Low-Resource Neural Machine Translation for Southern African Languages

by

Evander EL-Tabonah Nyoni

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Applied Mathematics in the Faculty of Science at Stellenbosch University

Department of Mathematical Sciences,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. Bruce A. Bassett
Co-supervisor: Prof. Willie Brink

December 2021

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature: .....
Evander EL-Tabonah Nyoni

Date: December 2021

Abstract

Low-Resource Neural Machine Translation for Southern African Languages

Evander EL-Tabonah Nyoni

Department of Mathematical Sciences,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MSc. (Applied Mathematics)

December 2021

The majority of African languages have not fully benefited from the recent advances in machine translation due to lack of data. Motivated by this challenge we leverage and compare transfer learning, multilingual learning and zero-shot learning on three Southern Bantu languages (namely isiZulu, isiXhosa and Shona) and English. We focus primarily on the English-to-isiZulu pair, since it has the smallest number of training pairs (30000 sentences), comprising just 28% of the average size of the other corpora. We demonstrate the significant importance of language similarity on English-to-isiZulu translations by comparing transfer learning and multilingual learning on the English-to-isiXhosa (similar) and English-to-Shona (dissimilar) tasks. We further show that multilingual learning is the best training protocol when there is sufficient data, with BLEU score gains of between 3.8 and 7.9 compared to transfer learning and zero-shot learning respectively for the English-to-isiZulu task. Our findings show that zero-shot learning is better than training a baseline model from scratch if there is not much English-to-isiZulu data. Our best model improves the previous English-to-isiZulu state-of-the-art BLEU score by more than 10. Taken together, our findings highlight the potential of leveraging the inter-relations within and between South Eastern Bantu languages to improve translations in low-resource settings.

Keywords: transfer learning, multilingual learning, zero-shot learning, BLEU.

Opsomming

Neurale masjienvertaling met lae hulpbronne vir Suider-Afrikaanse Tale

Evander EL-Tabonah Nyoni

Departement Wiskundige Wetenskappe,
Universiteit Stellenbosch,
Privaatsak X1, Matieland 7602, Suid-Afrika.

Tesis: MSc. (Toegepaste Wiskunde)

Desember 2021

Die meeste Afrikatale het weens die gebrek aan data nie ten volle gebaat by die onlangse vooruitgang in masjienvertaling nie. Gemotiveer deur hierdie uitdaging benut en vergelyk ons oordragleer, veeltalige leer en nul-skoot leer op drie Suidelike Bantoe-tale (naamlik isiZulu, isiXhosa en Shona) en Engels. Ons fokus hoofsaaklik op die Engels-tot-isiZulu-paar, aangesien dit die kleinste aantal opleidingspare (30000 sinne) het, wat slegs 28% van die gemiddelde grootte van die ander korpusse beslaan. Ons demonstreer die belangrikheid van taalgelykheid in vertalings tussen Engels en isiZulu deur die oordragleer en veeltalige leer op die take Engels-na-isiXhosa (soortgelyk) en Engels-na-Shona (verskillende) te vergelyk. Ons toon verder dat meertalige leer die beste opleidingsprotokol is as daar voldoende data is, met BLEU-tellingwinste van tussen 3.8 en 7.9 in vergelyking met onderskeidelik oordragleer en nul-skoot leer vir die Engels-na-isiZulu-taak. Ons bevindinge toon dat zero-shot-leer beter is as om 'n basislynmodel van voor af op te lei as daar nie veel Engels-tot-isiZulu-data is nie. Ons beste model verbeter ook die vorige Engels-tot-isiZulu SOTA BLEU telling met meer as 10. Ons bevindings beklemtoon die potensiaal om die onderlinge verhoudings binne en tussen Suid-Oosterse Bantoe-tale te benut om vertalings in lae-hulpbron-instellings te verbeter.

Sleutelwoorde: oordrag leer, veeltalige leer, nul-skoot leer, BLEU.

Acknowledgements

First and foremost, I am deeply indebted to the almighty God for His mercies and providences throughout this research. My most sincere gratitude goes to Professor Bruce A. Bassett and Professor Willie Brink for the guidance, support and patience during the course of my MSc study and research.

My special thanks are extended to AIMS (African Institute for Mathematical Sciences) for awarding me with the MSc bursary throughout my study.

It would be folly to not render gratitude to Mr N. Matshisela, Mrs P. Nsimba and Mr T. Tichareva for their encouragement. Finally, I express my profound gratitude to my family members Mr T. Kwidini, Mrs N. Khumalo, Mr E. Nyoni and Miss C. Kazembe for their continuous support and encouragement throughout my study. Without them, this success would not have been possible.

Dedications

To my late parents Emmanuel Nyoni and Bokani Kwidini

Publications

The following publication is an extract from this thesis:

- Low-Resource Neural Machine Translation for Southern African Languages.

The link to the publication is included in the model card appended at the end of the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Chapter Organisation

This chapter lays out a brief introduction to the study, with Section 1.2 being the motivation behind the research itself and is presented before the problem description in Section 1.3. Further on in Sections 1.4 and 1.5 the chapter presents the work that is related to this study and the general structure of this thesis, respectively.

1.2 Motivation

Languages are an indispensable instrument of communication, without which social order is difficult or impossible to achieve. They are undoubtedly the primary medium of interaction for people from heterogeneous and homogeneous cultural backgrounds. However, with over one hundred and forty-two known language families in existence [29], this may present a communication barrier between individuals of different languages. Languages that are used predominantly in high-status functions like education, media and other government enterprises are popularly known as *institutional languages* [26]. Concerning economic globalisation currently, the challenge of communication barrier is the principal motivation behind why so many individuals need to learn some institutional language. However, the learning process may prove to be a challenge to some individuals as some of these institutional languages are non-native. In most African countries, for example, the most dominant institutional languages are non-native [10].

Figure 1.1 shows the ranking of the 10 most spoken languages worldwide as either a native or second language. More than 88% of the world's population speak one of

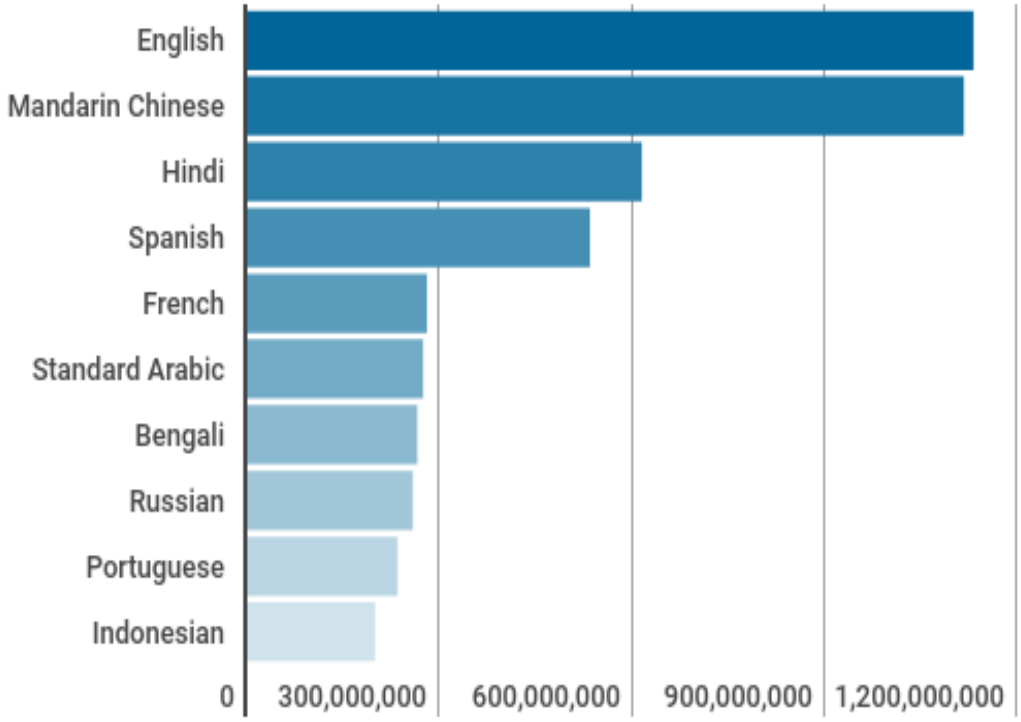


Figure 1.1: The ranking of the 10 most spoken languages worldwide as either a native or second language. As of year 2019, approximately 88% of the world's population spoke one of these languages as either a native or second language. [29]

these languages as either a native or second language. Second, to Asia, Africa has the second most indigenous languages, and unsurprisingly these two continents together account for two-thirds of all the tongues in the world. The vast majority of people worldwide use European or Asian languages, and this is mainly due to the sheer population in some localities and the colonial expansion in centuries past. Perhaps this explains why non-native languages are the most dominant institutional languages in Africa.

With a population that is over a billion, Africa is the second most populous continent and has the highest linguistic diversity, see Figure 1.2. This is mainly due to the enormous language diversity in the sub-Saharan-Africa region. Most of these languages are under the Niger-Congo language family [29]. The Niger-Congo and Austronesian lang-

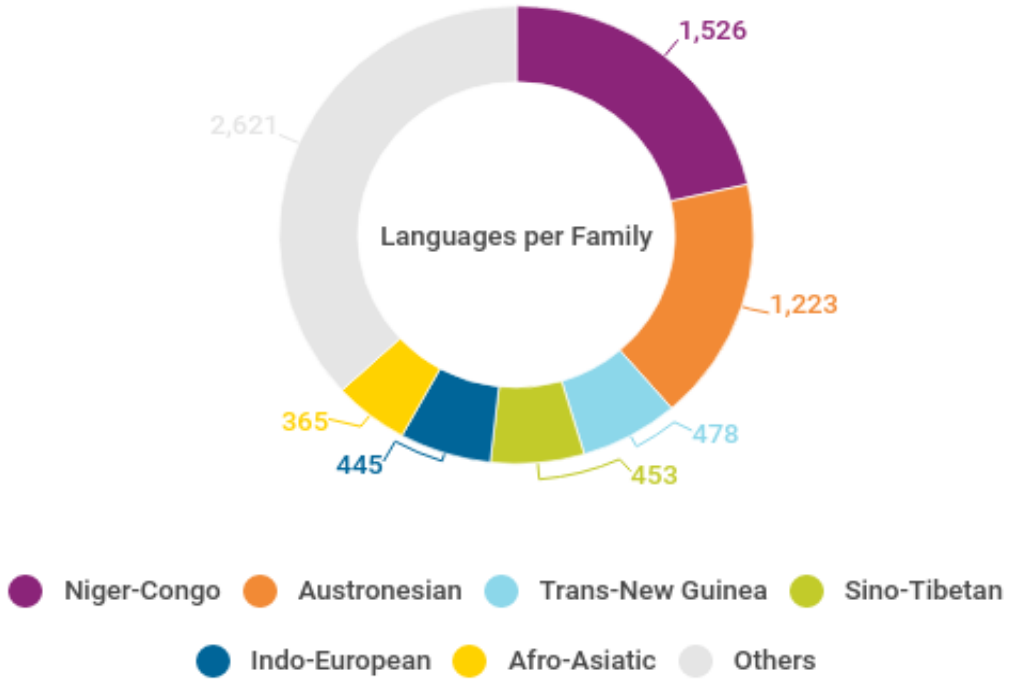


Figure 1.2: A 2019 summary of language count per language family. Each section of the donut pie chart represent the number of languages in a specific language family. [29]

uage families constitute over a thousand languages each, and they both form the two largest families, see Figure 1.2. Each of the six language families shown in Figure 1.2 account for no less than 5% of the total languages in the world, and together they make up two-thirds of the language families. With such language diversity, the spread of information, knowledge and ideas across different cultures becomes a challenge. As a result, translation becomes a viable tool of alleviating this problem, and thus making it a tool that promotes social congruity and peace.

Although human translation is the barometer for high quality translations across languages, the dynamics of human mobility render it imperative that we have decent and instantaneous translation systems. It is for this very reason that science has born a subdomain of computational linguistics called Machine Translation (MT), which focuses on

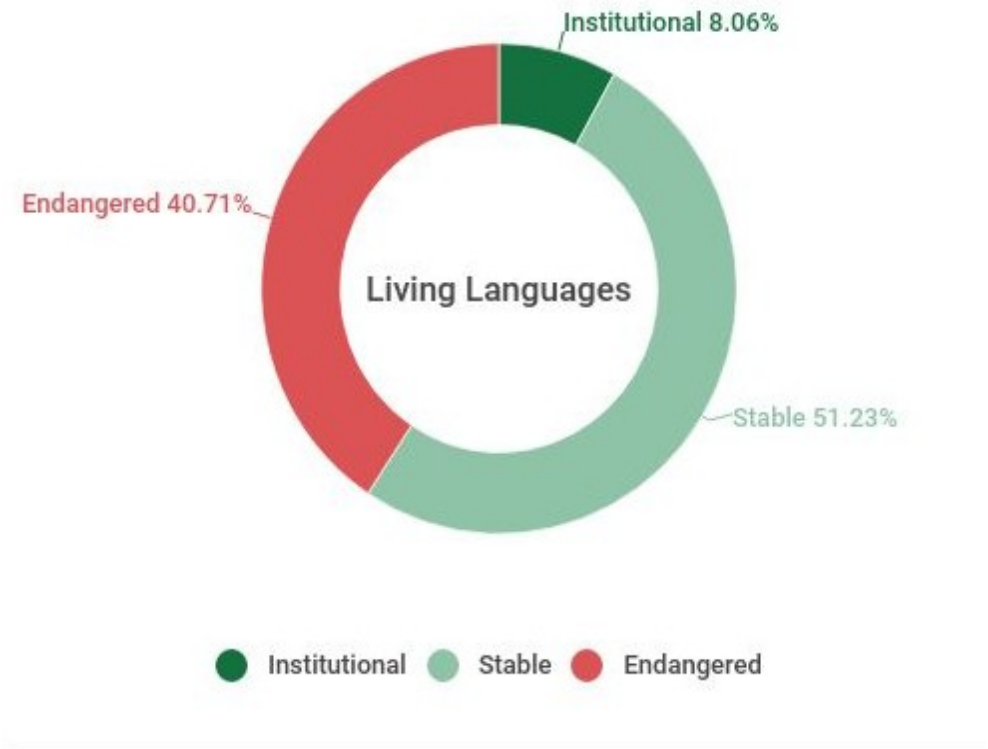


Figure 1.3: A donut pie chart illustrating the proportion of endangered languages, 2019. The institutional languages are in no danger at all. Should the speakers of the stable languages begin to teach a more dominant language to their offspring, this would result in the stable languages being endangered. [29]

automated language translation. Neural Machine Translation (NMT) has demonstrated promising results in the development of instantaneous translation systems and is proving to be superior to phrase-based Statistical Machine Translation (SMT) [84]. NMT is a MT model development technique that utilises neural networks to predict a probable translation while on the other hand SMT uses statistical models to make the predictions.

In most African educational sectors, a learner's understanding/development is greatly affected by their proficiency in the institutional language. As a strategy for improving African youth's education, it has been suggested that the learning process be augmented with online content [80]. For this reason, MT can help improve the learning process of African youth. With enormous educational content available online, translating it into

languages the learners are most comfortable with can help stimulate their interest in learning. This view can perhaps be amplified by quoting Nelson Mandela, "If you talk to a man in a language he understands, that goes to his head. If you talk to him in his language, that goes to his heart." In other words, unless Africa's educational systems employ native languages as the principal medium of communication, in-place of the foreign languages which largely dominate the government institutions, all attempts of establishing quality educational systems will ultimately benefit the elite and its posterity [11].

A language is said to be endangered whenever its native speakers begin to speak and teach a more dominant language to their off-springs. On the other hand, a language is said to be stable when a population's new generation are constantly learning and speaking that language. With approximately 41% and 51% of the world's languages being respectively endangered and stable as shown in Figure 1.3. Several African languages fall under the endangered category, as the non-native languages like English continue to encroach more territory. The stable languages are also at the risk of being endangered should its speakers begin to speak and teach a more dominant language to their children. Therefore MT can be a viable tool in saving most of these endangered languages.

1.3 Problem Formulation

Machine Translation has since the early 1900s experienced rapid growth due to the increase in the availability of parallel corpora and computational power [61]. Subject to the inception of neural machine translation (NMT) [14, 84], MT has seen substantial advancements in translation quality as modern MT systems draw closer and closer to human translation quality. Notwithstanding the progress achieved in the domain of MT, the idea of NMT system development being data-hungry remains a significant challenge in expanding this work to low resource languages. Unfortunately, most African languages fall under the low-resourced group and as a result, MT of these languages has seen little progress. Irrespective the great strategies (or techniques) that have been developed to help alleviate the low resource problem, African languages still have not seen any substantial research or application of these strategies. A result of African languages accounting for the least amount of linguistic resources available to natural language processing practitioners' use [70].

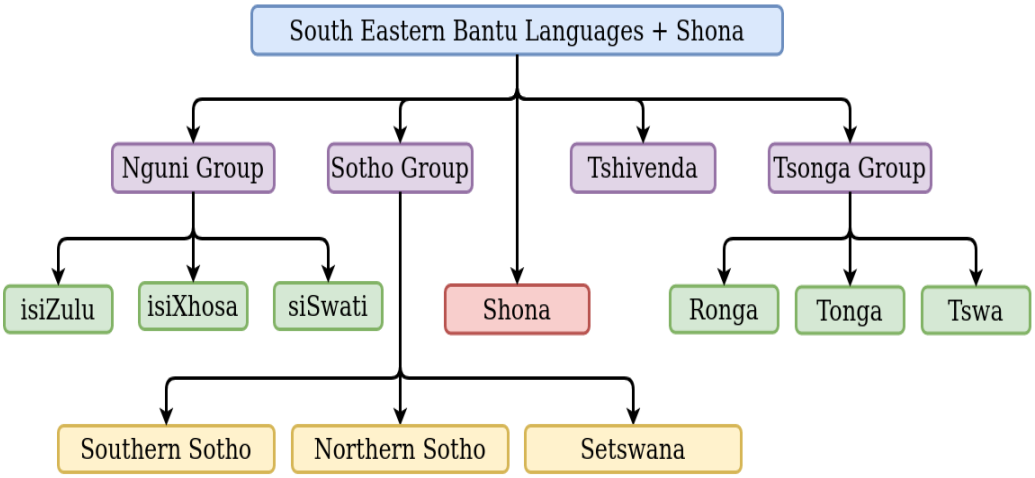


Figure 1.4: An illustration of the South-Eastern Bantu language family tree plus Shona. This tree demonstrates the relationships within language classes and sub-classes or sub-groups. In most cases, languages of the same subgroup exhibit high similarity, a property that is demonstrated by vocabulary overlap between language pairs. [27].

With a considerable number of African languages being endangered [29], there is dire need for MT tools to help save them from disappearing. In other words, this poses a challenge to the African community of NLP practitioners. This work examines the application of low-resource learning techniques on African languages of the Bantu family. Of the three Bantu languages under consideration, isiZulu, isiXhosa and Shona, the first two fall under the Nguni language sub-class indicating a close relationship between them, as shown in Figure A.9. Shona is not closely related to the Nguni language sub-class [40]. Comparing MT on these three gives us the opportunity to explore the effect of correlations and similarities between languages. We give a comparative analysis of three learning protocols, namely, *transfer learning*, *zero-shot learning* and *multilingual modeling*.

1.3.1 Aims and Objective of the study

The aim of this work is to use methods from natural language processing (NLP) and deep learning to build a machine translation system for selected African languages. To achieve this, the objectives are as follows:

1. Build machine translation systems for the following low resource language pairs:
    - English-to-isiXhosa

- English-to-Shona
- English-to-isiZulu
- isiXhosa-to-isiZulu

2. Improve the translation quality of a low resource task by applying *cross-lingual transfer learning* and *zero-shot learning*.

3. To determine the best learning technique.

All three target languages Shona, isiXhosa and isiZulu fall under the Bantu language group. Furthermore, isiXhosa and isiZulu further fall under the Nguni language subclass, as a result, they are more closely related and share lots of vocabulary. Based on the objectives of this study we employ the English-to-isiXhosa and English-to-Shona models in performing a comparative analysis of transfer learning performance.

1.4 Related Work

The inception of NMT [84] has led to great advancements in translation quality as modern MT systems draw closer and closer to human translation quality [76]. Notwithstanding the great advancements that have been made in the domain of MT, the idea of NMT system development being 'data-hungry' remains a major challenge in expanding this work to low resource languages. In essence, NMT system performance improves as training data increases [51]. Unfortunately, most African languages fall under the low-resourced language group, and as a result, MT of these languages has seen little progress.

Apart from the work that is done (or being done) by Google, most research on MT of African languages can be dated back to the year 2010, a time when SMT was the conventional technique of developing translation models. In the development of a translation model for English to Setswana, phrase-based SMT [92] was shown to be a promising technique of developing translation systems for African languages. This research employed both parallel and monolingual datasets. This technique, along with its variants were later extended to other language pairs as shown by the work of [94] when they developed a model that translates English to isiZulu. In this work, the authors used isiZulu syllables as their source tokens, a modification that proved to be efficient as it resulted in a 12.9% increase in performance. Furthermore, they went on to show that syllabification was languages dependent and could be extended to languages of the same family. As is

the case with modern MT research and all other MT research discussed in this thesis the *bilingual evaluation understudy* (BLEU) is the metric of evaluation.

Seeing that SMT was still the conventional technique of developing Machine Translation Systems (MTS) [65] developed a SMT model for translating English to Xitsonga. The training process was done on both parallel and monolingual corpus. The authors went on to investigate the effects of adding extra-linguistic information on the training data, data cleaning and decreasing data sparsity by using placeholders in place of certain words on translation quality. The final findings revealed little increase in translation quality for all conditions. The effects of unsupervised word segmentation coupled with phrase-based SMT were investigated by [86], in translating from English to Afrikaans, Northern Sotho, Tsonga and isiZulu. In their final analysis, the authors found their experiment to be efficient only for Afrikaans and isiZulu datasets.

As SMT techniques became less popular, owing to the robust NMT techniques most researchers began to adopt the novel NMT architectures in building translation models for African languages. For example, [7] developed a model that translates English to Setswana and the findings of this work indicated that NMT outperformed the previous SMT model by up to 5.33 BLEU scores. These findings prompted out the opportunities of extending NMT to other African languages. In addition to this work, [55] went on to set up a benchmark for translating English to five African languages, namely Swahili, Amharic, Tigrigna, Oromo and Somali. In their work, the authors compare each baseline model versus models trained using semi-supervised learning, transfer learning and multilingual modeling. The conclusions drawn from this research indicate that all three techniques show significant BLEU score improvement with the multilingual model having the most BLEU score gains at $+5$ scores. This thesis is closely related to the work of [55] as we both employ transfer learning and multilingual modeling in developing our models. However, this thesis is based on translating from English to southern Bantu languages and between bantu languages. We also compare three training techniques, namely zero-shot learning, transfer learning and multilingual training.

Seeing the need for further bench-marking of NMT for African languages [62] developed NMT models for translating from English to four South African languages, namely isiZulu, Northern Sotho, Setswana and Afrikaans. As one would expect, their findings

indicated that the fewer the data the lower the BLEU score (i.e. translation quality decreased as the training data decreased). Still on the same notion of bench-marking [64] went on to train ten models that translate between English and ten of South African languages. It comes as no surprise that the amount of research done in MT of African languages is relative to the resources available. As a result, this research aims at aiding in the advancement of NMT for African languages.

1.5 Study Outline

Subsequent to this introductory chapter are six chapters, namely: Background, Recurrent Neural Networks, Encoder-Decoder Sequence-to-Sequence Architectures, Low Resource Training Protocols and Evaluation Metrics, Results, and the Conclusion and Future work.

Chapter 2: Background

This chapter provides a review of the background and fundamental building blocks of NMT. For the purpose of explaining the conventional sequence modelling neural networks, this chapter introduces the basic neural network architecture along with its training algorithm.

Chapter 3: Recurrent Neural Networks

Having introduced the basic building blocks of neural networks, this chapter focuses on recurrent neural networks, a family of neural networks that are predominantly known for modeling sequential data. The complications of modelling sequential data, particularly text data, are discussed in this chapter which in turn leads to exploring the variants of recurrent neural networks that help address these complications.

Chapter 4: Encoder-Decoder Sequence-to-Sequence Architectures

This chapter focuses mainly on the encoder-decoder architecture for sequence to sequence modelling. Furthermore, the drawbacks with recurrent neural networks and its variants are discussed and consequently the solutions to these drawbacks. The transition from recurrent neural networks to the transformer network is also the subject of

discussion in this chapter.

Chapter 5: Low Resource Training Protocols and Evaluation Metrics

The main protocols employed for training NMT models in this thesis are the key topics of discussion in this chapter. In addition,the chapter also delves into the evaluation metrics used in this research.

Chapter 6: Results

This chapter presents and discusses the experimental results from each training protocol per language pair. Taking into consideration the objectives and motivation behind this work, the findings are evaluated and the protocols are compared.

Chapter 7: Conclusion and Future Work

This chapter summarizes the entire study and findings. Taking into consideration the findings of this research, possible avenues of future work are discussed in this chapter.

Chapter 2

Background

2.1 Chapter Organisation

To fully explain the model employed in this thesis, which has become the defacto standard for most NLP tasks, we take a bottom-up approach beginning with the basic building blocks of neural networks. In this chapter, we review the components needed to construct neural networks, and this gives the reader an understanding of the fundamental concepts behind neural machine translation architectures. We begin by giving an introduction to machine translation in Section 2.2. We then delve into the fundamentals of neural networks in Section 2.3 with the aim of explaining the fundamental units of neural machine translation systems later in this dissertation.

2.2 Machine Translation

The rise of MT began mid-twentieth century [90] and has since the early 1990s grown exponentially [61] due to the growth in the availability of multilingual corpora and computational power. Instigated by Petrovich Troyanskii [43], the research and development of MT systems has since been investigated at large-scale. Thereafter, Rule-based machine translation systems (RBMTS) became a classical technique of MT. A language one translates from is called the source language and the language one translates to is called the target language. For every source and target language pair, RBMT relies on a combination of a myriad of built-in linguistic rules and bilingual dictionaries. In other words, this technique (RBMTS) leverages the respective languages' linguistic information obtained from dictionaries and grammatical rules appertaining to the semantic, structural and syntactic consistencies of the language pairs.

RBMTS not only necessitate dictionaries of word semantics but translation rules designed by humans (especially linguists) for each language pair as well. Though they usually provide grammatically sound translations, RBMTS do not perform well under domain change [54]. As a result, example-based machine translation systems (EBMTS) were introduced by Makoto Nagao early in the 1980s [30]. At development, these EBMTS use parallel texts (also known as a bilingual corpus) as a knowledge pool for learning translations.

The earliest proposal of statistical machine translation (SMT) was by W. Weaver in the late 1940s [90] and it incorporated Claude Shannon's concepts of information theory. Made prominent by [19] in the late 1980s, SMT systems were based on statistical models that are learned from a pool of bilingual data. Afterwards, variants of SMT systems were then developed and these were similar to RBMTS and EBMTS as they required little or no linguistic rules at development [50]. Initially, most SMT systems were word-based [88] but with the progression of time phrase-based models [52] were introduced. The most recent advancement in SMT was the introduction of quasi-syntactic structures [21]. Before the advent of neural machine translation (NMT), SMT was the most prominent machine translation technique.

The resurgence of neural networks has led to great strides (or advances) in many areas of NLP [32]. However, the amalgamation of neural networks with MT was rather partial as the early developments only integrated neural networks into existing SMT systems [83]. The introduction of NMT [84] in 2014 has led to cognisable growth of NMT research. NMT systems differ from SMT in that they do not depend on manually curated linguistic rules of the respective language pairs. Instead, they learn translations through function approximation that is parametrised by neural networks [83]. Recent NMT results on some languages have shown impressive achievements by producing translations that are closely comparable to those of humans [67]. To gain an in-depth understanding of NMT systems we begin by introducing neural networks.

2.3 Neural Networks

The foundations of neural networks stem from the mimicry of the biological neural circuit structure [53], hence the name "Artificial Neural Networks" (ANN). The human

neural circuit comprises a myriad of interconnected neurons that pass on information when activated. Notwithstanding the biological comparison being deemed a poor caricature of the mechanism behind the human neural circuit [9], the fundamental principles of neuroscience have been of great significance in the construction of various artificial neural networks architectures.

2.3.1 Perceptrons

The main idea behind the development of perceptrons is that it is possible to mimic some aspects of the complex neural circuit network, despite limited knowledge of the biological neuron. Each biological neuron consists of three major parts, namely the cell body, axon and root-like structures called dendrites as depicted in Figure 2.1. The mammalian neuron dendrites receive impulses which are transmitted forward through the axon. The neurons in a neural circuit interconnect with one another at the synapse to form a large network. The synapse is a structure that facilitates the relaying of specific impulses/signals to the next neuron. The signal strength tends to vary, which in turn determines which neurons to activate. The neuron-to-neuron connections tend to become stronger or weaker relative to how often a specific type of signal is relayed in the network.

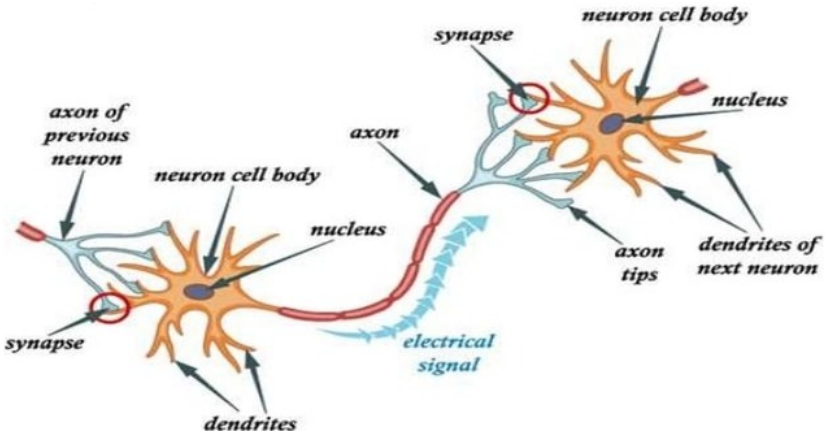


Figure 2.1: An illustration of electrical signal transmission in the mammalian neuron. The signal is passed on from the preceding neuron to the next via the dendrites on to the cell-body, then through the axon and finally to the axon tips [12].

Mimicking the biological neuron, the perceptron is the centrepiece of an artificial neural network and was proposed by Frank Rosenblatt in the 1950s [68]. Rosenblatt's perceptron (or system) comprised of a smart input-output connection, emulating the McCulloch-Pitts perceptron. The McCulloch-Pitts perceptron had earlier been developed by Walter Pitts (a logician), and Warren S. McCulloch (a neuroscientist) in 1943, as they sought to decipher the convoluted decision process of the human brain by exploiting the linear threshold unit. However, this perceptron could only solve binary problems without any learning (or adaptation). The smartness of Frank Rosenblatt's perceptron lay in its ability to learn representations through updating its weights as inputs are passed through the neuron, also known as a *node*.

Figure 2.2 (below) shows the basic structure of a perceptron. To describe the perceptron mathematically, we consider a case where each input-output pair is of the form (x_i, y), where the input is $x_i \in \{x_1, x_2, ..., x_n\}$ and the output $y \in \{-1, +1\}$ is a binary class variable, also called the *observed value*. At training, the input-output pairs are fed to the model with an objective of learning the required representations that predict the unobserved class values.

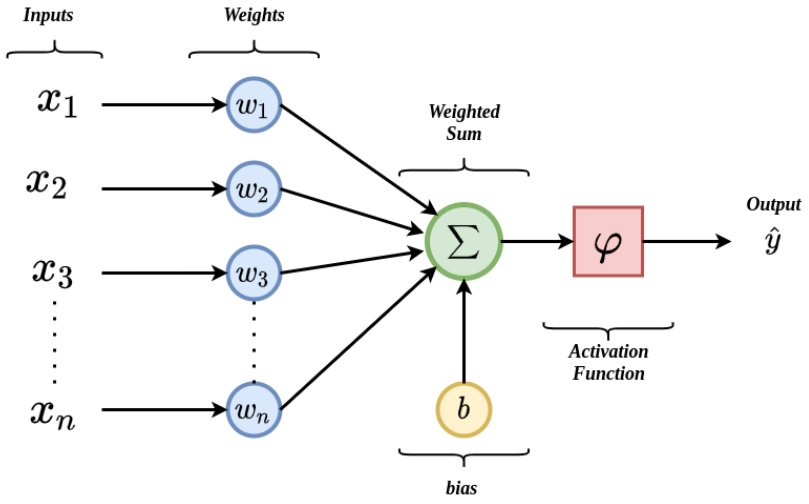


Figure 2.2: A simple artificial neuron, also known as the Perceptron model. The weighted sum of inputs are fed into the activation function which then generates output $\hat{y}$.

The term b in Figure 2.2 is called the bias, and serves the purpose of representing the

invariant part (with respect to the inputs) of the predictions. The activation function $g(\cdot)$ then receives as input, the weighted sum of input variables z (see Equation 2.3.1), and in-turn predicts the output variable $\hat{y}$ as shown in Equation 2.3.2. To get output value $\hat{y} \in \{-1, +1\}$ we employ the hyperbolic tangent activation function. However, the choice of activation function depends on the machine learning model one intends to simulate.

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b \tag{2.3.1}$$

$$\begin{aligned} \hat{y} &= g\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right) \\ &= g(z) \end{aligned} \tag{2.3.2}$$

In this thesis, we use the terms *"weighted sum of inputs"* and *"pre-activation value"* interchangeably. Similarly, we refer to the neuron's output $\hat{y}$ as the *"post-activation value"*.

2.3.2 Activation Functions

The process of choosing the appropriate activation function is a key component in designing ANN. In the previously mentioned perceptron binary classifier (see Section 2.3.1), the ideal activation unit would be threshold-based i.e. the neuron's output would either be a -1 or $+1$ whenever the weighted input sum is above or below a set threshold. Of the diverse activation functions that exist, the *softmax*, *tanh*, *sigmoid* and *rectified linear unit (ReLu)* are the most common.

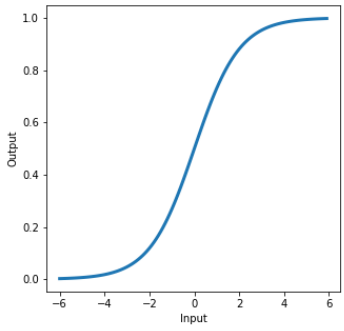


Figure 2.3: A plot of the sigmoid activation as described in Equation 2.3.3.

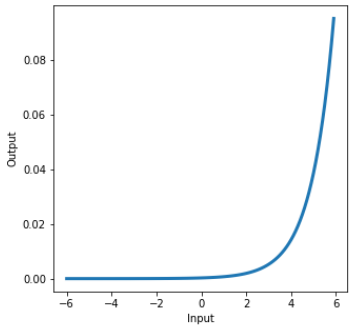


Figure 2.4: A plot of the softmax activation as described in by Equation 2.3.4.

When faced with a scenario where we seek to predict the probability of a specific binary class, the favourable activation function would be the sigmoid activation function (Equation 2.3.3), which is shown in Figure 2.3. In other words, the sigmoid activation function maps real *pre-activation values* to an interval [0,1].

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.3.3}$$

The softmax activation function shown in Figure 2.4, is defined by Equation 2.3.4 and is used when mapping non-normalised *pre-activation values* to a multi-class distribution. For example, supposing that we want to perform k-way classification, we employ the softmax activation function to map the pre-activation values $z = [z_1, z_2, ..., z_K]$ to the (set of) probability values of a multi-class distribution. To be specific, the i^{th} output is obtained in the following manner:

$$g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{-z_j}} \quad \forall i \in \{1, 2, ..., k\} \tag{2.3.4}$$

Put differently, the softmax activation function normalises each exponentiated element of vector z by dividing it with the sum of all the exponentials in the vector. As a result, the softmax activation output falls in interval $[0, 1]$. The ReLu activation function shown in Figure 2.5, gained popularity sometime early in the 21st century [66]. The ReLu function is open ended [17] and is defined as follows:

$$g(z) = max(0, z) \tag{2.3.5}$$

Unlike the tanh and sigmoid functions, the simple mathematical operation of the ReLu function makes it computationally inexpensive. Portrayed in Figure 2.6 is the tanh activation function and is expressed mathematically in the following manner:

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.3.6}$$

The tanh function maps a real *pre-activation value* to an interval [-1, 1] and it yields a zero-centered output [25] and as a result, the tanh activation function is often preferred over the sigmoid activation function.

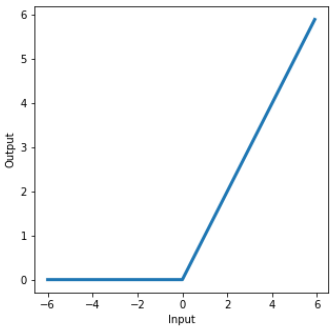


Figure 2.5: A plot of the relu activation function as defined by Equation 2.3.5.

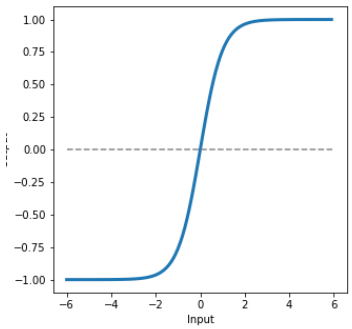


Figure 2.6: A plot of the tanh activation function as defined by Equation 2.3.6.

In summary, activation functions serve the purpose of mapping a *pre-activation value* to a different output space. The choice of an activation function is not limited to the four mentioned above. The four we have discussed are among the most popular ones in applications. However, the choice of an appropriate activation function depends on both the network architecture and the data at hand.

2.3.3 Neural Network Structure

The perceptron model described earlier in Section 2.3.1 consists of an input layer that relays data to the computation performing layer as depicted in Figure 2.2. This simple perceptron has become the fundamental unit that powers what are today known as deep neural networks. Deep artificial neural networks are an ensemble of perceptrons whereby each network contains multiple computational layers in-between the inlet and output layer as illustrated by Figure 2.7. These layers that lie between the inlet and outlet layers are are known as the *hidden layers*. The processing layers in a neural network learn data representations by performing several sequential level abstractions [57]. The entire network is sometimes called the *feed-forward network (FFN)*, a consequence of the successive layers feeding their *post-activation values* to their subsequent layers, in a sequential and forward manner, from the inlet to the outlet layer.

The feed-forward architecture is based on the assumption that each of the nodes in a single layer is connected to all the subsequent layer nodes. As a result, a neural net-

work architecture is almost entirely defined when we have the number of layers, their respective

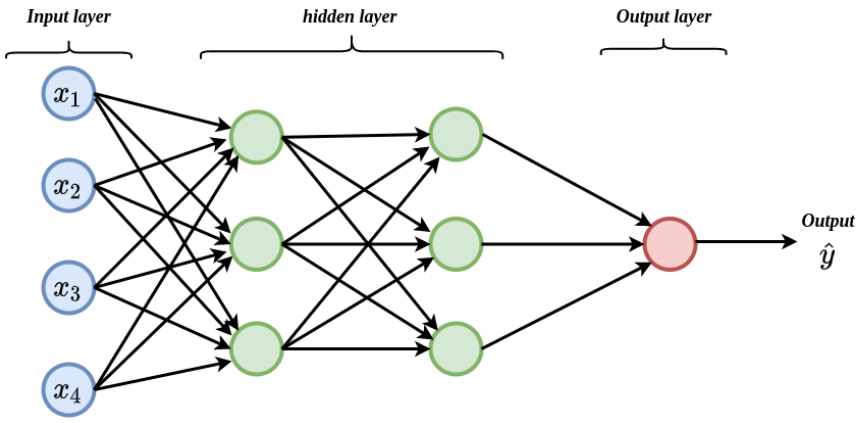


Figure 2.7: The basic architecture of a simple ANN with four layers. The first layer on the far left takes in input data and relays it to the hidden layers which in turn relays it to the single node output layer.

number of nodes and the node type. Which leaves us with the loss function (Section 2.3.4) as the only missing detail. Suppose we have a network of L layers and each layer contains n_1,n_2,...,n_L neurons respectively. We let $l \in \{1,...,L\}$ denote the l^{th} layer in the network. In sequel let z^l and z^l_i respectively denote the layer and i^{th} neuron pre-activation values. Similarly, we let a^l and a^l_i denote the layer and neuron post-activation values respectively. In like manner, $b^l \in \mathbb{R}^{n_l}$ represents the l^{th} layer's vector of biases, such that b^l_i is the respective layer's i^{th} neuron bias. The network's inlet and first hidden layers have their connecting weights contained in matrix $w^1 \in \mathbb{R}^{n_1 \times n_2}$. In the same manner, we denote the connecting weights between layers l and $(l-1)$ as $w^l \in \mathbb{R}^{n_{l-1} \times n_l}$. The edge weight between a k^{th} node belonging to layer $(l-1)$ and a j^{th} node belonging to subsequent layer l is denoted w^l_{jk}.

The network's mapping of input vector x from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$ is summarised by the following successive transformations:

$$a^1 = g\left(w^1 x + b^1\right) \tag{2.3.7}$$

$$a^{l+1} = g\left(w^{l+1} a^l + b^{l+1}\right) \ \forall\, l \in \{1,..,L-1\} \tag{2.3.8}$$

$$a^L = g\left(w^L a^{L-1} + b^L\right) \tag{2.3.9}$$

where $g(\cdot)$ denotes the activation function (one should probably notice that here the function g() applies on a vector value). Equations 2.3.7, 2.3.8 and 2.3.9 respectively represent the input to hidden layer transformation, hidden layer to hidden layer transformation and hidden layer to output layer transformation. It then follows that the post-activation value for the j^{th} node belonging to layer l is computed as follows:

$$\begin{aligned} a_j^l &= g\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \\ &= g\left(z_j^l\right), \end{aligned} \tag{2.3.10}$$

with the sum being over all the n_{l-1} neurons in layer $(l-1)$. The overall training objective of the network is to find parameters w^l and b^l that best map each input to its corresponding output. In quantifying the aforementioned objective, the network seeks to minimise the *cost* (discussed Section 2.3.4) associated with the mapping from the input (or domain-set) $\mathbb{R}^{n_1}$ to the output $\mathbb{R}^{n_L}$. It has been demonstrated [42] that networks that comprise of a single non-linear hidden layer that relays its output to a linear outlet layer can approximate almost any *rational* function.

2.3.4 Cost Function

The metric of quantifying model performance is called the cost function (or loss function). Selecting the appropriate evaluation metric is an essential requirement of machine learning as it helps characterise the model outputs in a way that is relative to a specific task or problem. For instance, let the training pairs be $\{x_i, y_i\}_i^m$, for which the i^{th} corresponding ground truth is $y(x_i)$. In training a model that comprises L layers, we define the quadratic loss function as follows:

$$\mathcal{L}(w,b) = \frac{1}{2m}\sum_x \|y(x) - a^L(x)\|^2, \tag{2.3.11}$$

where parameters w and b denote the network weights and biases. a^L represents the post-activation vector of the output layer. Intuitively, for each training pair (x, y), the loss function calculates the squared difference or magnitude between the ground-truth and model's output value. In a case where both ground-truth and predicted value are vectors, we take the difference in magnitude.

The quadratic cost function is usually preferred to those of higher powers predominantly because it has a single global minimum, which is not the case with cost functions

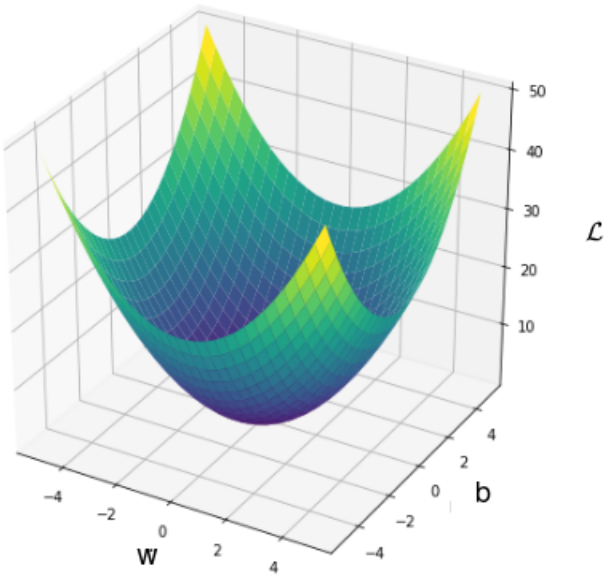


Figure 2.8: An illustration of a quadratic surface with weights and bias parameters on the x-axis and y-axis respectively. The loss function is denoted $\mathcal{L}$ and is represented by the z-axis.

of higher powers. Illustrated in Figure 2.8 is a representation of the quadratic surface with weights and bias parameters on the x-axis and y-axis respectively. The z-axis represents the cost associated with a pair of weight and bias parameters. Regardless of the type of loss function one chooses, it must satisfy two primary drivers. The first being that it must at all points be strictly greater than zero save for the global minimum as shown in Figure 2.8. It is at this global minimum that the network finds the optimal pair of parameters w and b. The second property of the cost function is that it must be differentiable, which means that the gradient of the cost function is defined at all points. To find the cost function's local minima, we employ a classical algorithm called *gradient descent* to be discussed in Section 2.3.5.

2.3.4.1 Training Conditional Distributions with Maximum Likelihood

Modern neural networks are predominantly trained using the maximum likelihood loss function [33], which implies that the cost function can be thought of as a *negative log likelihood* or *cross entropy* between the model distribution and the training data. The cost

function $\mathcal{L}(w, b)$ is then expressed as follows

$$\mathcal{L}(w, b) = -\sum_{(x,y)\in\mathcal{D}} \log p_{model}(y|x) \tag{2.3.12}$$

where $\mathcal{D}$ represents the set of training pairs (x, y) and p_{model} is the model distribution. The main advantage of using the log-likelihood as our loss function is that it eliminates the burden of defining a specific loss function for each model. In specifying a model $p(y|x)$ we simultaneously determine the loss function $-\log p(y|x)$.

When training neural networks, it is imperative that the magnitude of the cost function gradient be well-behaving to govern/assist the learning algorithm. In many cases, activation functions that vanish or explode tend to subvert this training objective. For this reason, the negative log likelihood helps alleviate this problem as some neural network output units have an exponent that saturates whenever it has an argument that is strongly negative. The logarithmic component in Equation 2.3.12 undoes the exponent in these output units. Another key factor of using the cross entropy cost function is that it usually does not have a maximum or minimum output value. In other words, for discrete output variables, we parametrise the model in such a way that it can not give as output a probability of 0 or 1. It then follows that the model's probabilities can only get arbitrarily close to 0 or 1.

2.3.5 Gradient Descent

We have shown that the criterion $\mathcal{L}(w, b)$ is a composition of multiple non-linear functions. The process of analytically computing the cost function's minimum is generally mathematically intractable. Fortunately, there exists a mathematical algorithm called *gradient descent* which finds a minimum value for differentiable functions. Gradient descent is a classical iterative algorithm and is by far the most favoured when optimising neural networks [77]. The function we seek to optimise is called the objective function or criterion. Considering our input-output pairs $\{x_i, y_i\}_i^m$ respectively, we define our training hypothesis as

$$h_\theta(x) = \theta_0 x + \theta_1, \tag{2.3.13}$$

along with loss function:

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \\ \text{Goal} &: \min_\theta \mathcal{L}(\theta) \end{aligned} \tag{2.3.14}$$

where θ represents the parameters to be optimised. For the purpose of breaking the symmetry, gradient descent sets off by randomly initialising parameter θ and then repeatedly updating it until it converges to a value that minimises $\mathcal{L}(\theta)$. The iterative updates are as follows:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{L}(\theta) \; \forall \; j = 0, .., m \;\; \text{and} \; \alpha > 0. \tag{2.3.15}$$

where *learning rate* hyper-parameter is denoted α. This learning rate controls the rate of convergence, such that small values of α lead to slow convergence and on the other hand, extremely large values result in the algorithm overshooting the minima and diverging. In implementing this algorithm one has to compute the partial derivative $\partial \mathcal{L}(\theta)/\partial(\theta_j)$ in Equation 2.3.15. Given a single training pair this partial derivative is derived as:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \mathcal{L}(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\ &= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{m} \theta_i x_i - y \right) \\ &= (h_\theta(x) - y) x_j, \end{aligned} \tag{2.3.16}$$

It then follows that a single update can be generalised to:

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \tag{2.3.17}$$

In the case of multiple training examples Equation 2.3.15 is generalised as follows:

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \;\; \text{for} \; \forall \; j \tag{2.3.18}$$

The summation in Equation 2.3.18 is equivalent to the partial derivative $\partial \mathcal{L}(\theta)/\partial \theta_j$ in Equation 2.3.15 which implies this is gradient descent on the original criterion $\mathcal{L}(\theta)$. This method is called *batch gradient descent* as the entire training set is used to compute the gradient at every iteration. Since for every update batch gradient descent requires that we compute the gradients for the entire training set, the algorithm tends to be slow for large datasets. Another form of gradient descent is *stochastic gradient descent* which for every update employs a single training and consequently increases the variance of parameter updates. A blend of both stochastic and batch gradient descent yields the

most favorable and widely variant called *mini-batch gradient descent*. Instead of using the entire training set at every update, it uses a subset or batch of b examples from m training data points. For illustrative purposes, we consider 1000 training pairs and a batch size of 10. After arbitrary parameter initialisation the parameters are then updated as follows:

$$\theta_j := \theta_j + \alpha \frac{1}{10} \sum_{k=i}^{i+9} (y^{(k)} - h_\theta(x^{(k)}))x_j^{(k)} \text{ for } \forall\, j = 0, .., n \text{ and } i = 1, 11, 21, ..., 991 \quad (2.3.19)$$

Updating the parameters with respect to each batch helps lessen the variance thus leading to more steady and smooth convergence.

Now we implement gradient descent in the training of our simple neural network shown in Figure 2.7 based on the following weight and bias update rules:

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} \quad (2.3.20)$$

$$b_j^{(l)} \leftarrow b_j^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial b_j^{(l)}} \quad (2.3.21)$$

We can view multilayered neural networks as a system of convoluted composite functions that are evaluated at each neuron. Taking the network in Figure 2.7 as an example, the computations in the l^{th} layer (output layer) can be expressed as the composite function $f(g_1(\cdot), g_2(\cdot), ..., g_{n_k}(\cdot))$, where $g_i(\cdot)$ denotes the output of a neuron belonging to layer $(l-1)$. The calculation of the loss at each layer involves a complex nested function of weights and biases of prior layers. The gradients of these nested functions are computed using a highly efficient algorithm called *back-propagation* as discussed in the following Section.

2.3.6 Back-propagation Algorithm

As discussed earlier in Section 2.3.5, gradient descent entails calculation of cost function gradients with respect to weight and bias parameters. In ANN training, the process of calculation gradients can be mathematically intractable, due to the large number of neurons and layers. Nonetheless, the advocacy of the back-propagation algorithm in the mid-1980s [99] has since brought about research renaissance in the field of deep learning, as the the training of complex network architectures had finally become feasible. Hence the application of ANN to a myriad of problems from various domains, which previously had been impossible due to computational constraints.

Back-propagation addresses the problem of learning good weights and biases for each hidden-layer neuron. Unlike the network output layer which has a target output, each hidden layer neuron has no specific target output, which renders the defining of an error function for each of these neurons impossible. In lieu of a specific target output, all the hidden layer neuron error functions would depend on the preceding and subsequent layer's parameters. In other words, the back-propagation seeks to compute the loss function partial derivatives $\partial\mathcal{L}/\partial w^l_{jk}$ and $\partial\mathcal{L}/\partial b^l_j$, with respect to the weights and biases respectively. Now we consider a single training pair (x, y) with quadratic cost function

$$\mathcal{L} = \frac{1}{2}\|y - a^K\|^2 \tag{2.3.22}$$

where a^K is the output neuron's post activation value, the only neuron on which the cost function depends. From this section on-wards we let $f'(x)$ denote the derivative of a function $f(x)$, such that the an activation function's derivative is denoted $\sigma'(x)$. Now we rewrite the post-activation value in Equation 2.3.10 in a vectorised form as

$$\begin{aligned} a^l &= g(w^l a^{l-1} + b^l) \\ &= g(z^l), \qquad\qquad \text{for} \quad l = 2, .., L, \end{aligned} \tag{2.3.23}$$

where $z^l \in \mathbb{R}^{n_l}$, such that z^l_j denotes the input to the j^{th} neuron in layer l. The computation of partial derivatives $\partial\mathcal{L}/\partial w^l_{jk}$ and $\partial\mathcal{L}/\partial b^l_j$ involves the introduction of intermediary term $\delta^l \in \mathbb{R}^{n_l}$ which denotes the l^{th} layer error term. It then follows that for each l^{th} layer we compute the j^{th} neuron error in the following manner:

$$\delta^l_j = \frac{\partial\mathcal{L}}{\partial z^l_j}, \qquad \text{for } 2 \leqslant l \leqslant L \text{ and } 1 \leqslant j \leqslant n_l \tag{2.3.24}$$

The back-propagation gives us a way of calculating the intermediate term δ^l_j, which is then linked with $\partial\mathcal{L}/\partial w^l_{jk}$ and $\partial\mathcal{L}/\partial b^l_j$. It is imperative to note that the calculating of δ^l_j will prove to be dependent on the error terms in the preceding layer. This is suggestive of the "backwards propagation of errors", hence the name back-propagation.

In the back-propagation process we start off by calculating the error term for the output layer as

$$\delta^L_j = \frac{\partial\mathcal{L}}{\partial z^L_j}. \tag{2.3.25}$$

By applying the chain rule, we then can express Equation 2.3.25 as a sum of all the neurons in the final layer

$$\delta_j^L = \sum_k \frac{\partial \mathcal{L}}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}, \tag{2.3.26}$$

where a_k^L is the k^{th} neuron post-activation value and it depends on pre-activation value z_j^L only, given that $j = k$. In other words $\partial a_k^L / \partial z_j^L$ would vanish whenever $k \neq j$. That being so, we can now rewrite Equation 2.3.26 as shown below:

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \tag{2.3.27}$$

Now it is worth noting that since $a_j^L = g(z_j^L)$, which implies we can rewrite $\partial a_j^L / \partial z_j^L$ as $g'(z_j^L)$. As a result, Equation 2.3.27 is simplified to

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} g'(z_j^L). \tag{2.3.28}$$

Rewriting Equation 2.3.28 in matrix notation yields the following:

$$\delta^L = \nabla_a C \odot g'(z^L), \tag{2.3.29}$$

where symbol $\odot$ and and $\nabla_a C$ denote the Hardamard product [41] and vector of partial derivatives $\partial \mathcal{L} / \partial a_j^L$. Having calculated the error term for the L^{th} layer, we then proceed to doing the same for the hidden layers. We now show how the calculation of δ^l is dependent on the error term of the $(l+1)^{th}$ layer. In other words we seek to express the error term $\delta_j^l = \partial \mathcal{L} / \partial z_j^l$ as a function of $\delta_k^{l+1} = \partial \mathcal{L} / \partial z_k^{l+1}$. To accomplish this, we once again employ the chain rule:

$$\begin{aligned} \delta_j^l &= \frac{\partial \mathcal{L}}{\partial z_j^l} \\ &= \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}. \end{aligned} \tag{2.3.30}$$

By the definition of intermediate term δ_j^l, the partial derivative $\partial \mathcal{L} / \partial z_k^{l+1}$ can be substituted with δ_k^{l+1} such that Equation 2.3.30 is simplified to

$$\delta_j^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{2.3.31}$$

For us to evaluate the term $\partial z_k^{l+1}/\partial z_j^l$, it is worth remembering that

$$\begin{aligned} z_k^{l+1} &= \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} \\ &= \sum_j w_{kj}^{l+1} g(z_j^l) + b_k^{l+1}. \end{aligned} \tag{2.3.32}$$

It then follows that calculating the partial derivative of z_k^{l+1} yields the following result:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} g'(z_j^l) \tag{2.3.33}$$

Substituting the above result (Equation 2.3.33) into Equation 2.3.31 will give

$$\delta_j^k = \sum_k w_{kj}^{l+1} \delta_k^{l+1} g'(z_j^l). \tag{2.3.34}$$

Expressing the above result in matrix notation yields

$$\delta^l = [(w^{l+1})^T \delta^{l+1}] \odot g'(z^l). \tag{2.3.35}$$

Now it is clear that for any l^{th} layer in the network, we can compute the error term δ^l by first computing Equation 2.3.29 followed by a repetition of Equation 2.3.35 until we reach the desired layer. Now we can express our partial derivatives as a function of δ_j^l. From Equation 2.3.10, we note that $\partial z_j^l/\partial b_j^l = 1$, therefore:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b_j^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} \\ &= \delta_j^l. \end{aligned} \tag{2.3.36}$$

Similarly, based on Equation 2.3.10 $\partial z_j^l/\partial w_{jk}^l = a_k^{l-1}$ and $\partial z_g^l/\partial w_{jk}^l = 0$ where $g \neq j$. It then follows that

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{jk}^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \\ &= \delta_j^l a_k^{l-1} \end{aligned} \tag{2.3.37}$$

Assuming we have an optimal learning rate α and arbitrary weight parameters w_{jk}^l, we summarise the back-propagation computations as follows:

Back-propagation Algorithm

1. **Feed-forward phase:** each training pair in $\{x_i, y_i\}_i^n$ is fed into the network to generate $\hat{y}_i$, z_j^l and a_j^l for each j^{th} neuron of the l^{th} layer.
2. **Backward phase:** for each neuron belonging to $(l-1)$ calculate $\partial\mathcal{L}/\partial w_{jk}^l$ and $\partial\mathcal{L}/\partial b_j^l$ based on each training pair in $\{x_i, y_i\}_i^n$.
   a) Calculate δ^L using Equation 2.3.29
   b) For each hidden layer calculate δ_j^l using Equation 2.3.34
   c) Evaluate the desired partial derivatives as given in Equations 2.3.37 and 2.3.36
3. **Summing gradients:** calculate $\partial\mathcal{L}(X, w)/\partial w_{jk}^l$ the total gradient of the training set $X = \{(x_i, y_i)\}_{i=1}^n$.
4. **Weight update:** update the weights and biases using Equations 2.3.20 and 2.3.21.

2.3.7 Regularisation for Neural Networks

The main challenge in training deep learning models is to ensure that the trained model is able to generalise well on an unseen data-set (or test data). To ensure this, machine learning practitioners keep track of the following key factors:

- model's proficiency on minimising the training error.
- model's proficiency on minimising the difference between the train and test error.

The above conditions are necessitated by the two fundamental challenges of training machine learning algorithms known as *underfitting* and *overfitting*. As shown in Figure 2.9, a scenario where a trained model fails to obtain an adequately low training error is referred to as underfitting. On the other hand, a scenario where the trained model fails to adequately minimise the error magnitude between the test and training data (see Figure 2.9) is called overfitting.

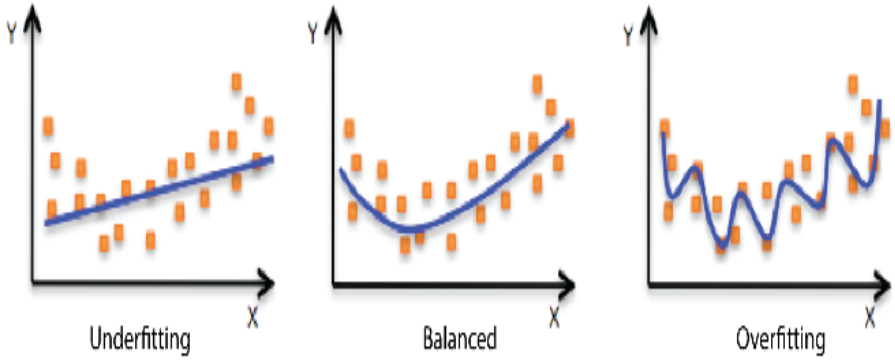
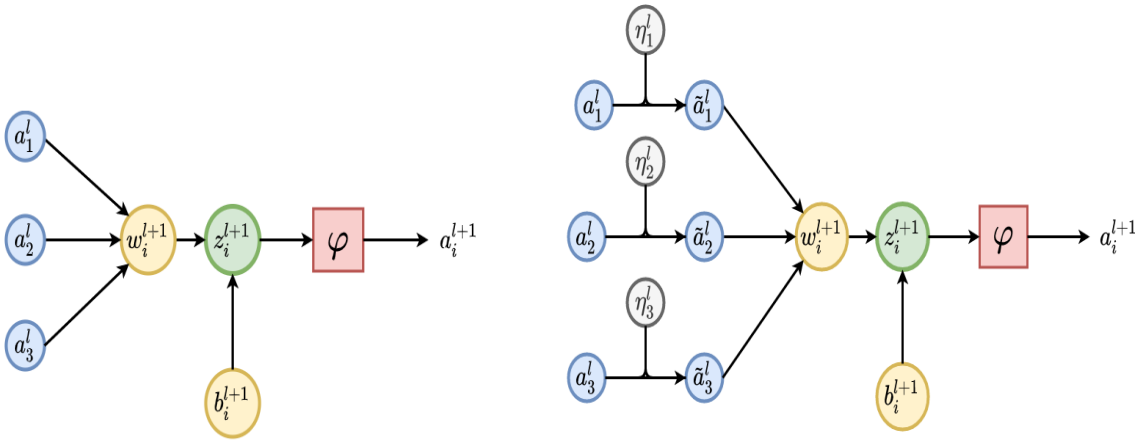


Figure 2.9: An illustration of underfitting vs overfitting [5]. To the far left is an illustration of model underfitting, on the far right is model overfitting. The sought-after generalisation is shown in the center image.



(a) An example of a simple neuron from the standard neural network in Figure 2.7

(b) A simple neuron with dropout, modified from the standard neural network.

Figure 2.10: A contrast between the standard neural network vs a neural network with dropout. Figure a) shows a standard neural network and on the other hand Figure b) shows a neural network with drop out.

Several techniques have been devised to reduce the test error. In this section we look at the one called *dropout regularisation* [81]. Regularisation is the modification of machine learning algorithms with an intent of reducing the generalisation error. It is worth noting that this reduction in generalisation error does not necessarily imply an improvement on the training error. Dropout regularisation mitigates the co-adaptation of the network's

individual hidden neurons by modifying the network architecture. This modification is done by randomly ignoring (or dropping) the neurons in the hidden layers at the forward pass, and the results are back-propagated through the modified network. The neurons are independently retained based on some fixed probability p. The weight and bias update is done over a mini-batch of examples. Thereafter the network is once again thinned by replacing the previously dropped neurons and dropping a new subset, after which the forward and back propagations are performed. This process is repeated a number of times to allow the network to learn a set of weights and biases.

Figure 2.10 shows a comparison between a standard neural network (see Figure 2.10a) and a neural network with dropout (see 2.10b). The forward pass in Figure 2.10b is computed as follows:

$$\begin{aligned} z_i^{l+1} &= w_i^{l+1}\tilde{a}^l + b_i^{l+1} \\ a_i^{l+1} &= g(z_i^{l+1}) \end{aligned} \tag{2.3.38}$$

where $\tilde{a}^l$ is the element wise product between the vector $\eta^l \sim Bernoulli(p)$ and a^l. At inference, the weights and biases of each neuron are scaled down by the probability of retaining each neuron during training. Thus ensuring that all hidden nodes have an expected output that is equivalent to the ground truth at test time.

2.4 Summary

In this chapter we have given a brief review on the history of ANN. A comprehensive review is given in [37]. A full review is beyond the scope of this thesis. We also give a brief outline of the fundamental building blocks of developing and training deep neural networks. In the next chapter, we then delve into a variant of neural networks that are commonly used in learning representations from sequential data.

Chapter 3

Recurrent Neural Networks

3.1 Chapter Organisation

In the previous chapter we delved into the basic building blocks of ANNs that typically are employed when modelling continuous data. In this chapter, we introduce a variant of ANNs that is normally used when modelling sequential data. The chapter organisation is as follows; Section 3.2 introduces the concept of sequential data and the importance of learning sequential dependencies. Subsequently, Section 3.3 then presents the fundamental units of a recurrent neural network conjointly with the respective training algorithm. The complications one may encounter in the training of recurrent networks are then discussed in Section 3.4. Section 3.5 then looks into the concept of alleviating long term dependencies with the aid of bidirectional recurrent neural networks. The next section introduces Long Short-Term Memory Neural Networks. Section 3.7 then presents a variant of the Long Short-Term Memory Neural Network called the Gated Recurrent Neural Network. The overall summary of the chapter is then presented in Section 3.8.

3.2 Introduction

The neural network architectures discussed earlier in Chapter 2 are inherently designed to process data types with attributes that are predominantly independent of each other. However, there are some data types whose attributes largely depend on each other (time sequential dependencies). Examples of time sequential data include but are not limited to, text sentences, time-series data and biological sequences in genetic code. The values of these sequential datasets can either be real-valued or symbolic. These sequential

dependencies are described as follows:

- Text data is a symbolic data type and is usually treated as a bag of words [98], however, in gaining reasonable/proper linguistic representations we have to take into consideration the order and dependence of words.
- In the case of a time-series dataset, each time-stamp (or position in the sequence) is real-valued and has an influence on the subsequent time stamps. Treating the information at each time-stamp as an independent feature results in the loss of information. For instance, given a sequential representation of some sentence, we would expect real-valued time-stamp t to be closely associated with its prior values, therefore we lose valuable information by independently treating this and other time-stamps.
- Biological sequences are of a symbolic nature and often comprise of nucleotide or amino acids which are the fundamental units of DNA [89]. Similarly, the nucleotides are arranged in sequential fashion and altering this order yields a different amino acid.

It is important to stress that while modelling sequential data, the model of choice should be capable of capturing these sequential dependencies. With most sequence-centric problems within the sphere of NLP, capturing the semantic representation is of paramount importance. To best illustrate this idea, we consider the following sentences:

Semantic Examples

Ex 3.2.1. The toothless bulldog is chasing the cat.

Ex 3.2.2. The cat is chasing the toothless bulldog.

The two sentences above are different, and to best capture the semantics in each sentence one has to utilize models that best take into consideration the relationship between a word and its neighbouring counterparts. From the sentences above, it is clear that a simple change in word order can result in a change in semantic connotations. Therefore, this shows how important it is for a model to capture word order. As a result, the two main prerequisites of processing sequential data are as follows:

- The capacity to process sequence input attributes cognizant of their ordering in the source sequence.

- All time-step inputs are treated in the same manner, relative to their respective preceding attributes.

Recurrent neural networks (RNNs) inherently address the above prerequisites. As a result, the RNN become relevant to NLP and other domains that leverage sequential data.

3.3 Recurrent Neural Network

The RNN is a variant of FFNs and has a unique design (or arrangement) that appertains to the concept of time-stratification [9]. Unlike the FFNs that have a varying number of input attributes in a solitary input layer, RNNs not only have a varying number of hidden layers but each respective layer has an input attribute that corresponds to a specific time-step. As a result, this enables the down-stream interaction of hidden layer attributes, relative to their respective sequence positions. To ensure uniform modelling at each time-step, each layer uses unique parameters, which results in a fixed number of network parameters. For this reason, the RNN can be thought of as the replication of a single layer-wise architecture over time, ergo the name *"recurrent neural networks"*.

3.3.1 RNN Architecture

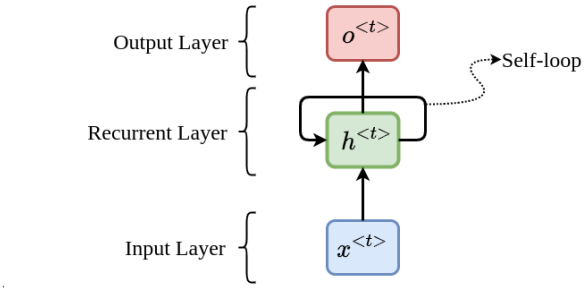


Figure 3.1: An illustration of the basic building block of a RNN. The self loop allows the passing of information from one time-step to the next, for example the learned representations at time-step $x^{<t-1>}$ is relayed to time-step t.

To best comprehend the RNN architecture we consider an input sequence of some arbitrary length T, represented as $\mathbf{x} = (x_1, x_2, x_3, ..., x_T)$, such that sequence $\mathbf{x}$ exists in some set $\mathcal{X}^*$ of all the sequences over an input space $\mathcal{X}$. We also let $\mathbf{y} = (y_1, y_2, y_3, ..., y_\tau)$ be the relative output sequence of length τ, belonging to some set $\mathcal{Y}^*$ over an output space $\mathcal{Y}$.

Suppose the RNN processes our sequence vector **x**, such that $x^{<t>}$ denotes the t^{th} token in the sequence. In principle, the RNN processes data in mini-batches, and for simplicity, the mini-batch index is omitted. The basic building block of an RNN architecture is similar to that of a traditional neural network as shown in Figure 3.1, with the addition of a self-loop being the major difference between the two. The self-loop represents the effect of the network's $< t-1 >^{th}$ value on the $< t >^{th}$ value.

Taking classification as an example use case, the generation of the respective output will not be done at every time-stamp. Instead, the output is generated at time-stamp T, as the sequence terminates. Notwithstanding that the input and output attributes may be available only to a select few time-stamps, we consider a simple scenario case wherein the attributes are accessible through out all time-stamps, such that

$$h^{<t>} = f(h^{<t-1>}, x^{<t>}). \tag{3.3.1}$$

In Equation 3.3.1 the recurrent layer, also known as the hidden state is given as a function of the preceding hidden state $h^{<t-1>}$ along with input vector $x^{<t>}$. The process of learning hidden state $h^{<t-1>}$ is done by utilizing some weights along with certain activation functions, as is the case in basic NNs. These weights are applied at every time-stamp. For this reason, even though the hidden state changes with time, the prime function $f(\cdot,\cdot)$ and weights are kept fixed through out all time-stamps, as the network is trained. The output layer generates unnormalized log probabilities from the hidden states and these are learnt through a different function

$$o^{<t>} = g(h^{<t>}). \tag{3.3.2}$$

It should be borne in mind that output vector $o^{<t>}$ comprises of continuous elements and has a dimensionality of vocabulary size i.e. $o^{<t>} \in \mathbb{R}^{|V|}$, where $|V|$ denotes the magnitude of the vocabulary. Applying a softmax layer (described in Section 2.3.2 of Chapter 2) to the output $o^{<t>}$ results in some probability vector of length $|V|$. Now, to help define the activation functions $f(\cdot,\cdot)$ and $g(\cdot)$ more concretely, we incorporate the weights into Equation 3.3.1 which yields the following

$$h^{<t>} = f(W_{hh}h^{<t-1>} + W_{hx}x^{<t>} + b_h) \tag{3.3.3}$$

likewise, Equation 3.3.2 becomes

$$o^{<t>} = g(W_{yh}h^{<t>} + b_y). \tag{3.3.4}$$

where coefficients W_{hh},W_{hx},W_{yh} b_h and b_y are the shared weights and biases respectively. By virtue of the recursive attribute of Equation 3.3.1 RNNs have the capacity to model functions of varying input lengths.

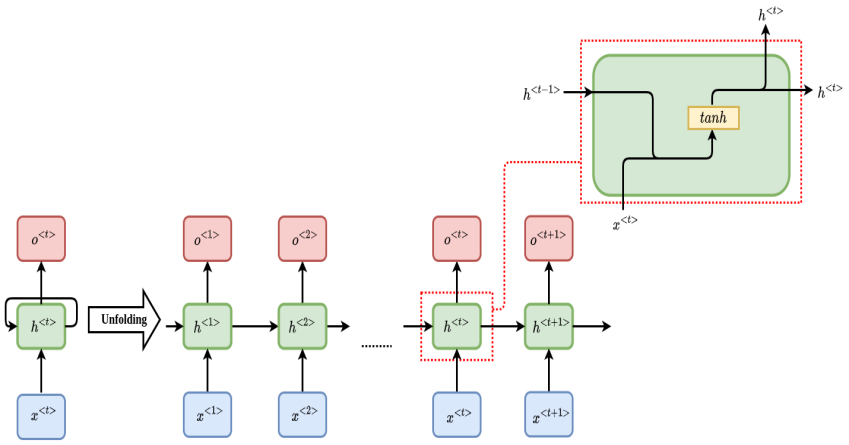


Figure 3.2: The unfolding of a RNN basic building block into a deep RNN network. Given a sequence of length t, the term "unfolding" simply means we transcribe a network for the entire sequence, to obtain a network of t layers, with each layer corresponding to it's respective time-stamp.

In essence we can unfold or expand the recursive operation of Equation 3.3.1 while maintaining its subjectivity to t, as illustrated in Figure 3.2. For instance, by beginning with some fixed constant vector $h^{<0>}$ which is also known as an initialization vector, we obtain

$$h^{<1>} = f(h^{<0>}, x^{<1>}). \tag{3.3.5}$$

and

$$h^{<2>} = f(f(h^{<0>}, x^{<1>}), x^{<2>}). \tag{3.3.6}$$

It is important to note that $h^{<1>}$ depends on $x^{<1>}$ only, whereas $h^{<2>}$ is determined by both $x^{<1>}$ and $x^{<2>}$. In summary, the forward pass initiates $h_{<0>}$ thereafter, for time-

stamp $t = 1$ to $t = T$ we implement the recursive process:

$$a^{<t>} = W_{hh}h^{<t-1>} + W_{hx}x^{<t>} + b_h \tag{3.3.7}$$

$$h^{<t>} = f(a^{<t>}) \tag{3.3.8}$$

$$o^{<t>} = W_{oh}h^{<t>} + b_o \tag{3.3.9}$$

$$\hat{y}^{<t>} = g(o^{<t>}). \tag{3.3.10}$$

The above system of equations represents a RNN mapping where both input and output are of the same length T, as shown in Figure 3.2. For this reason, to get the total loss, we would have to sum the losses over all t time stamps. For example, considering the aforementioned input $x = \{x^{<1>}, ..., x^{<t>}\}$ with respective output $y^{<t>}$ has a negative log-likelihood $\mathcal{L}^{<t>}$ as follows

$$\begin{aligned}\mathcal{L}(\{x^{<1>}, ..., x^{<\tau>}\}|\{y^{<1>}, ..., y^{<\tau>}\}) &= \sum_t \mathcal{L}^{<t>} \\ &= -\sum_t \log p_{model}(y^{<t>}|\{x^{<1>}, ..., x^{<t>}\}).\end{aligned} \tag{3.3.11}$$

with p_{model} denoting the probability distribution of the model. The process of computing gradients is computationally expensive as it involves a forward and backward pass of the unrolled network. In consequence, the time complexity of this process is $\mathcal{O}_T$ and parallelization can not help alleviate it, as the forward pass is intrinsically sequential. This implies that the computations at time-stamp $t + 1$ can only be computed after the computation of time-stamp t. Besides, the forward pass *"hidden states"* are kept in memory until back-propagation. As a result, both time complexity and memory cost are $\mathcal{O}_T$. The back-propagation process in RNNs is known as *back-propagation through time* (BPTT) and is explained in Section 3.3.3. Irrespective of their computational abilities the recursive nature of RNNs renders them computationally expensive and they take longer converge, to curb this complexity (i.e to help the network converge faster) we make use of a technique called *teacher forcing*, which we discuss in the forthcoming section.

3.3.2 Teacher Forcing

Teacher forcing is an ingenious method of training RNN to obtain quicker convergence. This technique is based on the maximum likelihood criterion, whereby during the training of time stamp $t - 1$, in-place of the predicted output $\hat{y}^{<t-1>}$ the RNN receives the ground truth $y^{<t>}$ as input. Given the nature of the maximum likelihood principle,

teacher forcing is by far the most common strategy for training RNNs [93]. For illustrative purposes, we consider two time-stamps with conditional maximum likelihood criterion

$$log\, p_{model}(y_1, y_2|x_1, x_2) = log\, p_{model}(y_2|y_1, x_1, x_2) + log\, p_{model}(y_1|x_1, x_2), \tag{3.3.12}$$

where p_{model} is the probability distribution of the model. The conditional distribution in Equation 3.3.12 represents a model whose goal is to maximise the conditional probability of y_2 taking into account both input and output sequence (x_1, x_2) and y_1 respectively.

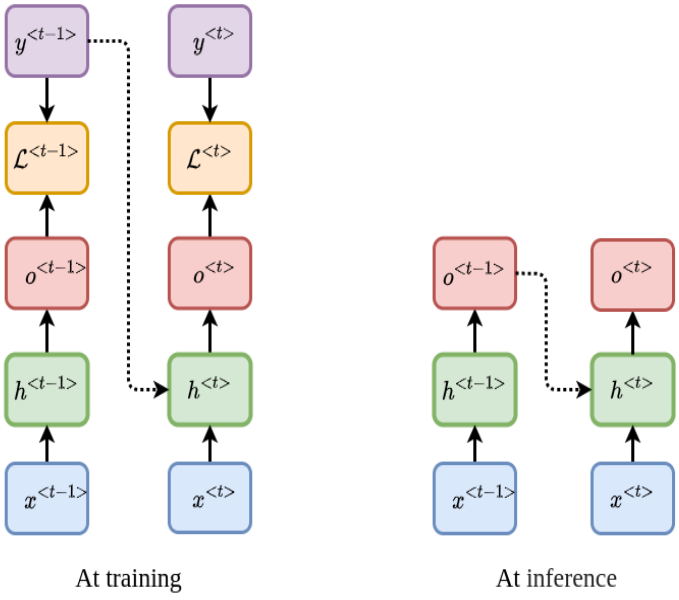


Figure 3.3: A pictorial representation of teacher forcing. In place of feeding state $h^{<t>}$ with state $h^{<t-1>}$ output, we instead feed it with its respective ground truth $y^{<t>}$. During inference the model feeds $y^{<t-1>}$ output to its subsequent hidden state $h^{<t>}$.

In training RNN with the gradient descent algorithm, teacher forcing helps inhibit various trajectories (from several initial states) from having the same point of convergence [74]. In summary, Equation 3.3.12 demonstrates that in-place of feeding the RNN with its generated outputs we feed it with the correct values (or ground truth) relative to the respective time-stamps as depicted in Figure 3.3.

3.3.3 Back-propagation Through Time

Back-propagation through time (BPTT) stems from the modification of the conventional back-propagation algorithm we discussed earlier in Section 2.3.6 and applies to the computing of gradients for RNNs. For illustrative purposes we make use of Equations 3.3.8 and 3.6.6 in explaining how BPTT algorithm operates. As in the case of Equation 3.3.11, we employ the negative log-likelihood loss function to the ground truth $y^{<t>}$ given $t-1$ inputs. We compute the model's cost function gradients with respect to its outputs $o^{<t>}$ as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \hat{o}_k^{<t>}} &= \frac{\partial \mathcal{L}}{\mathcal{L}^{<k>}} \frac{\mathcal{L}^{<k>}}{\partial \hat{o}_k^{<t>}} \\ &= \hat{y}_k^{<t>} - I_{(k,j_t)},\end{aligned} \tag{3.3.13}$$

where

$$I_{(k,j_t)} = \begin{cases} 1 & \text{if } k = j_t \\ 0 & \text{otherwise} \end{cases}$$

As is the case of simple neural networks, our computations take a backward direction starting from the time stamp $t = \tau$ to $t = 1$. Which implies the final hidden state $h^{<\tau>}$ possesses a single progeny $o^{<\tau>}$ and as such, its gradients are computed as follows:

$$\frac{\partial \mathcal{L}}{\partial h^{<\tau>}} = W_{oh}^{\top} \frac{\partial \mathcal{L}}{\partial o^{<\tau>}}. \tag{3.3.14}$$

However, in the case of time stamp $t = \tau - 1$ up to $t = 1$ the hidden state $h^{<t>}$ has $o^{<t>}$ and $h^{<t-1>}$ as its progeny. In this case $h^{<t>}$ gradients are given by

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial h^{<t>}} &= \left(\frac{\partial h^{<t-1>}}{\partial h^{<t>}}\right)^{\top} \left(\frac{\partial \mathcal{L}}{\partial h^{<t+1>}}\right) + \left(\frac{\partial o^{<t>}}{\partial h^{<t>}}\right)^{\top} \left(\frac{\partial \mathcal{L}}{\partial o^{<t>}}\right) \\ &= W_{hh}^{\top} \left(\frac{\partial \mathcal{L}}{\partial h^{<t+1>}}\right) \mathcal{D} + W_{oh}^{\top} \left(\frac{\partial \mathcal{L}}{\partial o^{<t>}}\right),\end{aligned} \tag{3.3.15}$$

where diagonal matrix $\mathcal{D} = diag(1 - (h^{<t+1>})^2)$, comprising of $< t+1 >^{th}$ elements, $1 - (h_i^{<t+1>})^2$. Which are the jacobians of the hyperbolic tangents in relation to hidden state i.

Having obtained the computational graph's internal node gradients we proceed to compute the weight node gradients. In doing this, implementing conventional back propagation discussed in Section 2.3.6 will only lead to the layer update conundrum as the weights are shared across different time stamps. This weight sharing has an influence

on the update process. To enable straightforward weight update we assume that the weights in different layers are independent of each other. In consonance with this assumption we introduce dummy variables $W_{hh}^{<t>}$, $W_{hx}^{<t>}$ and $W_{hy}^{<t>}$ for time stamp t. As a result, we can now perform standard back-propagation under the supposition that the shared weights are independent of each other. Thereafter we sum the gradient contribution of each temporal weight archetype to form a single weight update for each parameter. The gradients of the remaining parameters are computed as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_o} &= \sum_t \left(\frac{\partial o^{<t>}}{\partial b_o}\right)^\top \left(\frac{\partial \mathcal{L}}{\partial o^{<t>}}\right) \\ &= \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{<t>}}\right),\end{aligned} \tag{3.3.16}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_h} &= \sum_t \left(\frac{\partial h^{<t>}}{\partial b_h^{<t>}}\right)^\top \left(\frac{\partial \mathcal{L}}{\partial h^{<t>}}\right) \\ &= \sum_t diag(1-(h^{<t>})^2)\left(\frac{\partial \mathcal{L}}{\partial h^{<t>}}\right),\end{aligned} \tag{3.3.17}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_{oh}} &= \sum_t \sum_i \left(\frac{\partial \mathcal{L}}{\partial o_i^{<t>}}\right) \frac{\partial o_i^{<t>}}{\partial W_{oh}} \\ &= \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{<t>}}\right) h^{<t>\top}\end{aligned} \tag{3.3.18}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_{hh}} &= \sum_t \sum_i \left(\frac{\partial \mathcal{L}}{\partial h_i^{<t>}}\right) \frac{\partial h_i^{<t>}}{\partial W_{hh}^{<t>}} \\ &= \sum_t diag(1-(h^{<t>})^2)\left(\frac{\partial \mathcal{L}}{\partial h^{<t>}}\right) h^{<t-1>\top}\end{aligned} \tag{3.3.19}$$

$$\begin{aligned}\frac{\partial C}{\partial W_{hx}} &= \sum_t \sum_i \left(\frac{\partial \mathcal{L}}{\partial h_i^{<t>}}\right) \frac{\partial h_i^{<t>}}{\partial W_{hx}} \\ &= \sum_t diag(1-(h^{<t>})^2)\left(\frac{\partial \mathcal{L}}{\partial h^{<t>}}\right) x^{<t>\top}\end{aligned} \tag{3.3.20}$$

Computing the gradients with respect to $x^{<t>}$ is not necessary, owing to it not having any parent parameters in the loss defining computational graph. The gradient computations above follow a direct implementation of the multivariate chain rule. Much like all back-propagation algorithms with shared weights, we have benefited from the fact that partial derivatives of temporal avatar parameters (such as $W_{hh}^{<t>}$) with respect to

the primary parameter can be set to 1. Therefore, it follows that for us to compute the update equations we have to wrap the temporal assemblage around traditional back-propagation. This ingenious algorithm of back-propagating through time is believed to have been introduced in Werbos' fundamental work sometime in 1990 before RNN became more prominent [91].

3.4 The Vanishing and Exploding Gradient Problems

Given a set of weight matrices, the process of training RNN appears simple and straight forward, however, the recurrent connection makes it a complicated task. To be precise, the difficulty arises from the manner in which the gradients of preceding layers relate with their successive layers. To best conceptualize the reason for this difficulty we analyse the back-propagation of Equations 3.3.8 and 3.3.10. Having completed the forward pass computations, it is worth noting that during the backward pass the gradients may become too small or too large. Which results in a phenomenon called *vanishing and exploding gradient problem* [73], respectively. The conundrum of vanishing and exploding gradients was discovered by [39] [16] and [18], independently. One might anticipate that this problem may be solved by avoiding the regions with vanishing or exploding gradients. However, for the RNN to be able to save memories in a manner that remains sturdy under minor perturbations, it has to get into such a space [18] (the regions being avoided). This phenomenon is prominent when learning *long-term dependencies* in sequential data. [18] gives a detailed outline of how learning long term dependencies gives rise to the vanishing gradient problem. In essence, this phenomenon is unnoticeable as we go a few time steps back in time but is rather more evident when we go further back in time.

Vanishing of gradients make it difficult for optimization algorithms to know which direction (or slope) to take when optimizing the weights. On the other hand, the exploding of gradients results in an unstable learning process. Looking at the aforementioned equations of consideration, we only assess the weights that are associated with the input, output and recurrent nodes denoted W_{xh}, W_{yh} and W_{hh} respectively. In calculating the loss of a sequence of length τ we sum the losses of all the time-stamps. For illustrative purposes, we opt to compute only the gradients that are with respect to hidden-hidden

recurrence weights W_{hh}.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_{hh}} &= \frac{\partial \mathcal{L}}{\partial \hat{y}^{<t>}} \frac{\partial \hat{y}^{<t>}}{\partial h^{<t>}} \frac{\partial h^{<t>}}{\partial W_{hh}} \\ &= \sum_t \frac{\partial C^{<t>}}{\partial \hat{y}^{<t>}} \frac{\partial \hat{y}^{<t>}}{\partial h^{<t>}} \frac{\partial h^{<t>}}{\partial h^{<k>}} \frac{\partial h^{<k>}}{\partial W_{hh}}\end{aligned} \tag{3.4.1}$$

In Equation 3.4.1 above, we examine the term $\frac{\partial h^{<t>}}{\partial h^{<k>}}$ and apply the chain rule to compute its gradient as follows

$$\begin{aligned}\frac{\partial h^{<t>}}{\partial h^{<k>}} &= \frac{\partial h^{<t>}}{\partial h^{<t-1>}} \frac{\partial h^{<t-1>}}{\partial h^{<t-2>}} \cdots \frac{\partial h^{<k>}}{\partial h^{<k-1>}} \\ &= \prod_{j=k+1}^{t} \frac{\partial h^{<j>}}{\partial h^{<j-1>}}.\end{aligned} \tag{3.4.2}$$

The resulting product in Equation 3.4.2 represents a derivative between two vectors, which implies a Jacobian matrix of the time-step to time-step transition function. As a result, the term $\frac{\partial h^{<j>}}{\partial h^{<k>}}$ represents the product of Jacobian matrices that are each associated with a forward computation step. We further examine the term $\frac{\partial h^{<j>}}{\partial h^{<j-1>}}$ (in Equation 3.4.2) as follows

$$\frac{\partial h^{<j>}}{\partial h^{<j-1>}} = W_{hh}^{\top} \, diag(1 - (h_j)^2). \tag{3.4.3}$$

Taking the norm of Equation 3.4.3 yields the following

$$\begin{aligned}\left\| \frac{\partial h^{<j>}}{\partial h^{<j-1>}} \right\| &= \left\| W_{hh}^{\top} \, diag(1 - (h^{<j>})^2) \right\| \\ &\leq \left\| W_{hh}^{\top} \right\| \left\| diag(1 - (h^{<j>})^2) \right\| \\ &= \beta_w \beta_h,\end{aligned} \tag{3.4.4}$$

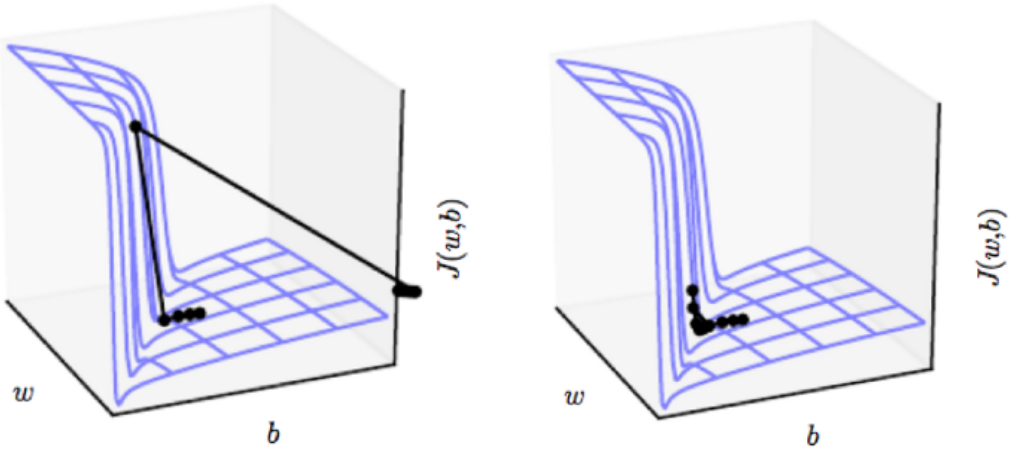
where $\beta_w = \left\| W_{hh}^{\top} \right\|$ and $\beta_h = \left\| diag(1 - (h^{<j>})^2) \right\|$. Now when we combine Equations 3.4.2 and 3.4.3 it yields

$$\left\| \frac{\partial h^{<t>}}{\partial h^{<k>}} \right\| = \left\| \prod_{h=k+1}^{t} \frac{\partial h^{<j>}}{\partial h^{<j-1>}} \right\| \leq (\beta_W \beta_h)^{t-k} \tag{3.4.5}$$

It then follows that as the sequence length t becomes larger ($t \to \infty$), Equation 3.4.5 tends to explode or vanish, regardless of the magnitude of $\beta_W \beta_h$. To overpass this phenomenon we employ a technique called *gradient clipping* [73], and or variants of RNN called long short-term memory (LSTM) and gated recurrent unit (GRU) networks. Yoshua Bengio [18] gives a detailed outline of how learning long term dependencies brings about vanishing gradients.

3.4.0.1 Gradient Clipping

As discussed earlier in Section 3.4, the non-linear functions computed by deep RNN prioritize gradients with magnitudes that are either too large or too small. Figure 3.4a below, illustrates a representation of gradient explosion where gradient descent updates toss the parameters into a region with a larger objective function. The tossing of parameters nullifies all the work done in obtaining the current solution. A simple solution to this problem is *gradient clipping*, a technique which was proposed by Tomas Mikolov [73].



(a) Learning without clipping (b) Learning with clipping

Figure 3.4: An illustration of clipping RNN gradients with the weight and bias parameters denoted w and b, respectively. On the left is a case where gradients overshoot which calamitously hurls the parameters of interest away from the solution neighbourhood. On the right side is the is a scenario whereby the gradient is scaled to a threshold which in turn restricts the step-size such that the parameter updates are within the vicinity of the solution.[33]

The main idea of gradient clipping is based on scaling the gradient of a magnitude exceeding a predetermined threshold into matching the set threshold. Figure 3.4b illus-

Algorithm 1 Gradient clipping.

1: **Result:** $g \leftarrow \frac{gv}{\|g\|}$
2: **if** $\|g\| > v$ **then**
3: $\quad g \leftarrow \frac{gv}{\|g\|}$
4: **end if**

trates the effects of gradient clipping. In contrast with the exploding gradients in Figure 3.4a, the clipped optimizer tends to perform best around a cliff. As the parameter updates ascend the cliff, we restrict the step-size such that the updates do not move further away from the solution neighbourhood [73].

3.5 Bidirectional Recurrent Neural Networks

One major drawback of the RNN discussed in earlier sections is that at hidden state $h^{<t>}$ computation the RNN only considers information about its prior hidden states but has no knowledge of the subsequent hidden states. In some applications of sequence modelling, the model performance is vastly improved by availing both historical and futuristic information to hidden state $h^{<t>}$. For example, to gain some intuition on the semantic representations of Examples 3.2.1 and 3.2.2 at each time-stamp t, the RNN has to leverage information about both prior and subsequent words which in turn results in improved context understanding. It is from this idea of learning sequences from both directions that the bidirectional recurrent neural network emanates from [79].

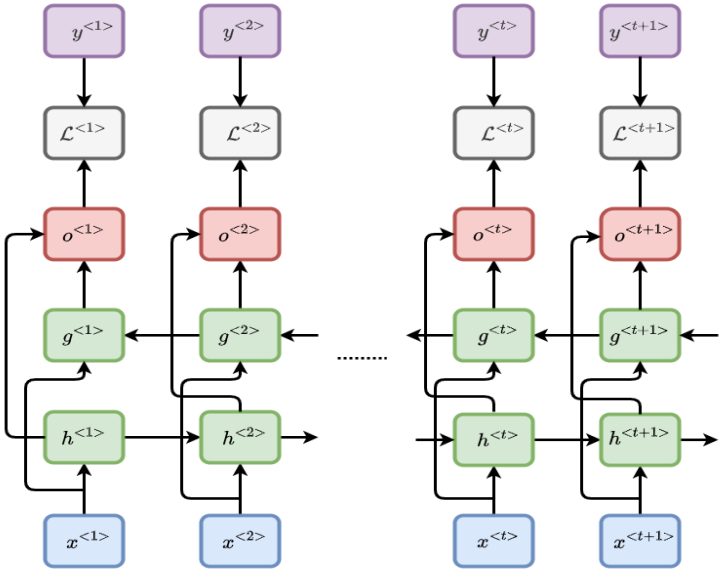


Figure 3.5: A standard bidirectional RNN that maps input vector x to target output vector y such that the cost function $\mathcal{L}^{<t>}$ represents the loss at time stamp t. At each time-stamp t the recurrences $h^{<t>}$ and $g^{<t>}$ the propagation of information in the forward and backward directions respectively. Therefore, output unit $o^{<t>}$ leverages historic and futuristic information from $h^{<t>}$ and $g^{<t>}$ computations respectively.

As illustrated in Figure 3.5, bidirectional RNNs combine two RNN whereby one propagates through time in the forward direction while the other propagates in the backward direction. As a result, at each time-stamp, bidirectional RNNs have separate independent hidden states $h^{<t>}$ and $g^{<t>}$. In other words, each hidden state $h^{<t>}$ only interacts with its prior hidden state $h^{<t-1>}$. Similarly, hidden state $g^{<t>}$ only interacts with its prior hidden state $g^{<t-1>}$. The major development in this network is the stacking of combinations of two unidirectional layers or sub-networks. On one layer, the forward-hidden states interact in the forward direction from the time-stamp $t = 1$ to $t = \tau$, where τ is the sentence length. On the other layer, hidden states interact in the backward direction from time-stamp $t = \tau$ to $t = 1$. However, both $h^{<t>}$ and $g^{<t>}$ inputs are from the same sequence, the same token $x^{<t>}$. Consequently, this enables the computation of output units $o^{<t>}$ to capture representations of information from both the prior and subsequent hidden states. Therefore, the representations in $o^{<t>}$ are strongly related to the inputs (or words) within the neighbourhood of input $x^{<t>}$. All this happens without specifying a fixed size neighbourhood window.

In the RNN under consideration we have a separate set of weights for the forward pass and backward pass sub-networks. We denote the forward pass weights as W_{xh}, W_{hh}, W_{hy} and b_h. The backward pass parameters are then denoted as W_{xg}, W_{gg}, W_{gy} and b_g, such that the recurrence propagation conditions are computed in the following manner:

$$h^{<t>} = tanh(W_{hh}h^{<t-1>} + W_{hx}x^{<t>} + b_h) \tag{3.5.1}$$

$$g^{<t>} = tanh(W_{gg}g^{<t-1>} + W_{gx}x^{<t>} + b_g) \tag{3.5.2}$$

$$o^{<t>} = W_{oh}h^{<t>} + W_{og}g^{<t>} + b_o. \tag{3.5.3}$$

From Equations 3.5.1 to 3.5.3 above, it can be seen that the recurrence conditions are a generalization of the standard RNN recurrence conditions in Equations 3.3.7 to 3.6.6. Another observation from these recurrence equations is that the hidden states $h^{<t>}$ and $g^{<t>}$ are independent of each other, meaning one can compute the forward pass hidden states and then compute the backward pass hidden states concurrently. Thereafter, combining each time-stamp's backward and forward pass hidden states in computing the respective output states.

Following the output computations, we apply back-propagation in computing the gradients of the various parameters. To achieve this we first compute the gradients with

respect to the output states. Thereafter, we compute the forward pass parameter gradients from time-stamp $t = \tau$ to $t = 1$. We then compute the backward pass gradients from time-stamp $t = 1$ to $t = \tau$. Finally, the gradients are aggregated based on the respective shared parameters. Consequently, BPTT can be modified to suit the case of bidirectional RNN as summarised below.

Bidirectional RNN training

1. **Step 1:** Independently and separately compute the forward pass and backward pass hidden states.
2. **Step 2:** Compute the time-stamp output states relative to the forward and backward pass hidden states.
3. **Step 3:** Compute the loss concerning each output state and output dummy parameter.
4. **Step 4:** Independently calculate the gradients concerning the forward and backward pass states. Use these results to evaluate the partial derivatives for each forward and backward-pass dummy parameters.
5. **Step 5:** Aggregate the gradients of the shared parameters.

The performance of Bidirectional RNN is almost similar to that of an ensemble of two separate RNNs, whereby one has as its input, the original input vector x and the other has the reversed vector x as its input. The difference between the ensemble and bidirectional RNN is that the bidirectional forward and backward-pass are jointly trained. However, this joint training exhibits some weakness as the forward and backward states are not directly related.

3.6 Long Short-Term Memory Neural Networks

As mentioned earlier, the major drawback with conventional RNNs is that they struggle with learning long term dependencies, a consequence of the vanishing and exploding gradient problem [15]. In addition to gradient clipping, another way of handling this problem is by using a version of RNNs known as *long short-term memory* (LSTM) neural networks [39]. The principal idea behind these LSTMs is to substitute the conventional

RNN hidden-hidden recurrence cell with an LSTM cell. The main objective of these LSTMs is to have control over the information stored in memory. Figure 3.6 below depicts a simple LSTM cell that replaces the conventional RNN cell.

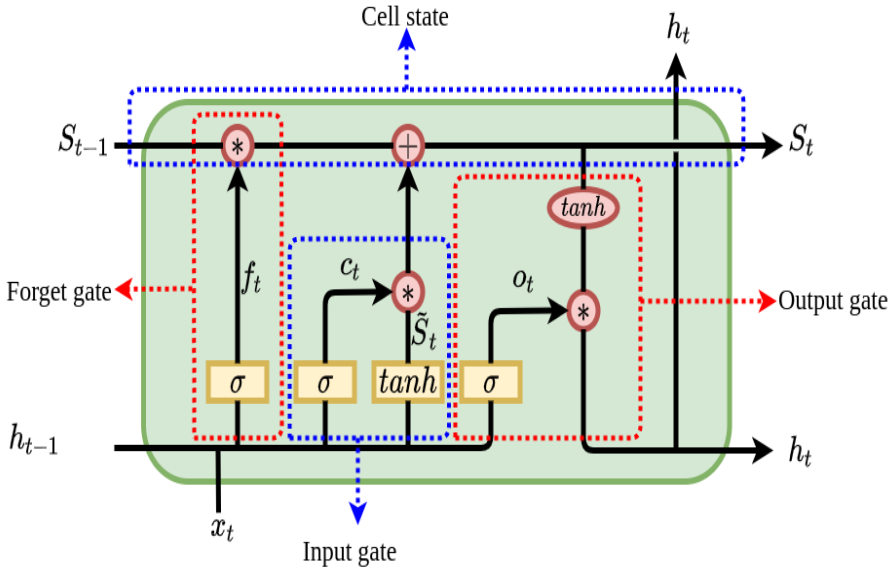


Figure 3.6: An illustration of an LSTM recurrent cell. The horizontal line at the top of the cell represents information flow along the cell state. The LSTM cell inputs and outputs are represented by $(x^{<t>}, h_{t-1}, S_{t-1})$ and $(h^{<t>}, S_t)$ respectively. The forget gate determines what information from previous cell state S_{t-1} to filter out. The input gate determines what information from the current time step is worth filtering out. The output gate determines the representations to pass as output relative to cell state S_t and filtered input o_t.

In contrast with conventional an RNN that essentially implements component-wise non-linearity to the affine transformation of the input sequence and hidden states, LSTMs comprise of a cell that posses an inherent interior self-loop. This self-loop generally implies that LSTM networks have two self-loops, one within the cells and the other being the outer conventional RNN self-loop. Moreover, LSTM RNN cells have a grid of gates that curb information flow. As in the case of conventional RNN, LSTMs have a unique or specific input and output with their key component being the *cell state*. In Figure 3.6, the cell state is represented by the horizontal line running from C_{t-1} to C_t and has a

couple of linear alterations to it. At time-stamp t the amount of information that flows through the network's i^{th} cell self-loop is regulated by the forget gate $f_i^{<t>}$ whose output values are between 0 and 1.

$$f_i^{<t>} = \sum_j W_{x,i,j}^f x_j^{<t>} + \sum_j W_{h,i,j}^f h_j^{<t>} + b_i^f \tag{3.6.1}$$

where W_x^f, W_h^f and b^f represent the input weight, hidden-hidden weights and biases respectively. The input and recurrence vectors are denoted $x^{<t>}$ and $h^{<t>}$ respectively. The following step subsequently determines what information to store in the cell state $\tilde{S}^{<t>}$. This information regulator has two phases, the first $c^{<t>}$ determines which values are to be updated next. The second phase $\tilde{S}^{<t>}$ generates a vector of candidate values which contribute towards the updating of the cell state $S^{<t>}$.

$$c_i^{<t>} = \sigma\left(\sum_j W_{x,i,j}^c x_j^{<t>} + \sum_j W_{h,i,j}^c h_j^{<t-1>} + b_i^c\right) \tag{3.6.2}$$

$$\tilde{S}_i^{<t>} = \tanh\left(\sum_j W_{x,i,j}^{\tilde{S}} x_j^{<t>} + \sum_j W_{h,i,j}^{\tilde{S}} h_j^{<t-1>} + b_i^{\tilde{S}}\right). \tag{3.6.3}$$

Equations 3.6.2 and 3.6.3 are then combined to update the cell state as follows

$$S_i^{<t>} = f_i^{<t>} S^{<t-1>} + c_i^{<t>} \tilde{S}_i^{<t>} \tag{3.6.4}$$

Finally, the LSTM cell then computes the output $h^{<t>}$, by multiplying the output gate $o^{<t>}$ by the filtered cell state in the following manner

$$h_i^{<t>} = o_i^{<t>} \tanh S_i^{<t>} \tag{3.6.5}$$

where

$$o_i^{<t>} = \sigma\left(\sum_j W_{x,i,j}^o x_j^{<t>} + \sum_j W_{h,i,j}^o h_j^{<t-1>} + b_i^o\right). \tag{3.6.6}$$

Compared to conventional RNN cells, LSTMs have proven to be more capable of handling/learning long term dependencies [38]. However, there are pertinent questions that arise from LSTM architectures, such as:

1. Which LSTM cell components are more important ?
2. How can one regulate the forget gate performance ?

In answering these questions, we delve into an ingenious variant of the LSTM cell called the *gated recurrent unit*.

3.7 Gated Recurrent Units

In response to the aforementioned questions (see Section 3.6) we consider the GRU which has crafty alterations to the LSTM cell [23]. An example of a GRU cell is shown in Figure 3.7 below. The GRU can be thought of as a simplified version of the LSTM with the absence of an explicit cell state as the major distinguishing factor. In contrast with the LSTM which has forget and output gates for controlling the flow of information within the cell, GRUs have this dual functionality handled by a single reset gate. In essence, this GRU gating unit simultaneously scales both the forget component and the state unit update.

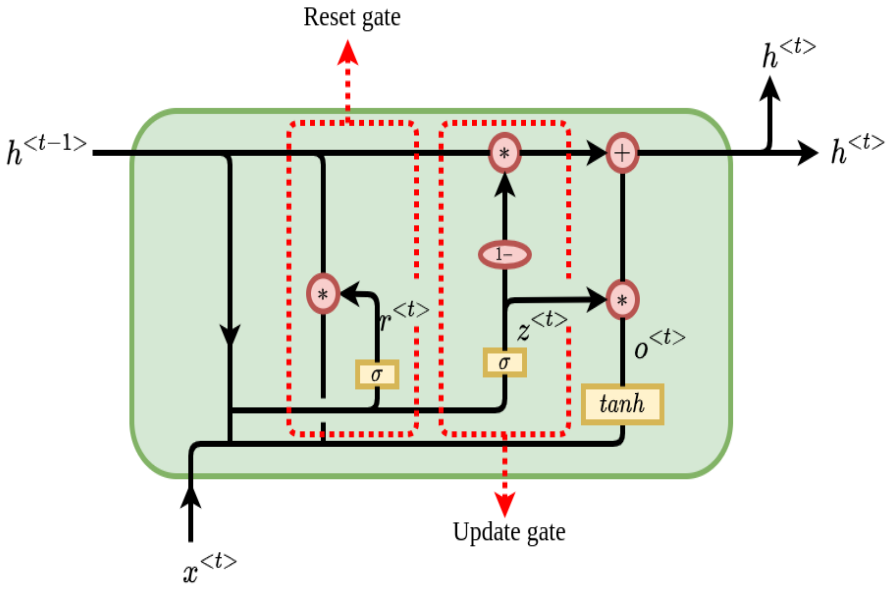


Figure 3.7: A pictorial representation of the GRU cell where inputs and outputs are denoted $(x^{<t>}, h^{<t-1>})$ and $h^{<t>}$ respectively. In contrast with the LSTM which has forget and output gates for controlling the flow of information within the cell, GRUs have this dual functionality handled by a single reset gate. The reset and update gates scale both the forget and state unit update.

Nonetheless, the partial resetting of the hidden states within the GRU cell closely resembles that of the LSTM. On this account the GRU cell has the following update equations:

$$h_i^{<t>} = z_i^{<t-1>} h_i^{<t>} + (1 - z_i^{<t-1>}) o_i^{<t>}, \tag{3.7.1}$$

where $z_i^{<t-1>}$ denotes an update unit and is defined as:

$$z_i^{<t-1>} = \sigma\left(\sum_j W_{x,i,j}^z x_j^{<t>} + \sum_j W_{h,i,j}^z r_j^{<t>} h_j^{<t-1>} + b_i^z\right), \tag{3.7.2}$$

and

$$o_i^{<t>} = \tanh\left(\sum_j W_{x,i,j}^o x_j^{<t>} + \sum_j W_{h,i,j}^o r_j^{<t>} h_j^{<t-1>} + b_i^o\right), \tag{3.7.3}$$

the reset unit $r^{<t>}$ is defined as:

$$r_i^{<t>} = \sigma\left(\sum_j W_{x,i,j}^r x_j^{<t>} + \sum_j W_{h,i,j}^r h_j^{<t-1>} + b_i^r\right). \tag{3.7.4}$$

These update and reset gates can sometimes ignore a portion of the state vector $h^{<t-1>}$. Reset gate $r^{<t>}$ regulates the proportion of states that are used when computing the target state $h^{<t>}$. In doing so, nonlinear relation between the $< t-1 >^{th}$ and $< t >^{th}$ state is introduced by the reset gate. An in-depth investigation on the performance of the LSTM variants is found in the work of [82] and that of [46].

3.8 Summary

This chapter has given a brief introduction to the modelling of sequential data with RNNs and its variants. Additionally, the chapter outlines the process of training RNN. Having discussed the training of RNNs, the chapter then gave an outline of the complications that arise from learning to model long sequences. As a means of alleviating these long term dependencies, then followed an analysis of bidirectional RNNs, LSTMs and GRUs architectures.

Chapter 4

Sequence-to-Sequence Encoder-Decoder Architectures

4.1 Chapter Organisation

In Section 4.2 we lay out an overview of the *many-to-one* and *one-to-many* mappings. The next section (Section 4.3) then introduces the basic building blocks of an encoder-decoder architecture. Thereafter, in Section 4.4 we then discuss the *attention mechanism*, an ingenious solution to the long term dependency problem. The discussion on the attention mechanism includes the *self-attention* mechanism which constitutes the fundamental component of the transformer network. The transformer network is then introduced in Section 4.5. A summary is given in Section 4.6.

4.2 Introduction

We define *sequence-to-sequence* modelling as the mapping of a fixed-length sequence to another sequence of a fixed length. In these mappings, the output sequence length can differ from that of the input. Earlier in Chapter 3, we demonstrated how RNNs map a fixed size input sequence to another fixed-size sequence. We refer to this type of mapping as a *many to many* mapping. This type of mapping has one major restriction, which is that both input and output sequence lengths must be equal. In Section 4.3 we discuss how this restriction is solved. Furthermore, RNN also allow us to perform *many to one* and *one to many* mappings. In addition to their ability to perform sequence modelling, the flexibility of RNN architectures with regards to the type of mapping is what has made them so prominent in the domain of sequence modelling.

4.2.1 Many to One Mappings

We have so far only discussed RNN models where both input and output sequences are of the same lengths. However, RNNs are also capable of mapping some sequence of fixed-length to a single output as shown in Figure 4.1. Similar to the RNN discussed in Section 3.3.1, Figure 4.1 represents a time unfolded RNN basic building block with the only difference being the single output in Figure 4.1. Many to one models are largely used in sentiment classification and when producing an output that is further processed by another network (such as the network in Section 4.2.2). When training the model, the gradient concerning the output $o^{<\tau>}$ is computed by back-propagating from the subsequent nodes.

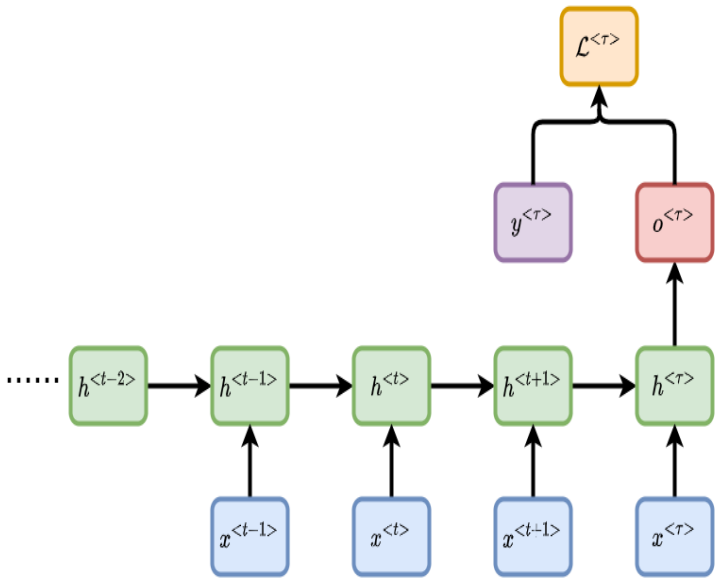


Figure 4.1: An illustration of a RNN that maps a fixed length input sequence to a single output. The network cells can be of any form of RNN variant. An input sequence **x** is fed into the network which then learns to map the sequence to a single output $o^{<\tau>}$.

4.2.2 One to Many Mappings

As discussed earlier in Section 2.3.4.1, any model that represents variable $p(y;(w,b))$ might be described as a conditional distribution $p(y|\theta)$ where $\theta = (w,b)$. If we express θ as a function of input x the model $p(y|\theta)$ can then be extended to $p(y|x)$. In general this represents distribution over y variables conditioned on a single input x. This condition-

ing is also called a one to many mapping. To achieve this, for each time-step t the RNN takes as input the target variable $y^{<t-1>}$ and the weighted input x as shown in Figure 4.2 below. The reciprocal relationship between the input vector x and each $h^{<t>}$ (the hidden state) is parametrized by the matrix of weighted input $x^\top W$, which was absent in the *many-to-many* and *many-to-one* model. At all time-steps the product $x^\top W$ is added to the hidden unit inputs.

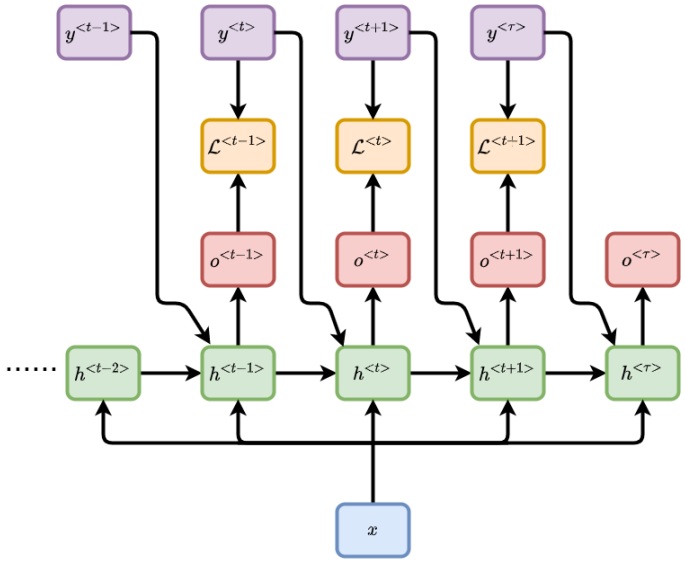


Figure 4.2: An illustration of a RNN that represents the distribution over some y variables conditioned on a fixed length input vector x. When training, each element $y^{<t>}$ serves both as an input (at time t) and as the target for time step $t-1$.

4.3 Encoder-Decoder

We have discussed how RNN can learn to model a many-to-many, many-to-one and one-to-many mappings (see Sections 3.3.1, 4.2.1 and 4.2.2). Based on these mapping variants, we now discuss how RNNs can be trained to model a sequence-to-sequence mapping where input and output sequences are of different lengths. In principle, such models have applications in the domain of NLP for example, machine translation, question answering and speech recognition. Although the input and output sequence lengths might be different, they tend to be related [33]. The encoder-decoder architecture is a type of network that maps a sequence of varying length to an output sequence of vari-

able length was first developed by [84] and [22].

The main idea of the encoder-decoder architecture is based on coupling the many-to-one with the one-to-many model discussed earlier in Sections 4.2.1 and 4.2.2. As depicted in Figure 4.3, the many-to-one RNN model output is passed onto the one-to-many RNN model as input. For nomenclature we denote our input and target sequences as $x = (x^{<1>}, x^{<2>}, ..., x^{<\tau_1>})$ and $y = (y^{<1>}, y^{<2>}, ..., y^{<\tau_2>})$ respectively. Where input and output sequence lengths are denoted by τ_1 and τ_2 respectively. For each input-output pair of sentences $(x, y) \in \mathcal{D}$ we add artificial tokens $< SOS >$ and $< EOS >$ to denote the *start-of-sentence* and *end-of-sentence*, such that $x^{<1>} =< SOS >$,$y^{<1>} =< SOS >$,$x^{<\tau_1>} =< EOS >$ and $y^{<\tau_2>} =< EOS >$. In essence, the

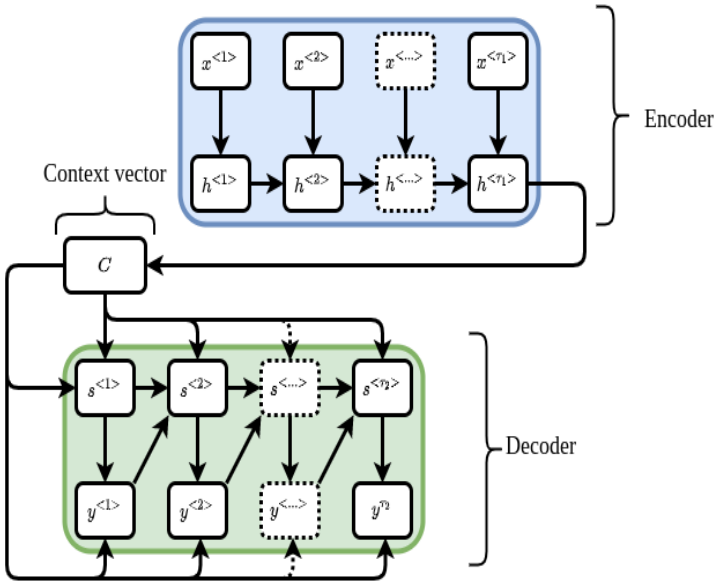


Figure 4.3: A pictorial representation of a RNN encoder-decoder architecture that maps sequence $x = (x^{<1>}, x^{<2>}, ..., x^{<\tau_1>})$ to sequence $y = (y^{<1>}, y^{<2>}, ..., y^{<\tau_2>})$. The encoder RNN processes the input to generate the context vector C, which is passed onto the decoder RNN as input. In essence, the encoder is viewed as a many-to-one mapping and the decoder as a one-to-many mapping.

processing steps of this architecture are as follows:

1. The encoder or many-to-one component learns representations that summarise the source sentence x into c, a vector of fixed-length. Through a RNN, these represen-

tations are learnt in the following manner

$$h^{<t>} = f(x, h^{<t-1>}) \tag{4.3.1}$$

and

$$c = g(h^{<1>}, h^{<2>}, ..., h^{<\tau_1>}) \tag{4.3.2}$$

where $h^{<t>} \in \mathbb{R}^n$ denotes time step t hidden state and c denotes the vector that summarises the source sentence semantic representation. Furthermore, f and g denote some nonlinear functions.

2. To generate the target sentence y, the decoder RNN (or one-to-many component) is conditioned on context vector c. At each time step the decoding process is said to be auto-regressive, meaning when generating the next target word the decoder uses as additional input the previously generated/predicted outputs [34]. In other words the decoder learns to predict the word $y^{<t>}$ in view of context vector c along with prior words $\{y^{<1>}, y^{<2>}, ..., y^{<t-1>}\}$. In essence, the decoder learns a conditional distribution over the target sentence y as follows:

$$p(y) = \prod_{t=1}^{\tau_2} p(y^{<t>} | \{y^{<1>}, y^{<2>}, ..., y^{<t-1>}\}, c). \tag{4.3.3}$$

Therefore, since the decoder uses a RNN architecture we express its conditional probability as

$$p(y^{<t>} | \{y^{<1>}, y^{<2>}, ..., y^{<t-1>}\}, c) = q(y^{<t-1>}, s^{<t>}, c) \tag{4.3.4}$$

whereby q represents a multilayered transformation that calculates the probability of $y^{<t>}$. The context vector is denoted by c. For each time step t, the RNN decoder hidden state is denoted $s^{<t>}$.

The objective of this model is to jointly train the encoder and decoder such that it maximizes the conditional log-likelihood as follows

$$\max_{\theta} \frac{1}{|x|} \sum_{(x,y)\in\mathcal{D}} \log p_\theta(y|x) \tag{4.3.5}$$

where $\mathcal{D}$ and θ are sets of training pairs and parameters respectively. The target translation $\hat{y}$ is predicted by the following procedure

$$\hat{y} = \arg\max_{y} p_\theta(y|x). \tag{4.3.6}$$

An encoder with bidirectional RNN computations (see Section 3.5) will comprise of a forward pass $\overrightarrow{h}$ and backward pass $\overleftarrow{h}$ which will each sequentially out put hidden states $(\overrightarrow{h}^{<1>}, \overrightarrow{h}^{<1>}, ..., \overrightarrow{h}^{<\tau_1>})$ and $(\overleftarrow{h}^{<1>}, \overleftarrow{h}^{<2>}, ..., \overleftarrow{h}^{<\tau_1>})$ respectively. It therefore follows that the annotation for each word $x^{<t>}$ is obtained by concatenating $\overrightarrow{h}$ and $\overleftarrow{h}$ in the following circumstances

$$h^{<t>} = \left[\overrightarrow{h}^{<t>\top}; \overleftarrow{h}^{<t>\top}\right]^{\top}. \tag{4.3.7}$$

Therefore, the annotation $h^{<t>}$ will contain the compressed representation of the subsequent and prior words. As a consequence of RNN being better at learning recent input representations, it then follows that $h^{<t>}$ is centered on the words that are around $x^{<t>}$. Which implies such an architecture will tend to struggle when learning *long-term dependencies*, a phenomenon we discussed in Section 3.4. Nonetheless, this architecture archived SOTA performance on MT tasks [22]. However, the performance of this encoder-decoder RNN architecture decreases with an increase in input sequence length. That is to say, as the input sequence length increases the encoder-decoder RNN struggles to compress long input representations into a single context vector [22]. The solution to this phenomenon (*long-term dependency problem*) was firstly proposed by [24] and is the subject of discussion in the upcoming sections.

4.4 Sequence to Sequence with Attention

As discussed earlier in Section 3.6 of Chapter 3 LSTM RNNs were from 2014-2017, the most popular RNN variant for machine translation tasks. Although LSTMs are usually the architecture of choice, for simplicity sake we shall in this section generalize our notation to suit any type of RNN. Despite the theoretical fact that LSTMs can alleviate the long-term dependency problem, in practice, they tend to perform badly when sentences become too long [51], for example when translating a paragraph or an article. The rationale being that the probability of keeping the learned representation of a word that is too far from the current word being learned decays exponentially [18] with the distance from it. Which implies that as the sentence length becomes too large, oftentimes the model will forget the distant representations. To alleviate this problem, researchers have developed a technique that translates sentences by paying *attention* to different tokens within the source sentence.

The attention mechanism can be defined as the mapping of some input vectors, known as the *query* and *value-key pairs* to some output vector of weighted sum of *values*. The

weights relating to these values are generated by means of some correspondence function which takes in as arguments the query and value. There are various ways of incorporating the attention mechanism into RNN machine translation models. The most prominent being *additive attention*, *global attention*, *local attention* and *self-attention*. Foundational to the other two is the additive attention also know as *translation with alignment* [14].

4.4.1 Additive Attention

The main objective of this technique and the modifications that stem from it is to help memorise long-term dependencies in sequential data. Figure 4.4 gives a pictorial representation of this technique. We can envision this technique as having three major steps

1. The processing of input data (source sentence) to produce vectors of distributed representations. In the case of a bidirectional RNN encoder, the time step processed representations are computed in the following manner

$$h^{<t>} = \left[\overrightarrow{h}^{<t>\top}; \overleftarrow{h}^{<t>\top} \right]^{\top}, \quad t = 1, .., \tau_1 \tag{4.4.1}$$

2. Storing the outputs $h^{<t>}$ into memory vector c (list of vectors which can be retrieved in any order). Such that the context vector $c^{<i>}$ denotes the sum of weighted hidden states and is computed in the following manner

$$c^{<i>} = \sum_{t=1}^{\tau_1} \alpha^{<it>} h^{<t>}. \tag{4.4.2}$$

The weights associated with each *value* $h^{<t>}$ (or hidden state) are computed in the following manner

$$\alpha^{<it>} = \frac{\exp\left(e^{<it>}\right)}{\sum_{k=1}^{\tau_1} \exp\left(e^{<ik>}\right)} \tag{4.4.3}$$

where alignment score $e^{<it>}$ is a function of $s^{<i-1>}$ and $h^{<t>}$, the decoder's prior hidden state and encoder hidden state respectively.

$$e^{<it>} = a(s^{<i-1>}, h^{<t>}) \tag{4.4.4}$$

whereby $s^{<i-1>}$ and $h^{<t>}$ are called the *query* and *key*. The alignment function a is a general feed-forward network jointly trained with the entire model. This alignment model calculates the scores that determine how well the inputs within the neighbourhood of element t match with the i^{th} output.

3. The decoder sequentially predicts the target outcome by exploiting the memory vector c. Therefore, we rewrite the conditional probability in Equation 4.3.4 as follows

$$p(y^{<i>}|\{y^{<1>},...,y^{<i-1>}\}) = q(y^{<i-1>}, s^{<i>}, c^{<i>}) \tag{4.4.5}$$

where hidden state $s^{<i>}$ is computed in the following manner:

$$s^{<i>} = q(s^{<i-1>}, y^{<i-1>}, c^{<i>}). \tag{4.4.6}$$

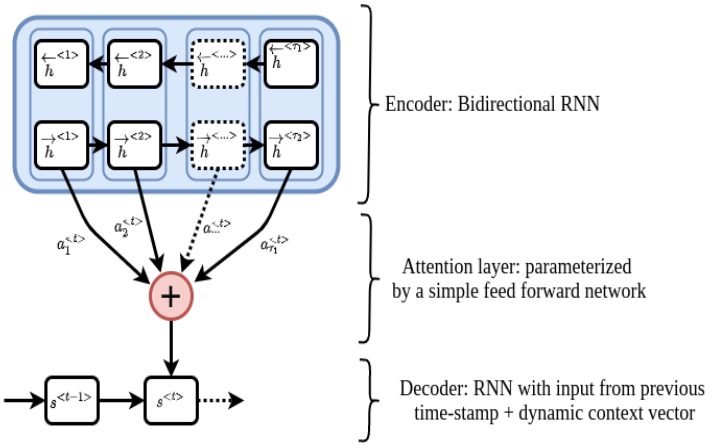


Figure 4.4: Illustration of an encoder-decoder architecture with a weighted average attention mechanism as introduced by [14]. The model is at a state where it is predicting the t^{th} target output given the input $(x^{<1>},...,x^{<\tau_1>})$.

As the model aligns the source words with its corresponding target words, it becomes possible to associate the source word with the target words that are closely related to it.

4.4.2 Global Attention

The global attention mechanism [60] is an improvement of the additive attention technique [14]. The main objective of this mechanism is to incorporate the learned representations from the encoder hidden states into the target hidden states by generating a new and improved target hidden state as shown in Figure 4.5. For the model to accomplish this at some time step t, the attention layer has to find a representation of the source sentence that is close to $s^{<t>}$, the output hidden state which is accomplished

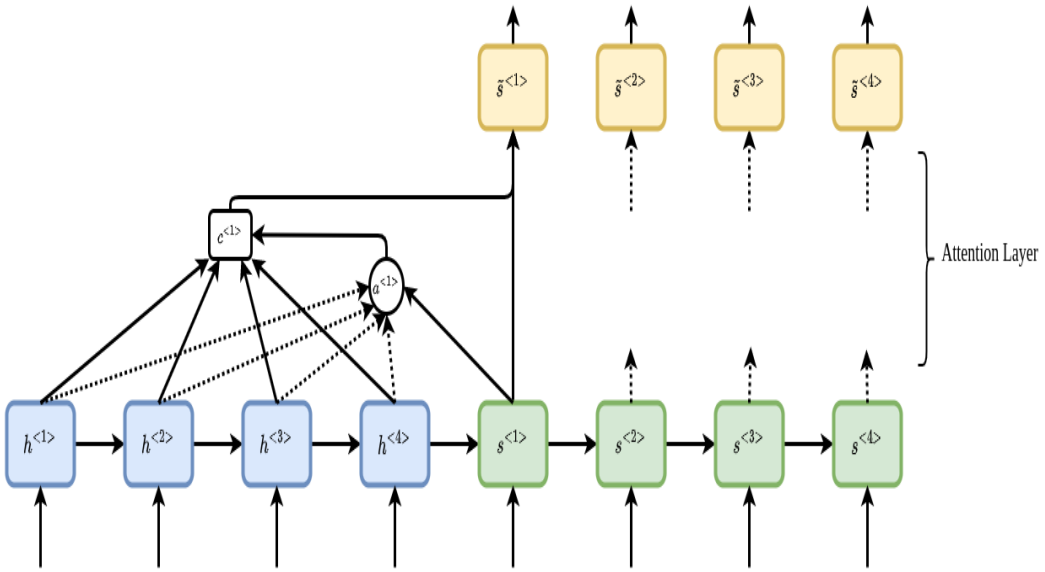


Figure 4.5: A pictorial representation of the global attention mechanism as introduced by [60]. For each time step t, the architecture employs state $s^{<t>}$ to infer alignment vector $a^{<t>}$. Thereafter, the sought-after content vector $c^{<t>}$ is then computed as a weighted average with respect to all the input hidden states and the alignment vector $a^{<t>}$.

by first computing context vector $c^{<t>}$. To compute context vector $c^{<t>}$, we apply a similarity-weighted average of the input hidden states as follows:

$$c^{<t>} = \frac{\sum_{j=1}^{\tau_1} \exp\left(h^{<j>} \cdot s^{<t>}\right) h^{<j>}}{\sum_{j=1}^{\tau_1} \exp\left(h^{<j>} \cdot s^{<t>}\right)} \tag{4.4.7}$$

Another way of representing this weighting is by way of an alignment variable $a(t,s)$ which determines the relevance of input word s to target output t:

$$a(t,s) = \frac{\exp\left(h^{<s>} \cdot s^{<t>}\right)}{\sum_{j=1}^{\tau_1} \left(h^{<j>} \cdot s^{<t>}\right)}. \tag{4.4.8}$$

As a result, our attention vector which relates to the t^{th} target word is denoted $a^{<t>} = [a(t,1), ..., a(t,\tau_1)]$. In other words this vector represents a set of probabilistic weights that sum to 1 and has a cardinality that is equal to the input sequence length τ_1. Now by substituting Equation 4.4.8 into Equation 4.4.7 we obtain

$$c^{<t>} = \sum_{j=1}^{\tau_1} a(t,j) h^{<j>}. \tag{4.4.9}$$

In principle, this technique seeks to determine a semantic representation of the input hidden states that are highly significant to the output hidden state of consideration. This significance is determined by the similarity dot product between the input and target hidden states as represented by the attention vector $a(t, s)$. The final step is to generate a target hidden state $\widetilde{s}^{<t>}$ which combines context vector $c^{<t>}$ representations and $s^{<t>}$, the original output hidden state:

$$\widetilde{s}^{<t>} = \tanh\left(W_c\left[\begin{smallmatrix} c^{<t>} \\ s^{<t>} \end{smallmatrix}\right]\right). \tag{4.4.10}$$

To predict the output $y^{<t>}$ the newly generated hidden state $\widetilde{s}^{<t>}$ is used in-place of $s^{<t>}$, the original hidden state.

4.4.3 Local Attention

The major setback with global attention is that it is computationally expensive, which is due to the fact that when predicting each target word the model has to attend to all the input words. Which makes it difficult to translate long sequences such as paragraphs or short articles. In response to this setback [60] further developed the local attention mechanism which derives inspiration from the application of attention in image captioning [96]. In the work of [96] they refer to global attention as *soft attention*, this is because the weights are softly placed on all the locales in the input image. On the other hand [96] proposed an attention mechanism which only attends to a single locale of an image at a time and this technique they called *hard attention*. Although hard attention is seemingly computationally inexpensive when compared to soft attention it being non-differentiable implies it requires sophisticated techniques of training such as reinforcement learning and variance reduction [60]. On the other hand, local attention discriminately focuses on a definite portion of context and this makes it computationally inexpensive.

In essence, this technique sets of by computing the alignment position $p^{<t>}$ with regards to each target word at time step t as shown in Figure 4.6. Thereafter, the model computes a weighted average over all the input hidden states within a specified window $[p^{<t>} - D, p^{<t>} + D]$, to generate context vector $c^{<t>}$. The window width D is chosen based on a trial and error criterion. In contrast with the variable-length alignment vector in the global attention mechanism, the local attention alignment vector has a fixed dimension of $a^{<t>} \in \mathbb{R}^{2D+1}$. This model has two variations of alignment:

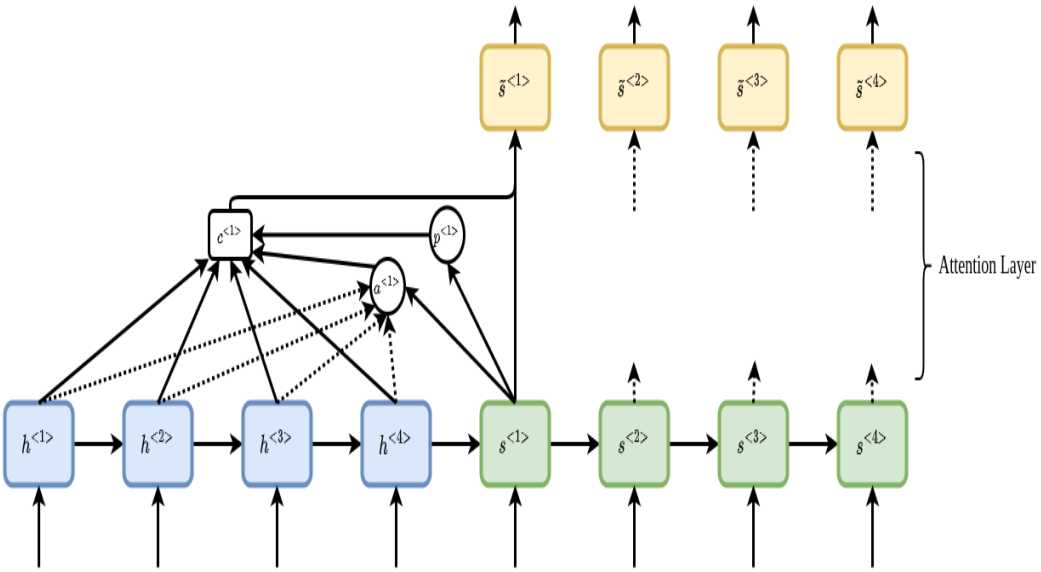


Figure 4.6: A pictorial representation of the local attention mechanism. At time step t the attention computations start of by predicting the alignment positions $p^{<t>}$. Thereafter, the model then computes context vector $c^{<t>}$ using a window that is centered around point $p^{<t>}$. During time step t inference, the model employs the input hidden state $h^{<t>}$ and output hidden state $s^{<t>}$ to compute the weights $a^{<t>}$.

1. Monotonic alignment: the model assumes that the input sentence is approximately aligned with the target sentence such that the alignment vector is defined as:

$$a(t,s) = \frac{\exp\left(h^{<s>} \cdot s^{<t>}\right)}{\sum_{j=1}^{\tau_1}(h^{<j>} \cdot s^{<t>})}. \tag{4.4.11}$$

2. Predictive alignment: the model assumes no predefined alignments. Instead it learns to predict these alignments as follows:

$$p^{<t>} = |x| \cdot sigmoid(v_p^\top \tanh\left(W_p h^{<t>}\right)) \tag{4.4.12}$$

where v_p and W_p denote the parameters which the model learns. On the other hand, $|x|$ denotes the input sequence length τ_1. It then follows that the alignment position $p^{<t>} \in [0, \tau_1]$, a consequence of the sigmoid function. In preference of the points within the neighbourhood of $p^{<t>}$, the model incorporates a Gaussian distribution that is centered at $p^{<t>}$. As a result, the alignment weight $a(t,s)$ is

modified to

$$\begin{aligned} a(t,s) &= a(h^{<s>}, s^{<t>}) \\ &= \exp\left(\frac{-(s-p^{<t>})^2}{2\sigma^2}\right), \end{aligned} \tag{4.4.13}$$

where the standard deviation is experimentally set to $\sigma = \frac{D}{2}$ and $s \in [p^{<t>} - D, p^{<t>} + D]$.

4.4.4 The input-feeding technique

The limitation with global and local attention is that the attention decisions are independently made, which renders these models sub-optimal [60]. This is different from the standard machine translation technique that simultaneously learns to align and translate [14]. To solve this difference [60] introduced an input feeding architecture as depicted in Figure 4.7. This architecture is fully cognizant of the previous alignment ordering. Which implies the model is capable of deciding which attention constraints are necessary. For each t^{th} decoding time step, the model concatenates the newly generated hidden state $\widetilde{s}^{<t>}$ with the $(t+1)^{th}$ input as shown in Figure 4.7.

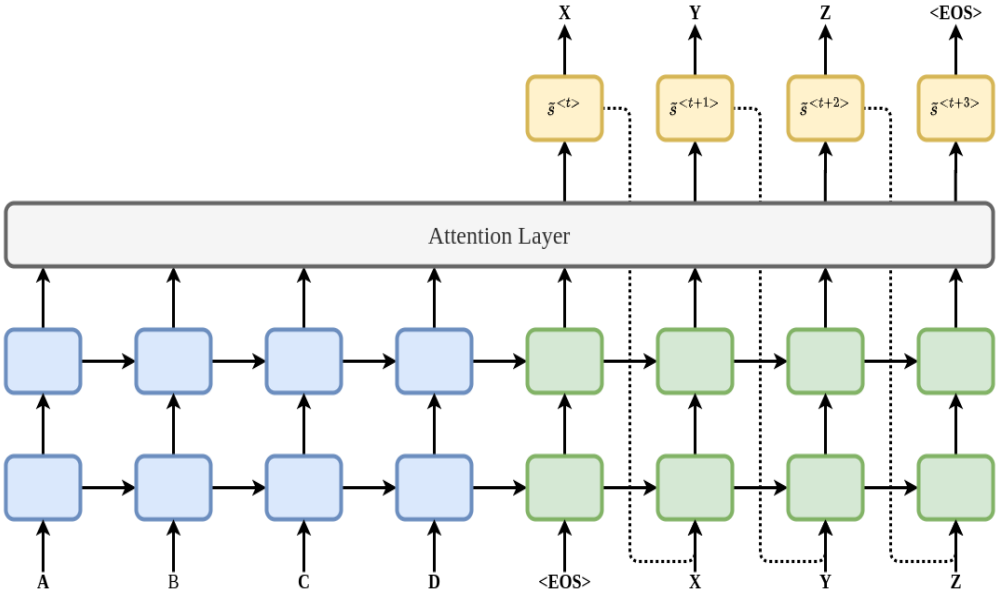


Figure 4.7: Illustration of an input-feeding architecture. At each time step t, the newly generated attention vector $h^{<t>}$ is passed as input at time step $t+1$. For t^{th} decoding time step, the model concatenates the newly generated hidden state $\widetilde{s}^{<t>}$ with the $(t+1)^{th}$ input.

4.4.5 Self-Attention

Suppose that $x = (x^{<1>}, ..., x^{<\tau_1>})$ is an input sequence of length τ_1, where $x^{<i>} \in \mathbb{R}^{d_x}$. The *self attention* [20] mechanism maps x to some output sequence $z = (z^{<1>}, ..., z^{<\tau_1>})$ where $z^{<i>} \in \mathbb{R}^{d_z}$. Each $z^{<i>}$ output is generated as the sum of weighted linear transforms of the input sequence:

$$z^{<i>} = \sum_{t=1}^{\tau_1} \alpha^{<it>}(x^{<t>}W_v) \tag{4.4.14}$$

where $\alpha^{<it>}$ computation is similar to Equation 4.4.3. The component $e^{<it>}$ is computed by means of a compatibility function known as the *scaled dot product* [87]

$$e^{<it>} = \frac{(x^{<i>}W_Q)(x^{<t>}W_K)^\top}{\sqrt{d_z}}, \tag{4.4.15}$$

where W_Q, W_K, $W_V \in \mathbb{R}^{d_x \times d_z}$ denote the model's attention parameters. This attention mechanism has some resemblance of *dot-product* attention [60], with the only difference being the scaling factor $1/\sqrt{d_z}$. It is worth noting that for long dimensions of z the compatibility function yields a dot product that is large in magnitude which catapults the softmax function into zones of vastly small gradients. It is for this reason that the scaling factor is incorporated into the compatibility function.

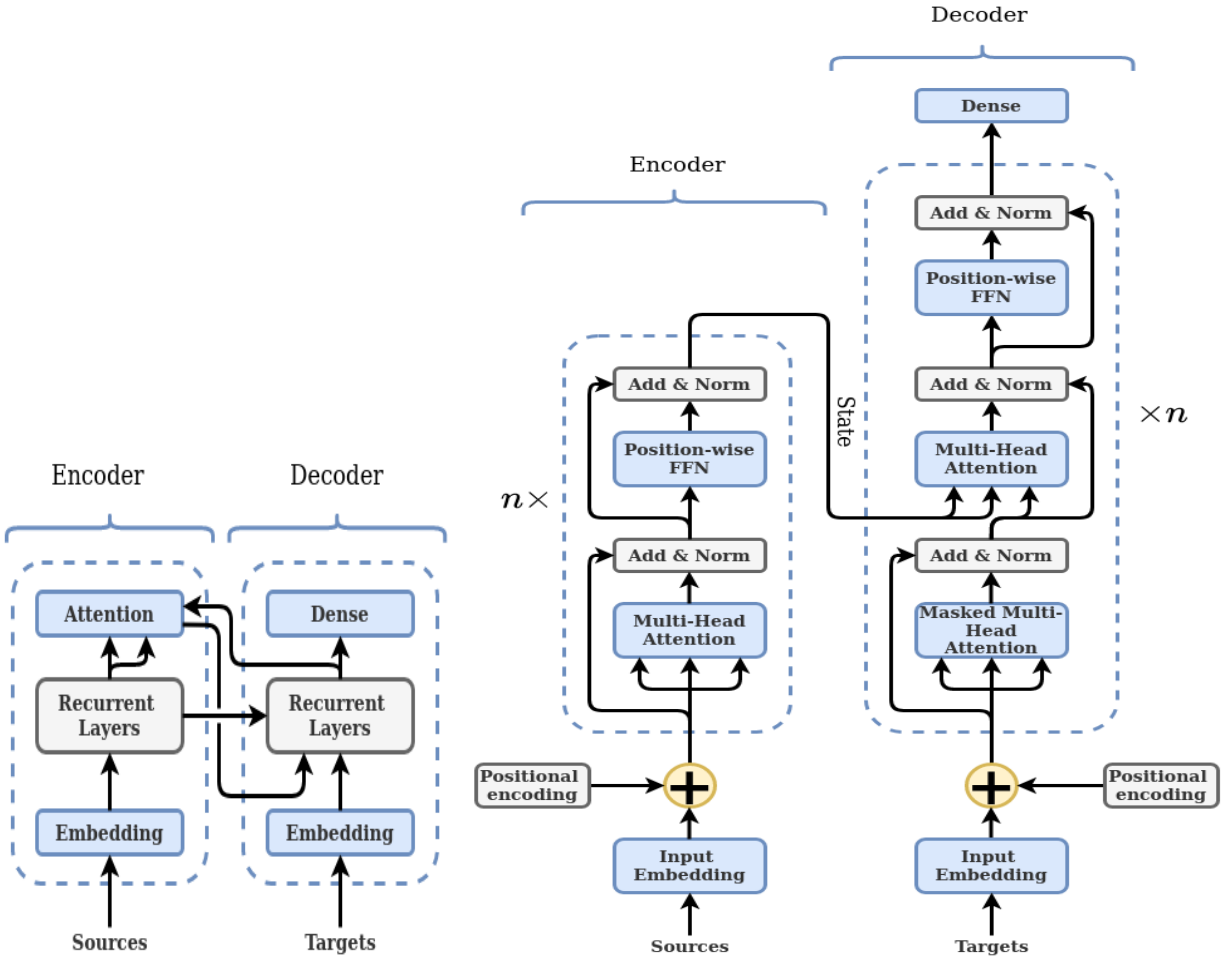
4.5 Transformer Model

The novel *transformer* architecture [87] is without a doubt one of the most influential developments in the domain of machine translation. This model's key component is a development of the global attention mechanism discussed in Section 4.4.2 which makes the modelling of sequence-to-sequence mappings without RNN feasible. In addition to the long-term dependency problem discussed in Section 4.4, RNN are difficult to parallelize due to their sequential computations. The transformer architecture aims at concurrently solving the variable length and parallelization problems. This is done by simultaneously encoding sequence token positions and capturing sequence-to-sequence recurrences by utilizing the self-attention mechanism [49]. As a result, the transformer is trained in a significantly shorter time frame.

Reminiscent of the modern sequence-to-sequence architectures the transformer also uses the encoder-decoder mechanism as shown in Figure 4.8b. For a comprehensive explanation of the transformer model, we give a side-by-side comparison between the trans-

former and RNN sequence-to-sequence model, as shown in Figure 4.8. The two architectures are both similar in the following manner:

- Both architectures take in as input the embedded source sentence into a block of encoder layers. Furthermore, both decoders take in as input, the encoder's output.
- In both models, the target sequence embeddings are fed into the decoder layers as part of the inputs. A densely connected layer (with a size equal to the vocabulary size) is used to generate the outputs of each decoder.



(a) RNN sequence-to-sequence with attention. (b) Transformer sequence-to-sequence model.

Figure 4.8: A side by side comparison of the RNN sequence-to-sequence architecture and transformer sequence-to-sequence architecture. Both architectures have an encoder-decoder mechanism. However, the transformer's encoder and decoder comprises of n stacked heads. The encoder feeds its output to the decoder for further processing.

On the other hand, the two networks differ in the following manner:

- **Transformer blocks:** the RNN layers are replaced with n transformer blocks. The encoder blocks contain multi-head self-attention layers along with position-wise FF layers. This is also the case for the decoder however the decoder has multi-head self-attention layers that take in as part of their input the encoder's output.
- **Add & Norm:** the outputs of both the fully connected and multi-head self-attention layers are fed into a layer that contains a residual connection and normalization component [13].
- **Positional Encoding:** with the self attention layers not being able to capturing sequential representations the transformer employs positional encoding layers instead.

4.5.1 Multi-Head Self-Attention

The self-attention mechanism constitutes the fundamental unit of multi-head self-attention. As a result, it is imperative that we discuss self-attention first. As shown in Figure 4.9 the self-attention model computes output $z^{<i>} \in \mathbb{R}^{d_z}$ based on the linearly

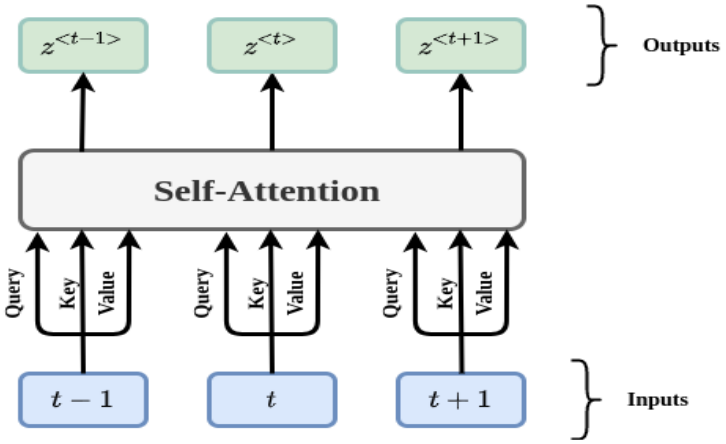


Figure 4.9: The self attention mechanism returns an output based on the query, key and value inputs. The output vectors are of the same length and unlike RNN architectures with attention, self attention can be computed in parallel and this results in efficient implementation.

projected query and value-key pairs. In contrast with the RNN layers, self-attention allows for the parallel computations of $z^{<i>}$ outputs, which results in more efficient computations. The multi-head self-attention layer comprises a fixed number of parallel self-attention layers h. Each of these parallel layers is called an attention head. Given a query, key and value with dimensions d_q, d_k and d_v respectively. Prior to feeding the query, value and keys into the self-attention layers, the model projects them with dense layers of hidden size dimensions p_q, p_vandp_k respectively. The attention heads then perform the attention transform on the projected query, key-value pairs in parallel, resulting in a p_v-dimensional output vector per head. The output vectors are first concatenated then fed into a dense layer as shown in Figure 4.10. The transformer decoder has a multi-head attention layer that takes in the encoder state as a part of its input. To be precise, for some query input $x^{<t>}$ at inference, the computation of output vector $z^{<t>}$ includes the past queries $\{x^{<1>}, x^{<2>}, ..., x^{<t-1>}\}$ as shown in Figure 4.11.

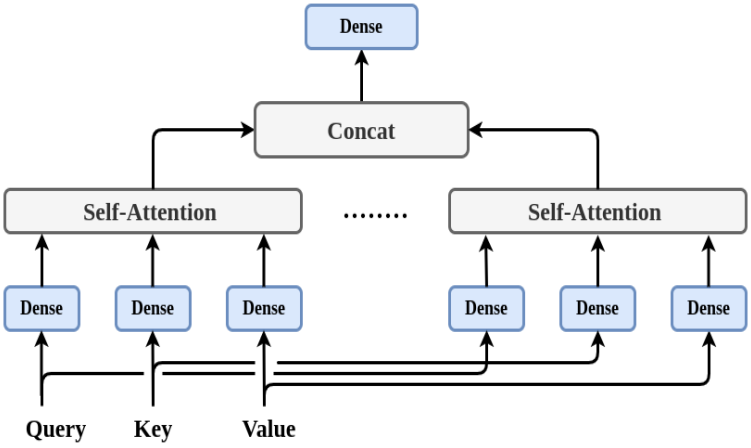


Figure 4.10: A pictorial representation of the multi-head self attention mechanism. The query, key and value are fed into dense layers that feed to the self attention heads respectively. The attention values are simultaneously computed by each head and concatenated before being fed into a dense layer.

The multiple self-attention heads enable the model to collectively learn from dissimilar representation sub-spaces at various positions, such that for every attention head $j = 1, ..., h$ the model learns parameters $W_q^{(j)} \in \mathbb{R}^{p_q \times d_q}$, $W_k^{(j)} \in \mathbb{R}^{p_k \times d_k}$ and $W_v^{(j)} \in \mathbb{R}^{p_v \times d_v}$. As

a result, each attention head generates the following outputs:

$$o^{(j)} = \phi(W_q^{(j)}q, W_k^{(j)}k, W_v^{(j)}v) \tag{4.5.1}$$

where ϕ denotes the dot-product attention. The p_v-dimensional outputs $o^{(j)}$ are concatenated into a hp_v dimensional output which in-turn is passed on to a dense layer comprising of d_o hidden units. This dense layer learn parameters $W_o \in \mathbb{R}^{d_o \times hp_v}$.

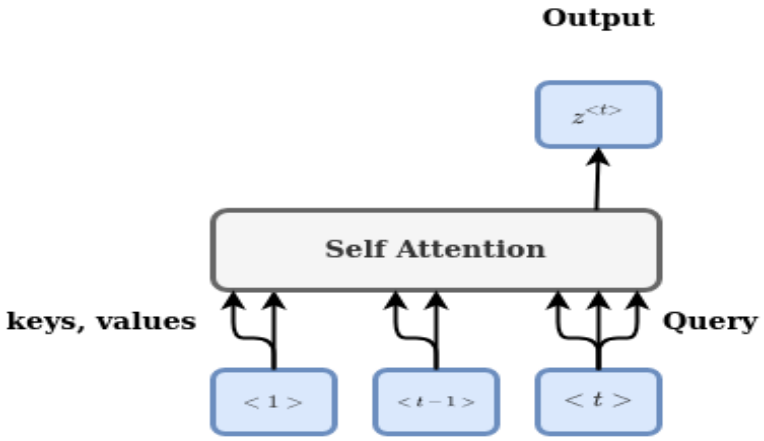


Figure 4.11: Pictorial representation of a decoder self-attention head at inference. For some query input $x^{<t>}$ at inference, the computation of output vector $z^{<t>}$ includes the past queries $\{x^{<1>}, x^{<2>}, ..., x^{<t-1>}\}$.

In general, the multiple attention heads generate a single attention output

$$o = W_o \begin{bmatrix} o^{(1)} \\ o^{(2)} \\ \vdots \\ o^{(h)} \end{bmatrix}. \tag{4.5.2}$$

In this work, we use $h = 16$ parallel attention heads and $d_o = 256$. It then follows that $p_q = p_k = p_v = d_o/h = 16$.

4.5.2 Position-wise Feed Forward Network

Along with the encoder-decoder attention sub-layers the transformer model contains position-wise feed forward (FFN) sub-layers. These FFNs accept as input a 3-dimensional tensor of shape $(batch - size, sequence - length, feature - size)$ and includes two linear

transformations which are separately and identically applied to each position. A RELU activation function lies between the transformations as follows

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2. \tag{4.5.3}$$

Though both linear transforms are identical throughout all the locations, they each employ weight parameters that are different from layer to layer.

4.5.3 Positional Encoding

Given that the transformer model has no recurrence components, it is important that the model have some mechanism of capturing positional sequence representations. To expound on the positional encoding component we suppose $e \in \mathbb{R}^{\tau_1 \times d}$ is an embedding of some input sequence x, such that τ_1 and d denotes the sequence length and embedding dimension respectively. The positional encoding layer encodes $P \in \mathbb{R}^{\tau_1 \times d}$ a position in e the input embedding and outputs $P + e$. Notwithstanding the numerous choices of positional encoding methods [31], as is the case in [87] we employ the sine and cosine functions for positional encoding as shown in Figure 4.12.

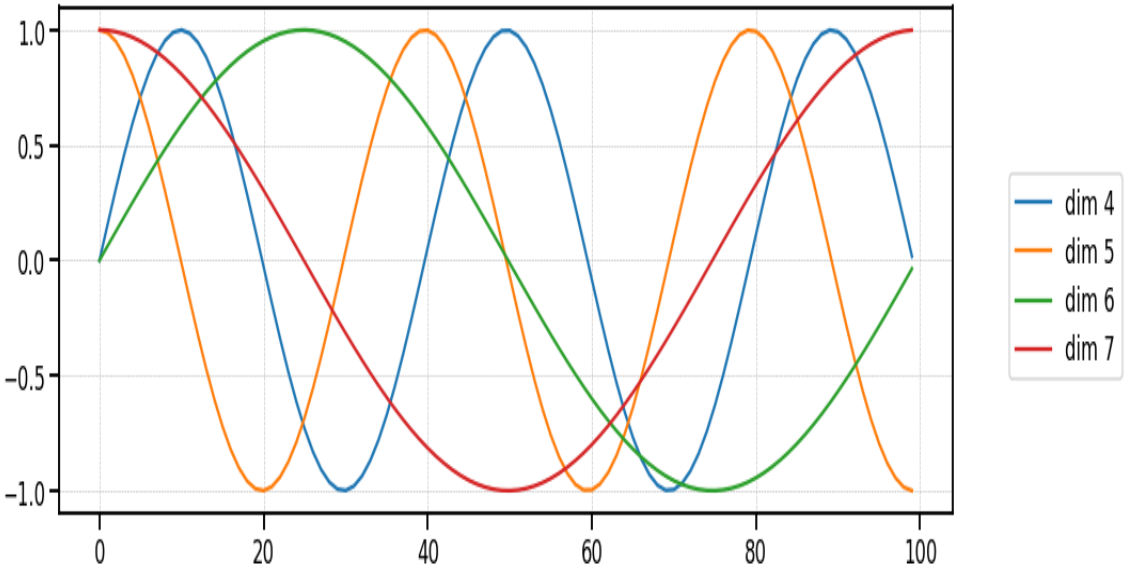


Figure 4.12: An illustration of a four dimensional positional encoding toy model. As shown above, the 4^{th} and 5^{th} dimension curves only differ in offset but they generally have the same frequency. Similarly, the 6^{th} and 7^{th} dimension curves only differ in offset but have the same frequency.

These sine and cosine functions are of varying lengths such that the value P is obtained by using the following equations:

$$\begin{aligned} P_{i,2j} &= \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \\ P_{i,2j+1} &= \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right), \end{aligned} \tag{4.5.4}$$

where i and j denote the sentence position and embedding dimension respectively. Such that each positional encoding dimension matches up with a sinusoid with wavelengths that resemble a geometric progression that ranges from 2π to $10000 \cdot 2\pi$.

4.5.4 Model regularization

The transformer architecture employs drop out regularization to the sub-layer outputs. A constant dropout rate $p_{drop} = 0.1$ is used through out all the layers with dropout. Firstly the model applies dropout to the encoder and decoder encoded embeddings. Thereafter, dropout is applied to all sub-layers before it is passed on as input to the *add-norm* layers.

4.6 Summary

We have given a comprehensive overview of the RNN encoder-decoder architecture models for machine translation and their drawbacks. As the main objective of this research is to employ the transformer model, we have highlighted how this architecture (transformer) solves the RNN drawbacks. Furthermore, we also gave a contrast between the RNN and transformer architectures. Hence the demystification of the transformer model.

Chapter 5

Low Resource Training Protocols and Evaluation Metrics

5.1 Chapter Organisation

The first section of this chapter is a discussion about the data. The hardware and data splitting technique are then discussed in Section 5.3. The next section then introduces transfer learning and how it can help solve the problem of low resources. Section 5.4.4 then discusses multilingual models and how they help improve translations on low-resource datasets. An extreme variant of transfer learning called zero-shot learning is examined in Section 5.4.5. Sections 5.5 then expounds on the evaluation metrics employed in this research. Finally, we give an overview of the content covered in this whole chapter.

5.2 Introduction

The conventional requirement for developing machine translation models is the availability of a large parallel dataset or corpus. It is by virtue of this requirement that much machine translation success is for institutional languages with millions/thousands of parallel text corpora. Albeit, of over 7000 human languages in existence, we have approximately 20 institutional languages. In most cases financial constraints make it difficult to source linguistically trained professionals (or speakers) to prepare machine translation datasets. Which leaves the great majority of human languages in great need of solutions or tools to solve this resource barrier. As a result, this research focuses on the application of *multilingual learning*, *transfer learning* and *zero-shot learning* to alleviate this

low-resource problem. Stated simply, these machine learning techniques for improving a modelling task by leveraging knowledge from a related task.

	En-Sh	**En-Zu**	**En-Xh**	**Xh-Zu**
Sentence count	77 500	30 253	128 342	125 098
Source token count	7 588	4 649	10 657	27 144
Target token count	16 408	9 424	33 947	25 465

Table 5.1: Summary statistics for each language pair. The sentence count denotes the number of examples each language. The source sentence token count denotes the number of words in the respective source languages. Similarly, the target token count denotes the number of words in the entire target language, respectively.

Our data sets comprise of parallel sentences obtained from [2] [3] [4] and [6]. The source and target languages for these parallel sentences are English-to-Shona (En-Sh), English-to-isiXhosa (En-Xh), English-to-isiZulu (En-Zu) and isiXhosa-to-isiZulu (Xh-Zu). These training pairs from different domains, with [2] supplying the En-Zu phrases for learning isiZulu. Similarly, [3] comprises of En-Xh and En-Zu phrases for learning both isiXhosa and isiZulu. [6] also contains more En-Xh phrases for learning isiXhosa. These three domains are not distant as they use similar writing styles. From the Orpus Corpus we obtain parallel sentences (En-Xh,En-Zu,En-Sh) from web crawls, some Wikipedia translations (En-Xh,En-Zu,En-Sh) and the En-Xh South African Navy corpus. As is the case with most machine learning projects, the data had to be cleaned and this was done through manually verifying the randomly selected alignments and dropping duplicates to avoid data leakage [78] . Manual alignment and other corrections were done with help of Shona, isiXhosa and isiZulu speakers.

Table 5.1 gives summary statistics for the four language pairs employed in this research. The number of tokens represents the the number of unique words in each dataset. The En-Xh pair had approximately 128 000 examples, making it the largest set. The second largest pair was the Xh-Zu pair with about 125 000 examples. The En-Sh and En-Zu pair were the third and fourth largest pairs with about 77 500 and 30 000 examples respectively. As stated in Section 1.3.1, one of our key objectives is to improve NMT of a low resource task by employ in cross-lingual transfer learning. In this work, we employ En-Xh and En-Sh language pairs as our high resource language pairs (see Table 5.1) for performing transfer learning.

5.3 Data splitting and Hardware

Our models were trained on High Performance Computer clusters comprising of fifth generation Intel CPUs [1]. We configured our node to 23 cores and 60gb of RAM. Furthermore, the models were also configured to train on machines with GPUs. Our models were developed in pytorch 1.7, a python machine learning package. Data-set splitting is an important concept of machine learning which helps prevent overfitting. We split our data sets into train, validation and test sets at a split ratio of $14:3:3$ respectively for each language pair. Figure 5.1 shows a visualization of this data splitting while Table 5.2 below gives a summary of the number of pairs per dataset subset.

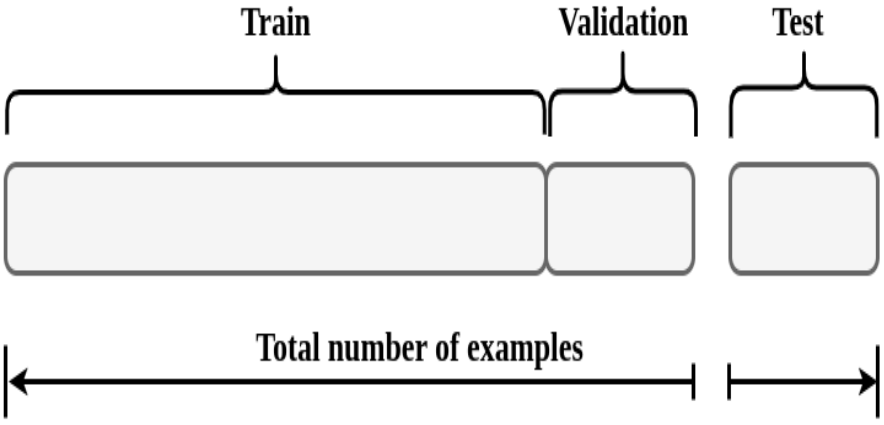


Figure 5.1: A visualization of the dataset splits. The dataset is first partitioned into two, on a ratio of 3:7. Thereafter, the smaller partition is equally partitioned into validation and test set.

	En-Sh	**En-Zu**	**En-Xh**	**Xh-Zu**
Train	54 250	21 177	88 192	87 570
Validation	11 625	4 538	20 075	18 765
Test	11 625	4 538	20 075	18 763

Table 5.2: Number of examples per partition. The dataset splitting was based on a ratio of 3:3:14 for the validation, test and training set respectively. Such that the total number of examples per language pair can be obtained by summing the number of train, validation and test set examples, respectively.

The training set contains 70% of the primary data set and it is on this subset that the model learns its generalization. The validation set contains 15% of the primary dataset (the entire set) and is used for computing an unbiased estimate of the model's generalization when finding and optimizing the best model. In other words, the trained model is occasionally allowed to see the validation subset without learning from it. We then use the validation results to optimise the model's hyper-parameters. As a result, the validation set has an indirect influence on the final model. For an unbiased estimate of our model performance, we employ an unseen test set for learning the model's overall generalization performance. Our test dataset comprises of the remaining 15% in the primary dataset. We refer to the testing of the model on the test an "unbiased evaluation" of the final model.

5.4 Low Resource Training Protocols

The process of training a machine learning task that learns to map an input (source sentence) to some output (target sentence) based on input-output pairs examples, is called supervised machine learning. In training these supervised learning tasks, especially deep learning models, the models tend to perform better with more data. For this reason, MT of low resource languages remains a challenge. Fortunately, there are a number of training protocols that have been developed to help alleviate the problem of low resources translation tasks. From these training protocols, we employ transfer learning, multilingual learning and zero-shot learning. To evaluate the performance of these training protocols we compare them against each other and the baseline models.

5.4.1 Baseline model training

We train our baseline models based on the NMT encoder-decoder protocol [84] discussed earlier in Section 4.3. However, in place of the RNN encoder-decoder architecture, we employ transformer-based encoder-decoder protocols. Two of these baseline models are then used as the parent or source model when performing transfer learning, a training protocol that leverages learned representations from a related task. A detailed discussion of transfer learning is given in the next section.

5.4.2 Transfer Learning

The human mind has an inherent ability to transfer knowledge across tasks. In other words, given two related tasks A and B, the knowledge we acquire while solving task

A can be leveraged to solve task B. It then follows that the more related the tasks at hand, the easier the cross-utilization of knowledge. As examples of knowledge cross-utilization, we consider the following

1. A mathematician/statistician $\rightarrow$ Learns machine learning.
2. One who knows how to play a guitar $\rightarrow$ Learns how to play the violin.
3. A baseball player $\rightarrow$ Learns how to play cricket.

In all the above examples, the learner does not have to learn everything from the onset. Instead, they focus on adapting to the new concepts at hand. Put differently the learner leverages and transfer knowledge gained from the prior task to the later. It is from this phenomenon that *transfer learning* in the domain of machine learning gathers its inspiration.

Conventional deep learning algorithms are usually developed to independently solve isolated tasks [69]. Such models have to be trained from scratch whenever we encounter a change in feature space distribution. This change is predominantly known as domain shift [48]. Deep learning models require a substantial amount of training data for better performance. On the contrary, the gathering of large sums of data (especially in the case of supervised learning) is challenging, taking into account the amount of time and effort required to label the data. Hence the motivation to apply transfer learning whenever we are faced with the problem of low resources (i.e. few data). The inception of transfer learning with application to machine learning research is believed to have cropped up at the Neural Information Processing Systems (NIPS) workshop back in 1995 [85] and has since then been used in multiple domains.

5.4.3 Understanding Transfer Learning

Transfer learning is a domain adaptation technique where a model that has been trained on one task is exploited to better the generalizations on a related task [33]. This technique is generally efficient when faced with a low resource problem. In other words, to accomplish this cross-utilization of learned features/knowledge we first employ a base dataset for training the base model, thereafter we leverage the learned features to train a second model on another task and dataset. This technique works best if the prior/base model features can be generalized to the second task, which means the base features have to be adaptable to the later task instead of them being task specific [97]. In the

case of our NMT experiments, we employ En-Xh and En-Sh translation tasks as the base models for training translations for En-Zu pairs.

Before we give a formal definition of transfer learning we start of by introducing our nomenclature. Suppose a domain $\mathcal{D}$ comprises two elements: a source feature space $\mathcal{X}$ over which is a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$, such that $\mathcal{D} = \{\mathcal{X}, P(X)\}$. A simple example is the case of a binary classification task like movie review sentiment analysis, the feature space $\mathcal{X}$ will consist of vectors x_i where $i \in \{0, .., n\}$. Here, n denotes the number of training examples, and each X denotes a learning sample. As a general rule, the variations between two domains may result in having dissimilar feature spaces which in-turn leads to dissimilar marginal probability distributions [71]. Taking $\mathcal{D} = \{\mathcal{X}, P(X)\}$ into consideration, our task $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ comprises of two elements: $\mathcal{Y}$, a target feature space and the respective objective function $f(\cdot)$. The objective predictive function $f(\cdot)$ is learnt from the paired training examples (x_i, y_i) where each $x_i \in X$ and $y_i \in \mathcal{Y}$, $\forall i \in \{1, ..., n\}$. Function $f(\cdot)$ is employed to generate the i^{th} commensurate output $f(x_i)$. From a probabilistic end-point, the function $f(x_i)$ can be expressed as $P(y_i|x_i)$. In our sentiment analysis example the target space $\mathcal{Y}$ consists of all the binary labels such that $y_i =$ "positive" or "negative".

With regards to the two tasks A and B mentioned earlier in Section 5.4.3 we denote the source domain data (i.e. task A data) as $D_S = \{(x_{S_1}, y_{S_1}), ..., (x_{n_S}, y_{n_S})\}$ where $x_{S_i} \in \mathcal{X}_S$ and $y_{S_i} \in \mathcal{Y}_S$ denote the i^{th} input feature and target label respectively. In like manner we denote our target domain data (i.e. task B data) as $D_T = \{(x_{T_1}, y_{T_1}), ..., (x_{n_T}, y_{n_T})\}$. In the context of a low resource problem it follows that $0 \leq n_T \leq n_S$.

Definition 5.4.1. *Accorded with source and target domains $\mathcal{D}_S$ and $\mathcal{D}_T$ respectively, along with their respective tasks $\mathcal{T}_S$ and $\mathcal{T}_T$, cross-lingual transfer learning intends to boost or enhance the performance of $f_T(\cdot)$, the target predictive function belonging to target domain $\mathcal{D}_T$. Performance enhancement is done by leveraging the learned representations from the source domain and source task $\mathcal{D}_S$ and $\mathcal{T}_S$, respectively, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$.*

In view of Definition 5.4.1 the domain is expressed as $D = \{\mathcal{X}, P(X)\}$. Therefore, $\mathcal{D}_S \neq \mathcal{D}_T$ insinuates that $\mathcal{X}_S \neq \mathcal{X}_T$ otherwise $f_S(\cdot) \neq f_T(\cdot)$. In context of the sentiment analysis example, this implies that source domain $\mathcal{D}_S$ and target domain $\mathcal{D}_T$ differ only on their input features (e.g different languages) or on the marginal probability distributions. In like manner, task $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ points out that $\mathcal{T}_S \neq \mathcal{T}_T$ implies that $f_S(\cdot) \neq f_T(\cdot)$

otherwise $\mathcal{Y}_S = \mathcal{Y}_T$. It is worth noting that whenever the conditions $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$ are met, the learning technique reverts to a conventional machine learning problem.

The difference in domains implies one of the following:

1. $\mathcal{X}_S \neq \mathcal{X}_T$ i.e. the feature spaces between the two problems are different.
2. $\mathcal{X}_S = \mathcal{X}_T$ but $f_S(\cdot) \neq f_T(\cdot)$ i.e. although the two feature spaces are the same, the objective predictive function are different.

In line with the sentiment analysis example, the above cases can respectively imply the following:

1. the two input datasets are of different languages.
2. the source and target domains focus on different sentiment classes.

A scenario where we are provided with task specific domains $\mathcal{D}_S$ and $\mathcal{D}_T$ with their respective tasks $\mathcal{T}_S \neq \mathcal{T}_T$ suggests the following:

1. $\mathcal{Y}_S \neq \mathcal{Y}_T$ i.e. the two domains have different output spaces.
2. $f_S(\cdot) \neq f_T(\cdot)$ i.e. the source and target objective predictive functions are different.

In context of the sentiment analysis example, the above conditions respectively correspond to the following:

1. the source task $\mathcal{T}_S$ is a binary classification task while target task $\mathcal{T}_T$ is a multinomial classification problem.
2. the number of classes in the source task is not equal to the number of classes in the target task.

Furthermore, the existence of some implicit or explicit relationship between $\mathcal{D}_S$ and $\mathcal{D}_T$ affirms the presence of some relation between the two domains [71].

5.4.4 Multilingual Model

Inter-lingual translation models are a quintessential technique of translating between two languages [44, 75]. Notwithstanding its distinguished background, the research on this technique had since the mid-twentieth century experienced little growth as it was computationally expensive to build individual models to handle such tasks. The inception of NMT systems [14, 47, 84] proved to be an essential tool for developing both inter-lingual and multilingual machine translation systems. The earliest development of multilingual NMT [28] was a modification of the encoder-decoder attention mechanism [14] and had an overall architecture that was based on having individual attention-based decoders for the respective target languages. [59] proposed a multitask learning encoder-decoder architecture without attention. This model had multiple encoders and decoders for each source-target language pair.

There are a number of multilingual NMT architecture proposals that are based on these early developments, however in this research we employ Google's multilingual NMT protocol [45] which surpasses all prior developments with regard to performance and computational efficiency. Our multilingual NMT system is based on the Transformer architecture [87]. The multilingual models leverage learned representations from individual translation pairs and is generally identical to single pair translation architecture.

Definition 5.4.2. *Given a training tuple (x_i, y_j) where $i, j \in \{1, .., T\}$, the multilingual model's task is to translate source language i to target language j. It then follows that the model's objective is to maximize the log-likelihood over all the training sets $D_{i,j}$ appertaining to all the accessible language pairs $\mathcal{S}$:*

$$\max_{\theta} \frac{1}{|\mathcal{S}| \cdot |D_{i,j}|} \sum_{(x_i, y_j) \in D_{i,j}, (i,j) \in \mathcal{S}} \log p_\theta(y_j | x_i, j^*), \tag{5.4.1}$$

where j^* denotes the target language ID.

To be precise, the model is provided with an artificial token j which acts as a target language ID and is added at the beginning of the input sequence. For example, if we consider an English-to-Zulu pair:

- Finally, he achieved his goal $\longrightarrow$ Ekugcineni, wafinyelela umgomo wakhe.

which is modified to:

- <2zu> Finally, he achieved his goal $\longrightarrow$ Ekugcineni, wafinyelela umgomo wakhe.

Language ID <2zu> is the key modification to the single pair translation model. The multilingual model learns a universal or shared representation space for all the available languages pairs, which in-turn facilitates *zero-shot learning* [56], an extreme variant of transfer learning .

5.4.5 Zero-shot Learning

Explicit bridging is a conventional technique of translating between languages with low or no resources. Explicit bridging employs an intermediate language (usually English) to which we first translate our source sentence to, and thereafter we translate from intermediate to target language. The major drawbacks with this approach are:

- It is computationally expensive.
- There is a high risk of losing information when translating to and from the intermediate language.

A more recent approach to this problem of translating between low resource languages is *zero-shot learning*, an implicit bridging technique. Zero-shot learning is a machine learning technique whereby a model aims to perform a task whose examples are not accessible during training [95]. The feasibility of this approach relies on the additional information that is exploited during training [33]. To give a unified definition of zero-shot learning, we adopt the definitions of a "domain" and "task" as stated earlier in Section 5.4.3.

Definition 5.4.3. *Given a domain or set $\mathcal{D} = \{\mathcal{X}, P(X)\}$ along with its respective source task $\mathcal{T}_S = \{\mathcal{Y}, P(Y|X)\}$ where input and output feature spaces are denoted by $\mathcal{X}$ and $\mathcal{Y}$ accordingly. The objective of zero-shot learning is to estimate the predictive function $P(Y|X, \mathcal{T}_S)$ where $\mathcal{T}_S$ denotes the target task to be learnt by the model.*

Applying the above definition to the context of multilingual NMT, a model can be trained on the following pairs:

- English $\longrightarrow$ isiXhosa
- isiXhosa $\longrightarrow$ isiZulu

and thereafter assigned a task of translating English $\longrightarrow$ isiZulu. The feasibility of zero-short learning lies on the fact that the overall parameter optimization of the model yields a shared feature space for all encoded inputs, from which the target language is then decoded [35].

5.5 Model Evaluation

The growth in MT system development has rendered it imperative for researchers to formulate language-independent MT evaluation metrics that will replace the tardy and expensive human evaluation approaches. Han Lifeng [36] gives a comprehensive survey on both the automatic and manual evaluation techniques of evaluation MT outputs. In this work, we choose *perplexity* and the *bilingual evaluation understudy*, the widely used metrics in modern NMT tasks. In the forthcoming sections, we give a detailed outline of these two metrics.

5.5.1 Perplexity

In information theory, perplexity quantifies the efficiency of a probability distribution in predicting some given test-set. Derived from exponentiating the entropy of a probability distribution, perplexity has been widely used in the domain of NLP. When computing perplexity, the entropy is the average amount of information required to encode the representations of a random variable. Suppose we have a random variable X and probability distribution $p(x)$. We compute $H(p)$, the entropy of probability distribution $p(x)$ in the following manner:

$$H(p) = -\sum p(x) \log_2 p(x) \tag{5.5.1}$$

Exponentiating Equation 5.5.1 yields the weighted average number of possible choices a random variable has (the perplexity, $PP(X)$).

$$\begin{aligned} PP(X) &= e^{-\frac{1}{N}\sum_{i=1}^{N} \ln q(x_i)} \\ &= e^{-\frac{1}{N}\sum_{i=1}^{N} \ln q(x_i)} \\ &= \prod_{i=1}^{N} q(x_i)^{-\frac{1}{n}} \\ &= \sqrt[N]{\frac{1}{q(x_1)q(x_2)....q(x_N)}} \end{aligned} \tag{5.5.2}$$

where q,N and X_i denote the proposed probability model,total number of words and i^{th} word respectively. As a result, smaller values of perplexity indicated better model performance.

5.5.2 Bilingual Evaluation Understudy

In the domain of MT, the bilingual evaluation understudy (BLEU) score is the extensively used evaluation metric. The BLEU metric depends on modified precision P_n, the degree of word pair (n-grams) overlap between the MTS outputs and the human translation references [72]. To compute the modified precision we first count the n-gram matches in each target sentence and thereafter add the clipped count of n-grams in all the candidate sentences. We then divide the resulting sum by the sum of probable n-grams in the test data.

$$P_n = \frac{\sum_{C \in \{Candidates\}} \sum_{ngram \in C} Count_{clip}(ngram)}{\sum_{C^* \in \{Candidates\}} \sum_{ngram^* \in C^*}} \tag{5.5.3}$$

The modified precision ensures that the candidate sentence is neither too long nor too short by penalizing words that do not appear in the reference sentences. In addition, modified precision also penalizes the words that appear all the more much of the time than their maximum reference count. This allows MTS to use a word however many occurrences as it could be allowed, while restricting its usage more than it appears in any of the reference sentences. To account for the exponential decay of P_n with n, a weighted average of the logarithm of P_n is taken. The weighted average is computed with uniform weights λ_n, making it equivalent to a geometric mean of Pn. In any case, P_n by itself has no restriction on sentence length. Therefore, as a measure of sentence length restriction, P_n is weighted by the brevity penalty BP

$$BP = \begin{cases} 1 & \text{if } c > \text{r} \\ e^{(1-\frac{r}{c})} & \text{if } c \leq r \end{cases} \tag{5.5.4}$$

where r denotes the total length of the respective corpus reference sentence, and c is the length of candidate translations in the corpus. The overall BLEU score is given as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} \lambda_n \log P_n\right) \tag{5.5.5}$$

5.6 Summary

We have given a brief discussion on the hardware used in training all our models. The data, data-cleaning,data-sources and partitioning are outlined in this chapter. The chapter also discusses the transfer learning in detail. Borrowing some ideas from transfer

learning, we also discuss zero-shot learning and multilingual learning. Lastly, we discuss the NMT evaluation metrics employed in this work, namely, the BLEU and perplexity.

Chapter 6

Results

6.1 Chapter Organisation

This chapter discusses the experimental findings from all the training protocols discussed earlier in Chapter 5. Section 6.2 gives a brief recap of the training protocols and evaluation procedure. The results are outlined in the following manner; presented first in Section 6.3 are the experimental findings from the four baseline models. Thereafter, multilingual learning, zero-shot learning and transfer learning results are presented in Sections 6.4, 6.5 and 6.6 respectively. Section 6.7 then presents the comparison of the training protocols. An overview of the model performance with increase in training data is then given in Section 6.8. Finally, Section 6.9 gives a summary of the entire chapter.

6.2 Introduction

As outlined earlier in Chapter 5, we employ perplexity and the BLEU score metrics for evaluating translation quality of the different translations produced by our models. The core concept of the BLEU score is to quantify MT quality by comparing it with the respective human translations i.e. how close the MT output is to the human translation. In respect of our experiments, we perceive translation quality as the similarity between our model's translation and the respective human translation. The BLEU score values are presented as numbers ranging from 0 to 100, with preference given to higher scores. On the other hand, perplexity quantifies how well a probability model predicts a sentence or the next word. Perplexity ranges from 0 to infinity with preference given to lower values. For model inference, we use fixed samples of randomly selected sentences from

unseen data. These sampled sentences are then used for comparing the performance of different training protocols. In all of our translation outputs, we focus primarily on the dataset of interest, the En-Zu language pair. However, in the case of the baseline models, we delve into the translation quality of these models. The translation examples of all other languages pairs are presented in the Appendix.

6.3 Baseline models

As discussed earlier in Section 5.4.1 baseline models are based on the conventional encoder-decoder training protocol without any modifications. The transformer architecture is employed for all training pairs (NB: in all our training protocols we employ the same architecture as described in Section 4.5). Furthermore, these baseline models are used as a benchmark against which we compare the performance of the other training protocols.

6.3.1 English to isiZulu model

As discussed earlier in Section 5.2, the En-Zu pair had the least amount of training examples. Consequently, the En-Zu transformer model performance was not good and gave a minimum training and validation entropy loss of 1.14 $\pm$0.07 and 2.54 $\pm$0.01, respectively. Similarly, the training and validation perplexity was high with values of 3.13 $\pm$0.43 and 12.68 $\pm$0.25, respectively (see Figure A.1 in appendix for overall training performance per epoch). On the test-set the model produced an entropy loss of 2.52 $\pm$0.05 and perplexity of 12.42 $\pm$0.88.

English source sentence	isiZulu reference sentence	isiZulu translation
i wish i knew what i should say .	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngithini .	ngifisa ukuthi ngabe <unk> ukuthi ngabe <unk> . <eos>
words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	ukubonakala <unk> <unk> <unk> <unk> . <eos>

Table 6.1: En-Zu baseline model translations illustrating how the model managed to correctly translate one tri-gram though failing to correctly predict any of the words in the other sentence. The reference sentences column denotes the actual translation or target translation and the model's prediction/translation is called the 'isiZulu translation'. Each source-target/reference pair is presented along with its respective model-translation.

The overall BLEU score on the test-set was 8.7 $\pm$0.3, however low this value is, it still falls within the range of SOTA translations, with a BLEU score range of $7 - 9$ [63]. This

low BLEU score is an indication of bad and unreliable translations. Table 6.1 shows some of the test-set translations produced by the model (see Table A.1 in appendix for more translations). The model translations were not good as shown by the number of unknown tokens and a few target words correctly predicted by the model. The model did not get any of the sampled sentences correct, save for a single tri-gram and a few words in the respective sentences.

6.3.2 English to isiXhosa model

The En-Xh language pair had the most training examples (see Section 5.2), indicating a higher likelihood of obtaining good translations. During training the model achieved a minimum entropy loss and perplexity of 0.61 $\pm$0.11 and 1.84 $\pm$0.41, respectively. In the case of the validation, we respectively obtained an entropy loss and perplexity of 2.00 $\pm$0.01 and 7.44 $\pm$0.15 (for overall training performance per epoch see Figure A.2 in appendix). Testing the model on an unseen data set yielded an entropy loss of 2.02 $\pm$0.06

Source sentence	isiXhosa reference sentence	isiXhosa translation
the question is what are you going to do about it .	umbuzo uthi wena uzakwenza ntoni ngaloo nto .	umbuzo ngowokuba nina nakwenza ntoni ngalo . <eos>
a deal is a deal .	isivumelwano sisivumelwano.	isivumelwano isivumelwano . <eos>

Table 6.2: En-Xh baseline translations showing how the model got some of the translations correct though opting for new words in some instances but still maintaining context. The reference sentences column denotes the actual translation or target translation and the model's prediction/translation is called the 'isiXhosa translation'. Each source-target/reference pair is presented along with its respective model-translation.

and perplexity of 7.57 $\pm$0.54. Table A.2 shows how the model's translations closely resemble the reference translations and this was evidenced by a BLEU score of 20.9 $\pm$1.4 and the number of n-grams predicted correctly. In a nutshell, the model produced reasonable translations on most of the sentences. In some cases, the model used words that did not exist in the reference translation though still maintaining the context. For example the source sentence "the question is what are you going to do about it ." was translated to "umbuzo ngowokuba nina nakwenza ntoni ngalo .", with the major difference between reference and model translation being the word "nina".

6.3.3 English to Shona model

The En-Sh language pair had more than double the number of training examples than the En-Zu language pair (See Section 5.2). As a result, the En-Sh model performed bet-

ter than the En-Zu model with a minimum training loss and perplexity of 0.72 $\pm$0.08 and 2.06 $\pm$0.33 respectively. On the validation set, the model obtained an entropy loss of 2.09 $\pm$0.13 and perplexity of 8.92 $\pm$0.84 (for overall training performance per epoch see Figure A.3 in appendix). Compared with the En-Zu model, this model obtained 3.76 $\pm$0.9 less validation perplexity.

Testing the model on an unseen dataset, we obtained an entropy loss of 2.24 $\pm$0.02 and perplexity of 9.40 $\pm$0.38. In comparison with the En-Zu model, the En-Sh model obtained 3.02 $\pm$0.96 less perplexity, making it a potential parent model for transfer learning. The model produced an overall BLEU score of 16.5 $\pm$ 0.8 on the test-set. In comparison with the En-Zu baseline model, the En-Sh model performed 7.8 $\pm$ 0.85 BLEU better. However, regardless of the BLEU score being higher than that of the En-Zu model, not all En-Sh model translations make sense.

English source sentence	Shona reference sentence	Shona translation
maybe i can show you .	pamwe ndinogona kukuratidza .	pamwe ndinofanira <unk> . <eos> < eos>
she showed me her new car .	akandiratidza mota yake nyowani .	akandiratidza mota yake . <eos>
i will go and get you some .	ndichaenda ndikutorere mamwe .	ndichaenda newe iwe . <eos>

Table 6.3: En-Sh baseline translations results illustrating how the model failed to maintain context in some sentences and in some it did maintain context though with "unknown" tokens.

Tabulated in Table 6.3 are some of the example translations from the test-set. In some translations the model predicts "unknown" tokens, which resemble unknown words i.e the words are not in the model's vocabulary. However, as much as the model did manage to predict some of the keywords in some of the sentences, some translations were far from the reference translations. For example, the model translation, "ndichaenda newe iwe ." which can be back-translated to "I will go with you", is far from the source sentence, "i will go and get you some.".

6.3.4 isiXhosa to isiZulu model

The Xh-Zu language pair had about thrice as much data as the En-Zu pair (see Section 5.2). With the second most amount of training examples, one would expect the Xh-Zu model to produce the second highest BLEU score. However, this training pair yielded the best translation results largely due to the similarities between the two languages

(source and target language). With most south eastern Bantu languages of the same subclass, there is a substantial word/vocabulary overlap between language pairs. Consequently, the Xh-Zu model produced the best BLEU and perplexity scores on both validation and test-set. While training, we obtained 0.48 ± 0.31 minimum entropy loss and 1.63 ± 0.21 perplexity. The model produced a validation entropy loss and perplexity of 1.79 ± 0.9 and 6.02 ± 0.82, respectively (for overall training and validation performance per epoch see Figure A.4 in appendix).

isiXhosa source sentence	isiZulu reference sentence	isiZulu translation
ndifuna ukukwazisa ukuba andikwazi ukuya kwintlanganiso yasemva kwemini .	ngifuna ukukwazisa ukuthi angikwazi ukuya emhlanganweni ntambama .	ngifuna ukukwazisa ukuthi kufanele uhlehlise umhlangano . <eos>
ndicinge ukuba ndingakufumana apha .	bengicabanga ukuthi ngingakuthola lapha .	bengicabanga ukuthi ngingakuthola lapha . <eos>
ukuba kukho nantoni na ofuna ukuyenza , kuya kufuneka uyenze .	uma kukhona ofuna ukukwenza , kuzofanele ukwenze .	uma ufuna ukwenza okuthile ofuna ukukwenza . <eos>

Table 6.4: Xh-Zu translation examples showing how close to the reference sentences the model's translations are. Each source-target/reference pair is presented along with its respective model-translation.

Tabulated in Table A.3 are some test-set translation examples for the Xh-Zu model which yielded a BLEU score of 34.9 ± 2.5. Most of the model translations are relatively close to their respective target sentences save for a few. For example, the reference sentence "uma kukhona ofuna ukukwenza , kuzofaneleukwenze ." with an english translation of "if there be anything you want to do, do it", was translated to "uma ufuna ukwenza okuthile ofuna ukukwenza", which has an english equivalent of "if you want to do anything". These results show that the Xh-Zu model translations were relatively close to the ground truth which hypothetically can be credited to the language similarity between the source and target languages.

6.4 Multilingual models

Our multilingual models were trained on a combination of sets of language pairs. For each language pair, we appended the target sentence token to each source sentence. For example, the source sentence "good morning" becomes "2zu good morning" where token "2zu" indicates that the sentence is to be translated to isiZulu. We trained three multilingual models, namely $\text{Multilingual}_{\text{A}}$, $\text{Multilingual}_{\text{B}}$ and $\text{Multilingual}_{\text{C}}$.

6.4.1 Multilingual$_A$

Multilingual$_A$ was trained on En-Zu, En-Xh and Xh-Zu pairs. On the training set, this model achieved a minimum entropy loss of 1.12 ± 0.09 and perplexity of 3.07 ± 0.45. On the validation set, the model achieved an entropy of 1.93 ± 0.21 and perplexity of 6.90 ± 0.95 (for overall performance per epoch see Figure A.5 in appendix.). The model was then tested on an unseen En-Zu test-set and achieved a minimum entropy loss of 2.16 ± 0.31 and respective perplexity of 8.71 ± 0.27. We obtained an overall BLEU score of 18.6 ± 1.0 on the En-Zu test-set.

English source sentence	isiZulu reference sentence	isiZulu translation
2zu he is one of my neighbours .	ungomunye womakhelwane bami .	ungomunye wabangane bami . <eos>
2zu the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela linikelwe amaphoyisa . <eos>

Table 6.5: Some examples of En-Zu translation results for Multilingual$_A$, illustrating how close to the reference translation the model translation are. Each source-target/reference pair is presented along with its respective model-translation.

Presented in Table 6.5 are some of the test-set translation examples for model Multilingual$_A$ on En-Zu. The translations are fairly close to the reference sentences and in some cases, the model opted for new words, which led to a slight change in meaning. For example, the reference sentence "ungomunye womakhelwane bami ." with source sentence "he is one of my neighbours .", was translated to "ungomunye wabangane bami . <eos>" which means, "he is one of my friends". Furthermore, we tested the model on En-Xh and Xh-Zu tests sets on which we obtained 18.5 ± 1.2 and 30.4 ± 1.5 BLEU scores respectively. The test-set translations for En-Xh and Xh-Zu language pairs are presented in the Appendix, see Tables A.4 and A.5. Both En-Xh and Xh-Zu translation are fairly good with the model opting for unused words in some translations.

6.4.2 Multilingual$_B$

Trained on En-Zu and En-Sh pairs, Multilingual$_B$ achieved an entropy loss of 0.71 ± 0.02 and perplexity of 2.04 ± 0.08 on the training set (for model performance per epoch see Figure A.6 in appendix). On the validation set containing both En-Zu and En-Sh language pairs, we obtained an entropy loss of 2.19 ± 0.06 and perplexity of 8.94 ± 0.34. The model was tested on an unseen dataset containing En-Zu language pairs, on which it yielded an entropy loss of 2.57 ± 0.01 and perplexity of 13.05 ± 0.13. Table A.6 presents some of the En-Zu test-set translations.

English source sentence	isiZulu reference sentence	isiZulu translation
2zu words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	<unk> <unk> . <eos>
2zu why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani ubaba <unk> ? . <eos>

Table 6.6: Some examples of En-Zu translation results for Multilingual$_B$ illustrating how bad the model's translations were. Each source-target/reference pair is presented along with its respective model-translation.

On the En-Zu test-set, we obtained an overall BLEU score of 6.0 ± 0.3. On most of the En-Zu translations, the model failed to predict the target words and instead opted for the "<unk>" tokens (see Table 6.6), which denote unknown words i.e the words were not in the model's vocabulary. The En-Sh translation examples are presented in the appendix (see Table A.9). On the En-Sh translations we obtained a BLEU of 14.3 ± 0.20 which is about 8 BLEU scores higher than the En-Zu translation score. This huge difference in BLEU scores is largely due to the En-Zu pairs having few training examples and as a result, the model favours the En-Sh generalizations when training. Furthermore, the two training pairs, (particularly the target languages) are distantly related therefore there is little to no information to be leveraged from either pair. Instead, the language pair dissimilarities intensify the complexity of learning the mappings.

6.4.3 Multilingual$_C$

Multilingual$_C$ was trained and validated on En-Zu and En-Xh languages pairs. On the training set, we obtained a minimum entropy loss of 0.70 ± 0.02 and perplexity of 2.01 ± 0.08. On the validation set, the model achieved a minimum entropy loss of 2.02 ± 0.04 and perplexity of 7.46 ± 0.59 (see Table A.7 in the appendix for model performance per epoch). Tested on an unseen En-Zu test-set, the model achieved a minimum entropy loss of 2.06 ± 0.12 and perplexity of 7.81 ± 0.16.

English source sentence	isiZulu reference sentence	isiZulu translation
2zu why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani wonke umuntu ezithanda ikati ? . <eos>
2zu he is one of my neighbours .	ungomunye womakhelwane bami .	<unk> omakhelwane bami . <eos>
2zu the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela <unk> amaphoyisa. <eos>

Table 6.7: En-Zu translations, illustrating how Multilingual$_C$ performed when translating on the test-set. Though the model failed to predict all the sentences correctly, it still manages to get a few n-grams correct. Each source-target/reference pair is presented along with its respective model-translation.

The model yielded a BLEU score of 14.6 ± 0.20 on the test-set. A few of the translations examples on the En-Zu test-set are presented in Table 6.7. The model translations were fairly close to the target translations, though with a number of "<unk>" tokens. For example, source sentence "2zu the thief was handed over to the police ." was translated to "isela <unk> amaphoyisa. <eos>" which can be back-translated to "the thief <unk> the police ". On the En-Xh pair we obtained an overall BLEU of 18.7 ± 0.80 and the respective translation examples are tabulated in the Appendix (see Table A.10).

6.5 Zero-shot learning model

In implementing zero-shot learning, we first train a multilingual model on En-Xh and Xh-Zu language pairs, and thereafter test the model on an unseen En-Zu test-set. The model achieved a minimum entropy loss of 1.19 ± 0.13 and 1.99 ± 0.07 on the training and validation sets, respectively. As for perplexity, the model achieved a minimum of 3.30 ± 0.47 and 7.31 ± 0.37 on the training and validation sets, respectively (see Figure A.8 in the appendix for model performance per epoch).

English source sentence	isiZulu reference sentence	isiZulu translation
i wish i knew what i should say .	ngifisa ukuthi ngabe ngazi ukuthi ku-fanele ngithini .	ngifisa ukuthi ngabe <unk> ukuthi nga-be <unk> . <eos>
2zu why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani wonke umuntu ezithanda izi-ngane ? . <eos>
2zu the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela labaleka nemali . <eos>

Table 6.8: Some examples of En-Zu translation results for zero-shot learning model. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.

On an unseen En-Zu test-set the model yielded a minimum entropy loss of 2.36 ± 0.03 and perplexity of 10.59 ± 0.39, respectively. Tabulated in Table 6.8 are some of the test translations on which the model achieved an overall BLEU of 10.60 ± 0.20. The examples indicate that the model did not get any of the translations correct. However, it still managed to predict a few target n-grams (i.e combination of n ordered words) correctly. For example, the source sentence "2zu why does everybody love cats ? ." was translated to "kungani wonke umuntu ezithanda izingane ? . <eos>" which can be back-translated

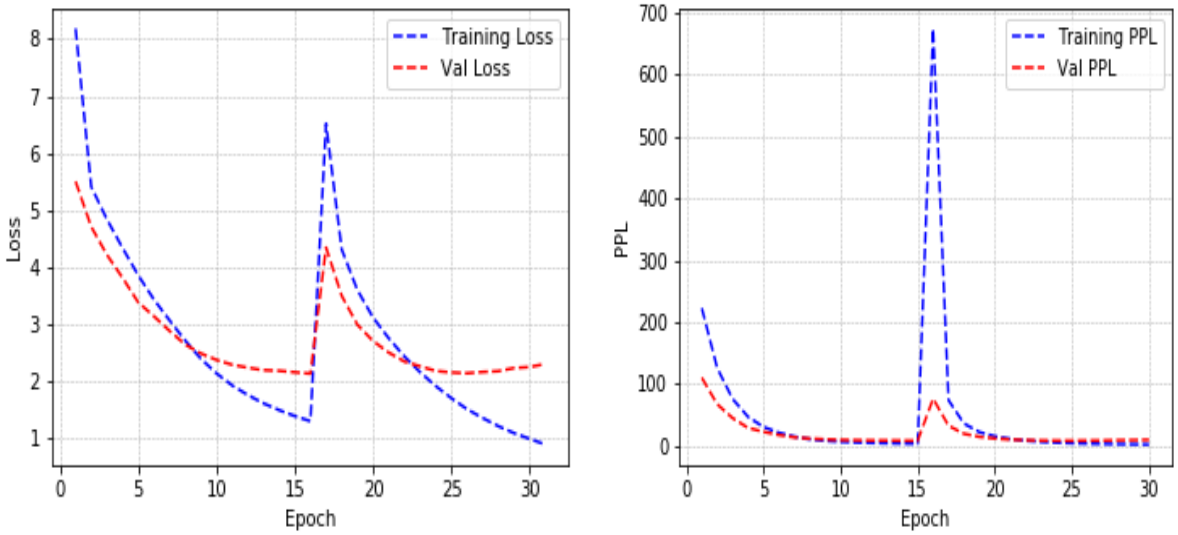
to "2zu why does everybody love kids ? .". This shows that with more training data, zero-shot learning has the potential of alleviating the low resource problem of Bantu languages.

6.6 Transfer learning models

Leveraging pre-trained models to improve the performance of a low resource task, we made use of the En-Xh and En-Sh baseline models to train separate En-Zu models.

6.6.1 English to isiXhosa parent model

Making use of the En-Xh baseline model weights to initialize a En-Zu model, the model exhibits an overall training performance depicted in Figure 6.1. All training and validation curves exhibit some exponential decay from epochs 1-17, denoting the baseline model training performance. Thereafter, the huge spike in both Figures 6.1a and 6.1b is caused by the introduction of a new language pair (a new domain). After epoch 17,



(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

Figure 6.1: A pictorial representation of the training and validation entropy loss and perplexity for transfer leaning with a En-Xh parent model, per epoch. The En-Xh baseline model is used for initializing the En-Zu model. The spikes in both entropy loss and perplexity denote the introduction of a new domain or language pair.

English source sentence	isiZulu reference sentence	isiZulu translation
i do not want you touching my stuff .	angikufuni uthinta izinto zami .	angifuni ukukuhlupha izinto zami . <eos>
we are going the wrong way .	sihamba ngendlela engafanele .	akulungile indlela . <eos>
the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela <unk> amaphoyisa . <eos>

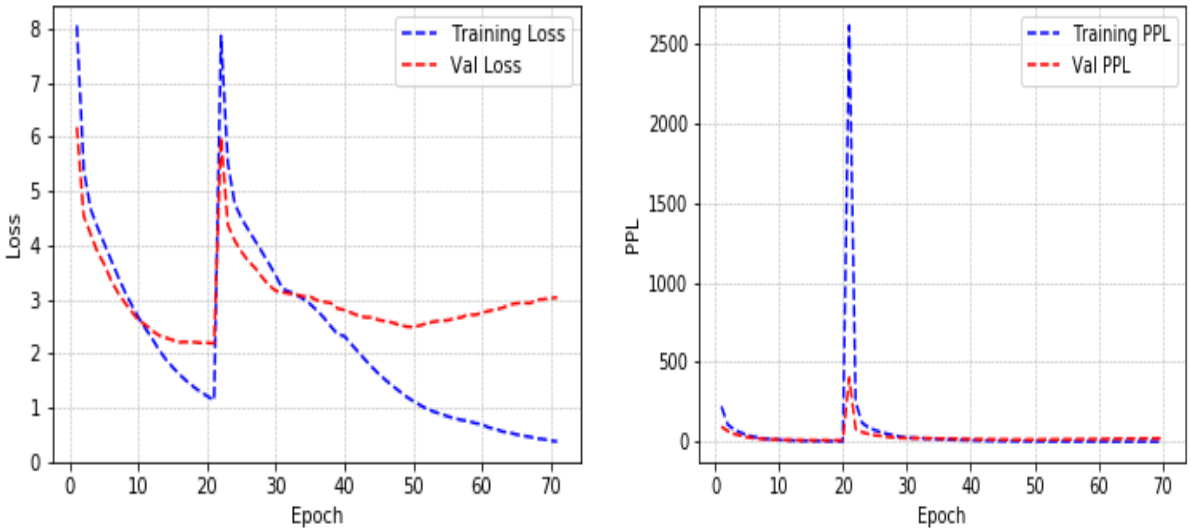
Table 6.9: Some examples of En-Zu translation results from the En-Xh parent model, showing how the model managed to predict a number of n-grams correctly. Each source-target/reference pair is presented along with its respective model-translation.

the model then begins to learn the new representations while leveraging the preexisting representations. This model achieved a minimum entropy loss of 0.49 ± 0.10 and perplexity 1.63 ± 0.18 on the training set. On the validation set, the model achieved 2.09 ± 0.08 minimum entropy loss and 8.06 ± 0.44 perplexity.

The model was tested on the En-Zu test-set and it yielded an entropy loss of 2.03 ± 0.04 which corresponds to a perplexity of 7.62 ± 0.31. This model achieved a test-set BLEU score of 14.8 ± 0.20. A sample of the test-set translations is tabulated in Table 6.9, from which we notice that the model managed to correctly translate a single sentence. The other translation examples show that the model was able to get some of the target n-grams correct. For example, the source sentence "the thief was handed over to the police ." was translated to "isela <unk> emaphoyiseni . <eos>" which can be backtranslated to "the thief <unk> police <oes> . ".

6.6.2 English to Shona parent model

As our second transfer learning-based model, we initialize another En-Zu model with the En-Sh parent model weights. Figure 6.2 gives a pictorial view of the entropy and perplexity loss per epoch. Similar to the transfer learning task discussed in Section 6.6.1, the decay from epochs 1-20 is based on the training of the parent model (or baseline model). The spike that then follows denotes the introduction of a new domain (or data set). After the 21^{st} epoch, the model begins to leverage the pretrained En-Sh representations to learn the new En-Zu representations. This model achieved a minimum entropy loss of 0.76 ± 0.03 on the training set and 2.46 ± 0.02 on the validation set. These entropy losses respectively correspond to perplexity scores of 2.14 ± 0.11 and 11.70 ± 0.38 on the train and validation set.



(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

Figure 6.2: Training and validation entropy loss and perplexity for transfer leaning with a En-Sh parent model per epoch, illustrating how the model behaves when we introduce a new domain. The En-Sh baseline model is used for initializing the En-Zu model.

English source sentence	isiZulu reference sentence	isiZulu translation
i do not want you touching my stuff .	angikufuni uthinta izinto zami .	angifuni ukulahlekelwa <unk> zami . <eos>
words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	amazwi <unk> <unk> <unk> . <eos>
why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani wonke umuntu <unk> ? . <eos>
the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela <unk> emaphepheni . <eos>

Table 6.10: Some examples of En-Zu translation results from the En-Sh parent model . The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.

To best infer the model performance we, tested the model on the En-Zu test-set and obtained a minimum entropy loss of 2.48 ± 0.01, corresponding to a perplexity of 11.94 ± 0.24. The model achieved an overall BLEU score of 9.6 ± 0.7. Tabulated in Table 6.10 are some of the test-set translation examples. The model managed to correctly translate the first tri-gram in reference translation "kungani wonke umuntu ewathanda amakati ? ." .

Though most of the translations are not correct, the model still managed to get a few of the n-grams correct. For some translations, however, the model failed to predict any of the words which is due to the model failing to generalize onto the new data-set.

6.7 Training protocol comparison

To best explore the training protocol comparisons, we focus mainly on the BLEU evaluation metric along with back translating the model outputs to English. All four training protocols were tested on the En-Zu language pair. The En-Zu baseline BLEU score of 8.7 ± 0.3 was used as the benchmark against which the other protocols are compared. The final BLEU scores for the four training protocols are summarised in Table 6.11. Looking at the baseline models, we notice an increase in model performance with an increase in training data and source-target language similarity. The effects of language similarity between source and target languages is shown by the Xh-Zu pair having the highest BLEU score. This high score is largely due to the huge vocabulary overlap between the source (Xh) and target language (Zu), as a result the model easily learns to generalize the source to target language transcriptions.

Model type	En-Zu	En-Xh	Xh-Zu	En-Sh	En-Zu Gain
Baseline	8.7 ±0.3	20.9 ±1.4	34.9 ±2.5	16.5 ±0.8	-
$\text{Multilingual}_{\text{A}}$	18.6 ±1.0	18.5 ±1.2	30.4 ±1.5	-	**9.9 ±1.0**
$\text{Multilingual}_{\text{B}}$	6.0 ±0.3	-	-	14.3 ±0.2	**- 2.7 ±0.4**
$\text{Multilingual}_{\text{C}}$	14.6±0.2	18.7 ±0.8	-	-	**5.9 ±0.3**
TL $\text{En-Xh}_{\text{parent}}$	14.8 ±0.2	-	-	-	**6.1 ±0.4**
TL $\text{En-Sh}_{\text{parent}}$	9.6 ±0.7	-	-	-	**0.9 ±0.8**
Zero-shot	10.6 ±0.2	18.1 ±1.5	34.0 ±2.4	-	**2.0 ±0.4**

Table 6.11: BLEU scores for the baseline, transfer learning, multilingual and zero-shot learning for the language pairs built from English (En), Shona (Sh), isiXhosa (Xh), isiZulu (Zu). The gains are calculated only for English-to-isiZulu (En-Zu), our target pair. Error bars are given by the standard deviations from ten separate re-training of the models in each case. $\text{Multilingual}_{\text{A}}$ is trained on En-Zu, En-Xh & Xh-Zu. $\text{Multilingual}_{\text{B}}$ is trained on En-Zu & En-Sh. $\text{Multilingual}_{\text{C}}$ is trained on En-Zu & En-Xh. Zero-shot learning applies only to En-Zu, built from a En-Xh & Xh-Zu multilingual model.

Model type	Train Loss	Val Loss	Test Loss	Train PPL	Val PPL	Test PPL
En-Sh	0.72 ±0.08	2.09 ±0.13	2.24 ±0.02	2.06 ±0.33	8.92 ±0.84	9.40 ±0.38
En-Zu	1.14 ±0.07	2.54 ±0.01	2.52 ±0.05	3.13 ±0.43	12.68 ±0.25	**12.42 ±0.88**
En-Xh	0.61 ±0.11	2.00 ±0.01	2.02 ±0.06	1.84 ±0.41	7.44 ±0.15	7.57 ±0.54
Xh-Zu	0.48 ±0.31	1.79 ±0.09	1.80 ±0.03	1.63 ±0.21	6.02 ±0.82	6.02 ±0.28
Multilingual$_A$	1.12 ±0.09	1.93 ±0.21	2.16 ±0.31	3.07 ±0.45	6.90 ±0.95	**8.71 ±0.27**
Multilingual$_B$	0.71 ±0.02	2.19 ±0.06	2.57 ±0.01	2.04 ±0.08	8.94 ±0.34	**13.05 ±0.13**
Multilingual$_C$	0.70 ±0.02	2.02 ±0.04	2.06 ±0.12	2.01 ±0.08	7.46 ±0.59	**7.81 ±0.16**
TL En-Xh$_{parent}$	0.49 ±0.10	2.09 ±0.08	2.03 ±0.04	1.63 ±0.18	8.06 ±0.44	**7.62 ±0.31**
TL En-Sh$_{parent}$	0.76 ±0.03	2.46 ±0.02	2.48 ±0.01	2.14 ±0.11	11.70 ±0.38	**11.94 ±0.24**
Zero-Shot	1.19 ±0.13	1.99 ±0.07	2.36 ±0.03	3.30 ±0.47	7.31 ±0.37	**10.59 ±0.39**

Table 6.12: Training, validation and test entropy and perplexity loss scores for all the models. All En-Zu test perplexity values are in bold. Error bars are given by the standard deviations from ten separate re-training of the models in each case.

Source sentence	we are going the wrong way .
Reference sentence	sihamba ngendlela engafanele .

Training protocol	Model translation	Back-translation
Baseline model	<unk> indlela . <eos>	<unk> the way . <eos>
Multilingual$_A$	sihamba ngendlela engalungile . <eos>	we are going the wrong way . <eos>
Multilingual$_B$	<unk> . <eos>	<unk> . <eos>
Multilingual$_C$	<unk> umgwaqo . <eos>	<unk> the road . <eos>
TL En-Xh$_{parent}$	akulungile indlela . <eos>	road not right . <eos>
TL En-Sh$_{parent}$	<unk> indlela . <eos>	<unk> way . <eos>
Zero-Shot	ngendlela esifanele kwenzeke . <eos>	the way we happen . <eos>

Table 6.13: A comparison of all the respective translations with back translated outputs. Though difficult to back translate some of the model outputs, we had native language speakers back translate the model translation to English. All multilingual model source sentences begin with target sentence token "2zu".

The overall BLEU score ranking of the six models (on the En-Zu task) is as follows:

1. Multilingual$_A$ 2. TL En-Xh$_{parent}$ 3. Multilingual$_C$ 4. Zero-shot learning
5. TL En-Sh$_{parent}$ 6. Baseline model 7. Multilingual$_B$.

These rankings are also demonstrated by the translation quality in Tables 6.13, 6.14 and 6.15. The minimum training, validation and test loss scores for all the models are tabulated in Table 6.12. Looking at Tables 6.11 and 6.12, we note that though Multilingual$_A$ has a perplexity that is greater than that of TL En-Xh$_{parent}$, its overall translation

Source sentence	2zu why does everybody love cats ? .
Reference sentence	kungani wonke umuntu ewathanda amakati ?

Training protocol	Model translation	Back-translation
Baseline model	kungani <unk> <unk> ? . <eos>	why <unk> <unk> ? . <eos>
Multilingual$_A$	kungani wonke umuntu ewathanda amakati ?. <eos>	why does everybody love cats ? . <eos>.
Multilingual$_B$	kungani ubaba <unk> ? . <eos>	why does father <unk> ? <eos>
Multilingual$_C$	kungani wonke umuntu ezithanda ikati ? . <eos>	why does everybody love cats ? . <eos>.
TL En-Xh$_{parent}$	kungani abantu bethanda ikati ? . <eos>	why do people love cat ? . <eos>
TL En-Sh$_{parent}$	kungani wonke umuntu <unk> ? . <eos>	why does everybody <unk> ? . <eos>
Zero-Shot	kungani wonke umuntu ezithanda izinganе ? <eos>.	why does everybody love kids ? . <eos>.

Table 6.14: A comparison of all the respective translations with back translated outputs. Though difficult to back translate some of the model outputs, we had native language speakers back translate the model translation to English. All multilingual model source sentences begin with target sentence token "2zu".

Source sentence	i wish i knew what i should say .
Reference sentence	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngithini .

Training protocol	Model translation	Back-translation
Baseline model	ngifisa ukuthi ngabe <unk> ukuthi ngabe <unk> . <eos>	I wish <unk> that <unk> . <eos>
Multilingual$_A$	ngifisa ukuthi ngazi ukuthi ngithini . <eos>	i wish i knew what i should say . <eos>
Multilingual$_B$	ngiyazibuza ukuthi ngiyakwazi yini . <eos>	I wonder if I know . <eos>
Multilingual$_C$	ngifisa ukuthi ngabe ngazi iqiniso . <eos>	I wish I knew the the truth <eos>
TL En-Xh$_{parent}$	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngazi . <eos>	I wish I knew that I should know <eos>
TL En-Sh$_{parent}$	ngifisa ukuthi ngabe ngazi ukuthi ngabe <unk> . <eos>	I wish I knew that that <unk> <eos>
Zero-Shot	ngifisa ukuthi ngabe <unk> ukuthi ngabe <unk> . <eos>	I wish that <unk> that <unk> <eos>

Table 6.15: A comparison of all the respective translations with back translated outputs. Though difficult to back translate some of the model outputs, we had native language speakers back translate the model translation to English. All multilingual model source sentences begin with target sentence token "2zu".

quality better than that of the later, as shown by both BLEU score and translation examples in Tables 6.13, 6.14 and 6.15. Furthermore, this phenomenon of TL En-$\text{Xh}_{\text{parent}}$ perplexity being lower than that of $\text{Multilingual}_{\text{A}}$ shows the importance of BLEU score as it has a high correlation with the reference sentence. Whereas perplexity focuses on the most likely word which allows room for word variations.

$\text{Multilingual}_{\text{A}}$ produced the highest BLEU score (18.6 $\pm$1.0), with a gain of 9.9 ± 1.0. However, this model resulted in a slight decrease in En-XH and Xh-Zu BLEU scores, due to the complexity of learning to generalize multiple language pair representations. The other reason as to why $\text{Multilingual}_{\text{A}}$ gave the highest score, is the language similarity between Xh and Zu. In this particular case the model only learns to transcribe the source sentence to either isiZulu or isiXhosa, two languages with a high similarity. This is forcibly illustrated in Tables 6.13, 6.14 and 6.15, where $\text{Multilingual}_{\text{A}}$ outperforms all the other models in correctly translating the respective source sentences.

Since both isiXhosa and isiZulu fall under the Nguni language family and not only are they similar but they have a high vocabulary overlap. For this reason, models $\text{Multilingual}_{\text{A}}$ and $\text{Multilingual}_{\text{C}}$ leverage the isiXhosa representations in decoding to isiZulu. Still on the notion of language similarity, En-$\text{Xh}_{\text{parent}}$ model achieved the second largest BLEU gain of 6.1 ± 0.4, and this is also due to transfer learning leveraging the pretrained En-Xh representations, for which there is a significant word overlap and similarity between the source task target language (Xh) and the target task target language (Zu). As a result, the En-$\text{Xh}_{\text{parent}}$ model greatly improves the En-Zu generalizations.

The performance of the En-$\text{Sh}_{\text{parent}}$ also demonstrates the importance of languages similarity when performing transfer learning. This model achieved a BLEU gain of 0.9 ± 0.8, which is 5.2 ± 0.85 BLEU less than the En-$\text{Xh}_{\text{parent}}$. Compared to TL En-$\text{Xh}_{\text{parent}}$, the En-$\text{Sh}_{\text{parent}}$ model's lower BLEU gain bears witness to the fact that there were fewer representations to be leveraged from this parent model. Tables 6.13, 6.14 and 6.15 translations give evidence of languages similarity importance as En-$\text{Xh}_{\text{parent}}$ translations gives the second most accurate results. En-$\text{Sh}_{\text{parent}}$ translations on the other hand are better than both baseline and $\text{Multilingual}_{\text{B}}$.

$\text{Multilingual}_{\text{B}}$ had the least BLEU gain with a score of -2.7 ± 0.4. The decrease in BLEU corroborates the idea of language similarity, as the model is faced with a daunting task of learning to translate unrelated pairs. Faced with a task of generalizing on distant lan-

guage pairs, Multilingual$_B$ favoured the pair with the most training examples. With the third highest BLEU gain of 5.9 ± 0.3, Multilingual$_C$ also demonstrates the importance of language similarity and this gain is due to both target languages being similar, thus simplifying the decoding process. Zero-shot learning on the other hand achieved a BLEU gain of 2.0 ± 0.4, having not seen the En-Zu language pair at training. This result shows that with the necessary data, zero-shot learning can be used as a training protocol for unavailable pairs.

6.8 Model performance with increase in data

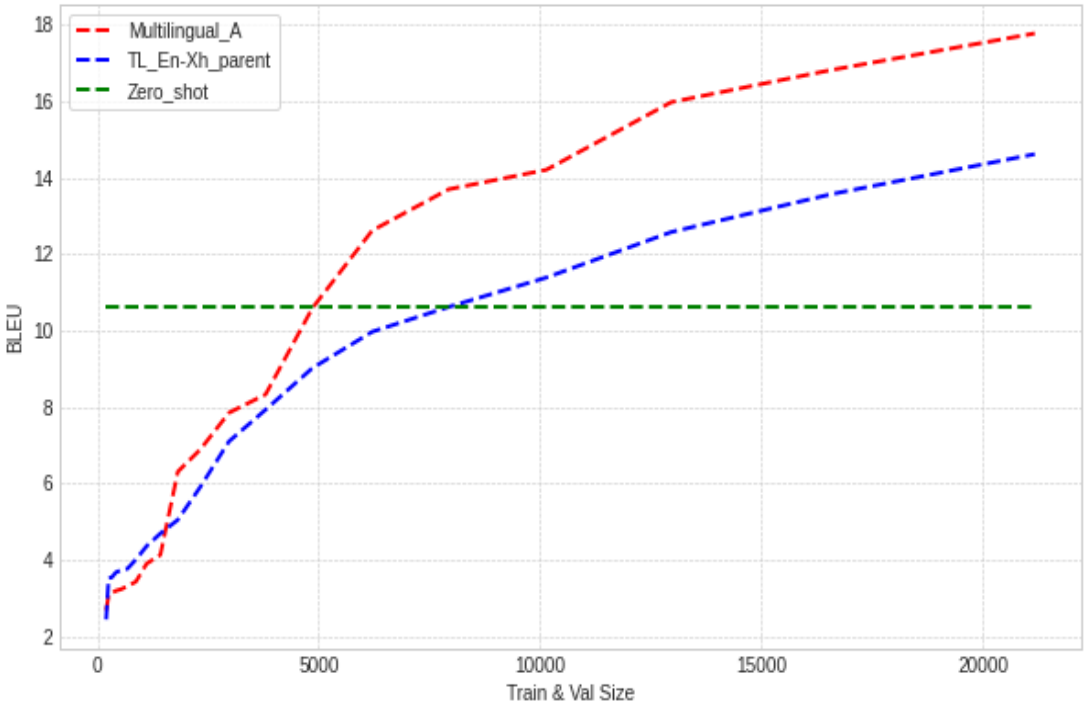


Figure 6.3: A graphical representation of BLEU score results for examining model performance with increase in data. Multilingual$_A$ and TL En-Xh$_{parent}$ models trained on En-Zu data-sets of different sizes. The zero-shot learning model is not retrained as its learned representations do not depend on the amount of En-Zu training pairs. The multilingual model outperforms transfer learning for any reasonable training set size.

To assess the effects of the amount of training data on multilingual and transfer learning training protocols, we train and validate our respective models on En-Zu pairs of increasing sizes. We exponentially increase the size of the training and validation sets while maintaining a constant test-set size. The zero-shot learning model was not retrained, as its learned representations do not depend on the size of the En-Zu target pair. Figure 6.3 shows the model performance for the respective protocols. We notice a clear exponential trend growth of both multilingual and transfer learning performance.

Taken together, the multilingual and transfer learning result prove a known reality, that deep learning models perform better with increase in data. An interesting observation shown Figure 6.3 is that to reach a BLEU score of 10.6, the equivalent of the zero-shot learning model, the multilingual model required less training data compared to transfer learning model. To attain 10.6 BLEU score TL En-$\text{Xh}_{\text{parent}}$ model required 38.78% more training data than $\text{Multilingual}_{\text{A}}$. This result is a consequence of multilingual learning leveraging the representations from the closely related En-Xh pair.

Furthermore, the decoding of En-Zu also makes the most of the Xh-Zu representations as the target languages in both pairs (En-Zu and Xh-Zu) are similar. A summary of the BLEU scores is presented in Table A.11. The result of this experiment cements the idea of opting for multilingual models over transfer learning whenever one is faced with a low-resource task and with the necessary data. Furthermore, these results illustrate the potentials of leveraging language similarity in a low resource setting. $\text{Multilingual}_{\text{A}}$ leverages the existing relations and word overlap between Zu and Xh, which makes the model perform better than the En-$\text{Xh}_{\text{parent}}$ model.

6.9 Summary

In this chapter, we have discussed the findings of this research. A comparative analysis of the training protocols has been given, from which we found Multilingual learning to be the favourable protocol, followed by transfer learning and zero-shot learning, successively. Furthermore, we have demonstrated the significance of language similarity in leveraging low resource training protocols, namely Zero-shot learning, transfer learning and multilingual learning.

Chapter 7

Conclusion and Future Work

7.1 Chapter Organisation

In this Chapter, we present the conclusions to the experimental objectives outlined in Chapter 1. We also discuss the possible avenues of furthering this work.

7.2 Conclusion

The main aim of this work was to explore the potential of leveraging low-resource learning protocols, namely transfer learning, zero-shot learning and multilingual learning on NMT of interrelated languages of the Southern-Bantu family. Furthermore, we examine the importance of language similarity on multilingual and transfer learning. Of all the Southern-Bantu languages, we focus on English-isiZulu, English-isiXhosa, English-Shona and isiXhosa-isiZulu languages pairs. Of these language pairs, isiXhosa and isiZulu fall under the same subclass called the Nguni class. As a result, these two languages are closely related, with a lot of vocabulary overlap. Shona, on the other hand, is distantly related to the two Nguni languages.

In training all our models, we employ the transformer architecture. Before discussing the underlying theory of the transformer architecture we discuss the fundamental building blocks of neural networks in Chapter 2, along with a background of NMT. Thereafter, we discuss sequential modelling and how RNNs help these problems in Chapter 3. To understand the sequence to sequence models, we discuss the encoder-decoder architecture in Chapter 4. In addition, we also outline the drawbacks of RNN and its variants. As a means of alleviating these drawbacks, we introduce the transformer architecture

in Chapter 4 by giving a side by side comparative analysis of the transformer with the RNNs architecture.

A comprehensive analysis of the underlying theory behind transfer learning, zero-shot learning and multilingual learning is outlined in Chapter 5. We also explain how NMT models are evaluated in the fifth chapter. We train four baseline models, namely En-Zu, En-Xh, En-Sh and Xh-Zu. The En-Xh and En-Sh models are then employed as parent models for transfer learning on En-Zu translation task. We also train three multilingual models on {En-Zu, En-Xh, En-Sh}, {En-Zu, En-Sh} and {En-Zu, En-Xh} pairs and these models are called $\text{Multilingual}_\text{A}$, $\text{Multilingual}_\text{B}$ and $\text{Multilingual}_\text{C}$ respectively. All transfer learning zero-shot and multilingual models were tested on common datasets.

Using multilingual learning ($\text{Multilingual}_\text{A}$) we achieve a BLEU score of 18.6 $\pm$1.0 for En-Zu pair, more than doubling the previous state-of-the-art and yielding significant gains (9.9 in BLEU score) over the baseline En-Zu transformer model. Multilingual learning for this dataset outperforms both transfer learning and zero-shot learning, though both of these techniques are better than the baseline model, with BLEU score gains of 6.1 and 2.0 respectively. The significance of language similarity is emphasised by $\text{Multilingual}_\text{A}$ and $\text{Multilingual}_\text{C}$ surpassing $\text{Multilingual}_\text{B}$ by more than 9.9 and 5.9 BLEU gains respectively.

We further demonstrate that transfer learning is a highly effective technique for training low resource translation models for closely related Southern-Eastern Bantu languages. Using the En-Xh baseline model, transfer learning to isiZulu had a BLEU score gain of 6.1 while using the En-Sh baseline model for transfer learning yielded no significant gain at all (0.9 ± 0.8). Since isiXhosa is similar to isiZulu while Shona is quite different, these findings illustrate the performance gains that can be achieved by exploiting language inter-relationships with transfer learning. The significance of leveraging language similarity is further emphasised or demonstrated by the fact that zero-shot learning, in which no En-Zu training data was available, outperformed transfer learning using the En-Sh baseline model. In conclusion, this work has led us to affirm that in addition to multilingual, transfer learning and zero-shot learning having tremendous opportunity of alleviating the low resource problem language similarity is of great significance. Furthermore, under similar experimental conditions (i.e the amount of data) we conclude that it is better to employ zero-shot learning than training baseline models from scratch.

With enough data multilingual learning is the best protocol for developing low resource NMT systems, followed by transfer learning.

7.3 Future work

In as much as this work has demonstrated the potential of leveraging the inter-relations within the sub-classes of the Bantu languages, namely the Nguni sub-class. An apparent extension of this research is to apply transfer learning, zero-shot learning and multilingual learning on other Bantu language sub-classes. Some potential techniques that are worth investigating are as follows:

- **Cross-lingual Word Embeddings**: this technique involves the training of word embeddings of multiple languages such that the final representations have words with the same semantic representation being closer to each other. The trained embeddings are then used in the final NMT model.
- **Back-Translation**: a simple data augmentation technique which does not involve any alterations to the pre-defined MT task. Instead, it involves the training of a target to source language translation task which is then used to generate more training examples for the predefined translation task.
- **Joint Back-Translation and Transfer Learning**: this technique employs the conventional back-translation method discussed above. Back-translation is incorporated into hierarchical transfer learning architectures to improve the quality of target translations. [58]
- **Self-Training enhanced Back-Translation**: this technique leverages the conventional back-translation method discussed above. Back-translation is incorporated into hierarchical transfer learning architectures to improve the quality of target translations. [8]

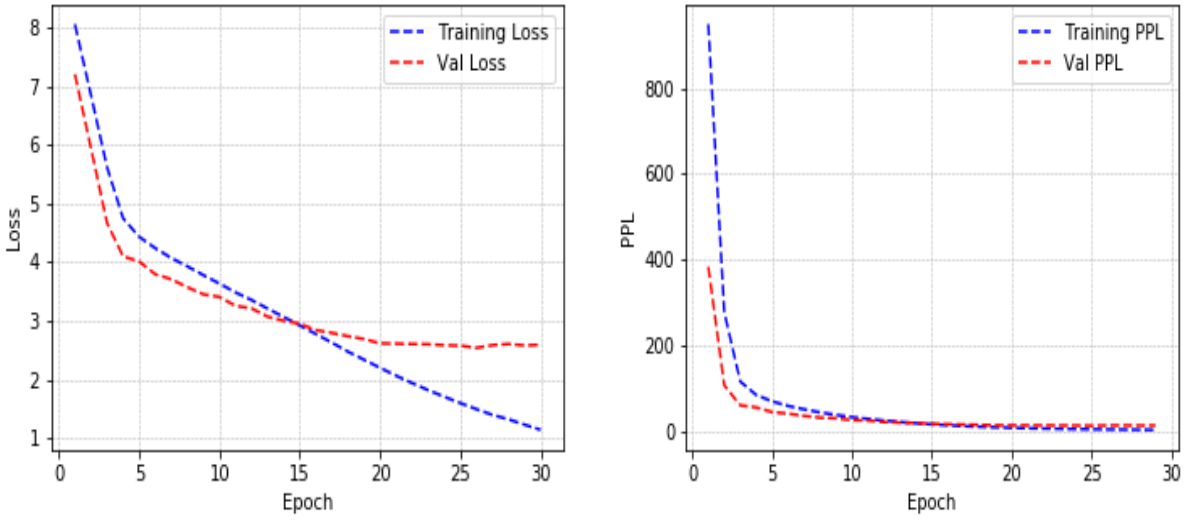
Considering the challenge of low resources, the above techniques are worth investigating as they do not require huge parallel corpus, save for the training of word embeddings which requires large monolingual corpora. Monolingual data is not as much a low resource as parallel corpus.

Appendix A

Appendix

A.1 Training and Evaluation Results

In this appendix we present a comprehensive summary of results for all of the different models we study, both in terms of training and validation as a function of epoch, as well as examples of translations.



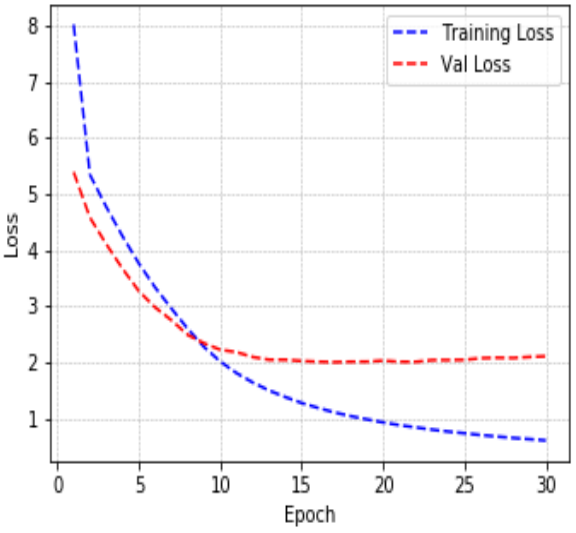
(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

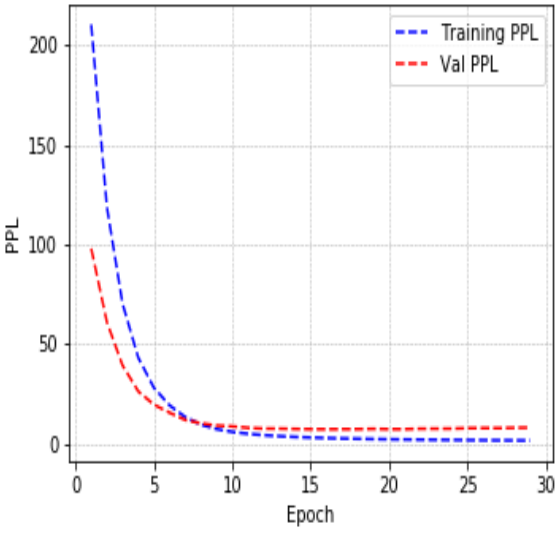
Figure A.1: Model loss and perplexity for En-Zu language pair. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.

English source sentence	isiZulu reference sentence	isiZulu translation
we talked about what we could do .	sakhuluma ngalokho esingakwenza .	<unk> ukwenza lokho kunalokho ngikwenze . <eos>
i do not want you touching my stuff .	angikufuni uthinta izinto zami .	angifuni izinto <unk> yami . <eos>
i wish i knew what i should say .	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngithini .	ngifisa ukuthi ngabe <unk> ukuthi ngabe <unk> . <eos>
it is very surprising .	kuyamangaza kakhulu .	<unk> kakhulu . <eos>
we are going the wrong way .	sihamba ngendlela engafanele .	<unk> indlela . <eos>
he can only pay twenty dollars at most .	angakhokha amadola angamashumi amabili kuphela . .	<unk> <unk> ezintathu . <eos>
words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	ukubonakala <unk> <unk> <unk> <unk> . <eos>
why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani <unk> <unk> ? . <eos>
he is one of my neighbours .	ungomunye womakhelwane bami .	<unk> <unk> wami . <eos>
the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela <unk> isela <unk> <unk> . <eos>

Table A.1: Some examples of En-Zu translation results for baseline model. These translations illustrate how the model performed on the test-set. The translation quality is far from being decent with the model producing many "unknown" tokens.

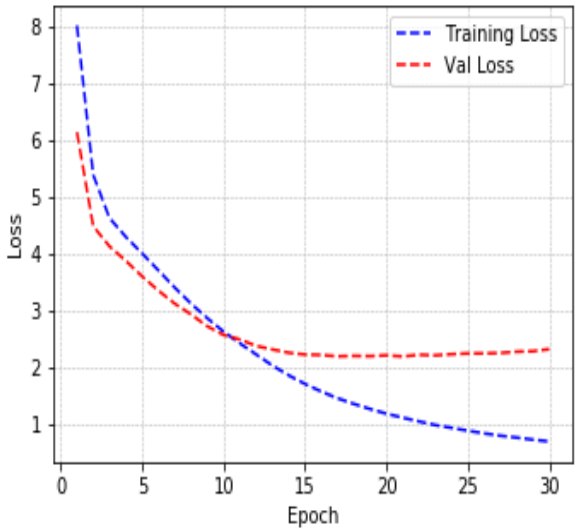


(a) Training and validation entropy loss per epoch

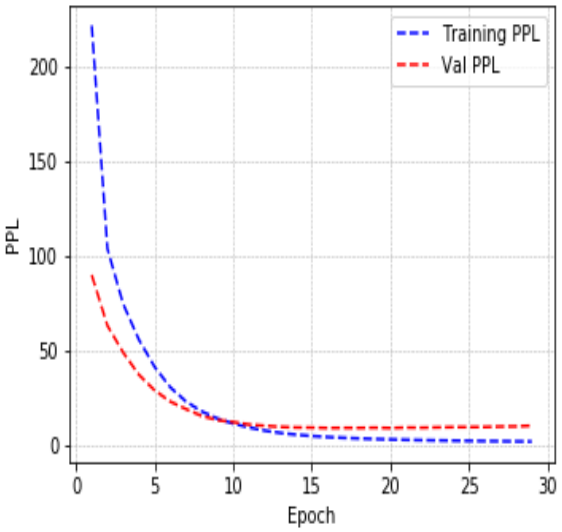


(b) Training and validation perplexity per epoch

Figure A.2: Model loss and perplexity for En-Xh language pair. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.



(a) Training and validation entropy loss per epoch

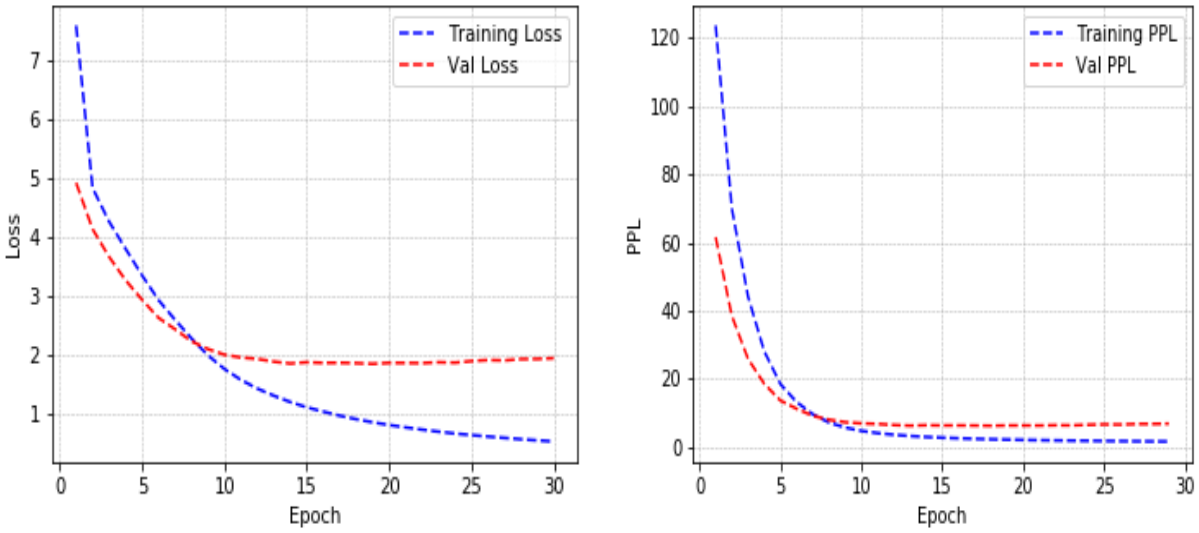


(b) Training and validation perplexity per epoch

Figure A.3: Model loss and perplexity for En-Sh language pair. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.

English source sentence	Shona reference sentence	Shona translation
today is extremely hot .	nhasi kwakanyanya kupisa .	nhasi kupisa chaizvo . <eos>
her story turned out to be true .	nyaya yake yakazove chokwadi .	zano rake yakazove chokwadi . <eos>
maybe i can show you .	pamwe ndinogona kukuratidza .	pamwe ndinofanira <unk> . <eos> < eos>
she showed me her new car .	akandiratidza mota yake nyowani .	akandiratidza mota yake . <eos>
i will go and get you some .	ndichaenda ndikutorere mamwe .	ndichaenda newe iwe . <eos>
they are looking for a complaint .	vari kutsvaga kunyunyuta .	<unk> . <eos>
we are all going to die anyway .	tese tichafa .	isu tese <unk> . <eos>

Table A.2: Some examples of En-Sh translation results for baseline model. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.



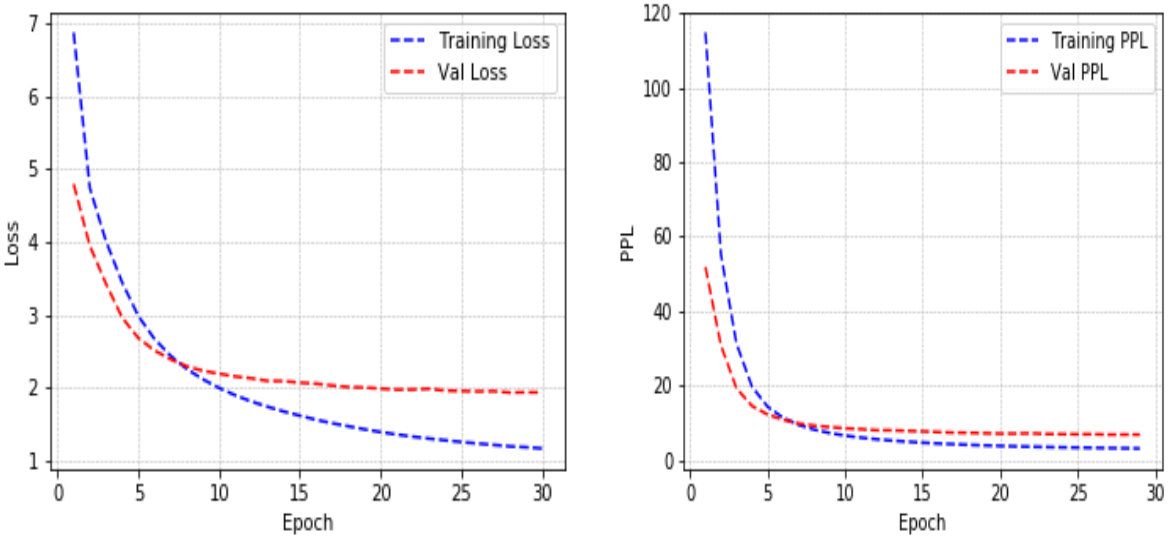
(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

Figure A.4: Model loss and perplexity for Xh-Zu language pair. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.

isiXhosa source sentence	isiZulu reference sentence	isiZulu translation
xa ndandisengumntwana , ndandidla ngokuhamba elunxwemeni rhoqo ngehlobo .	lapho ngiseyingane , ngangivame ukuya olwandle njalo ehlobo .	lapho njalo njalo ngiseyingane , ngangivame ukuya echibini . <eos>
ndifuna ukukwazisa ukuba andikwazi ukuya kwintlanganiso yasemva kwemini .	ngifuna ukukwazisa ukuthi angikwazi ukuya emhlanganweni ntambama .	ngifuna ukukwazisa ukuthi kufanele uhlehlise umhlangano . <eos>
ndicinge ukuba ndingakufumana apha .	bengicabanga ukuthi ngingakuthola lapha .	bengicabanga ukuthi ngingakuthola lapha . <eos>
ubonakala umhle ngeenwele zakho ezimfutshane .	ubukeka muhle ngezinwele zakho ezimfushane .	ubukeka muhle ngezinwele zakho ezimfushane . <eos>
ukuba kukho nantoni na ofuna ukuyenza , kuya kufuneka uyenze .	uma kukhona ofuna ukukwenza , kuzofanele ukwenze .	uma ufuna ukwenza okuthile ofuna ukukwenza . <eos>
ngaba usakhumbula ixesha esadibana ngalo okokuqala ? .	usasikhumbula isikhathi esahlangana ngaso okokuqala ? .	usasikhumbula isikhathi esahlangana okokuqala ? . <eos>
awunalungelo lakuhlala apha .	awunalo ilungelo lokuhlala lapha .	awunalo ilungelo lapha <unk> . <eos>
uye khona endaweni yam .	uye endaweni yami .	uye endaweni yami . <eos>

Table A.3: Xh-Zu translation examples showing how close to the reference sentences the model's translations are. Each source-target/reference pair is presented along with its respective model-translation.



(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

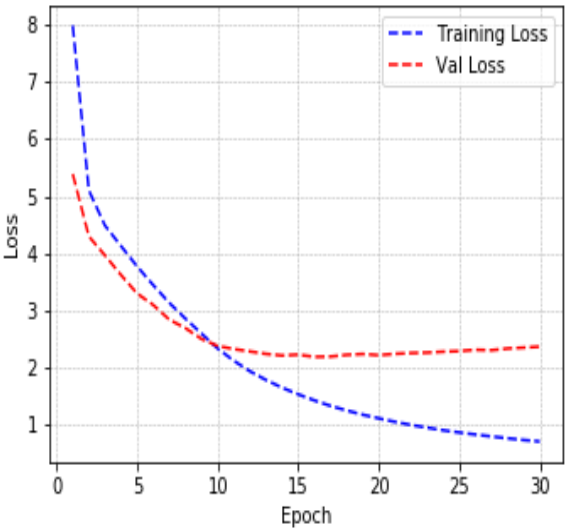
Figure A.5: Model loss and perplexity for Multilingual$_A$ trained on En-Zu, En-Xh and Xh-Zu pairs. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.

Source sentence	Target sentence	Translation
2xh we need to make sure that you are legally entitled to the grant .	kufuneka siqinisekise ukuba uselungelweni ngokusemthethweni lokusifumana isibonelelo eso .	kufuneka siqinisekise ukuba unelungelo lokusifumana isibonelelo eso . <eos>
2xh sassa is the branch of governmental that oversees the distribution of social grants to the people of south africa .	isassa licandelo elisisongezo kunikezelo lweenkonzo zikarhulumente nelijongene nokunikezela kweenkonzo nezibonelelo zoluntu kubemi basemzantsi afrika .	isassa sisongezo sengalo lweenkonzo zikarhulumente esilawula unikezelo lweminikelo kubemi bomzantsi afrika . <eos>
2xh the domestic violence act intends to establish shelters for victims of abuse .	umqulu wokuhlukunyezwa ezindlini ubonelela ngokusekwa kwamaziko okhuseleko .	umthetho wobundlobongela basekhaya unolungiselelo lokusekwa lokusekwa kwamakhaya angamakhusi . <eos>
2xh the plane crash was only last week .	ukuqhekeka kwenqwelomoya bekukho kwiveki ephelileyo .	ingozi yenzeke ibigqithile kwiveki ephelileyo . <eos>
2xh you can probably guess what happens though .	unokuqikelela ukuba kwenzeka ntoni na .	akudingeki wazi ukuthi kwenzekeni . <eos>
2xh i thought we could do it .	ndacinga ukuba singayenza .	bengicabanga ukuthi ngizokwenza . <eos>
2xh the question is what are you going to do about it .	umbuzo uthi wena uzakwenza ntoni ngaloo nto .	umbuzo ngowokuba nina nakwenza ntoni ngaloo nto . <eos>
2xh most of our houses have not yet complied with the policy change from quantity to quality housing .	izindlu zethu ezininzi azikaluthobeli ushenxiso olukhankanyiweyo kumgaqonkqubo oluthetha ukungaleqi ukwakha izindlu ezininzi kusuke kugxilwe ekwakheni izindlu ezikowona mgangatho uphezulu nowamkelekileyo .	'uninzi lwezindlu zethu azikathobelani nokutshintsha nokutshintsha nokutshintsha izindlu kwinani ukuya kumgangatho wezindlu . <eos>'
2xh a deal is a deal .	isivumelwano sisivumelwano.	'isivumelwano <unk> . <eos>'

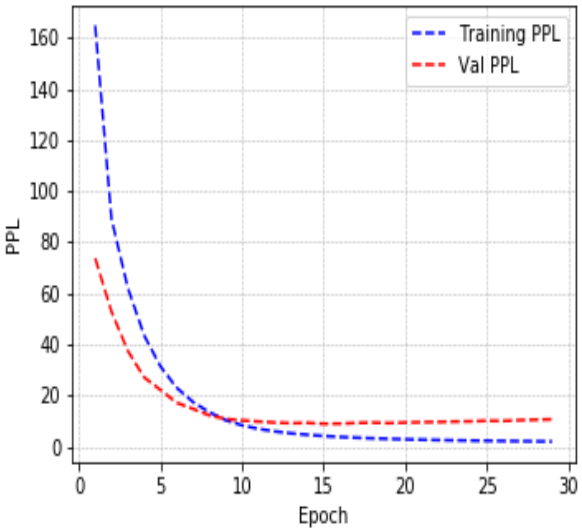
Table A.4: Some examples of En-Xh translation results for Multilingual$_A$. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.

Xhosa source sentence	Zulu reference sentence	Zulu translation
2zu xa ndandisengumntwana , ndandidla ngokuhamba elunxwemeni rhoqo ngehlobo .	lapho ngiseyingane , ngangivame ukuya olwandle njalo ehlobo .	lapho njalo njalo ngiseyingane , ngangivame ukuya echibini . <eos>
2zu ndifuna ukukwazisa ukuba andikwazi ukuya kwintlanganiso yasemva kwemini .	ngifuna ukukwazisa ukuthi angikwazi ukuya emhlanganweni ntambama .	ngifuna ukukwazisa ukuthi kufanele uhlehlise umhlangano . <eos>
2zu ndicinge ukuba ndingakufumana apha .	bengicabanga ukuthi ngingakuthola lapha .	bengicabanga ukuthi ngingakuthola lapha . <eos>
2zu ubonakala umhle ngeenwele zakho ezimfutshane .	ubukeka muhle ngezinwele zakho ezimfushane .	ubukeka muhle ngezinwele zakho ezimfushane . < eos>
2zu ukuba kukho nantoni na ofuna ukuyenza , kuya kufuneka uyenze .	uma kukhona ofuna ukukwenza , kuzofanele ukwenze .	uma ufuna ukwenza okuthile ofuna ukukwenza . <eos>
2zu i thought we could do it .	ndacinga ukuba singayenza .	ndacinga ukuba singayenza . <eos>
2zu ngaba usakhumbula ixesha esadibana ngalo okokuqala ? .	usasikhumbula isikhathi esahlangana ngaso okokuqala ? .	usasikhumbula isikhathi esahlangana okokuqala ? . <eos>
2zu awunalungelo lakuhlala apha .	awunalo ilungelo lokuhlala lapha .	awunalo ilungelo lapha <unk> . <eos>
2zu uye khona endaweni yam .	uye endaweni yami .	uye endaweni yami . <eos>

Table A.5: Some examples of Xh-Zu translation results for Multilingual$_A$. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.



(a) Training and validation entropy loss per epoch

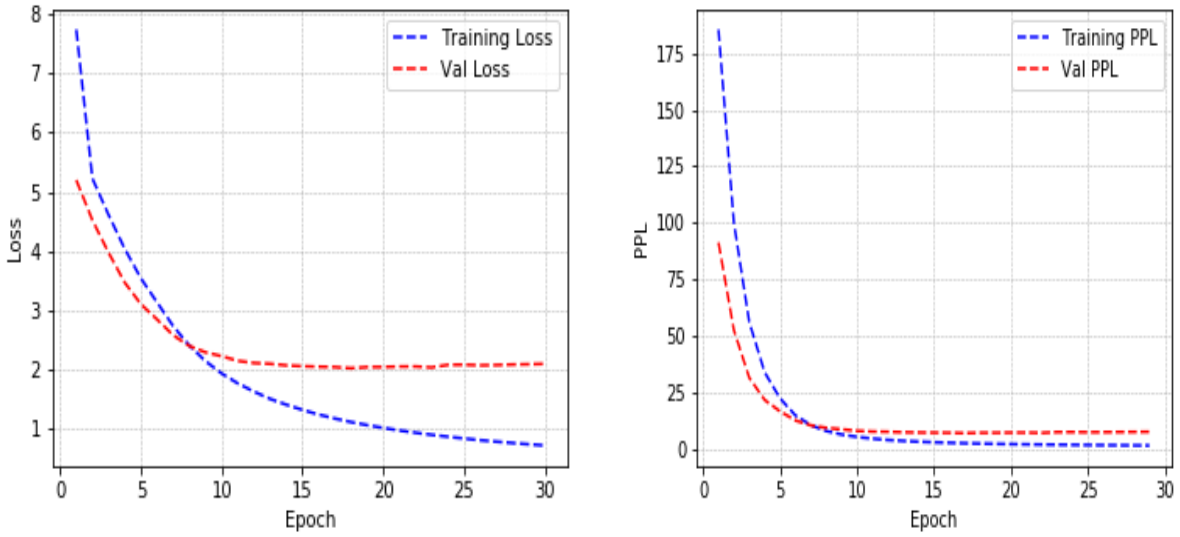


(b) Training and validation perplexity per epoch

Figure A.6: Model loss and perplexity for Multilingual$_B$ trained on En-Zu and En-Sh pairs. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.

English source sentence	isiZulu reference sentence	isiZulu translation
2zu we talked about what we could do .	sakhuluma ngalokho esingakwenza .	<unk> isenzo ngakho . <eos>
2zu i do not want you touching my stuff .	angikufuni uthinta izinto zami .	angifuni <unk> yami . <eos>
2zu i wish i knew what i should say .	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngithini .	ngiyazibuza ukuthi ngiyakwazi yini . <eos>
2zu it is very surprising .	kuyamangaza kakhulu .	<unk> . <eos>
2zu we are going the wrong way .	sihamba ngendlela engafanele .	<unk> . <eos>
2zu he can only pay twenty dollars at most .	angakhokha amadola angamashumi amabili kuphela . .	<unk> amabili <unk> . <eos>
2zu words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	<unk> <unk> . <eos>
2zu why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani ubaba <unk> ? . <eos>
2zu he is one of my neighbours .	ungomunye womakhelwane bami .	ungumfowethu . <eos>
2zu the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela <unk> nemali . <eos>

Table A.6: Some examples of En-Zu translation results for Multilingual$_B$. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.



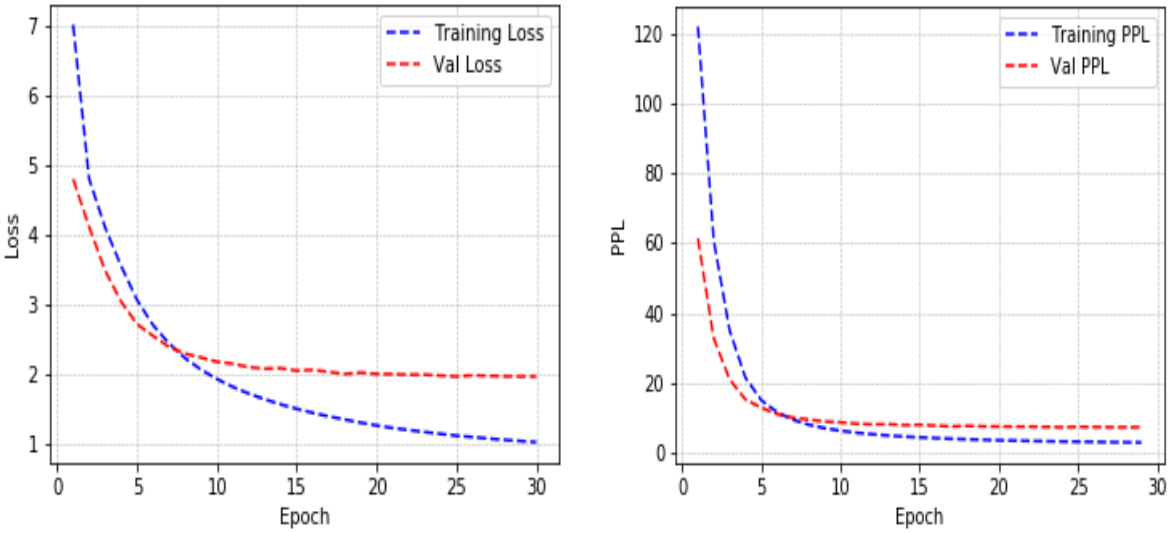
(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

Figure A.7: Model loss and perplexity for Multilingual$_C$ trained on En-Zu and En-Xh pairs. The graph on the left presents the training and validation entropy loss per epoch. Similarly, the graph on the right depicts the training and validation perplexity per epoch.

English source sentence	isiZulu reference sentence	isiZulu translation
2zu we talked about what we could do .	sakhuluma ngalokho esingakwenza .	sitshele konke <unk> ngakho . <eos>
2zu i do not want you touching my stuff .	angikufuni uthinta izinto zami .	angifuni <unk> <unk> zami . <eos>
2zu i wish i knew what i should say .	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngithini .	ngifisa ukuthi ngabe ngazi iqiniso . <eos>
2zu it is very surprising .	kuyamangaza kakhulu .	kuyamangalisa kakhulu . <eos>
2zu we are going the wrong way .	sihamba ngendlela engafanele .	<unk> umgwaqo . <eos>
2zu he can only pay twenty dollars at most .	angakhokha amadola angamashumi amabili kuphela . .	<unk> <unk> amathathu <unk> . <eos>
2zu words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	<unk> <unk> <unk> <unk> . <eos>
2zu why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani wonke umuntu ezithanda ikati ? . <eos>
2zu he is one of my neighbours .	ungomunye womakhelwane bami .	<unk> omakhelwane bami . <eos>
2zu the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela <unk> amaphoyisa. <eos>

Table A.7: Some examples of En-Zu translation results for $\text{Multilingual}_{\text{C}}$. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.



(a) Training and validation entropy loss per epoch

(b) Training and validation perplexity per epoch

Figure A.8: Zero-shot learning model training and validation entropy loss and perplexity per epoch. The multilingual model was trained on and validated on a combination of En-Xh and Xh-Zu language pairs. Thereafter, the model used for zero shot learning on the En-Zu language pair.

English source sentence	isiZulu reference sentence	isiZulu translation
i wish i knew what i should say .	ngifisa ukuthi ngabe ngazi ukuthi kufanele ngithini .	ngifisa ukuthi ngabe <unk> ukuthi ngabe <unk> . <eos>
2zu it is very surprising .	kuyamangaza kakhulu .	kuyathakazelisa kakhulu . <eos>
2zu we are going the wrong way .	sihamba ngendlela engafanele .	ngendlela esifanele kwenzeke . <eos>
2zu words rarely have only one meaning .	amagama akuvamile ukuba nencazelo eyodwa .	amagama kuphela <unk> . <eos>
2zu why does everybody love cats ? .	kungani wonke umuntu ewathanda amakati ? .	kungani wonke umuntu ezithanda izingane ? . <eos>
2zu he is one of my neighbours .	ungomunye womakhelwane bami .	ungomnye wabafundi sabo . <eos>
2zu the thief was handed over to the police .	isela linikelwe emaphoyiseni .	isela labaleka nemali . <eos>

Table A.8: Some examples of En-Zu translation results for zero-shot learning model. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.

English source sentence	Shona reference sentence	Shona translation
2sh i should not have to tell you to do your homework .	handifanire kukuudza kuti uite basa rako remumba .	usakanganwa kuita kuti uite basa rako remumba . <eos>
2sh today is extremely hot .	nhasi kwakanyanya kupisa .	nhasi <unk> . <eos>
2sh her story turned out to be true .	nyaya yake yakazove chokwadi .	<unk> nyaya yacho . <eos>
2sh maybe i can show you .	pamwe ndinogona kukuratidza .	<unk> . <eos>
2sh she showed me her new car .	akandiratidza mota yake nyowani .	akandiratidza mota yake . <eos>
2sh i will go and get you some .	ndichaenda ndikutorere mamwe .	enda <unk> . <eos>
"2sh hurry up , or you will miss the last train " .	" kurumidza kumhanyisa , kana iwe uchapotsa chitima chekupedzisira . " .	" kurumidza kumhanyisa , chitima . " . <eos>
2sh they are looking for a complaint .	vari kutsvaga kunyunyuta .	<unk> . <eos>
2sh we are all going to die anyway .	tese tichafa .	tese tichava ipapo . <eos>

Table A.9: Some examples of En-Sh translation results for Multilingual$_B$. The reference sentences are also called the target sentences and the model's prediction is called the translation. Each source-target/reference pair is presented along with its respective model-translation.

Source sentence	Target sentence	Translation
2xh we need to make sure that you are legally entitled to the grant .	kufuneka siqinisekise ukuba uselungelweni ngokusemthethweni lokusifumana isibonelelo eso .	kufuneka siqinisekise ukuba unelungelo lesibonelelo ngokusemthethweni . <eos>
2xh sassa is the branch of governmental that oversees the distribution of social grants to the people of south africa .	isassa licandelo elisisongezo kunikezelo lweenkonzo zikarhulumente nelijongene nokunikezela kweenkonzo nezibonelelo zoluntu kubemi basemzantsi afrika .	isassa sisolulo sengalo karhulumente yobonelelo nenikela ngezibonelelomali kubemi basemzantsi afrika . <eos>
2xh the domestic violence act intends to establish shelters for victims of abuse .	umqulu wokuhlukunyezwa ezindlini ubonelela ngokusekwa kwamaziko okhuseleko .	umthetho wobundlobongela basekhaya unolungiselelo lokusekwa kwamakhaya angamakhusi . <eos>
2xh the plane crash was only last week .	ukuqhekeka kwenqwelomoya bekukho kwiveki ephelileyo .	ingozi <unk> ibigqithile kwiveki ephelileyo . <eos>
2xh you can probably guess what happens though .	unokuqikelela ukuba kwenzeka ntoni na .	inokuba uyikholelwe into eyenzekileyo . <eos>
2xh i thought we could do it .	ndacinga ukuba singayenza .	ndacinga ukuba singayenza . <eos>
2xh the question is what are you going to do about it .	umbuzo uthi wena uzakwenza ntoni ngaloo nto .	umbuzo ngowokuba nina nakwenza ntoni ngaloo nto . <eos>
2xh most of our houses have not yet complied with the policy change from quantity to quality housing .	izindlu zethu ezininzi azikaluthobeli ushenxiso olukhankanyiweyo kumgaqonkqubo oluthetha ukungaleqi ukwakha izindlu ezininzi kusuke kugxilwe ekwakheni izindlu ezikowona mgangatho uphezulu nowamkelekileyo .	uninzi lwezindlu zethu azikathobelani nokutshintsha komgaqonkqubo osuka kwinani ukuya kumgangatho wezindlu . <eos>
2xh a deal is a deal .	isivumelwano sisivumelwano.	isivumelwano . <eos>

Table A.10: Some examples of En-Xh translation results for Multilingual$_C$. The reference sentences and model predictions are labeled target sentences and translation respectively. Each source-target/reference pair is presented along with its respective model-translation.

Train & Val Size	Multilingual$_A$	TL En-Xh$_{parent}$	Difference
200	2.68 ±0.14	2.44±0.1	0.24±0.17
256	3.07±0.24	3.53±0.11	−0.46 ± 0.26
327	3.13±0.25	3.54±0.15	−0.41 ± 0.29
418	3.19±0.27	3.67±0.16	−0.48 ± 0.31
534	3.23±0.28	3.73 ±0.17	−0.50 ± 0.33
682	3.31 ±0.32	3.77 ±0.17	−0.58 ± 0.36
872	3.43 ±0.35	4.01 ±0.16	−0.58 ± 0.38
1114	3.90±0.37	4.36 ±0.19	−0.46 ± 0.42
1424	4.13±0.4	4.69±0.19	−0.56 ± 0.44
1820	6.31 ±0.44	5.05±0.18	1.21 ± 0.48
2327	6.89±0.47	5.91±0.17	0.98 ± 0.50
2974	7.85±0.48	7.1±0.19	0.75 ± 0.52
3801	8.33 ±0.53	7.93 ±0.21	0.40 ± 0.57
4858	10.59±0.61	9.02 ±0.2	1.57 ± 0.64
6209	12.62±0.62	9.97±0.19	2.65 ± 0.65
7936	13.7±0.64	10.62±0.2	3.08 ± 0.67
10143	14.21 ±0.69	11.39±0.21	2.82 ± 0.72
12964	15.98 ±0.75	12.58±0.21	3.40 ± 0.78
16569	16.82±0.80	13.57±0.22	3.35 ± 0.83
21177	17.78 ±0.82	14.62±0.23	3.16 ± 0.85

Table A.11: BLEU score results for examining model performance with increase in data. Multilingual$_A$ and TL En-Xh$_{parent}$ models trained on En-Zu data-sets of different sizes. Each model is trained and validated on data-sets of the same size and then tested on a fixed size test-set from which we record the difference between Multilingual$_A$ and TL En-Xh$_{parent}$ BLEU scores.

Model Card for - Low Resource Neural Machine Translation for Southern African Languages.

The applications of zero-shot learning, transfer learning and multilingual learning on translation tasks of a select few Bantu languages (isiXhosa, isiZulu and Shona) and English are analysed on this card.

This model card gives an outline of the training procedures, capabilities and limitations of the models.

Model Details

Organisation:
Stellenbosch University & African Institute for Mathematical Sciences

Model date	Model type	Input
02 March 2021	Translation model	Text

Model architecture
We employ the transformer architecture with 6 encoder-decoder blocks, 8 attention heads, a 256-dimensional word representation, a dropout rate of 0.1 and positional feed-forward layers of 1024 inner dimension

Input & Output
The models input a source language and in turn, gives a target language output. In the case of multilingual models, the input sequences begin with a target language token.

Read the full paper here:
https://arxiv.org/abs/2104.00366

Access public code here:
https://github.com/Evandernyoni/Low-Resource-Neural-Machine-Translation-forSouthern-African-Languages

Citation details
Title: Low-Resource Neural Machine Translation for Southern African Languages.
Authors: Evander Nyoni & Bruce A. Bassett
Journal: arXiv preprint arXiv: 2104.00366
Year: 2021

Metrics

Model performance measures
We employ both perplexity and the BLEU score for evaluating the model performance. However, the BLEU score is our primary metric.

Training Data

Our models were trained on parallel sentences obtained from different domains, with the major data source being the Orpus Corpus. The Orpus Corpus comprises data from Wikipedia (En-Xh, En-Zu & En-Sh) and the South African Navy corpus (En-Xh). The rest of the data (En-Zu & En-Xh) was obtained from phrases for learning both isiZulu and isiXhosa.

Intended Use

1. Leveraging the low resources training protocols to African languages. Thus improving the understanding of the importance of language similarity when exploiting the aforementioned training protocols.

2. Improving translation tasks and other applications through fine-tuning.

Intended Users

NLP researchers.

Ethical Considerations

We entrust the users to, control and combat the negative consequences of misusing such models. We hope that this work contributes to NLP research of African languages will continue to grow.

Recommendations

The multilingual models were trained on few language pairs, a recommendation would be to train the models on more language pairs.

The models can also be trained with cross-lingual word embeddings. This technique involves the training of word embeddings of multiple languages such that the final representations have words with the same semantic representation being closer to each other.

Self-Training enhanced Back-Translation: this technique leverages the conventional back-translation method discussed above. Back-translation is incorporated into hierarchical transfer learning architectures to improve the quality of target translations.

List of references

[1] Center for High Performance Computing. http://wiki.chpc.ac.za/quick:start. [Accessed: 7-July-2020].

[2] Linguanaut Foreign Language Learning, Copyright © 2013 Linguanaut. http://www.linguanaut.com/index.htm. [Accessed: 15-September-2019].

[3] Omniglot the online encyclopedia of writing systems and languages. https://omniglot.com/language/phrases/zulu.php. [Accessed: 10-May-2020].

[4] ORPUS corpus the open parallel corpus. http://opus.nlpl.eu/. [Accessed: 10-September-2019].

[5] Underfitting vs. Overfitting. https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html. [Accessed: 7-July-2020].

[6] Wild Coast Xhosa phrase book. https://www.wildcoast.co.za/xhosa-phrasebook. [Accessed: 10-October-2019].

[7] Jade Z Abbott and Laura Martinus. Towards neural machine translation for african languages. *arXiv preprint arXiv:1811.05467*, 2018.

[8] Idris Abdulmumin, Bashir Shehu Galadanci, and Abubakar Isa. Using self-training to improve back-translation in low resource neural machine translation. *arXiv preprint arXiv:2006.02876*, 2020.

[9] Charu C Aggarwal. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[10] Neville Alexander. Evolving african approaches to the management of linguistic diversity. *Keynote address presented at AILA*, 2008.

[11] Neville Alexander. *The potential role of translation as social practice for the intellectualisation of African languages*. PRAESA Cape Town, 2010.

[12] DAISY JANE ANTIPUESTO. How nerves transmit impulses. https://www.nurseszone.in/nurseszone/how-nerves-transmit-impulses/46.html. [Accessed: 19-September-2019].

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[15] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[16] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE, 1993.

[17] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

[18] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[19] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederick Jelinek, John Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[20] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[21] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics, 2005.

[22] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[23] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[24] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

[25] Hoon Chung, Sung Joo Lee, and Jeon Gue Park. Deep neural network using trainable activation functions. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 348–352. IEEE, 2016.

[26] Ellen Cushman. Critical literacy and institutional language. *Research in the Teaching of English*, pages 245–274, 1999.

[27] Clement Martyn Doke, Benedict Wallet Vilakazi, DM Malcolm, and Mzilikazi Khumalo. *English-isiZulu/isiZulu-English Dictionary*. NYU Press, 2014.

[28] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, 2015.

[29] Gary F. Simons Eberhard, David M. and Charles D. Fennig. Ethnologue: Languages of the world. twenty-second edition. https://www.ethnologue.com. [Accessed: 19-September-2019].

[30] A Elithorn, R Banerji, and MA Nagao. Framework of a mechanical translation between japanese and english by analogy principle. artifical and human inteligence. *New York: Elsevier Science Publishers Corporation;*, pages 173–180, 1984.

[31] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

[32] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[34] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[35] Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. Improved zero-shot neural machine translation via ignoring spurious correlations. *arXiv preprint arXiv:1906.01181*, 2019.

[36] Lifeng Han. Machine translation evaluation resources and methods: A survey. *arXiv preprint arXiv:1605.04515*, 2016.

[37] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[38] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] Clare Janaki Holden and Ruth Mace. Spread of cattle led to the loss of matrilineal descent in africa: a coevolutionary analysis. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1532):2425–2433, 2003.

[41] Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[42] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[43] John Hutchins and Evgenii Lovtskii. Petr petrovich troyanskii (1894–1950): A forgotten pioneer of mechanical translation. *Machine translation*, 15(3):187–221, 2000.

[44] William John Hutchins and Harold L Somers. *An introduction to machine translation*, volume 362. Academic Press London, 1992.

[45] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

[46] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350, 2015.

[47] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

[48] Vicky Kalogeiton, Vittorio Ferrari, and Cordelia Schmid. Analysing domain shift factors between videos and images for object detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(11):2327–2334, 2016.

[49] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[50] Philipp Koehn and Kevin Knight. Statistical machine translation, November 24 2009. US Patent 7,624,005.

[51] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872*, 2017.

[52] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[53] Dai Kusumoto and Shinsuke Yuasa. The application of convolutional neural network to stem cell biology. *Inflammation and regeneration*, 39(1):14, 2019.

[54] Antonio-L Lagarda, Vicent Alabau, Francisco Casacuberta, Roberto Silva, and Enrique Díaz-de Liaño. Statistical post-editing of a rule-based machine translation system. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 217–220, 2009.

[55] Surafel M Lakew, Matteo Negri, and Marco Turchi. Low resource neural machine translation: A benchmark for five african languages. *arXiv preprint arXiv:2003.14402*, 2020.

[56] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[58] Gong-Xu Luo, Ya-Ting Yang, Rui Dong, Yan-Hong Chen, and Wen-Bo Zhang. A joint back-translation and transfer learning method for low-resource neural machine translation. *Mathematical Problems in Engineering*, 2020, 2020.

[59] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[60] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[61] José B Marino, Rafael E Banchs, Josep M Crego, Adrià de Gispert, Patrik Lambert, José AR Fonollosa, and Marta R Costa-jussà. N-gram-based machine translation. *Computational linguistics*, 32(4):527–549, 2006.

[62] Laura Martinus and Jade Z Abbott. Benchmarking neural machine translation for southern african languages. *arXiv preprint arXiv:1906.10511*, 2019.

[63] Laura Martinus and Jade Z. Abbott. A focus on neural machine translation for african languages, 2019.

[64] Laura Martinus, Jason Webster, Joanne Moonsamy, Moses Shaba Jnr, Ridha Moosa, and Robert Fairon. Neural machine translation for south africa's official languages. *arXiv preprint arXiv:2005.06609*, 2020.

[65] Cindy A McKellar. An english to xitsonga statistical machine translation system for the government domain. In *Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium*, pages 229–233, 2014.

[66] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[67] Graham Neubig. Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*, 2017.

[68] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, USA:, 2015.

[69] Emilio Soria Olivas, Jos David Mart Guerrero, Marcelino Martinez-Sober, Jose Rafael Magdalena-Benedito, L Serrano, et al. *Handbook of Research on Machine*

Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques. IGI Global, 2009.

[70] Iroro Orife, Julia Kreutzer, Blessing Sibanda, Daniel Whitenack, Kathleen Siminyu, Laura Martinus, Jamiil Toure Ali, Jade Abbott, Vukosi Marivate, Salomon Kabongo, Musie Meressa, Espoir Murhabazi, Orevaoghene Ahia, Elan van Biljon, Arshath Ramkilowan, Adewale Akinfaderin, Alp Öktem, Wole Akin, Ghollah Kioko, Kevin Degila, Herman Kamper, Bonaventure Dossou, Chris Emezue, Kelechi Ogueji, and Abdallah Bashir. Masakhane – machine translation for africa, 2020.

[71] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[72] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[73] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[74] Fernando J Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4(3):216–245, 1988.

[75] Richard H Richens. Interlingual machine translation. *The Computer Journal*, 1(3):144–147, 1958.

[76] Matīss Rikters and Mārcis Pinnis. Debugging translations of transformer-based neural machine translation systems. *Baltic Journal of Modern Computing*, 6(4):403–417, 2018.

[77] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[78] Ravi K Samala, Heang-Ping Chan, Lubomir Hadjiiski, and Sathvik Koneru. Hazards of data leakage in machine learning: a study on classification of breast cancer using deep neural networks. In *Medical Imaging 2020: Computer-Aided Diagnosis*, volume 11314, page 1131416. International Society for Optics and Photonics, 2020.

[79] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[80] Nicholas Spaull. South africa's education crisis: The quality of education in south africa 1994-2011. *Johannesburg: Centre for Development and Enterprise*, pages 1–65, 2013.

[81] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[82] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[83] Felix Stahlberg. Neural machine translation: A review, 2019.

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

[85] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[86] Daniel R van Niekerk. Exploring unsupervised word segmentation for machine translation in the south african context. In *Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium*, pages 202–206, 2014.

[87] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[88] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics, 1996.

[89] Michael S Waterman, Temple F Smith, and William A Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.

[90] Warren Weaver. Translation. *Machine translation of languages*, 14:15–23, 1955.

[91] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[92] Ilana Wilken, Marissa Griesel, and Cindy McKellar. Developing and improving a statistical machine translation system for english to setswana: a linguistically-motivated approach. In *Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa*, page 114, 2012.

[93] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[94] Friedel Wolff and Gideon Kotzé. Experiments with syllable-based zulu-english machine translation. In *Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium*, pages 217–222, 2014.

[95] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2251–2265, 2018.

[96] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[97] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[98] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[99] David Zipser and Richard A Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679–684, 1988.