkpoint is saved
step: 1364, loss: 0.70472741127
step: 1369, loss: 0.731468617916
step: 1374, loss: 0.779067277908
step: 1379, loss: 0.610720217228
```

Figure 43: Training log pg 9

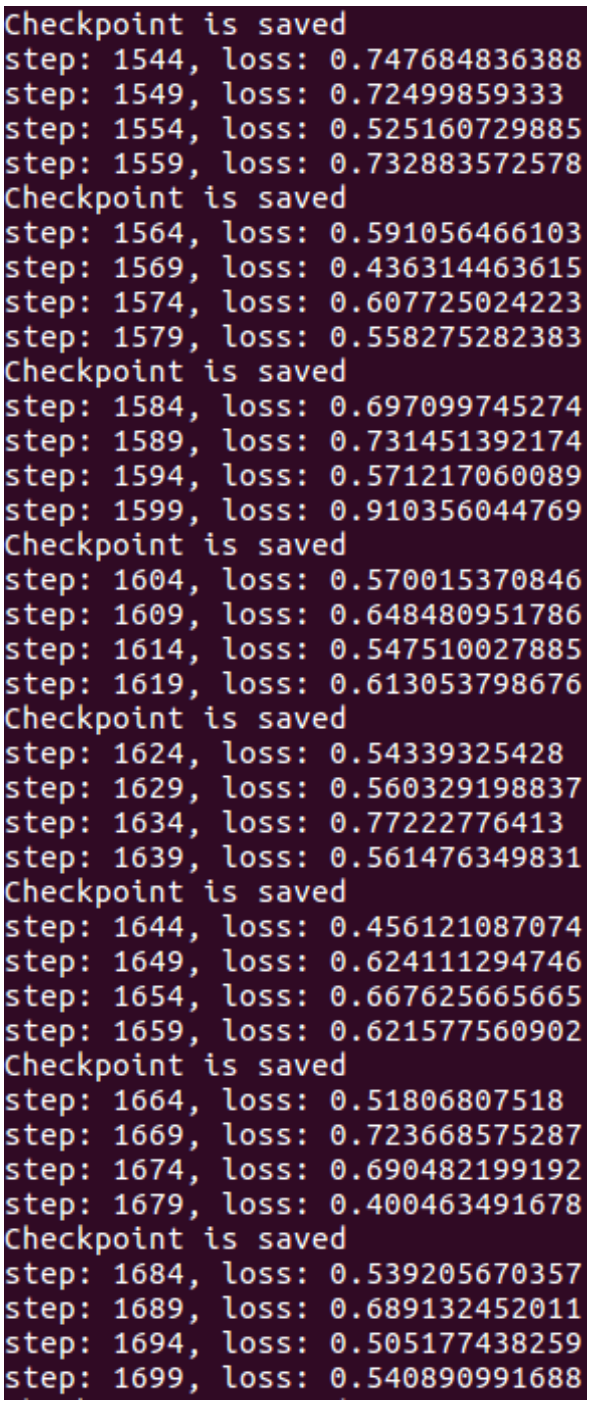


Figure 44: Training log pg 10

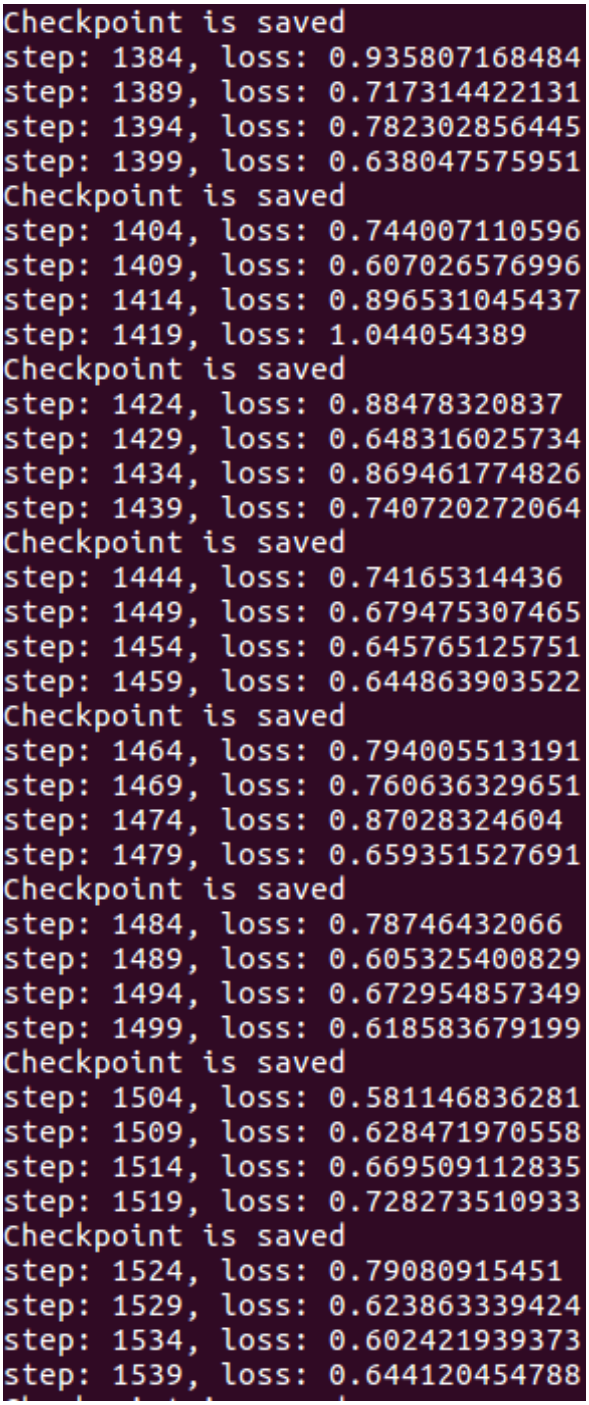


Figure 45: Training log pg 11

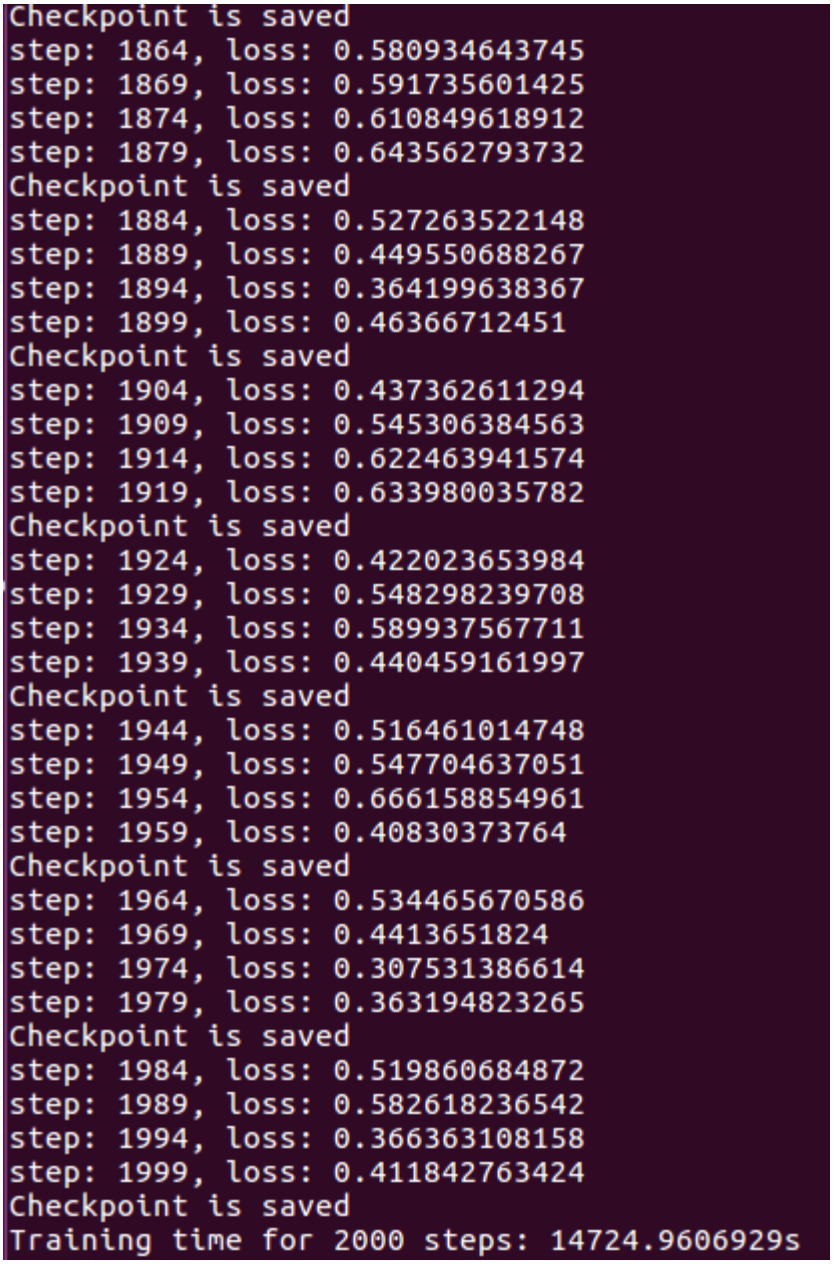


Figure 46: Training log pg 12

We can see form the training logs that the loss is slowly going lower as the session reaches step 2000. In this particular iteration of the training run the Student set the training steps to 2000 for better accuracy. A validation accuracy of 72% and a Bleu score of 26.4 was achieved in this translation.

7.1.2 PLOT FOR LOSS

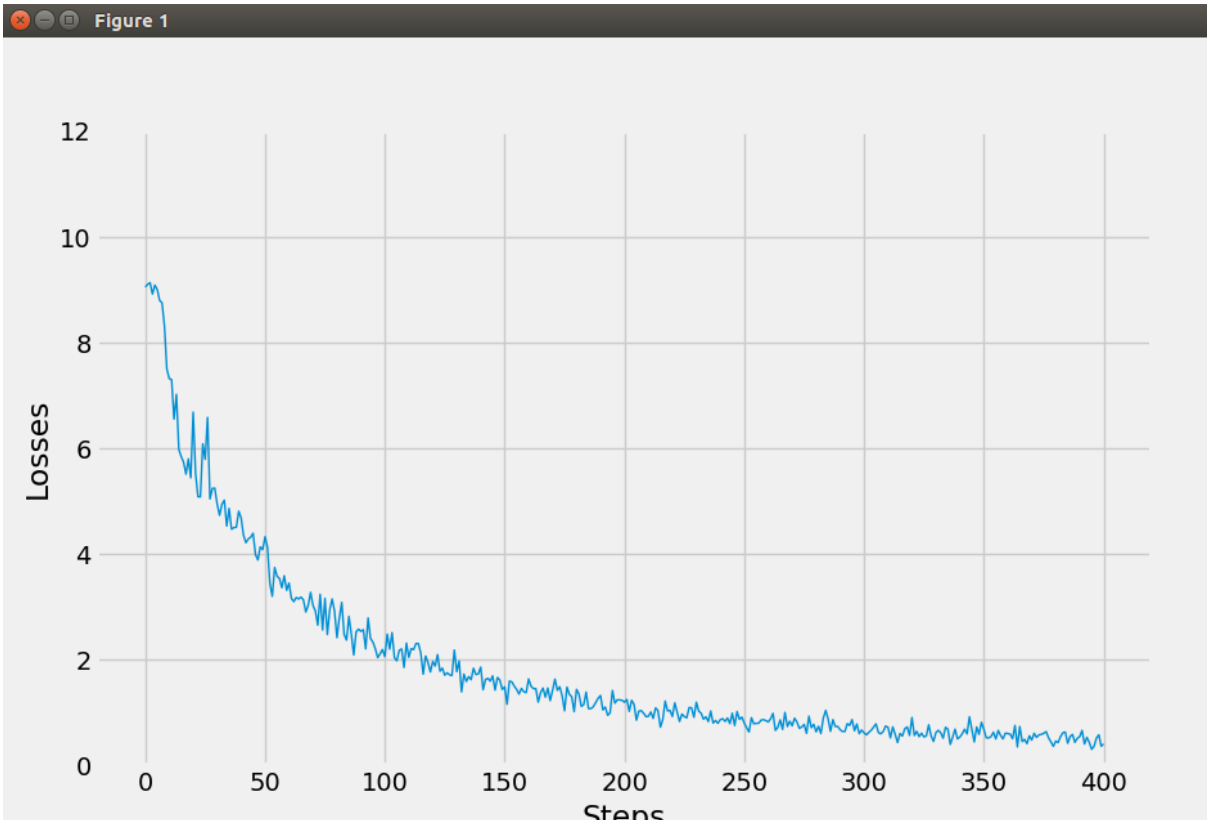


Figure 47: Loss Plot

We can see from this plot that the loss is lowering steadily with respect to the steps.

7.1.3 TRANSLATIONS

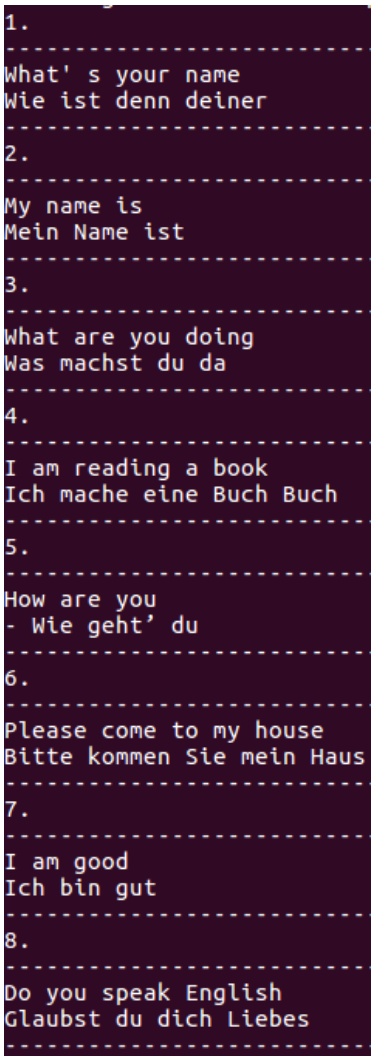


Figure 48: Snapshot of the translated outputs 1

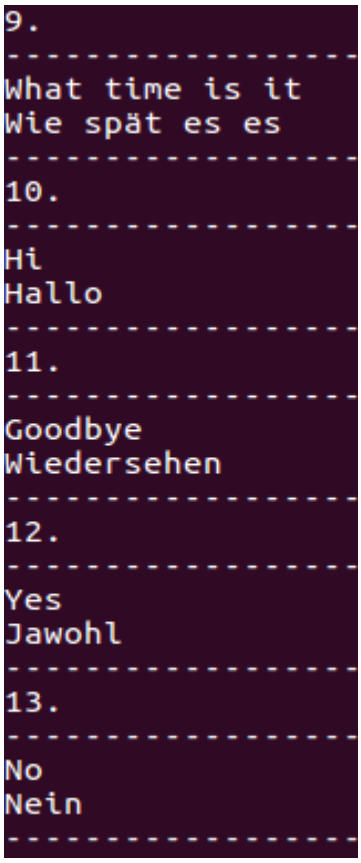


Figure 49: Snapshot of the translated output 2

These translations are nearly accurate. This might be because of the quality of the training dataset. Accuracy could be further improved by using multiple GPU's and bi-directional RNN's.

7.2 ENGLISH TO SPANISH

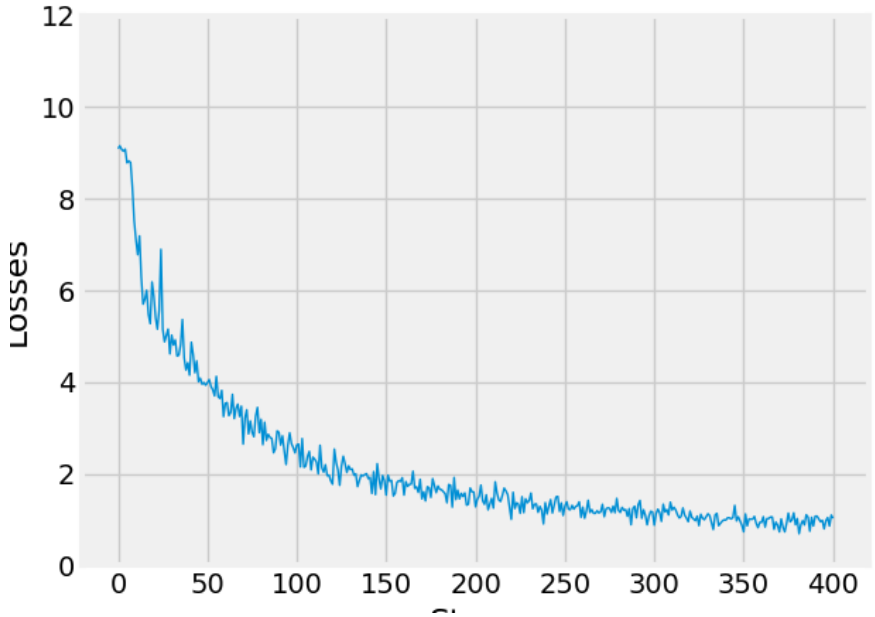


Figure 50: Loss plot for Eng-Sp

TRANSLATIONS

```
1.
--------------------------
What' s your name
¿Cuál es su nombre
--------------------------
2.
--------------------------
My name is
Mi nombre es
--------------------------
3.
--------------------------
What are you doing
¿Qué estás haciendo
--------------------------
4.
--------------------------
I am reading a book
Soy un libro un libro
--------------------------
5.
--------------------------
How are you
¿Cómo estás bien
--------------------------
6.
--------------------------
Please come to my house
Por favor a mi casa
--------------------------
7.
--------------------------
I am good
Yo te lo bien
--------------------------
8.
----------------------
Do you speak English
Usted basta inglés
----------------------
9.
----------------------
What time is it
¿Qué hora es
----------------------
10.
----------------------
Hi
Hola
----------------------
11.
----------------------
Goodbye
Adiós
----------------------
12.
----------------------
Yes
Sí
----------------------
13.
----------------------
No
No
----------------------
```

Figure 51: Snapshot of translated output

The accuracy for the English to Spanish translation was done by using the same model but just a different dataset. The accuracy for English to Spanish was found to be 85%.

At this point the student tried using a less widely used language to gauge the accuracy of the model. Student chose to use Arabic.

7.3 ENGLISH TO ARABIC

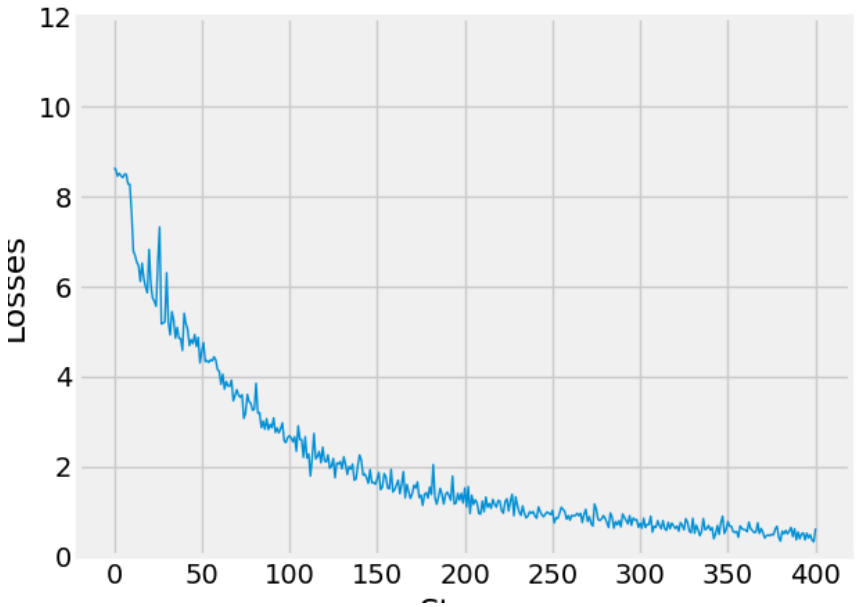


Figure 52: Loss plot for Engilsh to Arabic

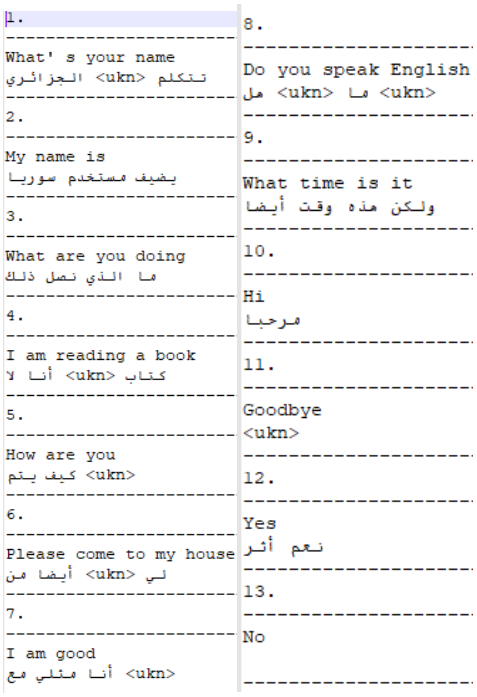


Figure 53: Snapshot of Translation output for en- ar

The results English to Arabic were very poor. This is mainly due to the quality of translations in the training set.

8. CONCLUSION

In this project, the student addressed the task of machine translation and presented an evaluation of Deep- learning-based LSTM encoder decoder system by employing an attention mechanism and a softmax activation. The student aimed at creating an RNN model which could be implemented on a standard PC without fancy GPU and a lot of processing power. The highest validation accuracy achieved by the model was 85% for English to Spanish translations. This could deteorate in case of a large validation set of sentences.

The training and testing datasets were procured by the student from an open source website mentioned in the report above. The student wrote a data-utils script to pre-process the sentences to change from raw text files to datasets which are then broken into a training set and a testing set. Student has used in data-utils class the python built-in cPickle module at achieve this. A small dataset would have deprecated training and validation losses would have increased.

The results from 2000 step training sessions too almost 1 hour and 30 minutes to run and were far superior than the 1 hour runtime 1000 step training sessions. Stronger hardware would also have greatly improved the performance of the model. With the increase in hardware capability the student could have tried to implement a GRN which requires far more processing power than a conventional RNN. It would also have been able to handle a much larger dataset, and possibly reduced the training time exponentially.

To combat limited memory problem in the model the student incorporated an attention mechanism in the model. This increased the accuracy by querying the output of the previous LSTM and checking if the relevant changes were being made.

The student greatly increased his knowledge of working in python language and also his knowledge of different deep learning architectures. His attraction to this area has exponentially increased over the past few months as the project came closer to fruition.

Results of the student’s experiments with different architectures and their varying hyperparameters confirmed much of the general intuition that the student had about RNNs that he gained by reading about them. Overall, the student accepts that LSTMs are capable of much more than just machine translation. Deep learning networks appear to fulfil the promises made for them in the literature, and the student considers this project his own small step in getting closer to the fields like Machine learning and Artificial Intelligence.

9. FUTURE WORK AND RECOMMENDATIONS

Given the limited time and resources for the project, there is significant room for improvement in the model's performance. The student believes, if the project were to be worked on in the future, the following recommended design and implementation aspects could be investigated:

- Stronger hardware support to improve network strength, it would not require any additional changes in code.
- Stronger memory support (RAM or GPU) enabling deeper architectures as models would be able to hold more parameters, even for larger images and/or datasets.
- Try to procure better quality datasets.
- The performance could be further enhanced by playing around with the hyperparameters, the number of steps and even the batch size.
- The student would have liked to work with more of the different types of LSTMs available in tensorflow.

REFERENCES

[1] J. Brownlee, “A Gentle Introduction to Neural Machine Translation,” *Machine Learning Mastery*, 21-Nov-2017. [Online]. Available: https://machinelearningmastery.com/introduction-neural-machine-translation/. [Accessed: 16-Mar-2018].

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Frechen: MITP, 2018.

[3] I. Pestov, “A history of machine translation from the Cold War to deep learning,” *freeCodeCamp*, 12-Mar-2018. [Online]. Available: https://medium.freecodecamp.org/a-history-of-machine-translation-from-the-cold-war-to-deep-learning-f1d335ce8b5. [Accessed: 16-Mar-2018].

[4] A. Geitgey, “Machine Learning is Fun Part 5: Language Translation with Deep Learning and the Magic of Sequences,” *Medium*, 21-Aug-2016. [Online]. Available: https://medium.com/@ageitgey/machine-learning-is-fun-part-5-language-translation-with-deep-learning-and-the-magic-of-sequences-2ace0acca0aa. [Accessed: 16-Mar-2018].

[5] M. D. Okpor , “A Brief Study of Challenges in Machine Translation,” *International Journal of Computer Science Issues*, vol. 14, no. 2, pp. 54–57, 2017.

[6] “Example-based machine translation,” *Wikipedia*, 17-Feb-2018. [Online]. Available: https://en.wikipedia.org/wiki/Example-based_machine_translation. [Accessed: 15-Mar-2018].

[7] P. Koehn, F. Och and D. Marcu, Statistical Phrase-Based Translation. Ft. Belvoir: Defense Technical Information Center, 2003.

[8] P. Koehn, *Statistical machine translation*. Cambridge: Cambridge University Press, 2014.

[9] F. Ghasemi, A. Mehridehnavi, A. Fassihi, and H. Pérez-Sánchez, “Deep neural network in QSAR studies using deep belief network,” *Applied Soft Computing*, vol. 62, pp. 251–258, 2018.

[10] I. E. E. E. Staff, *2012 IEEE Conference on Computer Vision and Pattern Recognition*. Place of publication not identified: IEEE, 2012.

[11] M. Klapper-Rybicka, N. N. Schraudolph, and J. Schmidhuber, “Unsupervised Learning in LSTM Recurrent Neural Networks,” *Artificial Neural Networks — ICANN 2001 Lecture Notes in Computer Science*, pp. 684–691, 2001.

[12] Wang, Lu, Zhou, Jie, Liu, and Qun, “Deep Neural Machine Translation with Linear Associative Unit,” [1705.00861] Deep Neural Machine Translation with Linear Associative Unit, 02-May-2017. [Online]. Available: https://arxiv.org/abs/1705.00861. [Accessed: 19-Mar-2018]

[13] Wu, Mike, Chen, Zhifeng, Mohammad, Wolfgang, Krikun, Cao, Yuan, Gao, Qin, Klaus, Klingner, Jeff, Shah, Johnson, Melvin, Liu, Xiaobing, Kaiser, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Hideto, Stevens, Keith, Kurian, Patil, Wang, Wei, Young, Smith, Jason, Rudnick, Alex, Corrado, Greg, Macduff, Jeffrey, Oriol, and Hughes, “Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” [1609.08144] Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 08-Oct-2016. [Online]. Available: https://arxiv.org/abs/1609.08144. [Accessed: 19-Mar-2018].

[14] C. V. Nicholson and A. Gibson, “A Beginner's Guide to Recurrent Networks and LSTMs,” A Beginner's Guide to Recurrent Networks and LSTMs - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM. [Online]. Available: https://deeplearning4j.org/lstm.html. [Accessed: 20-Mar-2018].

[15] 7 The Simple Recurrent Network: A Simple Model that Captures the Structure in Sequences, 16-Dec-2015. [Online]. Available:

https://web.stanford.edu/group/pdplab/pdphandbook/handbookch8.html. [Accessed: 20-Mar-2018].
[16] Chung, Junyoung, Caglar, Cho, Bengio, and Yoshua, "Gated Feedback Recurrent Neural Networks," [1502.02367] Gated Feedback Recurrent Neural Networks, 17-Jun-2015. [Online]. Available: https://arxiv.org/abs/1502.02367. [Accessed: 20-Mar-2018].
[17] S. Hochreiter and J. Schmidhuber , "Long Short-Term Memory," *Neural Computation*. [Online]. Available: https://dl.acm.org/citation.cfm?id=1246450. [Accessed: 20-Mar-2018].
[18] A. Graves, S. Fernández, and F. Gomez, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," CiteSeerX. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.75.6306. [Accessed: 20-Mar-2018].
[19] "Release notes," Release notes | Anaconda: Documentation. [Online]. Available: https://docs.anaconda.com/anaconda/release-notes. [Accessed: 20-Mar-2018].
[20] "Anaconda End User License Agreement," *Anaconda End User License Agreement | Anaconda: Documentation*. [Online]. Available: https://docs.anaconda.com/anaconda/eula. [Accessed: 20-Mar-2018].
[21] A. Unruh Feed, "What is the TensorFlow machine intelligence platform?," Opensource.com. [Online]. Available: https://opensource.com/article/17/11/intro-tensorflow. [Accessed: 20-Mar-2018].
[22] "NumPy¶," NumPy - NumPy. [Online]. Available: http://www.numpy.org/. [Accessed: 20-Mar-2018].
[23] Pedregosa, Fabian, Varoquaux, Gaël, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Jake, Passos, David, Brucher, Matthieu, Perrot, and Duchesnay, "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, 01-Jan-1970. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html. [Accessed: 20-Mar-2018].
[24] "Installation¶," *Installation - Matplotlib 2.2.2 documentation*. [Online]. Available: https://matplotlib.org/. [Accessed: 20-Mar-2018].
[25] "Neural Machine Translation (seq2seq) Tutorial | TensorFlow," TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/seq2seq. [Accessed: 20-Mar-2018].

APPENDICES

APPENDIX A

Source code: translator.py

```
1.  # import dependencies
2.  import tensorflow as tf
3.  import numpy as np
4.  from sklearn.model_selection import train_test_split
5.  import time
6.  import data_utils
7.  import matplotlib.pyplot as plt
8.  from nltk.translate.bleu_score import sentence_bleu
9.
10. # read dataset
11. X, Y, en_word2idx, en_idx2word, en_vocab, de_word2idx, de_idx2word, de_vocab = data
    _utils.read_dataset('data.pkl')
12.
13. # inspect data
14. print 'Sentence in English - encoded:', X[0]
15. print 'Sentence in German - encoded:', Y[0]
16. print 'Decoded:\n------------------------'
17.
18. for i in range(len(X[1])):
19.     print en_idx2word[X[1][i]],
20.
21. print '\n'
22.
23. for i in range(len(Y[1])):
24.     print de_idx2word[Y[1][i]],
25.
26. # data processing
27.
28. # data padding
29. def data_padding(x, y, length = 15):
30.     for i in range(len(x)):
31.         x[i] = x[i] + (length - len(x[i])) * [en_word2idx['<pad>']]
32.         y[i] = [de_word2idx['<go>']] + y[i] + [de_word2idx['<eos>']] + (length-
    len(y[i])) * [de_word2idx['<pad>']]
33.
34. data_padding(X, Y)
35.
36. # data splitting
37. X_train,  X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1)
38.
39. del X
40. del Y
41.
42. # build a model
43.
44. input_seq_len = 15
45. output_seq_len = 17
46. en_vocab_size = len(en_vocab) + 2 # + <pad>, <ukn>
47. de_vocab_size = len(de_vocab) + 4 # + <pad>, <ukn>, <eos>, <go>
48.
49. # placeholders
50. encoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'encoder{
    }'.format(i)) for i in range(input_seq_len)]
51. decoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name = 'decoder{
    }'.format(i)) for i in range(output_seq_len)]
52.
```

```
targets = [decoder_inputs[i+1] for i in range(output_seq_len-1)]
# add one more target
targets.append(tf.placeholder(dtype = tf.int32, shape = [None], name = 'last_target'))
target_weights = [tf.placeholder(dtype = tf.float32, shape = [None], name = 'target_w{}'.format(i)) for i in range(output_seq_len)]

# output projection
size = 512
w_t = tf.get_variable('proj_w', [de_vocab_size, size], tf.float32)
b = tf.get_variable('proj_b', [de_vocab_size], tf.float32)
w = tf.transpose(w_t)
output_projection = (w, b)

outputs, states = tf.contrib.legacy_seq2seq.embedding_attention_seq2seq(
                                            encoder_inputs,
                                            decoder_inputs,
                                            tf.contrib.rnn.BasicLSTMCell(size),
                                            num_encoder_symbols = en_vocab_size,
                                            num_decoder_symbols = de_vocab_size,
                                            embedding_size = 100,
                                            feed_previous = False,
                                            output_projection = output_projection,

                                            dtype = tf.float32)

# define loss function

# sampled softmax loss - returns: A batch_size 1-D tensor of per-example sampled softmax losses
def sampled_loss(labels, logits):
    return tf.nn.sampled_softmax_loss(
                          weights = w_t,
                          biases = b,
                          labels = tf.reshape(labels, [-1, 1]),
                          inputs = logits,
                          num_sampled = 512,
                          num_classes = de_vocab_size)

# Weighted cross-entropy loss for a sequence of logits
loss = tf.contrib.legacy_seq2seq.sequence_loss(outputs, targets, target_weights, softmax_loss_function = sampled_loss)

# define helper functions

# simple softmax function
def softmax(x):
    n = np.max(x)
    e_x = np.exp(x - n)
    return e_x / e_x.sum()

# feed data into placeholders
      def feed_dict(x, y, batch_size = 64):
          feed = {}

          idxes = np.random.choice(len(x), size = batch_size, replace = False)

          for i in range(input_seq_len):
              feed[encoder_inputs[i].name] = np.array([x[j][i] for j in idxes], dtype = np.int32)

          for i in range(output_seq_len):
              feed[decoder_inputs[i].name] = np.array([y[j][i] for j in idxes], dtype = np.int32)

```

```
            feed[targets[len(targets)-1].name] = np.full(shape = [batch_size], fill_value = de_word2idx['<pad>'], dtype = np.int32)

            for i in range(output_seq_len-1):
                batch_weights = np.ones(batch_size, dtype = np.float32)
                target = feed[decoder_inputs[i+1].name]
                for j in range(batch_size):
                    if target[j] == de_word2idx['<pad>']:
                        batch_weights[j] = 0.0
                feed[target_weights[i].name] = batch_weights

            feed[target_weights[output_seq_len-1].name] = np.zeros(batch_size, dtype = np.float32)

            return feed

        # decode output sequence
        def decode_output(output_seq):
            words = []
            for i in range(output_seq_len):
                smax = softmax(output_seq[i])
                idx = np.argmax(smax)
                words.append(de_idx2word[idx])
            return words

        # ops and hyperparameters
        learning_rate = 5e-3
        batch_size = 64
        steps = 2000

        # ops for projecting outputs
        outputs_proj = [tf.matmul(outputs[i], output_projection[0]) + output_projection[1] for i in range(output_seq_len)]

        # training op
        optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(loss)

        # init op
        init = tf.global_variables_initializer()

        # forward step
        def forward_step(sess, feed):
            output_sequences = sess.run(outputs_proj, feed_dict = feed)
            return output_sequences

        # training step
        def backward_step(sess, feed):
            sess.run(optimizer, feed_dict = feed)

        # Train the model

        losses = []

        # save a checkpoint so we can restore the model later
        saver = tf.train.Saver()

        print '------------------TRAINING------------------'

        with tf.Session() as sess:
            sess.run(init)

            t = time.time()
            for step in range(steps):
                feed = feed_dict(X_train, Y_train)

```

```
173.                 backward_step(sess, feed)
174.
175.                 if step % 5 == 4 or step == 0:
176.                     loss_value = sess.run(loss, feed_dict = feed)
177.                     print 'step: {}, loss: {}'.format(step, loss_value)
178.                     losses.append(loss_value)
179.
180.                 if step % 20 == 19:
181.                     saver.save(sess, 'checkpoints/', global_step=step)
182.                     print 'Checkpoint is saved'
183.
184.             print 'Training time for {} steps: {}s'.format(steps, time.time() - t)
185.
186.         # plot losses
187.
188.         with plt.style.context('fivethirtyeight'):
189.             plt.plot(losses, linewidth = 1)
190.             plt.xlabel('Steps')
191.             plt.ylabel('Losses')
192.             plt.ylim((0, 12))
193.
194.         plt.show()
195.
196.         #Testing the model
197.
198.         with tf.Graph().as_default():
199.
200.             # placeholders
201.             encoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name
    = 'encoder{}'.format(i)) for i in range(input_seq_len)]
202.             decoder_inputs = [tf.placeholder(dtype = tf.int32, shape = [None], name
    = 'decoder{}'.format(i)) for i in range(output_seq_len)]
203.
204.             # output projection
205.             size = 512
206.             w_t = tf.get_variable('proj_w', [de_vocab_size, size], tf.float32)
207.             b = tf.get_variable('proj_b', [de_vocab_size], tf.float32)
208.             w = tf.transpose(w_t)
209.             output_projection = (w, b)
210.
211.             # change the model so that output at time t can be fed as input at time
    t+1
212.             outputs, states = tf.contrib.legacy_seq2seq.embedding_attention_seq2seq(

213.                                                             encoder_inputs,
214.                                                             decoder_inputs,
215.                                                             tf.contrib.rnn.BasicLSTMCell
    (size),
216.                                                             num_encoder_symbols = en_voc
    ab_size,
217.                                                             num_decoder_symbols = de_voc
    ab_size,
218.                                                             embedding_size = 100,
219.                                                             feed_previous = True, # <---
    --this is changed----->
220.                                                             output_projection = output_p
    rojection,
221.                                                             dtype = tf.float32)
222.
223.             # ops for projecting outputs
224.             outputs_proj = [tf.matmul(outputs[i], output_projection[0]) + output_pro
    jection[1] for i in range(output_seq_len)]
225.
226.             # Translate these sentences
227.             en_sentences = ["What' s your name", 'My name is', 'What are you doing',
     'I am reading a book',\
```

```
                            'How are you','Please come to my house', 'I am good', 'Do you speak English', 'What time is it', 'Hi', 'Goodbye', 'Yes', 'No']
            en_sentences_encoded = [[en_word2idx.get(word, 0) for word in en_sentence.split()] for en_sentence in en_sentences]

            # padding to fit encoder input
            for i in range(len(en_sentences_encoded)):
                en_sentences_encoded[i] += (15 - len(en_sentences_encoded[i])) * [en_word2idx['<pad>']]

            # restore all variables - use the last checkpoint saved
            saver = tf.train.Saver()
            path = tf.train.latest_checkpoint('checkpoints')

            with tf.Session() as sess:
                # restore
                saver.restore(sess, path)

                # feed data into placeholders
                feed = {}
                for i in range(input_seq_len):
                    feed[encoder_inputs[i].name] = np.array([en_sentences_encoded[j][i] for j in range(len(en_sentences_encoded))], dtype = np.int32)

                feed[decoder_inputs[0].name] = np.array([de_word2idx['<go>']] * len(en_sentences_encoded), dtype = np.int32)

                # translate
                output_sequences = sess.run(outputs_proj, feed_dict = feed)

                # decode seq.
                for i in range(len(en_sentences_encoded)):
                    print '{}.\n--------------------------------'.format(i+1)
                    ouput_seq = [output_sequences[j][i] for j in range(output_seq_len)]
                    #decode output sequence
                    words = decode_output(ouput_seq)

                    print en_sentences[i]
                    for i in range(len(words)):
                        if words[i] not in ['<eos>', '<pad>', '<go>']:
                            print words[i],


                    print '\n--------------------------------'
```

APPENDIX B

Source code: data_utils.py

```
1.  import cPickle as pickle
2.  from collections import Counter
3.
4.  def read_sentences(file_path):
5.      sentences = []
6.
7.      with open(file_path, 'r') as reader:
8.          for s in reader:
9.              sentences.append(s.strip())
10.
11.     return sentences
12.
13. def create_dataset(en_sentences, de_sentences):
14.
15.     en_vocab_dict = Counter(word.strip(',." ;:)(][?!') for sentence in en_sentences
     for word in sentence.split())
16.     de_vocab_dict = Counter(word.strip(',." ;:)(][?!') for sentence in de_sentences
     for word in sentence.split())
17.
18.     en_vocab = map(lambda x: x[0], sorted(en_vocab_dict.items(), key = lambda x: -
    x[1]))
19.     de_vocab = map(lambda x: x[0], sorted(de_vocab_dict.items(), key = lambda x: -
    x[1]))
20.
21.     en_vocab = en_vocab[:20000]
22.     de_vocab = de_vocab[:30000]
23.
24.     start_idx = 2
25.     en_word2idx = dict([(word, idx+start_idx) for idx, word in enumerate(en_vocab)]
    )
26.     en_word2idx['<ukn>'] = 0
27.     en_word2idx['<pad>'] = 1
28.
29.     en_idx2word = dict([(idx, word) for word, idx in en_word2idx.iteritems()])
30.
31.
32.     start_idx = 4
33.     de_word2idx = dict([(word, idx+start_idx) for idx, word in enumerate(de_vocab)]
    )
34.     de_word2idx['<ukn>'] = 0
35.     de_word2idx['<go>']  = 1
36.     de_word2idx['<eos>'] = 2
37.     de_word2idx['<pad>'] = 3
38.
39.     de_idx2word = dict([(idx, word) for word, idx in de_word2idx.iteritems()])
40.
41.     x = [[en_word2idx.get(word.strip(',." ;:)(][?!'), 0) for word in sentence.split
    ()] for sentence in en_sentences]
42.     y = [[de_word2idx.get(word.strip(',." ;:)(][?!'), 0) for word in sentence.split
    ()] for sentence in de_sentences]
43.
44.     X = []
45.     Y = []
46.     for i in range(len(x)):
47.         n1 = len(x[i])
48.         n2 = len(y[i])
49.         n = n1 if n1 < n2 else n2
50.         if abs(n1 - n2) <= 0.3 * n:
51.             if n1 <= 15 and n2 <= 15:
```

```
                X.append(x[i])
                Y.append(y[i])

    return X, Y, en_word2idx, en_idx2word, en_vocab, de_word2idx, de_idx2word, de_v
ocab

def save_dataset(file_path, obj):
    with open(file_path, 'wb') as f:
        pickle.dump(obj, f, -1)

def read_dataset(file_path):
    with open(file_path, 'rb') as f:
        return pickle.load(f)

def main():
    en_sentences = read_sentences('data/data.en')
    de_sentences = read_sentences('data/data.de')

    save_dataset('./data.pkl', create_dataset(en_sentences, de_sentences))

if __name__ == '__main__':
    main()
```

COPY OF POSTER

Machine Translation using Deep Learning Neural Networks

UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

Yashwardhan Kaul

BEng in Electronic and Computer Engineering

Introduction

One of the earliest goals for computers was the automatic translation of text from one language to another. Classical, rule based systems were used for this task, which were replaced in the 1990s by statistical methods. More recently, deep neural network models have achieved state-of-the-art results in the aptly named field of **neural machine translation**.

This project considers a machine learning approach – Deep Learning – to machine translation. This is achieved by using Recurrent Neural Networks (RNNs) which are particularly useful in the identification of patterns in data.

Since human language is essentially just one big, complicated pattern, RNNs are particularly suited for use in many areas of natural language processing.

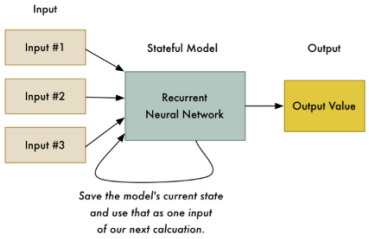


Fig 1: Working of a RNN

Aim

- Obtain an understanding of Deep Learning and Recurrent Neural Networks (RNNs), and how they are used to carry out translation between languages.
- Build a machine translation system by developing and training a deep learning RNN architecture, and see how it performs on a machine with standard processing power.

Recurrent Neural Networks (RNNs)

- Recurrent Neural Networks consist of looped networks which allow information to persist.

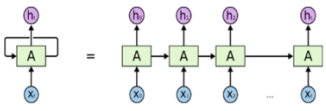


Fig 2 : An unrolled RNN

- The chain-like nature reveals that RNNs are intimately related to sequences and lists.
- At a high level, the model consists of two recurrent neural networks.
- The encoder RNN simply consumes the input source words.
- The decoder RNN processes the target sentence while predicting the next words.
- RNNs encode and then decode the semantics of a sentence instead of memorizing phrase-to-phrase translation of the sentences.
- The specific type of RNNs used in this project are Long-Short-Term-Memory RNN's or LSTMs, which are much better at capturing long-term dependencies than vanilla RNNs.

Long Short Term Memory (LSTM)

- LSTMs are a special kind of RNN, capable of learning long-term dependencies.
- They were introduced by Hochreiter & Schidhuber (1997), and were refined and popularised by many people in following years.
- This project uses the LSTM encoder/decoder with an attention mechanism model to achieve translation.

How the Model Works

- A specialised data class (utils) within tensorflow is used to process and prepare data for training.
- The vocabulary size/number of words that are going to be trained from the data is defined.
- The data utils class is used to read data from the data directory and return the formatted and tokenised words in both the languages.
- A recurrent LSTM network encodes a sentence in language A.
- The RNN spits out a hidden state **S** which represents the vectorized contents of the sentence.
- The attention mechanism queries on the output of the encoder LSTM. Each encoder output gets a relevancy score which gets converted to a probability score by applying a softmax activation to it.
- A weighted summation of the encoder outputs, known as a context vector, is extracted based on the relevancy of each output.
- The weighted outputs are then fed to the decoder LSTM RNN which then generates the translated sentence in language B.

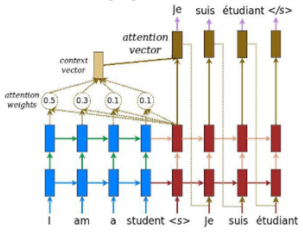


Fig 3: Project Model

Data

- This project uses a collection of multilingual corpora for training.
- The corpora consists of open source movie subtitles downloaded from http://opus.nlpl.eu/OpenSubtitles.php
- The corpora is downloaded in moses format and processed using the data utils class in tensorflow before training.

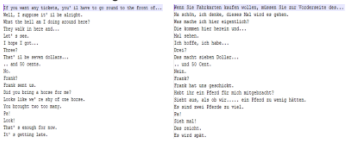


Fig 4: Parallel Corpora of English and German Text

Results

- The model was trained on 70,534 sentence pairs in English and German for 2000 steps.

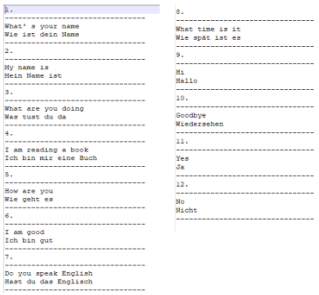


Fig 5: Translated sentence output for English-German

- The training took 1hr and 35 minutes.

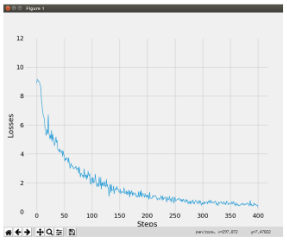


Fig 6: Variation in losses through the steps

Conclusion and Personal Reflection

- The model implementation was successful in achieving fairly good accuracy in translation.
- Due to memory constraints, the network was not made deeper which could have increased the accuracy.
- Multiple GPUs would have provided the processing power needed for Bidirectional RNNs which have state of the art accuracy but result in degradation in speed as more layers are used.

Overall, the work done on this project helped develop the following key attributes:

- Knowledge: Research into the topic of deep learning provided enhanced knowledge in the areas of RNNs and NMT modelling.
- Responsibility: This project helped to improve skills of self supervision and time management as they both played a very critical role in the timely completion of the project.

Acknowledgements

I would like to thank Dr Colin Flanagan for all his insight and guidance to bring this project to fruition. I would also like to thank all of my peers for their support and advice, the influence of which instilled in me the skills required to successfully complete this project.

E&CE Department of Electronic & Computer Engineering