Universität Zürich[UZH]

Masterarbeit
zur Erlangung des akademischen Grades
Master of Arts
der Philosophischen Fakultät der Universität Zürich

Normalization of Swiss German WhatsApp Messages with Statistical and Neural Sequence-to-Sequence Methods

Verfasser: Massimo Lusetti
Matrikel-Nr: 15-706-435

Referent: Prof. Dr. Martin Volk

Betreuerin: Dr. Tanja Samardžić

Institut für Computerlinguistik

Abgabedatum: 15.06.2018

Abstract

Natural language processing (NLP) tools and applications require standard text as input, which makes the task of text normalization a fundamental step, in that it converts non-canonical text into a standardized form that can be processed in a NLP pipeline. Examples of non-canonical text are historical texts (dating back to periods with varying spelling conventions), texts written in dialects or languages without an orthographic standard, and texts characterized by various irregularities, such as computer-mediated communication (CMC). Swiss German WhatsApp messages are an expression of two of these domains, for being an instance of CMC written in a dialect with no orthographic standard. I propose a method for automatic text normalization based on a neural sequence-to-sequence architecture and I compare its performance with the current state-of-the-art statistical systems. In machine translation tasks, neural frameworks have replaced traditional statistical methods. However, in text normalization they still lag behind. The primary reason for this is that the strength of neural networks emerges when a large amount of training data is available. In text normalization, unlike machine translation, parallel corpora are created by experts mainly for research purposes and are of limited size, due to the fact that manual normalization is expensive and time-consuming. The method I propose, which has already been applied to other tasks such as morphological segmentation, integrates additional word-level language models into a basic character-level sequence-to-sequence neural framework. The results show that this approach outperforms the statistical system and open new promising scenarios in neural text normalization, thanks to the flexibility of neural frameworks in combining components, such as language models trained on different tokenization levels, that can help achieve and improve state-of-the-art performance when little training data is available.

Zusammenfassung

Werkzeuge und Anwendungen der Maschinellen Sprachverarbeitung (Natural Language Processing - NLP) benötigen Standardtext als Eingabe. Das Verfahren der Textnormalisierung spielt daher eine wichtige Rolle, um nicht-kanonischen Text in eine standardisierte Form umzuwandeln, der in einer NLP-Pipeline verarbeitet werden kann. Beispiele für nicht-kanonischen Text sind historische Texte (aus Epochen mit unterschiedlichen Rechtschreibungen), Texte in Dialekten oder Sprachen ohne orthographischen Standard, und Texte, die von verschiedenen Unregelmässigkeiten gekennzeichnet sind, wie z.B. Computervermittelte Kommunikation (computer-mediated communication - CMC). Schweizerdeutsche WhatsApp-Nachrichten sind ein Ausdruck zwei dieser Domänen, da sie eine Instanz von CMC, und in einem Dialekt ohne orthographischen Standard geschrieben sind. In dieser Arbeit stelle ich eine Methode zur automatischen Textnormalisierung vor, die auf einer neuronalen Sequence-to-Sequence Architektur basiert, und vergleiche deren Leistung mit statistischen Systemen, die den aktuellen Stand der Technik darstellen. Obwohl bei der maschinellen Übersetzung neuronale Netzwerke traditionelle statistische Methoden ersetzt haben, gilt dies nicht für die Textnormalisierung. Der Hauptgrund dafür ist, dass die Stärke neuronaler Netzwerke erst dann entsteht, wenn eine grosse Menge an Trainingsdaten verfügbar ist. Allerdings werden bei der Textnormalisierung parallele Korpora von Experten zu Forschungszwecken erstellt und sind aufgrund der Tatsache, dass die manuelle Normalisierung teuer und zeitaufwendig ist, von begrenzter Grösse. Die von mir vorgestellte Methode, die bereits in anderen Bereichen wie die morphologische Segmentierung angewendet wurde, integriert zusätzliche Sprachmodelle auf Wort-Ebene in ein grundlegendes neuronales Framework, das auf Zeichen-Ebene wirkt. Diese Methode übertrifft den statistischen Ansatz. Die Ergebnisse eröffnen neue vielversprechende Szenarien in der neuronalen Textnormalisierung, indem die Flexibilität neuronaler Frameworks genutzt wird, zusätzliche Komponenten zu kombinieren, z. B. auf verschiedenen Tokenisierungsstufen trainierte Sprachmodellen. Das trägt dazu bei, Leistung nach dem Stand der Technik zu erreichen und zu übertreffen, auch wenn wenig Trainingsdaten verfügbar sind.

Acknowledgement

I would like to express my sincere gratitude to all the people who helped and supported me in this project. First, I would like to thank my supervisor, Dr. Tanja Samardžić, for her valuable and patient advice and guidance, for her enlightening courses and for providing me with the idea for the topic of this thesis. My thanks also go to Prof. Dr. Martin Volk, who has always been a source of inspiration and motivation during my studies in Multilingual Text Analysis.

I would like to thank the following people at the University of Zürich, whose help has been of the utmost importance: Prof. Dr. Elisabeth Stark, for making the corpora available and for allowing me to carry out this work in the context of a research project at the department of Romance Studies; Anne Göhring and Simone Ueberwasser, for the help in getting access to the corpora; and Tatyana Ruzsics, for the expertise and technical skills that she shared with me in the experimental phase.

I am also grateful to my parents, for their unconditional support in my quest for knowledge since I was a child, and to Enrico, for constantly reminding me, especially when I am on the verge of losing faith, that nothing is impossible. Special thanks go to Tanja G., for putting up with me during all my studies, and for encouraging me to believe in my potential and pursue my dreams.

Contents

List of Figures

List of Tables

List of Acronyms

CMC	Computer-Mediated Communication
CNN	Convolutional Neural Network
CSMT	Character-Level Statistical Machine Translation
ED	Encoder-Decoder
GRU	Gated Recurrent Unit
LM	Language Model
LSTM	Long-Short Term Memory
MT	Machine Translation
NER	Named Entity Recognition
NLP	Natural Language Processing
NMT	Neural Machine Translation
OOV	Out Of Vocabulary
PBSMT	Phrase-Based Statistical Machine Translation
POS	Part Of Speech
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SMT	Statistical Machine Translation
WSMT	Word-Level Statistical Machine Translation

1 Introduction

1.1 Motivation

The main goal of natural language processing (NLP) is to develop formal models and algorithms that enable a computer to interpret, process and generate natural language. Part-of-speech tagging, sentiment analysis, information extraction, speech recognition, machine translation, syntactic parsing and dialogue systems are only a few of the many fields of application of NLP. Some of these tasks can be part of the same sequential process, and they are usually organized in a structured pipeline in which each step is a prerequisite to carry out the next one.

When the texts to be processed are characterized by non-standard language, text normalization becomes necessary to make them suitable for subsequent NLP applications. Language can be non-standard, for example, when written texts do not obey specific orthographic conventions (e.g., dialects), or when they display various types of irregularities, such as in computer-mediated communication (CMC). In this thesis I describe a process for text normalization of WhatsApp messages written in Swiss German. This term denotes a family of dialects spoken in the German-speaking part of Switzerland.

Several factors contribute to making this task particularly challenging and, at the same time, fascinating. The first factor has already the traits of a paradox: people write Swiss German although it is not meant to be written. Since Swiss German dialects do not have a standardized orthography, their speakers have traditionally used Standard German in written contexts, with the exception of children's and popular literature. However, the rise and spread of CMC, which shares properties of the written and spoken medium, has produced an interesting phenomenon that cannot be ignored by the NLP community: people write CMC in Swiss German. Here, two complications arise: firstly, as already mentioned, due to the lack of a writing system, people write words as they assume they should be written. Secondly, Swiss German is not one single dialect, but rather a family of dialects. Both these complications result in a high degree of variation. The more variation we have, the

harder it is for computers to make sense of human language. For example, we can imagine a machine translation task in which a computer has to translate text from Swiss German into English. Machine translation is itself a challenging task, but it becomes even trickier if the same source word is spelled in many different ways in a text. It would be certainly desirable to be able to map all the possible variants of a word to a form, maybe even a constructed one, that can be considered the representative of them all. This is exactly what text normalization aims at.

The second factor is that Swiss German is a low-resource language, which means that we do not have much material from which a computer can learn how to process it, and not many computational tools have been developed for this purpose. In our digital society, there exists a risk that low-resource languages are at a disadvantage in being analyzed, preserved and be made understandable by computers. Despite the lack of written resources, Swiss German is by no means an endangered language, by virtue of the high number of speakers and of the important status that it enjoys in the German-speaking part of Switzerland. However, it remains a low-resource language, and I hope that this work can represent a contribution toward a stronger position of Swiss German in the world of NLP.

1.2 Task, Challenges and Goals

Normalization is typically viewed as a machine translation task, where the source language is Swiss German and the target language is its normalized form. Within this project, this task is carried out with statistical machine translation (SMT) and neural sequence-to-sequence methods. Both approaches require a parallel corpus of manually normalized texts. The corpus is used to train and evaluate a model that can learn from the data how to automatically normalize previously unseen texts.

Statistical methods are being gradually abandoned in machine translation. The recently introduced neural methods achieve much better performance, providing at the same time a more flexible framework for designing and testing different models. They, however, require large training sets. Since machine translation (MT) enjoys widespread and consolidated applications for cultural, commercial and diplomatic purposes, large corpora for training MT systems are freely available. By contrast, in the field of text normalization, data sets are of limited size and are created by experts specifically for the task. Several attempts have been made to train neural normalization models, but the resulting systems could not reach the performance of SMT.

The process of normalization is affected by the ambiguity that is inherent in language, for example when there are many possible normalization forms for a given source word. In such cases, the context can help in disambiguating the meaning of a word. The Swiss German word *bi* corresponds to the normalized forms *bei* ('by') and *bin* ('am'). A model that does not use the context might have no other choice but to select the form that is most common in the training set, which inevitably results in errors. If the context is available, it is easy for the model to select the form *bin* if, for example, the previous word is *ich* ('I').

With respect to WhatsApp messages written in Swiss German, normalization is necessary because of the high degree of variation that characterizes them. This variation is due to multiple factors. Firstly, due to the lack of an orthographic standard, texts written in Swiss German are characterized by regional, inter-speaker and intra-speaker variation. As a consequence, the same word may be written in different ways by speakers of different dialects, but also by two speakers of the same dialect and even by the same speaker within the same text. For example, the normalized form *viel* ('much') is associated throughout the corpus used in this project (see Section 5.1) to many source words, which all have the same meaning but are spelled differently, such as *viel*, *viil*, *vill* and *viu*.

Secondly, it must be pointed out that the model has to deal with a number of peculiarities which are due to the very nature of WhatsApp messages (e.g., abbreviations, slang, typos, character reduplication and other irregularities) and which increase the degree of variation of the source language. For example, the source text may contain "misspelled" words such as *shriibe* ('write'). Although Swiss German does not have an orthographic standard, which means that there are no fixed rules that prescribe how a given word should be written, *schriibe* is the word that Swiss German speakers would normally use to convey the normalized form *schreiben*. The form *shriibe* is thus most probably the result of a typo rather than of an arbitrary choice. An ideal model should be capable of learning that both the misspelled and the expected form map to the same normalized form. However, irregularities like typos do not obey systematic patterns so that, in practice, learning such mapping poses a challenge that causes the model to have difficulty converting the character sequence *sh* into *sch* and thus to output a wrong normalization.

1.3 Research Question

Text normalization is commonly addressed as a machine translation task. Until the recent introduction of neural network frameworks, statistical systems have been for

a long time the traditional approach to machine translation. Many MT services available on-line, such as Google Translate, have switched to neural machine translation (NMT), which is gradually becoming the dominant paradigm. However, NMT is still struggling to achieve state-of-the-art results in text normalization. One of the main reasons for this is that the corpora available for training and evaluating text normalization systems are relatively small. The research question that shall be answered in this thesis is thus the following:

Is it possible to develop neural models that, despite relying on small data sets, are able to achieve state-of-the-art performance in the task of automatic text normalization?

In order to address the research question, I will run experiments with both SMT and NMT systems. I will use a method, which is innovative in text normalization, aimed at improving the performance of neural sequence-to-sequence models by augmenting them with additional integrated components.

1.4 Thesis Structure

Chapter 2 provides a factual background for the task of text normalization of WhatsApp messages written in Swiss German. First I describe the Swiss German dialects, their main features, such as the lack of a standardized spelling, and their differences with Standard German. I then briefly touch on the peculiarities of computer-mediated communication, and describe the task of text normalization as a conversion from non-canonical language to a representation that is suitable for natural language processing tools.

In Chapter 3, I provide a theoretical and technical background and introduce the two approaches adopted (statistical and neural machine translation) with focus on character-level methods, and I describe the evaluation metrics used to assess the performance of the two models.

Chapter 4 presents an overview of the related work on the topics that are linked to the task of text normalization with statistical and neural methods.

In Chapter 5, I first describe the corpora used as data sets and then provide an overview of the tools, resources and computational frameworks on which this work is built: the Moses toolkit for phrase-based statistical machine translation; the Python programming language; the library Theano, that provides support for linear algebra operations; Blocks, a framework that helps build neural networks on top of Theano;

and SGNMT, which builds on all the previous frameworks, with focus on neural machine translation and sequence tasks.

In Chapter 6, after presenting the methods adopted to establish a baseline (a minimum threshold for the performance, that can be achieved with simple methods) and a ceiling (the maximum level of performance that can be expected), the two approaches of SMT and NMT are compared and their implementations are analyzed in detail, with particular focus on the chosen architectures, settings and parameters.

Chapter 7 presents the results of the two approaches as well as an error analysis aimed at identifying common weaknesses, possible improvements and differences in the two models.

In Chapter 8, I summarize the findings and formulate the answer to the research question of this thesis.

2 Factual Background

In Chapter 1, I have introduced text normalization as a task that aims at converting non-canonical text into a standard form, suitable for NLP tasks. In this chapter I describe the factors that, due to the variation that they cause, make WhatsApp messages written in Swiss German non-canonical text. I then provide an overview of the task of text normalization, with details about its challenges and applications.

2.1 The Swiss German Dialects

In this section I describe one important factor that contributes to the variation that characterizes written Swiss German, namely its lack of an orthographic standard. Switzerland has four national languages: German, French, Italian and Romansh. This situation, in which multiple languages are spoken in the same community, is known as multilingualism. German is spoken by 63% of the Swiss population. The German-speaking part of Switzerland is characterized by a phenomenon known as diglossia, i.e. two different varieties of the same language are used within a community in different social situations. Diglossia is described by Ferguson (1959) as follows:

> Diglossia is a relatively stable language situation in which, in addition to the primary dialects of the language (which may include a standard or regional standards), there is a very divergent, highly codified (often grammatically more complex) superposed variety, the vehicle of a large and respected body of written literature, either of an earlier period or in another speech community, which is learned largely by formal education and is used for most written and formal spoken purposes but is not used by any section of the community for ordinary conversation.

In the German-speaking part of Switzerland, the superposed variety is known as Standard Swiss German, that is the variety of Standard German that is accepted as the norm in Switzerland. It is used in most written contexts (literature, newspapers, private correspondence, official documents), in formal and official spoken contexts

(education, parliament speeches, news reports in radio and television programs) and in interactions with foreigners. Differences between Standard Swiss German and Standard German can be observed in the orthography (e.g., Standard German *ß* vs. Standard Swiss German *ss*), phonology, morphology and, most notably, vocabulary (e.g., Standard German *Fahrrad* vs. Standard Swiss German *Velo* ('bycicle')). The other variety, that is the dialect, is known as Swiss German and is used in everyday life, within the family as well as in most radio and television programs, except news reports. Since Swiss German does not have a standardized orthography,[1] it is rarely used in written contexts. However, nowadays we observe an increasing use of the dialect in written computer-mediated communication (CMC). This phenomenon has multiple and interesting repercussions, since it makes valuable material available for NLP tasks, thus granting Swiss German a better position among the languages studied in the NLP community. However, given the high degree of variation described in Section 1.2, the need for text normalization becomes immediately evident.

It must be pointed out that the definition Swiss German does not refer to one single and uniform dialect, but rather to a family of mutually intelligible dialects belonging to the Alemannic group of German dialects. The main linguistic division within Swiss German is into Low, High and Highest Alemannic dialects, with further more fine-grained divisions within each group. According to Rash [1998], since the first half of the 20th century, proposals to introduce a national dialect have unleashed a heated debate. The detractors of such proposals argue that a national dialect would be an artificial imposition that clashes with the multilingual and plural identity of Switzerland, while its supporters believe that it would strengthen the status of the dialect. However, even among this latter group there are disagreements, mainly due to the criteria to be adopted in creating such national dialect. While some advocate a mixed dialect that draws from all the already existing dialects, others believe that one of these should be elevated to represent them all, a scenario that is itself highly controversial, since the choice of one dialect would hardly be accepted by speakers of the others.

Swiss German is commonly regarded as archaic for not having undergone many changes that have affected other German dialects and Standard German since the Middle Ages. Rash [1998] takes "the shared characteristics of the High Alemannic dialects to represent a type of norm for Swiss German", and gives an exhaustive list of features that differ from Standard German. Such differences are important as the normalized form of Swiss German, in many cases, corresponds to Standard

[1]Eugen Dieth's orthographic handbook for Swiss German [Dieth, 1938] is a set of guidelines that are mainly used by experts and linguists, but hardly followed by speakers when they use Swiss German in written contexts.

German (see Section 2.3). Features affecting the phonology, which are reflected in the orthography and are responsible for significant intra-word changes, play a very important role with respect to text normalization. Table 1 shows a selection of such features.

SG feature	**StdG feature**	**Example (SG / StdG)**	**English gloss**
long vowel	short vowel	praacht / brachte	brought
long monophthong	diphthong	Huus / Haus	house
diphthong	long monophthong	Brüeder / Brüder	brothers
MHD *u* preserved	MHD *u* → *o*	Sunne / Sonne	sun
MHD *i* rounded	-	wüsse / wissen	know
vowel lengthening	-	Doorff / Dorf	village
syncope	-	Gsellschaft / Gesellschaft	society
apocope	-	Straass / Straße	street
scht / *schp*	*st* / *sp*	Poscht / Post	post (noun)
loss of final *n*	-	mache / machen	make
intrusive *n*	-	früener / früher	earlier

Table 1: Main phonological differences between Swiss German (SG) and Standard German (StdG), which are reflected in the orthography in those cases in which Swiss German is written. MHD: Middle High German.

Further differences can be observed at the morphological, syntactic, semantic and lexical level. In particular, a typical morphological feature of Swiss German is the merging of pronouns with the preceding verb (e.g., Swiss German *hani*, Standard German *habe ich* 'have I', a construction in which the intrusive *n* is used when two vowels become in contact). Moreover, lexical differences can result in non-cognate words that may pose a challenge to a machine translation system that operates at the character level, which, as we will see in the next chapters, is the case in this work.

2.2 Computer-Mediated Communication

A second important factor that increases the degree of variation of texts written in Swiss German manifests itself when such texts represent instances of computer-mediated communication (CMC). CMC is defined as any human communication that relies on electronic devices for conveying a message. The term refers to those forms of communication that occur via computer-mediated formats, e.g. instant messaging, email, chat rooms, online forums and social network services, but it has

also been applied to other forms of text-based interaction such as text messaging. CMC represents a hybrid form of communication that employs features of the written mode (e.g., the visual channel, persistence, and the possibility to plan, revise and edit a message) and the spoken mode, with varying degrees of synchronicity, interaction and time pressure. CMC is characterized by distinct linguistic features, whose frequency and type may vary according to the specific communication format. Typical CMC features affect the orthography and consist in omitted or unconventional punctuation, arbitrary use of lowercase and uppercase characters, and spelling irregularities: typos, unconventional spelling ('Are you there?' vs. 'R u there?'), vowel reduplication ('Coooool' vs. 'Cool'), omitted vowels ('lt dnr' vs. 'late dinner'). Further features are the frequent use of abbreviations, emoticons, emojis and other symbols. The boundaries between these feature types are not always clear-cut, and in some cases (e.g., the use of 'their' rather than 'they're') it is impossible to establish if writers are using an abbreviated form or just making a mistake. Section 5.1 gives an overview of the typical CMC features with respect to Swiss German.

I have described two factors for which WhatsApp messages written in Swiss German are characterized by non-standard language: the lack of a standardized orthography and the peculiarities typical of CMC. Since NLP tools and applications require standard language as input, the task of text normalization becomes very important. An introduction to text normalization is given in the next section.

2.3 Text Normalization

Text normalization aims at converting texts written in non-canonical language, such as historical texts, dialect and CMC, into a standardized representation that enables subsequent employment along the NLP pipeline. Historical texts can be considered non-canonical for obeying different spelling conventions if compared to the variety of a specific language currently in use. For example, in ancient texts the German words *Mitteilung* ('message') and *Kenntnis* ('knowledge') can be found in the forms *Mittheilung* and *Kenntniß*. Moreover, texts might date back to a period in which orthography was not standardized, so that a word might appear written in different forms in the same document, depending on dialect influences, personal preferences and arbitrary choices of a specific writer. When a dialect is written, in the absence of a standardized orthography writers may have no other option but to write a word as they believe it should be written, which inevitably results in a high degree of arbitrariness and variation. As mentioned in Section 2.2, CMC is characterized by various idiosyncrasies and irregularities that make normalization necessary for

further NLP tasks. Since the goal of this work is to normalize WhatsApp messages written in Swiss German, the normalization task that I describe combines two fields of application of text normalization, that is dialect and CMC normalization.

With respect to Swiss German, in many cases words are normalized by mapping them to their Standard German translation. In some cases, words are converted into a constructed form that can only be interpreted by experts and linguists, but better expresses the etymology of the dialect word. One of the reasons for this choice is due to lexical mismatches, i.e. Swiss German words that are dissimilar from their Standard German translation for having a different etymological origin. For example, it must be decided whether the Swiss German word *luege* ('to look') should be normalized as *schauen*, which is its Standard German translation, or as the constructed form *lugen*. This form, of unclear origin, though probably related to English 'to look', is documented in Middle High German (*lūgen*, *luogen*) and in Old High German (*luogēn*), and is nowadays perceived as obsolete in Standard German.[2] A similar challenge is posed, for example, by the Swiss German and Standard German pairs *öpper* and *jemand* ('someone'), *gheie* and *fallen* ('to fall'), *gäbig* and *praktisch* ('useful'). In some cases of lexical mismatches, the origin of the dialect word is obscure and cannot be clearly traced back to an earlier stage of the German language, thus making the constructed form not easily interpretable by non-experts.

In this chapter I have described the features that characterize WhatsApp messages written in Swiss German and the challenges posed by the lack of a standardized orthography and by the nature of written CMC. In Chapter 3, I provide a theoretical and technical background of the two methods used to carry out the task of automatic text normalization: statistical machine translation and neural machine translation.

[2] `https://www.duden.de/rechtschreibung/lugen`

3 Technical Background

In this chapter I review the basic principles that underlie the two methods applied in my project to automatically normalize Swiss German WhatsApp messages. I have already mentioned that this task is addressed as a machine translation task. Both methods (statistical machine translation and neural machine translation) require a parallel corpus of manually normalized text to train a model that can then be used to normalize previously unseen texts. In our case, the source language is Swiss German, and the target language is its normalized form. The ability of a computer program to use data to acquire experience and knowledge that can be applied to perform a specific task on new data is known as machine learning.

3.1 Statistical Machine Translation

The core idea behind SMT is the noisy-channel principle, where two basic components are combined: the *translation model* $p(f|e)$, responsible for the adequacy of the translation from source to target sentence, and the *language model* $p(e)$, responsible for the fluency of a sentence in the target language.[1] Equation 3.1 shows how the Bayes rule is used to combine the two models to find the best translation e^*, among all target sentences E, given a source sentence f.

$$e^* = \underset{e \in E}{\operatorname{argmax}}\, p(e|f) = \underset{e \in E}{\operatorname{argmax}}\, p(e)p(f|e) \tag{3.1}$$

The translation model is trained on bilingual corpora, whereas the language model is trained on monolingual corpora in the target language.

[1] In the sections where neural machine translation is described, X refers to the input sequence and Y to the output sequence. Here I follow SMT convention and use e to refer to the target sequence and f to the source sequence.

3.1.1 Translation Model

A pre-requisite in a typical SMT setting is that the source and target texts are aligned at the sentence level. The first step in training the translation model is then automatic word alignment for each pair of source and target sentences. We will see in 3.1.4 that in the character-level approach adopted throughout this work, the initial alignment is at the word level and automatic alignment is then performed on characters of each source and target word pair. Automatic word alignment originated with the development of the IBM models, based on the expectation-maximization algorithm [Brown et al., 1993]. Koehn et al. [2003] improved the mono-directional IBM alignments, which allow a target word to be aligned with at most one source word, with a heuristic method based on a bidirectional alignment. In our case, Swiss German is aligned with the normalization form and vice versa. The two resulting word alignments must then be merged. One possible method is to use the intersection of the two alignments. This method is characterized by high-confidence alignment points: the identified alignments are very likely to be correct, though the drawback is that some correct alignments may be left out. An alternative would be to use the union of the two alignments. In this case, there is a high likelihood that most of the correct alignments are captured, though this comes at the cost of some faulty alignment points. To use concepts from information retrieval, we say that intersection has a high precision, and union has a high recall. The most common approaches use heuristic methods to explore the space between intersection and union.

To achieve better context-sensitive source-target mappings, traditional SMT systems rely on phrase-level translation models [Koehn et al., 2003]. These models allow to build a phrase table to store aligned phrase pairs, in the source and target language, that are consistent with the single-word alignments established by the IBM models with the expectation-maximization (EM) algorithm. Phrases are word sequences that are not necessarily linguistically motivated.

The phrase-based approach has several advantages if compared to models that use single words as fundamental translation units: in addition to help resolving ambiguities, it allows many-to-one and one-to-many mappings and to memorize long phrases from a training corpus, that can then be used as building blocks for the target sentence. In phrase-based models, the translation model in Equation 3.1 is decomposed as follows:

$$p(\bar{f}_1^I|\bar{e}_1^I) = \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i) d(start_i - end_{i-1} - 1) \tag{3.2}$$

In Equation 3.2, $\bar{f}_1^I$ and $\bar{e}_1^I$ are sequences of phrases, $\phi(\bar{f}_i|\bar{e}_i)$ is the probability distribution that models phrase translation and $d(start_i - end_{i-1} - 1)$ the probability distribution that models the reordering of the target phrases.

3.1.2 Language Model

The language model measures how likely a sentence is in the target language. It is important since the translation model alone could generate an output sentence that expresses very well the meaning of the source, but does not sound natural and fluent in the target language. Language models are typically based on n-grams, i.e. sequences of n words. The probability of a sequence W of length n can be computed by applying the chain rule of probability:

$$p(w1, w2, ..., w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w1, w2, ..., w_{n-1}) \tag{3.3}$$

However, long sequences of words might not occur in the text at all, with the consequence of zero probability for a given sequence. A solution to this problem, known as data sparseness, is based on the Markov assumption, which states that only a limited number of previous words affect the probability of the next word. Most commonly, trigram language models are used, which consider a two-word history to predict the third word. This requires a monolingual corpus in the target language, for collecting the statistics over sequences of three words.

3.1.3 Decoding

After training a language model based on n-gram occurrences in a corpus, and a translation model based on the phrase-table, a reordering (also called distorsion) model is built to account for different word order in the source and target sentences. The model components (translation, reordering and language model) are combined in a log-linear model, in which each one of them is weighted to scale its contribution to the final translation. The reordering model is ignored when reordering is disabled under the assumption of a monotonic translation, which is the case in my experiments. The term *decoding* denotes the actual translation process. The decoder employs a beam search algorithm. As explained in Koehn et al. [2003], the search

begins in an initial state where no source input words have been translated and no target output words have been generated. New states are created by extending the target output with a phrasal translation that covers a sequence of the source input words not yet translated. The current cost (whereby a high cost corresponds to a low probability) of the new state is the cost of the original state multiplied with the translation, distortion and language model costs of the added phrasal translation. Final states in the search are hypotheses that cover all source words. Among these, the hypothesis with the lowest cost (highest probability) is selected as best translation. The hypotheses are stored in stacks. For each stack, only a beam of the best n hypotheses is kept.

3.1.4 Character-Level Statistical Machine Translation

The experiments that I run in this project rely on a character-level approach. In character-level statistical machine translation (CSMT), we simply replace words with characters as the symbols that make up a phrase. In the language model, n-grams are sequences of n characters rather than words. CSMT is a suitable approach for those tasks in which many word pairs in the source and target languages are formally similar, such as the Swiss German word *Sunne* normalized as *Sonne* ('sun'), or are characterized by regular transformation patterns that are not captured by word-level systems, such as the pattern *ii* → *ei*, which is responsible for the transformations *Ziit* → *Zeit* ('time'), *wiiter* → *weiter* ('further'), *Priis* → *Preis* ('price').

This setting requires to pre-process the parallel corpus by replacing spaces between words with underscores and adding spaces between characters. This converts the corpus alignment unit *hani* ↔ *habe ich* into *h a n i* ↔ *h a b e _ i c h*. As a result, the characters are now the tokens[2] of the alignment units, phrases are sequences of characters and the language model is based on character n-grams. It is important to point out that in CSMT longer sequences have proven to be more effective than trigrams. The CSMT output must then be post-processed, by removing spaces and underscores, before evaluation with a reference.

[2]Tokenization is the process of segmenting a text into smaller units (tokens) that are considered relevant for a specific task. Tokens are usually words, though in character-level tasks they are characters. In English and many other languages, words are often separated from each other by blanks (whitespace), but there are special cases. For example, *Los Angeles* should be treated as an individual word, whereas *you're* consists of the two words *you* and *are*.

3.2 Neural Machine Translation

In the previous sections I have described an approach to machine translation based on statistical methods. In this section I describe the approach based on neural networks. Neural machine translation (NMT) is a recently developed method for machine translation characterized by a different framework and methodological approach if compared to traditional phrase-based SMT systems. The underlying architecture of the NMT system used in this work is based on recurrent neural networks and, in particular, on a sequence-to-sequence model that allows to create a representation of an input sequence by means of an encoder and produce an output sequence by means of a decoder. This section provides an overview of the notions that are relevant for neural network frameworks.

I have briefly mentioned that machine learning means using data in order to allow a computer to learn how to perform a specific task on new data. In many machine learning frameworks, including neural networks, instances of a data set are viewed as vectors in a multi-dimensional space, in which each dimension corresponds to a feature of the instances. Appendixes A.1 and A.2 provide a basic introduction to the linear algebra notions that might be useful in better understanding the neural methods described in this thesis. Appendix A.3 provides an introduction to the logistic regression algorithm, which is an important building block in the architecture of a neural network.

3.2.1 Logistic Regression

In order to output a prediction $\hat{y}$, a logistic regression model combines the parameters $\boldsymbol{w}$ (the vector of the weights of each feature) and b (the intercept) in a linear function as follows:

$$z = \boldsymbol{w}^T \boldsymbol{x} + b \tag{3.4}$$

The output z of the linear function is then fed into a non-linear function (the sigmoid function σ) to produce probability values between 0 and 1. The cross-entropy loss function is commonly used to measure how good the prediction is with reference to the true label y:

$$\mathcal{L}(\hat{y}, y) = -(ylog(\hat{y}) + (1-y)log(1-\hat{y})) \tag{3.5}$$

If i represents a single training example and m is their total number, it is then possible to compute the cost function J of the parameters w and b, referred to the entire training set:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{3.6}$$

The goal is then to find the parameters w and b that minimize the cost function J. This process, known as optimization, is achieved by employing the gradient descent algorithm. Equation 3.7 shows how the parameters θ (w and b) are updated at each iteration in this search for the minimum value of J:

$$\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \tag{3.7}$$

The symbol := indicates that θ is updated with the right part of the equation, whereas α denotes the learning rate. The process of computing the loss function $\mathcal{L}$ is called forward propagation, whereas back-propagation denotes the computation of the derivatives that are then used to update the parameters.

3.2.2 Feed-Forward Neural Networks

In Section 3.2.1 I have introduced the logistic regression algorithm. The basic architecture of a feed-forward neural network consists in repeating logistic regression multiple times, where each repetition represents a hidden layer of the network. These models are called feed-forward because information flows from an input x through the intermediate computations used to define a function f, to the output $\hat{y}$. When this type of architecture was first introduced, it was believed that it had a clear resemblance to the way the human brain works, hence the name neural networks. Although nowadays this analogy is regarded as being a bit contrived, the name has stuck and is commonly used both in the scientific community and in popular culture. The initial features constitute the input layer of the neural network, the output of the first hidden layer (defined as $a^{[1]}$) is used as input for the second hidden layer and so on, until the final layer outputs the predicted value. This operation of stacking together multiple layers may result in networks that are characterized by a deep and intricate structure. For this reason, the term deep learning is commonly associated with neural networks.

Figure 1 shows an example of a 2-layer neural network with three input features,

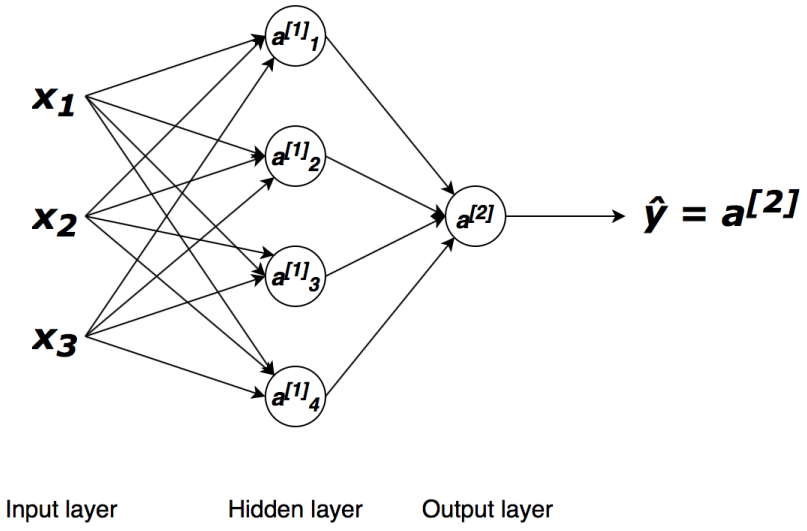

Figure 1: Example of a feed-forward neural network with three input features and one hidden layer that consists of four hidden units. The superscript and the subscript values of a denote the hidden layer and the hidden unit respectively.

one hidden layer consisting of four hidden units, and an output layer. The units of the hidden layer act in parallel, each representing a vector-to-scalar function. Each hidden unit in the hidden layer uses features x_1, x_2 and x_3, together with the weight vector $\boldsymbol{w}^{[1]}$ and the intercept $b^{[1]}$ (which in a neural network framework is called the bias) to compute $z^{[1]}$, where the superscript value 1 indicates the first layer. Equation 3.8 shows this operation for all the units of the hidden layer, so that the four z values form the vector $\boldsymbol{z}$:

$$\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]} \tag{3.8}$$

The result is than fed into a non-linear function to compute the vector $\boldsymbol{a}^{[1]}$:

$$\boldsymbol{a}^{[1]} = \sigma(\boldsymbol{z}^{[1]}) \tag{3.9}$$

Each value in this vector would be the final output $\hat{y}$ in logistic regression performed by each hidden unit. However, in a neural network, the process continues and the vector $\boldsymbol{a}^{[1]}$ is used as input for the following layer:

$$z^{[2]} = \boldsymbol{w}^{[2]T}\boldsymbol{a}^{[1]} + b^{[2]} \tag{3.10}$$

Note that z is here a scalar instead of a vector, since we have only one hidden unit. At this stage, there could be further hidden layers that would make the network even deeper. However, since this example describes a 2-layer network, the sigmoid function is here employed to compute the final predicted value:

$$a^{[2]} = \hat{y} = \sigma(z^{[2]}) \tag{3.11}$$

The output $\hat{y}$ can then be used to compute the cost function, so that back-propagation can be applied to find the derivatives, which will be in turn used in the gradient descent algorithm to update the parameters. The whole process is then repeated in further iterations until convergence. In order to extend the previous equations to a network with L layers, we can rewrite them in their general forms as follows:

$$\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]} \tag{3.12}$$

$$\boldsymbol{a}^{[l]} = \sigma(\boldsymbol{z}^{[l]}) \tag{3.13}$$

It is worth pointing out that the sigmoid function lends itself to be used in the output layer of a neural network, since it converts the result of a linear function into a value between 0 and 1, which represents a probability and is suitable as value for the prediction $\hat{y}$ in binary classification (Equation 3.11). However, other functions are commonly used in the hidden layers. One of them is the hyperbolic tangent function (tanh), whose formula is:

$$a = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.14}$$

The tanh function, illustrated in Figure 2, can have the disadvantage that, for large values of z, the slope of the curve is close to 0, which corresponds to a low value of the derivative. As a consequence, according to Equation 3.7, the parameters undergo only minor updates and learning may become very slow. This drawback can be overcome by employing the rectified linear unit (relu) activation function (Figure 3), whose formula is:

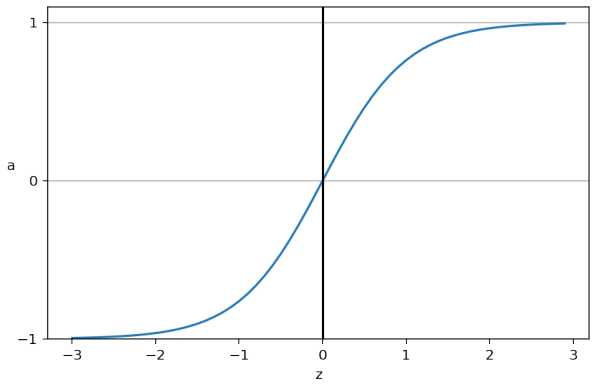

Figure 2: The hyperbolic tangent (tanh) activation function.

$$a = relu(z) = max(0, z) \tag{3.15}$$

In the relu function, z values that are larger than 0 are left unchanged, so that the derivative is equal to 1, whereas for z values that are lower than 0, the curve has no slope and the derivative is 0.[3]

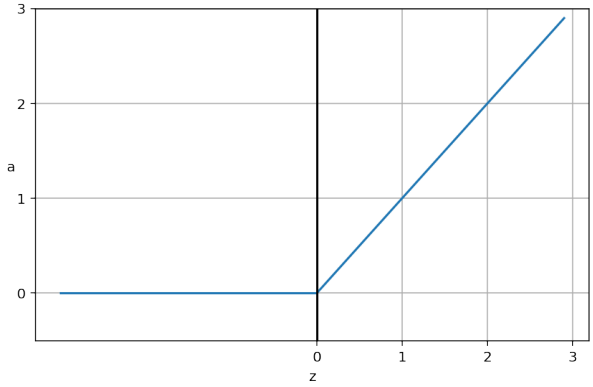

Figure 3: The rectified linear unit (relu) activation function.

The functions introduced above (sigmoid, tanh and relu) are known as activation functions. The need for a non-linear activation function in the hidden layers of a neural network is due to the fact that a repetition of several linear functions, such as the one in Equation 3.8, is itself a linear function, which would reduce the whole neural network to a simple logistic regression and nullify the advantages of having

[3]In theory, in the relu activation function the derivative is undefined when $z = 0$. For practical reasons, in the unlikely event of the value of z being exactly 0, common software implementations make the derivative to be either 0 or 1.

multiple layers. A typical example of a problem that cannot be solved by a linear model is the XOR problem.

As mentioned above, the goal of neural networks is to learn the values of w and b that minimize the cost function J. While w and b represent the parameters of the algorithm, there are also hyper-parameters that have an impact on the overall performance and must therefore be carefully chosen: the learning rate α, the number of iterations, the number of hidden layers and of hidden units per layer, the type of non-linear activation function etc. Though in some cases experience from previous experiments may help to choose the most suitable hyper-parameters, deep learning is usually an iterative process of trial and error, in which several different settings must be tried in order to find the most efficient one.

So far I have described a problem of binary classification, i.e. the system has to predict one of two classes. There are also problems of multi-class classification, in which the goal is to predict one among many classes. In this case, the network shown in Figure 1 would have a number of hidden units in the output layer that is equal to the number of classes C. Each output unit computes the probability of one class i and the output $\hat{\boldsymbol{y}}$ is a C-dimensional vector with each entry $\hat{y}_i = P(y = i|\boldsymbol{x})$. Each component of $\hat{\boldsymbol{y}}$ has a value between 0 and 1, and the entire vector sums to 1 so that it represents a valid probability distribution. After computing $\boldsymbol{z}$ with Equation 3.12, we can use a function, known as softmax function, to convert the set of values in $\boldsymbol{z}$ into such probability distribution:

$$\hat{y}_i = softmax(\boldsymbol{z})_i = \frac{exp(z_i)}{\sum_{j=1}^{c} exp(z_j)} \tag{3.16}$$

In multi-class classification, the loss function expressed in Equation 3.5 can be generalized as follows:

$$\begin{aligned} \mathcal{L}(\hat{y}, y) &= -\sum_{j=1}^{C} y_j log(\hat{y}_j) \\ &= -y_t log(\hat{y}_t) \\ &= -log(\hat{y}_t) \end{aligned} \tag{3.17}$$

Since all values of y_j are equal to 0 except for the one that corresponds to the real class t, which is 1, the right part of Equation 3.17 reduces to $-log(\hat{y}_t)$. It becomes clear how maximizing $\hat{y}_t$ results in a lower loss.

3.2.3 Convolutional Neural Networks

The first neural methods applied to machine translation were based, at least in some of their components, on convolutional neural networks (CNN). These have also proven to be highly effective in computer vision tasks, such as object detection and face recognition. Since the architecture and implementation of CNNs is beyond the scope of this thesis, what follows is only a brief overview, without delving into the details. The main building block of a CNN is the convolution operation, that, if we use the example of a face recognition task, enables the earlier layers of a neural network to detect basic features such as edges, the middle layers to detect more complex features (combinations of edges that form eyes, nose etc.) and the later layers to detect the complete face of a person. The convolution is achieved by applying a filter to the matrix representing an image. Since the filter matrix is smaller than the image matrix, the filter can be pasted on different regions of the image. The results of the element-wise product of the two matrices are then added together to produce a single number. The operation is repeated on all the regions of the image, producing an output matrix that, in practice, works as a feature detector. Then, a bias is added and an activation function is applied to the output matrix. It is possible to apply multiple filters to the same input. Each filter produces a different output matrix, and stacking up all these matrices results in the output of the convolutional layer. In the training phase, a CNN automatically learns the values of its filters. The convolutional layers described above are commonly used in combination with pooling layers, that reduce the size of the representation and help speed up computation: the matrix that results from a convolutional layer is divided into boxes containing multiple elements, and a new reduced matrix is created, where each element is the maximum value of the box (max pooling) or its average value (average pooling).

In addition to machine translation, CNNs have recently been successfully applied to NLP tasks such as learning semantic representations and text classification, where the input is, for example, a sentence or a document instead of an image. Each filter specializes in detecting a specific feature. If the feature occurs somewhere in the input sequence, applying the filter to that part of the sequence will yield a large value. Each row of the input matrix corresponds to the embedding of one element of the sequence, i.e. a word or a character, and the filters slide over rows of the matrix.

3.2.4 Recurrent Neural Networks

This section describes recurrent neural networks (RNN), the neural framework that is most commonly used in NLP and that also underlies the models used in the present work. One of the most interesting features of recurrent neural networks is their ability to process an input sequence and produce an output sequence of variable length. For this reason, they are particularly suitable in tasks such as language modeling, speech recognition and machine translation.

3.2.4.1 Basic Architecture

Figure 4 shows a basic configuration of a RNN that predicts an output at each time step, so that the input and the output sequence are of equal length. Other configurations might have a single output at the end, that summarizes the input sequence in a fixed-size representation, which in turn is used as input for further processing.

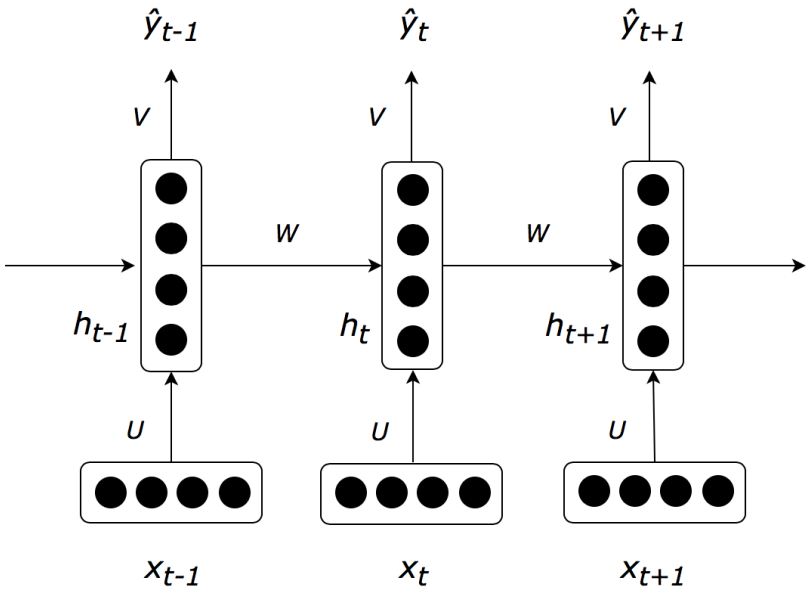

Figure 4: A recurrent neural network that predicts an output at each time step, where x is the input element of a sequence, h the hidden layer, and $\hat{y}$ the output; U, W, and V are the weight matrices for the input-to-hidden, hidden-to-hidden and hidden-to-output connections respectively.

At time step t, $\boldsymbol{h}$ is the hidden layer of the neural network, $\hat{y}$ is the output and $\boldsymbol{x}$ is the vector representing the word (more generally, the element of the sequence) that appears at that time step. The $\boldsymbol{x}$ vectors can be initialized randomly or by using

pre-trained values. In order to output a label, each hidden layer uses information from the input word vector and the previous hidden layer, which can be viewed as a summary of the information coming from the past. While in Figure 4 the output $\hat{y}$ is not fed into the following hidden layer, there are also variants of a RNN that have recurrent connections from the output at one time step to the hidden units at the next time step. The hidden layer is computed as follows:

$$\boldsymbol{h}_t = tanh(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}) \tag{3.18}$$

In Equation 3.18, $\boldsymbol{W}$ and $\boldsymbol{U}$ are the weight matrices that parametrize the hidden-to-hidden and the input-to-hidden vector respectively, and they are the same for each time step. Similarly to multi-class logistic regression, a softmax function is then applied to the output of Equation 3.18 to obtain a probability distribution over all the elements of the vocabulary, which, in a machine translation task, consists of the set of all the words in the target language.

$$\hat{\boldsymbol{y}}_t = softmax(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{c}) \tag{3.19}$$

In Equation 3.19, $\boldsymbol{V}$ is the weight matrix that parametrizes the hidden-to-output vector, and $\boldsymbol{c}$ is a second bias vector used for this particular step. One possible limitation of the architecture described above is that, at each time step, only information from the past is used to produce the output $\hat{y}$. However, there are tasks in which future elements in a sequence may provide useful information. This limitation is overcome by using a bidirectional recurrent neural network (see Section 3.2.4.3).

Similarly to logistic regression and feed-forward neural networks, a loss function is applied to determine how good the prediction $\hat{y}$ is with reference to the true label y. The cost function is then the sum of the losses over all the time steps. Finally, back-propagation is used to compute the gradient and update the parameters in the optimization process.

One of the problems associated with long sequences is that gradients may decrease or increase exponentially as a function of the number of time steps. In the former case (known as the vanishing gradient problem) learning might become extremely slow. In the latter (the exploding gradient problem), the model becomes unstable, resulting in large changes in loss from update to update. In both cases there is the additional risk of reaching NaN values due to numeric underflow or overflow.[4] While

[4]In a computer program, numeric underflow and overflow are conditions in which the result of a calculation is a number of smaller or larger absolute value than the one the computer can

exploding gradients can be prevented by setting an upper threshold to the value of the gradients, the employment of gated units in sequence-to-sequence models, described in Section 3.2.4.2, can help combat the problem of vanishing gradients.

3.2.4.2 Sequence-to-Sequence Models

In Section 3.2.4.1 I have described a RNN that maps an input sequence to an output sequence of equal length. It is possible to combine two RNNs, so that the first (encoder) maps an input sequence to a fixed-sized vector $\boldsymbol{c}$ that summarizes it and the second (decoder) maps this vector to an output sequence. This allows a mapping between input and output sequences of variable length. These models, whose basic architecture is illustrated in Figure 5, were introduced in machine translation by Cho et al. [2014] and Sutskever et al. [2014], who called them encoder-decoder and sequence-to-sequence models respectively. These two definitions will be used interchangeably in this thesis.

Sutskever et al. [2014] based their model on the long-short term memory (LSTM) unit [Hochreiter and Schmidhuber, 1997]. They found that the LSTM learns much better when the source sequences are reversed. They hypothesize that by doing so, the first few words in the source language become very close to the first few words in the target language. The problem's minimal time lag is thus greatly reduced, which has a positive impact on back-propagation and performance.

Cho et al. [2014] introduced a new type of hidden unit called gated recurrent unit (GRU), that consists of a reset gate and an update gate. In the encoder, the reset gate is computed by

$$r_j = \sigma([\boldsymbol{W}_r \boldsymbol{x}]_j + [\boldsymbol{U}_r \boldsymbol{h}_{t-1}]_j) \tag{3.20}$$

and the update gate is computed by

$$z_j = \sigma([\boldsymbol{W}_z \boldsymbol{x}]_j + [\boldsymbol{U}_z \boldsymbol{h}_{t-1}]_j) \tag{3.21}$$

where subscript j denotes the j-th element of a vector. Biases are omitted to make the equations more readable. Due to the use of the sigmoid function, the two gates will have values between 0 and 1. The reset gate r is then used to compute a candidate hidden state $\tilde{h}_j^t$:

actually represent in memory on its CPU. The computer uses NaN (not a number) to replace such values.

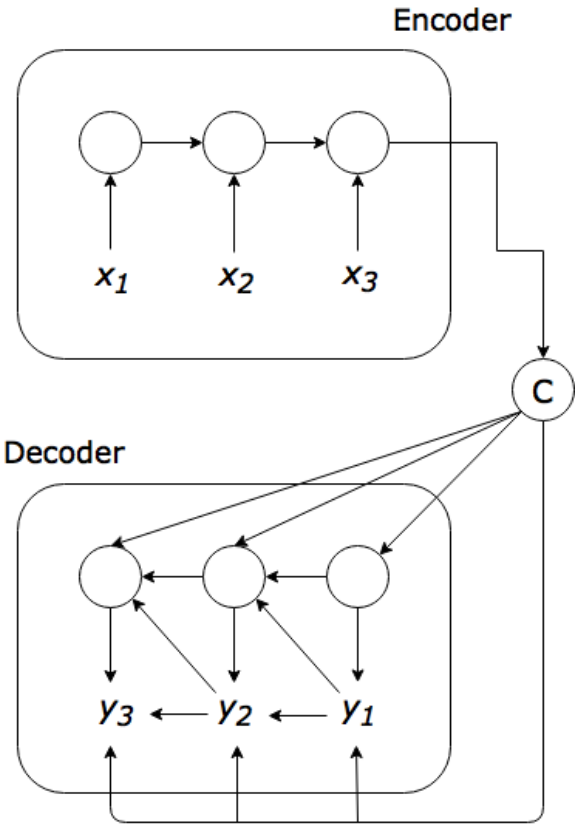

Figure 5: Example of an encoder-decoder or sequence-to-sequence RNN architecture. Though here both the input and the output sequence are of length 3, these models are able to process input and output sequences of variable length.

$$\tilde{h}_j^t = tanh([\boldsymbol{W}\boldsymbol{x}]_j + [\boldsymbol{U}(\boldsymbol{r} \odot \boldsymbol{h}_{t-1}]_j) \tag{3.22}$$

The symbol $\odot$ denotes the element-wise product between two vectors. If the reset gate is close to 0, information from the previous hidden state is ignored, and the only source of information is the current input vector $\boldsymbol{x}$. The final hidden state can be then computed as follows:

$$h_j^t = z_j h_j^{t-1} + (1 - z_j)\tilde{h}_j^t \tag{3.23}$$

The update gate z controls the extent to which the past influences the current time step. When the update gate has values close to 1, the candidate hidden state $\tilde{h}_j^t$ is ignored, which basically corresponds to copying the previous time step. This has

also the advantage of preventing the risk of vanishing gradients. As each hidden unit learns its own reset and update gates, each unit will specialize in capturing either short-term or long-term dependencies.

The equations for the decoder are similar, with the difference that the gates and the hidden states are also a function of the summary vector $\boldsymbol{c}$ (parametrized by a weight matrix $\boldsymbol{C}$), in addition to the previous hidden state and the previous output.

3.2.4.3 Bidirectional Neural Networks

Bidirectional neural networks [Schuster and Paliwal, 1997] have the advantage that information from both the past and the future can be captured at each hidden state $\boldsymbol{h}_t$. The model converts the input sequence into a fixed-dimensional vector representation, with a bidirectional encoder consisting of a forward and a backward RNN. The forward RNN reads the input sequence of embeddings $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_x})$ in forward direction and encodes them into a sequence of vectors representing forward hidden states:

$$\overrightarrow{\boldsymbol{h}}_{\boldsymbol{t}} = f(\overrightarrow{\boldsymbol{h}}_{\boldsymbol{t-1}}, \boldsymbol{x}_t), \quad t = 1, \ldots, n_x \tag{3.24}$$

while the backward RNN reads the sequence in the opposite direction and produces backward hidden states:

$$\overleftarrow{\boldsymbol{h}}_{\boldsymbol{t}} = f(\overleftarrow{\boldsymbol{h}}_{\boldsymbol{t-1}}, \boldsymbol{x}_t), \quad t = n_x, \ldots, 1 \tag{3.25}$$

where f denotes a recurrent unit (such as the GRU). The hidden state $\boldsymbol{h}_t$ for each time step is obtained by concatenating the forward and backward state, so that $\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}}_{\boldsymbol{t}}; \overleftarrow{\boldsymbol{h}}_{\boldsymbol{t}}]$.

3.2.4.4 The Soft Attention Mechanism

Bahdanau et al. [2014] identified a potential limitation in the practice of compressing all the information carried by the input sequence into a single fixed-length vector. They introduced a soft attention mechanism that, at each step during decoding, focuses on the most relevant part of the input. This can be achieved by adaptively selecting the vectors of the input sequence at decoding time.

The decoder RNN transforms the input representation into a variable length output sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n_y})$. At each prediction step t, the decoder reads the previous output $\boldsymbol{y}_{t-1}$ and the current context vector $\boldsymbol{c}_t$ and outputs a hidden state representation $\boldsymbol{s}_t$:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t), \quad t = 1, \ldots, n_y \tag{3.26}$$

Following the notation used by Bahdanau et al. [2014], the hidden state is here denoted by $\boldsymbol{s}_t$. Unlike the traditional GRU architecture, where there is only one vector $\boldsymbol{c}$, here the context vector $\boldsymbol{c}_t$ is computed at each output step from a sequence of annotations. Each annotation $\boldsymbol{h}_k$ contains information about the whole input sequence, with a strong focus on the parts surrounding the k-th element of the sequence. The context vector is computed by:

$$\boldsymbol{c}_t = \sum_{k=1}^{n_x} \alpha_{tk} \boldsymbol{h}_k, \tag{3.27}$$

The weight α_{tk} of each annotation is calculated by a function that learns how much attention should be given to the inputs around position k to generate the output at position t:

$$\alpha_{tk} = \phi(\boldsymbol{s}_{t-1}, \boldsymbol{h}_k), \tag{3.28}$$

where ϕ is a feed-forward neural network.

3.2.4.5 Neural Networks Summary

In Section 3.2 I have provided a theoretical and technical background of neural networks, with specific focus on recurrent neural networks, which will be the framework used in the neural approach to the task of text normalization described in this thesis. As I will show in Chapter 6, the RNN framework is based on a gated recurrent unit (Section 3.2.4.2) with bidirectional encoder (3.2.4.3) and soft attention mechanism (3.2.4.4).

3.3 Evaluation Metrics

This thesis describes statistical and neural sequence-to-sequence methods employed in a task of text normalization of words taken in isolation. This section provides an overview of some common evaluation metrics that are used to assess the performance of a model and of a baseline system, with particular emphasis on those that are suitable for our task.

- **Precision** is the number of correctly translated tokens divided by the length of the system translation.
- **Recall** is the number of correctly translated tokens divided by the length of the reference (human) translation.
- **Accuracy** refers to the proportion of correctly labeled instances.
- **BLEU score** [Papineni et al., 2002]. BLEU (Bilingual Evaluation Understudy) computes n-gram precision by comparing the n-grams of the machine translation (candidate) with the n-grams of the human translation (reference) and counting the number of position-independent matches. This results in a high score for sentences that have a high number of matches. To prevent the model from generating multiple copies of common n-grams, that is n-grams which are likely to be found in the reference sentence and result in high precision, the score is modified so that a candidate n-gram that matches a reference n-gram prevents the latter from being used again for comparison. Moreover, the model may have the tendency to generate short sequences to increase precision. These should be penalized by the fact that they produce a low recall. However, in machine translation, recall is rarely used, since there is usually more than one reference translation. For this reason, BLUE uses a brevity penalty factor, that penalizes a candidate translation that does not match the reference translations in length.

In a character-level machine translation framework, where alignment units consist of single words, evaluation measures such as precision, recall and BLEU may provide information on the extent to which a unit normalized by the model, viewed as a sequence of characters, differs from its reference. In other words, they express the magnitude of the intra-word error. However, for the purposes of this work I chose to simply check if a source unit has been correctly normalized or not by the system. For this reason, the accuracy score is used to evaluate the various models implemented.

4 Related Work

Several automatic approaches have been adopted in normalization of historical texts, such as rule-based methods that learn rules from training data [Bollmann, 2012], edit distance methods [Baron and Rayson, 2008; Pettersson et al., 2013] and character-level statistical machine translation (CSMT).

CSMT has been initially applied to closely related languages, such as Spanish and Catalan [Vilar et al., 2007]. Although it did not produce better results if compared with word-level statistical machine translation (WSMT), the translation quality improved when the two systems were combined by using CSMT on unknown words. Tiedemann [2009] used it for Norwegian and Swedish, concluding that, although it makes more errors than WSMT, many errors are of small entity, in that the translated word is very close to the reference. Since the BLEU score computed on word n-grams overly penalizes such errors, he proposed to replace it with other evaluation metrics, such as LCSR (longest common subsequence ratio). Moreover, he observed that CSMT can also learn mappings between words that are not formally similar.

CSMT has been used for transliteration between languages with different writing systems. Matthews [2007] used CSMT for Chinese-English and Arabic-English transliteration of proper names. Tiedemann and Nabende [2009] addressed the fact that each language has its own transliteration rules for the same source language and used CSMT to convert between the transliterated form, in different languages, of the same Russian proper name. Tiedemann [2012] proposed to use CSMT to translate an under-resourced source language into a pivot language for which more data is available, and then translate the latter into the target language with traditional WSMT. For example, when applied to the translation of Macedonian into English, via the pivot language Bulgarian, this approach produced a BLEU score of 21.10, as opposed to 20.24 obtained by directly translating from source to target language with WSMT.

More recent applications of CSMT include text normalization of historical texts: Sánchez-Martínez et al. [2013] for old Spanish; Pettersson et al. [2014] for old En-

glish, German, Hungarian, Icelandic, Swedish; and Scherrer and Erjavec [2016] for historical Slovene. De Clercq et al. [2013] used CSMT to normalize Dutch user-generated content, and Ljubešić et al. [2014] to normalize Slovene tweets.

The work describing how CSMT has been applied to dialect normalization is highly relevant for the task addressed in this thesis. In particular, in the task of normalizing the Archimob corpus (transcriptions of interviews conducted in various Swiss German dialects), Samardžić et al. [2015] and Samardžić et al. [2016] established a strong baseline by selecting, for each source word in the test set, its most common normalization in the training set. They observed that CSMT outperforms the baseline only in the normalization of unknown words, but not of unique words (words associated with only one normalized form in the training set) and ambiguous words (words associated with more than one normalized form in the training set). Scherrer and Ljubešić [2016], using the same corpus, applied CSMT to all the above-mentioned categories of words and showed that, although the performance on unique and ambiguous words is worse than the corresponding baseline performance, the improvement on unknown words is such that it results in an overall better accuracy. They also showed that normalizing segments instead of words in isolation helps exploiting the disambiguating effect of context, thus improving accuracy. The authors identify some potential improvements: a language model that operates on the word level and a neural language model that could learn morpho-syntactic regularities and long distance dependencies. Following up on their work, in my thesis I compare CSMT and an implementation of a recurrent neural network framework that not only integrates a word-level language model, but also extends the neural approach to the whole normalization process.

Bengio et al. [2003] applied neural networks to language models. They address the problem of data sparseness, that might result in the 0 probability of an n-gram (a sequence of symbols, e.g. words) in the test set, if the n-gram has not been seen in the training set. They use a distributed representation for words that allows generalization, in that unknown n-grams receive high probability if they are made of words that are similar to words that appear in already seen sentences, whereby similarity is based on word embeddings. Their network consists of an input layer of word features, one hidden layer with tanh activation function and an output layer with a softmax function, that outputs a probability distribution over words given a previous word sequence.

Schwenk [2012] extended the neural network approach to translation models and integrated it in the standard pipeline of a phrase-based SMT system, in order to re-score the list of most likely translations (n-best list) given a source sentence.

Recurrent Neural Networks (RNN) were proposed as a new approach to machine translation, as opposed to the traditional phrase-based SMT methods, by Kalchbrenner and Blunsom [2013]. Their recurrent continuous translation model consists of two components, a convolutional sentence model conditioned on the source sentence and a recurrent language model for the generation of the target translation.

Cho et al. [2014] introduced encoder-decoder models, in which one RNN encodes an input sequence of symbols into a fixed-length vector representation, and another decodes the representation into an output sequence of symbols. These models were used to compute conditional probabilities of phrase pairs, which were then integrated as an additional feature into the log-linear model of a SMT system, resulting in better performance. Sutskever et al. [2014] used a similar framework, but based on the long-short-term memory (LSTM) unit. Their model outperformed the standard SMT system and was able to correctly translate very long sequences. In both the above-mentioned papers the authors emphasize the ability of their models at capturing syntactic and semantic information.

Bahdanau et al. [2014] extended the encoder-decoder model with a soft attention mechanism that, at each step during decoding, focuses on the most relevant part of the input. They view the practice of compressing all the information carried by the input sentence into a single fixed-length vector as a potential limitation, particularly when long sequences are translated. They propose to encode the input sentence into a sequence of vectors which are then selected adaptively at decoding time: each time a target word is predicted, the soft attention mechanism focuses on the most relevant part of the input. They show that their approach performs better than the basic encoder-decoder model.

Honnet et al. [2017] developed a MT system for translation from Swiss German into Standard German and addressed the problem of out-of-vocabulary words (OOV), i.e. source words whose translation has not been learned in the training phase and are therefore simply copied when producing the corresponding target word. They propose a pre-processing method, to be used before SMT, that tries to convert such unknown words into a new form. If this turns out to be a known Swiss German word, it can be translated by the system. If it is a known Standard German word, it can be copied with a high likelihood of resulting in a correct translation. The conversion is carried out by applying three methods: explicit spelling conversion rules; replacement of the OOV word with a word that has the same pronunciation according to phonetic representations obtained with a grapheme-to-phoneme converter; and character-level NMT with a quasi-recurrent neural network (QRNN), which combines a convolutional and a recurrent architecture.

Though RNNs have historically outperformed CNNs in translation tasks, one possible limitation of RNNs is that they process one sequence element at a time, so that computation cannot be fully parallelized. By contrast, CNNs can compute all elements simultaneously, and allow to process information hierarchically, which makes it easier to capture complex relationships in the data. Gehring et al. [2017] introduced an architecture for sequence-to-sequence models that achieves state-of-the-art results and is entirely based on convolutional neural networks.

Gulcehre et al. [2016] integrated a language model into an encoder-decoder framework to augment the parallel training data with additional monolingual corpora on the target side. Both components of their system are trained on the same type of units, i.e. characters. Kann et al. [2016] applied soft attention to the task of morphological segmentation. Ruzsics and Samardžić [2017] addressed the same task by integrating a language model and improved performance by training their system – unlike Gulcehre et al. [2016] – at two levels: the basic encoder-decoder component is trained on character sequences and the language model component is trained on the sequences of morphemes.

The neural model that I implement in this work is based on the encoder-decoder model of Cho et al. [2014], with the soft attention mechanism proposed by Bahdanau et al. [2014]. The basic architecture has been augmented with additional language models, following Gulcehre et al. [2016] and Ruzsics and Samardžić [2017].

5 Data, Tools and Resources

5.1 The Corpus

I have previously emphasized the fact that both statistical and neural methods require a parallel corpus of manually normalized texts to train models for the task of text normalization. The data for the present thesis consists of parallel corpora of manually normalized Swiss German texts. Since the focus is on normalization of WhatsApp messages, the main corpus consists of messages from this CMC form. A second corpus of manually normalized SMS messages will be used to run further experiments with augmented models, as described in detail in Chapter 6.

The **WUS** (What's up Switzerland) corpus is the corpus of WhatsApp messages [Stark et al., 2014; Ueberwasser and Stark, 2017]. The entire collection contains 763,650 messages in the four national languages of Switzerland: German, French, Italian and Romansh. A portion of the data has been manually normalized according to specific normalization guidelines. The manually normalized data used for the normalization task described in this thesis consists of 5,345 messages in Swiss German, resulting in a total of 54,229 parallel alignment units. Examples of alignment units in the WUS corpus are shown in Table 4. This is a relatively small data set, particularly if compared with the standard size of those normally used to train machine translation systems. For the sake of comparison, the German-English section of the Europarl parallel corpus[1] [Koehn, 2005], extracted from proceedings of the European Parliament, consists of 1,920,209 aligned sentences, with 44,548,491 German words and 47,818,827 English words. Even the least represented language pair, Romanian-English, has 399,375 sentences, with 9,628,010 Romanian words and 9,710,331 English words. The modest size of our data set means that the normalization system can rely on little training data to learn how to perform the task on new data. Despite having replaced statistical methods in machine translation, neural methods are not able yet to outperform them in text normalization. One of the reasons for this gap is to be found in the size of the corpora. Neural models can

[1] `http://www.statmt.org/europarl/`

take advantage of large data sets, but these are seldom available in the field of text normalization, since their creation by experts and linguists for research purposes is expensive and time-consuming. By contrast, the Europarl corpus, which can be used to train machine translation systems, constantly increases in size, at no cost for the NLP community, simply because the proceedings of the European Parliament have to be translated in all the languages of the European Union and made public.

The WUS corpus was collected in 2014 from WhatsApp protocols sent by WhatsApp users who responded to a call appeared on the media. Many users also filled in an anonymous questionnaire and thus provided demographic information about themselves. Since the chats also contain texts written by third parties, privacy was a major issue in the collection of the data. Several manual processing steps have been performed on the corpus, including language identification (which was done manually to overcome the difficulties posed to automatic methods by the non-standard nature of the data and the frequent code-switching) and normalization of a portion of the data. Automatic processing includes tokenization, POS tagging and lemmatization of a portion of the manually normalized Swiss German data. Table 2 shows the number of chats, messages and tokens in the corpus for each Swiss national language.

Language	Chats	Messages	Tokens
Swiss German	275	506,984	3,611,033
French	141	197,255	1,397,375
Italian	87	42,559	293,567
Romansh	77	29,094	283,909

Table 2: Number of chats, messages and tokens per language in the WUS corpus.

We shall recall that text normalization aims at converting non-canonical text into a standardized form. Manual normalization (Table 3) was carried out on the Swiss German messages with an adapted version of the on-line glossing tool described in Ruef and Ueberwasser [2013]. The normalization guidelines are based on those defined in the sms4science project [Stark et al., 2009–2015], with some adaptations aimed at accounting for features and linguistic phenomena specific to WhatsApp messages.

POS tagging and lemmatization of the manually normalized Swiss German messages were performed with the TreeTagger[2] [Schmid, 1994] using the Standard German model. Messages initially identified as French were automatically normalized, POS

[2] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

Language	Messages	Tokens
Swiss German	5,345	54,229
French	6,999	51,234
Italian	4,399	40,625
Romansh	7,117	77,818

Table 3: Number of manually normalized messages and tokens per language in the WUS corpus.

tagged and lemmatized with the MElt[3] sequence labeler [Denis and Sagot, 2012] and its normalization wrapper. Manual evaluation, performed by checking a sample of 1,314 tokens from 160 randomly selected messages, resulted in an accuracy of 95.74% for the normalization task. It will be interesting to compare this score with the automatic normalization of Swiss German messages described in this thesis (see Chapter 7).

The **SMS** corpus is the corpus of SMS messages written in the four national languages of Switzerland [Stark et al., 2009–2015; Ueberwasser, 2015], and is entirely manually normalized. The Swiss German portion contains 10,674 messages, for a total of 262,494 alignment units.

A parallel corpus consists of a source side and a target side. The two sides are aligned in such a way that each line represents an alignment unit consisting of the source sequence and its correct translation or normalization. Aligned sequences can be of various types, for example sentences or tokens. The parallel corpus is split into training, tuning (or development) and test set. The training set is used to train the model by showing it how humans normalize texts, and the way a system learns from the data varies according to the learning algorithm used. The purpose of the tuning set is to offer the model the possibility to apply what it has learned to data that it has not seen before. The predictions made by the system are compared to the manual annotation (the gold standard), and the score of correct predictions is computed. The system can then adjust certain parameters of the model in order to achieve the best possible performance on the tuning set. Finally, the model is applied to the test set, which also consists of unseen data, and the performance thus obtained expresses the performance of the model.

I have mentioned that in this work I address the task of text normalization as a

[3] `https://team.inria.fr/almanach/melt/`

machine translation task. In machine translation, two important components in the generation of the output sequence are the translation model, which is responsible for finding the best translation for a given source sequence, and the language model (LM), which is responsible for the fluency of the target sequence. Both sides of the WUS parallel corpus are essential for building a translation model for this task, whereas the language model only needs data from the target side of the corpus. The target side of the SMS corpus is used in this task to train additional language models, which can be employed with the purpose of improving the fluency of the target language. In particular, one LM is trained at the character-level in the SMT approach. In the neural approach, there is no notion of a separate language model and translation model, but just a single sequence model, whose prediction of one symbol at a time is conditioned on the entire source sequence and the already produced target sequence. However, I will show in Chapter 6 how it is possible to integrate additional language models trained on the SMS corpus into the basic neural sequence-to-sequence model trained on the WUS corpus.

Given the nature of the source texts, one interesting feature of the WUS corpus is the frequent use of emojis. These are symbols and pictographs depicting facial expressions, common objects, places and animals, that are used in CMC to express emotions and convey information. For this reason, rather than representing mere decoration, they can be considered an intrinsic part of a text. Despite the fact that emojis are not proper words, in 2015, Oxford Dictionaries named the 😂 emoji the Word of the Year, for best reflecting the ethos, mood, and preoccupations of 2015.[4] Emojis are replaced in the WUS corpus by a sequence of characters consisting in the description of the symbol. For example, the emoji 😃, which corresponds to the Unicode hexadecimal character code &#x1F603, is rendered as:

`emojiQsmilingFaceWithOpenMouth`

Stark et al. [2014] point out that, for corpora that can be queried on-line, replacing symbols with their descriptions allows a user to write queries that look for a specific emoji or a group of them with common characteristics. For example, a user might use regular expressions to find all emojis in which there is a smiling face, or all those in which the mouth plays a role.

Such long sequences may however pose a problem to a character-level automatic normalization system, since they have a negative impact on training time. Moreover, they might produce normalization errors which could be avoided if a sequence were simply copied from source to target. Therefore, in building a normalization model,

[4] `https://en.oxforddictionaries.com/word-of-the-year/word-of-the-year-2015`

it is important to choose a strategy that accounts for how to treat such sequences. For this purpose, two versions of the WUS corpus have been used within the present work. The *original* version contains emojis as descriptions, whereas the *modified* version contains emojis as symbols. In addition, in the latter version 19 instances of hyper-links have been removed. These are potentially problematic long sequences, such as the following example:

`https://pbs.twimg.com/media/bnjl_1oceaa73_q.jpg`

	alignment type	**source form**	**normalized form**	**English gloss**
1	one-to-one	hüt	heute	today
2	one-to-many	inere	in einer	in a
3		hani	habe ich	have I
4		heschen	hast ihn	(you) have ... him
5		hämmers	haben wir es	have we ... it
6	many-to-one	b esser	besser	better
7		aweg riise	wegreissen	tear away
8		morge sport	Morgensport	morning gym
9	many-to-many	über chunts	überkommt es	receives it

Table 4: Examples of alignment units in the WUS corpus.

Table 4 shows the different types of aligned units that occur in the WUS corpus. Most of the alignments are pairs of single tokens (one-to-one alignments), as in row 1. There are also frequent contracted forms corresponding to multiple normalized words (one-to-many alignments). These are typically merged constructions of preposition and article (2), and verb forms merged with a subject pronoun (3), an object pronoun (4), or with both (5). The few cases of many-to-one alignments are due to typos (6), arbitrarily split verb prefixes (7)[5] and arbitrarily split compounds (8). Finally, different combinations of the factors listed above can result in many-to-many mappings (as in row 9, a combination of 4 and 7). Since one-to-one alignments represent the largest proportion of alignment units, from now onward, for the sake of simplicity, the terms *alignment unit* and *word pair* will be used interchangeably. The same applies to *source sequence* and *source word*.

[5]In German, certain verbs have a prefix that in the infinitive form is attached to the verb. A distinction exists between verbs with inseparable prefix (e.g., *verstehen* 'understand') and verbs with separable prefix (e.g., *aufstehen* 'stand up'), where the prefix is separated from the verb in certain grammatical constructions. Example 7 in Table 4, though belonging to the latter category, is in the infinitive form. The prefix should therefore not be separated from the verb.

feature	original form	intended form	normalized form	English gloss
typo	shriibe	schriibe	schreiben	write
vowel reduplication	gaaaanz	ganz	ganz	all, quite, whole
unconventional abbr.	ev	evt. / evtl. /eventuell	eventuell	possibly
omitted vowels	vllt	vilicht	vielleicht	maybe
unconventional casing	fReud	Freud	Freude	joy, pleasure

Table 5: Examples of linguistic features in the WUS corpus.

Table 5 shows a list of the linguistic peculiarities of the source language in the corpus, already mentioned in Section 1.2, with one example for each feature type. Each entry in the column *intended form* is only one of many possible spellings that would be plausible in "regular" written Swiss German. The boundaries between different features, and between what can be considered regular and irregular, are not always clear-cut in CMC and in a language without an orthographic standard. In some cases, such as the example of vowel reduplication, it seems reasonable to ascribe the spelling of the word *gaaaanz* to a deliberate choice made by the user with the purpose of emphasizing a particular word, rather than to uncertainty due to the lack of a standard spelling. The example of omitted vowels is more likely to be the result of a common strategy aimed at saving space and time, rather than a typo.

5.2 Tools

In this section, I describe the tools and computational resources that I have employed in the two approaches of SMT and NMT. Further details on their practical implementation can be found in Chapter 6, whereas Appendix B provides details on the various programming scripts used in this thesis.

5.2.1 The Moses Toolkit for SMT

I have used the Moses toolkit[5] [Koehn et al., 2007] to perform the experiments involving phrase-based statistical machine translation (PBSMT). Moses also offers the possibility to carry out hierarchical PBSMT, syntax-based translation and factored

[5] `http://www.statmt.org/moses/index.php?n=Main.HomePage`

translation, an extension of PBSMT that allows to add additional linguistic information to a phrase-based system. Moses consists of various components, such as the training pipeline and the decoder. The training pipeline is a collection of tools, mainly written in the Perl programming language, which process parallel texts to learn a translation model. The decoder is an application written in the C++ programming language, which uses the translation model to translate a source sentence into a target sentence. In addition, various external tools can be integrated, for example tools that can learn a language model from monolingual data, that is used by the decoder to ensure the fluency of the output.

5.2.2 The Python Programming Language

Python[6] is a high-level, interpreted programming language that supports object-oriented programming (OOP). It is object-oriented in that it allows to define classes and create instances of them. Defining a class means creating a new type of data, specifying the operations that can be performed on it. For example, the type *integer* is a pre-defined type in most programming languages, and it is associated to a number of operations that have a certain effect on it (addition, subtraction, etc.). Python offers the possibility to create new classes according to the needs of a specific task. Like many other programming languages, it is possible to import and use external modules, packages and libraries. The advantage in Python is that these resources are very extensive and constantly improved. A module is a file containing Python definitions and statements. Packages are a way of structuring Python's modules. As the Python documentation says, we can think of packages as the directories on a file system and modules as files within directories, though the analogy is an oversimplification and we should take it with a grain of salt. When a module or package is "published", people often refer to it as a library. A package that is commonly used in scientific computing, including the present work, is NumPy, which enables various functionalities such as operations with logarithms, generation of random numbers and broadcasting. Broadcasting is particularly useful in operations where inputs do not have exactly the same shape, such as the sum of a vector and a scalar, where it is assumed that the scalar is added to each entry of the vector. In addition to the Python scripts used for pre-processing, post-processing and evaluation in the SMT approach, the NMT approach heavily relies on Python, both in terms of scripts used for training the NMT model, and of additional libraries and frameworks such as Theano, Blocks and SGNMT, which are described in Section 5.2.3.

[6] `https://docs.python.org/3/index.html`

5.2.3 Neural Network Libraries and Frameworks

5.2.3.1 Theano

Theano[7] [Theano Development Team, 2016] is a Python library that allows to define, optimize, and evaluate mathematical expressions and linear algebra operations involving multi-dimensional arrays. A neural network relies on linear algebra operations, such as vector and matrix addition and multiplication, in the steps of forward propagation (computing the loss function), back-propagation (computing the gradient) and optimization (using the gradient to update the model parameters). Theano is not a programming language in the strict sense because a program must be written in Python in order to build expressions for Theano. However, it shares some properties of programming languages in that it allows to declare variables, build expressions that put those variables together, and compile expression graphs to functions, so that they can be used for computation. In a simple Theano implementation for logistic regression, for example, the variables to be declared are the input features and the target labels of a data set. The parameters of the model (the weights and the bias) are then initialized to random or constant values. These variables are combined in expressions such as the dot product to compute the value of z (Equation 3.12). An expression graph is then built by arranging the expressions in a structured sequence that reflects the architecture of the algorithm. In logistic regression, the graph consists of the following components: compute z, apply the sigmoid function to z to output a prediction, compute the cross-entropy cost function, compute the gradient by finding the derivatives of the cost function with respect to each parameter, use the gradient to update the parameters and repeat the process until convergence.

5.2.3.2 Blocks

Blocks[8] [van Merriënboer et al., 2015] is a framework that helps build neural networks on top of Theano. It supports and provides algorithms for model optimization as well as tools that allow to construct parametrized Theano operations, called "bricks". A brick is defined by a set of attributes (e.g., the number of input and output units) and a set of parameters of the brick object that will vary during learning (e.g., the weights and the biases). Moreover, Blocks allows to save and resume training, and monitor and analyze values during the training progress. While

[7] `http://deeplearning.net/software/theano/index.html`

[8] `http://blocks.readthedocs.io/en/latest/index.html`

Theano operates at the abstract and general level of linear algebra operations, Blocks represents a step toward a lower level of abstraction and builds on Theano with specific focus on neural network frameworks. In addition to standard activations and transformations used in feed-forward neural networks, a variety of more advanced recurrent neural network components are available, like LSTM, GRU, and support for attention mechanisms. Blocks allows, for example, to use Theano for declaring an input matrix as follows:

```
from theano import tensor
source_sentence = tensor.lmatrix('source')
```

It then offers various modules to set up neural architectures that process tensors like the matrix above for the training of a model and for decoding. For example, a bidirectional encoder (see Section 3.2.4.3) can be imported as follows:

```
from blocks.bricks.recurrent import Bidirectional
```

5.2.3.3 SGNMT

SGNMT[9] [Stahlberg et al., 2017] is an open-source framework, compatible with Blocks, that goes one further level of abstraction down in that it relies on Theano for linear algebra operations and on Blocks for applying them to neural networks, but it focuses on neural machine translation and other sequence prediction tasks. The tool provides a flexible platform which allows pairing a NMT model with various other models, such as language models. The option to combine several models will become very useful in the NMT approach (see Section 6.4.1) for two reasons: firstly, it allows to build an ensemble of NMT models whose performance is usually better than the performance of each single model; secondly, it offers the possibility to integrate language models that are trained on the same corpus but on a different level of granularity (e.g, words instead of characters) or on the same level but on a different corpus (as I will do by training a character-level LM on the SMS corpus). The two central concepts in SGNMT are predictors and decoders. Predictors are modules which define scores over the target language vocabulary given the current internal predictor state, the history and the source sequence. They compute a score for each symbol in the target vocabulary (in character-level MT, the set of all the characters that occur in the target language), so that the symbol with the highest

[9] https://ucam-smt.github.io/sgnmt/html/index.html#

probability is predicted. For example, the SRILM language model[10] [Stolcke, 2004] is implemented in SGNMT as SRILMPredictor and can be imported as follows:

```
from cam.sgnmt.predictors.ngram import SRILMPredictor
```

It can then be specified when calling the Python code for decoding (`decode_segm.py`) using the `--predictor` argument. Decoders are search strategies for traversing the search space which is spanned by the predictors. Examples of such search strategies are the greedy search and the beam search. In greedy search, at each prediction step the most likely symbol given the source sequence and the previous predictions is selected. In beam search, the n most likely predictions are considered, where n is a parameter that defines the beam width.

[10] `http://www.speech.sri.com/projects/srilm/`

6 Methods

In this chapter I describe the implementation and experimental setting of the two methods that I have used to carry out the task of text normalization of Swiss German WhatsApp messages. We shall recall that text normalization aims at converting non-canonical text into a standardized form which can be processed by NLP tools, and is commonly addressed as a machine translation task. The two approaches are character-level statistical machine translation and neural sequence-to-sequence models. Further details on the various programming scripts mentioned throughout this chapter can be found in Appendix B. In the next section, I describe how I computed a baseline and a ceiling, which are two important measures that can be useful in placing the performance score of a model in a frame of reference.

6.1 Baseline and Ceiling

A model that performs an automatic task is usually evaluated by comparing its performance against a baseline, that is a benchmark obtained by applying a straightforward and rather naive method or by using a very simple model. In a supervised machine learning framework, for example, the baseline for classification problems can be set by assigning, to all the test set instances, a random label or the label that occurs more frequently in the training set, and by computing the accuracy thus achieved. In POS tagging, where each token in a text is labeled with its corresponding part of speech, the baseline consists of the accuracy obtained by a system that assigns to each token its most frequent POS in the training set. Unknown tokens are labeled as proper nouns, that is if a test set token has not been seen during training, the system assumes it is a proper noun. This simple method, in which the system does not even have to learn a model, but simply to memorize the training set, usually results in an accuracy of around 90%, which represents a very strong baseline. If a POS tagger that relies on a sophisticated algorithm achieves an accuracy score of 90%, we might be tempted to consider it a satisfactory score. Only when we compare this score to the baseline do we get a sense of how poor the performance actually is. The baseline gives us a measure of the minimum score that

we should expect from a model. In developing a system, it is essential to outperform the baseline and the goal is to do it by the largest possible margin.

The baseline for the task of text normalization described in this thesis has been established following Samardžić et al. [2015]. For each word in the test set, its most frequent normalization in the training set is chosen. In particular, we can distinguish between *unique* source words, which are always normalized the same way in the training set, and *ambiguous* source words, which have more than one normalization form. In case of tie, the normalized form is chosen randomly. Words that have not been seen in the training set (*new* words) are simply copied. This method, applied to the WUS corpus, produces an accuracy score of 84.45%. Moreover, there seems to be a ceiling, mainly due to the ambiguity described in Section 1.2, that prevents accuracy to go beyond a certain threshold, unless the context is used. Such ceiling can be computed by applying the same approach used for the baseline, but training on the whole data set. This method reduces the error rate thanks to the fact that there are no *new* words in the test set, and produces an accuracy score of 93.19%. Table 6 shows the baseline and ceiling scores and the proportion of words for each of the three above-mentioned categories.

	Baseline		**Ceiling**	
Word category	**Proportion**	**Accuracy**	**Proportion**	**Accuracy**
Unique	55.05	98.43	63.87	99.51
Ambiguous	32.70	81.22	36.13	82.03
New	12.25	30.27	0	0
Total	100	84.45	100	93.19

Table 6: Proportion of words and breakdown of accuracy score for each category, in percentage, for baseline and ceiling (WUS corpus).

An analysis of the table reveals that when computing the baseline, accuracy on *unique* words is close to 100%, so most of the errors are due to *ambiguous* and *new* words. When computing the ceiling, since the whole data set is used for training, all words in the test set have been seen and the proportion of *new* words is thus 0%. Since also for the ceiling the accuracy on *unique* words is close to 100%, we can conclude that virtually all the error is due to *ambiguous* words. This means that even if we allow the system to see and learn the test set during training, accuracy cannot be better than 93.19% due to ambiguity.

An example might be useful to see how ambiguity can cause such systematic error. The source word *mir* is normalized in the WUS corpus 8 times as *mit der* ('with the'), 96 times as *wir* ('we') and 111 times as *mir* ('me' as indirect object). This

means that whenever the source word *mir* occurs in the test set, the method used to compute the ceiling will choose the normalization form *mir*, which is the most common. This results in a correct prediction when the source word is used with the meaning 'me', but in an incorrect prediction when it is used with one of the other two meanings.

In the next sections I describe two approaches to text normalization that aim at outperforming this very strong baseline and get as close as possible to the ceiling. The first approach consists in character-level statistical machine translation (CSMT), the second is based on neural machine translation (NMT) with recurrent neural networks.

6.2 Pre-processing

The term pre-processing denotes the operations that are carried out in order to make a data set suitable for a specific NLP task. The following pre-processing steps are common to both the CSMT and the NMT approach.

6.2.1 Shuffling the Corpus

The WUS corpus is a collection of WhatsApp chats structured, in most cases, in one word per line (see Section 5.1). The dialogue sequence, however, is preserved, in that one could read the text vertically. I split the corpus in three subsets: the training, tuning and test set. It is essential to make sure that these subsets are not biased. Bias could arise if one of the subsets is dissimilar from the others. For example, if the test set has features that are different from the training set, performance on the test set might be low. Even if the three subsets are extracted from the same corpus, there is a risk of bias if the subsets are not created by picking random words from the whole corpus, but rather by simply cutting it in three sequential parts. It might happen, for example, that many of the last chats in the corpus are characterized by a less formal language than the others, with many occurrences of the irregularities that are typical of CMC, such as typos, abbreviated forms and slang. Since they are at the bottom of the corpus, they will end up in the test set. This means that the model, at test time, will have to deal with a task that is more challenging than the one it has learned during training. I thus decided to shuffle the corpus before splitting it, by using the Unix command `shuf`.

6.2.2 Lowercasing

Lowercasing consists in converting all characters into their lowercased form. The decision of whether to perform lowercasing as a pre-processing step of a NLP task is subject to a trade-off between the need for consistency and disambiguation. In word-level SMT, omitting this step could result in an increased number of OOV words. It could happen that the test set word *You* at the beginning of a sentence cannot be translated because it has been seen in the training set only in its lowercased form. On the other hand, lowercasing could increase ambiguity by losing the difference, for example, between *US* (the country) and *us* (the pronoun), or between *Brown* (family name) and *brown* (color), with negative repercussions in tasks such as MT and named entity recognition (NER). In Standard German, the role of letter casing is even more important, since orthographic norms prescribe that nouns be written with uppercased initial. For example, there is a difference between *verfahren* ('to proceed') and *Verfahren* ('procedure'). Due to the informal and spontaneous nature of CMC messages, the same Swiss German word can appear in different forms in the corpus (all lowercased, all uppercased, first letter uppercased, mixed) without obeying systematic patterns. Since omitting lowercasing with the aim of reducing ambiguity might result in overly inconsistent source data, I have opted for full lowercasing of both corpora, using the script `lowercase.perl` (M.1).

6.2.3 Preparing the Data for Character-Level Processing

Whitespace is added between characters so that each one of them can be treated as a token. In addition, underscores in CSMT and the pipe symbol | in NMT are used to replace original whitespace and thus signal word boundaries in alignment units where the source side, the target side or both consist of more than one word, e.g. one-to-many alignments.

6.2.4 Splitting the Corpus

The WUS corpus has been randomly shuffled and split in 80% training set, 10% tuning set and 10% test set. The term *tuning set*, which is common in SMT, is often replaced by the equivalent *development set* in a machine learning framework.

6.3 Methods for CSMT

The CSMT approach has been carried out by employing the Moses tool [Koehn et al., 2007]. The following sections describe each step of the Moses pipeline. Moses is launched with the script `run_one.sh` (L.4), which was originally written to perform multiple runs of Moses on different shuffles of the corpus. In its final version, it is launched only once on the corpus split used throughout the whole project. The script calls `run_moses_one_split.sh` (L.5), which handles the Moses pipeline.

6.3.1 Training the Language Model

The language model is responsible for ensuring the fluency of the output and is trained on monolingual corpora in the target language. Moses uses the language model to select the most likely target language sentence from a set of "possible" translations it generated using the phrase table and reordering table. In the basic implementation of the work described in this thesis, only the target side of the WUS corpus is used for training. I have run further experiments with an additional language model consisting of the target side of the SMS corpus. Moses offers the possibility to use various external language models in the training pipeline. I have opted for the default KenLM language model toolkit [Heafield, 2011]. Following Scherrer and Ljubešić [2016], the order of the LM has been set to 7, which means that the LM is trained on character 7-grams. The language model is trained with the code `lmplz`(K.1) and is then binarized with the code `build_binary` (K.2). The binary format allows LMs to be efficiently stored and loaded.

6.3.2 Training the Translation Model

The translation model aims at finding the most likely translation for a source sequence, and is trained by using a parallel corpus. In a task of word-level machine translation, the source and target sides of the corpus must be previously aligned at the sentence level. Word alignment is then performed automatically, usually through an expectation-maximization algorithm. GIZA++ is a freely available implementation of the IBM models for word alignment, integrated in Moses. The word alignments are taken from the intersection of bidirectional runs of GIZA++ (the first run from source to target language, the second vice versa). Additional alignment points are obtained from the union of the two runs. A phrase table is built to store the mapping between source and target phrases, where phrases are units that

are not necessarily linguistically motivated (i.e., they are mere sequences of words). The phrase table is then consulted in the decoding phase, that is when an unseen source sentence is translated. In a task of character-level machine translation, the source and target sides of the corpus must be previously aligned at the word level. Characters are then automatically aligned with GIZA++ and the resulting phrases are sequences of characters. I've used the script `train-model.perl` (M.2) to train the translation model. I have set Swiss German as the source language and the standardized form as target language (in Moses denoted by the flags `-f` and `-e`, respectively). To establish alignments based on the above-mentioned bidirectional GIZA++ runs, I have used the default `grow-diag-final` heuristics.[1]

6.3.3 Disabling Reordering

The distortion model is a Moses component that allows for reordering of the input sequence. In WSMT, this can improve translation between languages that are characterized by a different syntactic structure and word order. In this character-level task, I assume monotone alignment between source and target sequence. I have therefore disabled the reordering model with the script `disableDistorsion.py` (U.3).

6.3.4 Tuning

Tuning refers to the process of finding the optimal weights for a linear model whose components are: the language model; the translation model obtained from the phrase table; the reordering model; and the phrase and words penalty (character penalty in CSMT), which ensure that the translations do not get too long or too short. Tuning in Moses is launched with the script `mert-moses.pl` (M.3). In the tuning phase, source language sequences in the tuning set are translated by the system, and the output is compared to a set of reference (human) translations. The weights of the components of the linear model are adjusted with the aim to improve the translation quality. This process continues through numerous iterations until an optimal translation quality is reached. Translation performance is usually measured with the BLEU score (see Section 3.3) in WSMT. Following Scherrer and Ljubešić [2016], I have replaced BLEU with word error rate (WER) – which becomes character error rate in a character-level framework – by specifying the flag `--mertargs='--sctype WER'`. WER is based on the Levenshtein distance

[1] A pseudo code that illustrates how the various alignment heuristics work in Moses is available at `http://www.statmt.org/moses/?n=FactoredTraining.AlignWords`

[Levenshtein, 1966], which is the minimum number of substitutions, deletions and insertions that have to be performed to convert the automatically normalized text into the reference text. I have used the default number of iterations, which is 25, with the option of stopping earlier if none of the weights changes more than a value of 0.00001 from one iteration to the next.

The weights are initialized with the following values:

```
LM0 = 0.500000
LM1 = 0.500000
Word Penalty = -1.000000
Phrase Penalty = 0.200000
Translation Model = 0.200000 0.200000 0.200000 0.200000
```

LM0 and LM1 denote the WUS and SMS language model, respectively. The translation model has four different scores, which correspond to the following components:

- inverse phrase translation probability: $\phi(f|e)$
- inverse lexical weighting: $lex(f|e)$
- direct phrase translation probability: $\phi(e|f)$
- direct lexical weighting: $lex(e|f)$

The weights computed in the tuning phase, which determine the contribution of each model component to the final translation, are saved by Moses in the configuration file `moses.ini`.

6.3.5 Decoding

This step consists in the actual translation of unseen text from the source test set into the target language, performed with the code `moses` (M.4). For a given input sequence, the phrase table is consulted and a translation is constructed progressively by adding together partial translations (hypotheses), where the search for the best translation is governed by a number of alternative search heuristics. Decoding relies on a linear model in which each component (phrase table, language model, reordering model and word penalty) has been paired with a weight in the tuning phase.

6.3.6 Post-processing

This phase consists in removing whitespace between characters and underscores between words that had been added in the beginning in order to allow the machine translation process to be carried out at the character level. This is done with the code `removeUnderscores.py` (U.2). The output of this step allows to compare the reference text and the automatically normalized text, and thus compute accuracy, that is the proportion of correctly normalized words.

6.3.7 Evaluation

The results of CSMT are shown in Chapter 7. I have written the Python code `baseline_cat.py` (L.3) to compute accuracy and to break down the score according to the three word categories described in Section 6.1. As an example, the code outputs the evaluation on the baseline system as follows:

```
Total number of source words:  5422

AMBIGUOUS   {'Incorrect': 333, 'Total': 1773, 'Correct': 1440}
Accuracy on AMBIGUOUS words: 81.22%
Proportion of AMBIGUOUS words: 32.7%

UNIQUE   {'Incorrect': 47, 'Total': 2985, 'Correct': 2938}
Accuracy on UNIQUE words: 98.43%
Proportion of UNIQUE words: 55.05%

NEW   {'Incorrect': 463, 'Total': 664, 'Correct': 201}
Accuracy on NEW words: 30.27%
Proportion of NEW words: 12.25%

Overall accuracy:  84.45 %
```

6.4 Methods for NMT

The following sections describe the NMT approach that I have employed for the task of text normalization. The approach is based on a neural sequence-to-sequence

model (Section 3.2.4.2) with a bidirectional encoder (Section 3.2.4.3) and soft attention mechanism (Section 3.2.4.4).

6.4.1 NMT Pipeline

The NMT pipeline is launched with the shell script `run-all-for-all-hyper.sh` (L.6). This was originally meant to launch NMT multiple times as a loop over different settings in terms of embedding and hidden dimensions. These were finally set to 300, which I found to be the optimal value. This script calls the shell script `Main-Norm-lm-ch-2-hyper.sh` (L.7), which handles the NMT pipeline.

6.4.1.1 Building the Alphabets

In NMT, a vocabulary of the source and target languages is extracted. If the translation unit is the sentence, each entry in the vocabulary is a word. This usually implies a trade-off: if we want as many words as possible to be in the vocabulary, we lose computation efficiency. If we restrict the size of the vocabulary, we gain efficiency but increase the number of unknown words, that cannot be translated by the system. In the character-level approach, the translation unit is the isolated word and the vocabulary becomes an alphabet of characters. This has the advantage that a set of symbols of limited size can contain all the characters used in the corpus. I define the two discrete alphabets Σ and Σ_n, consisting of the characters of the source and target language respectively. The mapping from a source character $x \in \Sigma^*$ to its normalized form $y \in \Sigma_n^*$ is then learned by applying an encoder-decoder model with soft attention. In the *original* corpus, the vocabulary consists of 136 characters in the source side and 114 in the target side. The vocabulary of the *modified* corpus is larger, since each emoji represents an additional entry: 250 characters for the source and 228 for the target language.

6.4.1.2 Converting Characters into Dense Vectors

Each character in the alphabet is converted into an integer, which represents an index in a one-hot vector. A one-hot vector is a vector in which all entries have value 0, except for the entry corresponding to the index, which has value 1. Each vector is then converted into a dense representation (embedding) with the number of dimensions specified as hyper-parameter (Section 6.4.2.3). Each entry of the dense vector is a parameter which is updated during training in order to find the set of parameters that minimize the loss function.

6.4.1.3 Training the Model

Since the model is sensitive to the order of the training sequences, five models are trained on different shuffles of the training set. The NMT approach uses an ensemble of the five models in the decoding phase. According to the soft attention mechanism described in Section 3.2.4.4, the conditional probability over output characters is estimated at each step as a function of the previous output y_{t-1}, the current decoder hidden state s_t and the current context vector c_t:

$$p(y_t|y_1, \ldots, y_{t-1}, \boldsymbol{X}) = g(y_{t-1}, s_t, c_t) \tag{6.1}$$

where g is a deep output layer [Pascanu et al., 2013] with a single maxout nonlinearity [Goodfellow et al., 2013]. The training objective is to maximize the conditional log-likelihood of the training corpus:

$$L = \frac{1}{N} \sum_{(x,y)} \sum_{t=1}^{n_y} \log p(y_t|y_1, \ldots, y_{t-1}, \boldsymbol{X}) \tag{6.2}$$

where N is the number of training pairs (x, y).

6.4.1.4 Integrating Language Models

The integration of additional language models represents the main innovation proposed in this thesis in the attempt to allow neural models to achieve state-of-the-art performance in a text normalization task where little training data is available. This method is inspired by Ruzsics and Samardžić [2017], who successfully applied it to the task of morphological segmentation by training their system at two levels: the basic encoder-decoder component is trained on character sequences and the language model component is trained on the sequences of morphemes.

In the application to text normalization here described, a plain encoder-decoder (ED) and a language model (LM) are trained separately before the integration. The ED model is trained on character sequences in the parallel WUS corpus consisting of aligned source words and their normalized forms. The ED model learns a conditional probability distribution over the normalized character sequences given the original sequences, as shown in Equation 6.1. This probability captures context-sensitive individual character mappings, therefore already including the information provided by a usual LM. This model is augmented with additional LMs, separately trained only over the target side of a corpus, applying two different methods. The first,

following Gulcehre et al. [2016], augments the training set with additional target-side data, in our case from the SMS corpus. The second, following Ruzsics and Samardžić [2017], trains a word-level LM and fuses it with the character-level ED using a "synchronization mechanism", which allows to combine the ED character-level with word-level scores at the decoding stage. I have conducted experiments with word-level LMs from both the WUS and the SMS corpus.

As described in Ruzsics and Samardžić [2017], the "synchronized" decoding approach relies on a beam search to find the prediction steps where different scores are combined. The beam search is run at two levels of granularity. First, it produces the output sequence hypotheses (candidates) at the character level using ED scores until the time step s_1, where K best hypotheses end with a boundary symbol. The boundary symbol is of two types: space, which marks the end of a word in a partially predicted sequence, and a special *EOW* symbol, which marks the end of a fully predicted sequence. For example, in the normalized unit *habe ich*, the space is the boundary between the two words, and the *EOW* symbol is the boundary after the whole unit. The step s_1 is the first synchronization step where the normalization hypotheses are re-scored with a weighted sum of the ED score and the LM score:

$$\log p(y_{s1}|y_1, \ldots, y_{s1-1}, X) = \log p_{ED}(y_{s1}|y_1, \ldots, y_{s1-1}, X) + \alpha_{LM} \log p_{LM}(y_1, \ldots, y_{s1}) \quad (6.3)$$

At this step, $y_1, \ldots, y_{s1}$ is considered a sequence of $s1$ characters by the ED system and one word by the LM. After the first synchronization point, the model continues to produce the re-scored hypotheses using ED scores until the next synchronization point. The search process ends at the synchronization point where the last symbol of the best scored hypotheses (using the combined ED and LM score) is the symbol *EOW*, which indicates the end of a complete prediction.

The decoding process scores the segmentation hypotheses at two levels: normally working at the character level with ED scores and adding the LM scores only when it hits a boundary symbol. In this way, the LM score helps to evaluate how likely the last generated word is, based on the predicted word history, that is the sequence of words generated at the previous synchronization time steps. The synchronization mechanism is used in this work to integrate both extended character-level LMs and higher word-level LMs, though in principle it can be used to add any kind of potentially useful predictors or scores obtained separately from different data sources.

6.4.1.5 Ensemble of Models

In data mining and machine learning, combining the output of different models in an ensemble usually produces better results than those achieved by each single model. As Witten and Frank [2005] observe, the performance of these methods is astonishingly good, and yet machine learning researchers have struggled to understand why, since it is not easy to understand in intuitive terms what factors are contributing to the improved decisions.

According to Hansen and Salamon [1990], optimization methods yield optimal parameters (weights and biases) which differ greatly from one run of the algorithm to the next, showing a high degree of randomness depending on the parameter initialization and sequencing of the training examples. This randomness tends to differentiate the errors, so that the networks will be making errors on different subsets of the input space. The collective decision produced by the ensemble is less likely to be in error than the decision made by any of the individual networks.

The framework SGNMT that I have used to build the neural network offers the possibility to perform ensembling by combining different predictors, i.e. by averaging the scores of the individual models at each decoding step. In my implementation I have used five models. Moreover, ensembling can be done with models at multiple tokenization levels, which in our case allows to combine the five basic character-level encoder-decoder models with additional language models. The character-level LM of the WUS corpus is already implemented as part of the basic encoder-decoder framework. Additional language models that I have integrated are two word-level LMs from the WUS and SMS corpora, and a character-level LM from the SMS corpus (see Section 6.4.1.4). In the NMT pipeline, when calling the Python script `decode_segm.py`, the ensemble of ED models and the additional language models can be integrated by using the argument `--predictors`, as in the following example:

```
--predictors $nmt_predictors, srilm, word2char_srilm
```

6.4.1.6 Post-processing Output for Model Comparison

The file with the predictions on the test set must be post-processed in order to be compared with the CSMT output and the reference. The NMT output appears as in the following examples:

```
e i n e
f ü r | d e n
```

A first `sed` command (`sed 's/ //g'`) can be used to eliminate whitespace between characters. Next, a second `sed` command (`sed 's/|/ /g'`) replaces the *pipe* symbol with whitespace.

6.4.2 Hyper-parameters

So far I have described the role of weights and biases in the various phases of a process based on neural networks. These constitute the parameters of the model. As mentioned in Section 3.2.2, there are however other settings that contribute to the configuration of a neural network and might have an impact on the performance and computational efficiency of the model. These settings, whose values are determined by the programmer or user, are called hyper-parameters. The following sections describe the choice of hyper-parameters for my NMT model.

6.4.2.1 Optimization Algorithm

In Section 3.2.1, I have described how gradient descent is used in a process called optimization to minimize the cost function (Equation 3.7). For the NMT experiments, I have tried two different optimization algorithms: stochastic gradient descent with momentum and AdaDelta.

Gradient descent can take on different connotations depending on the number of training examples that are processed before updating the parameters. In batch gradient descent, all training examples are processed (one pass over all training examples is defined as an epoch) before one step can be taken in the direction of the point in the cost curve that corresponds to the minimum cost.[2] If we plot the value of the cost function relative to the number of epochs, we expect to observe a steady and smooth decrease. However, this approach might be inconvenient, especially if the training set is particularly large, since we have to process all training examples before we can update the parameters. A valid alternative consists in mini-batch gradient descent, in which the training set is split into subsets (mini-batches) and the parameters are updated after each mini-batch has been processed. If we plot the value of the cost function relative to the number of mini-batches processed, the curve will still show an overall decrease, but might display a more erratic behavior. This is because each mini-batch is some sort of small training set, with possibly different characteristics from the previous one, which may cause the cost function

[2] Though the term curve refers to a two-dimensional space, I use it in its extended sense to include hyper-surfaces in a multi-dimensional space.

to temporarily increase. One special case of the mini-batch setting is stochastic gradient descent (SGD), in which each mini-batch has size equal to one, so that the parameters are updated after each single training example has been processed.

Other algorithms have been developed with the purpose of speeding up training. In gradient descent with momentum, we add a fraction γ of the update vector of the past iteration to the current update vector:

$$\boldsymbol{v}_t = \gamma \boldsymbol{v}_{t-1} + \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{6.4}$$

The hyper-parameter γ has values between 0 and 1 and controls the extent to which past observations affect the value of the update vector $\boldsymbol{v}$ at iteration t. For values of γ close to 1, the weighted average of observations from the past will have a substantial impact on the update vector. As a result, if the gradient for a particular parameter oscillates erratically in directions that are irrelevant for the purposes of reaching the minimum, previous fluctuating observations will tend to cancel out and dampen the effect of the current update vector. As γ decreases, the dampening effect is reduced and the update will be more sensitive to the effect of the current update vector. Equation 3.7 for updating the parameters becomes:

$$\theta := \theta - v_t \tag{6.5}$$

While it is possible to choose the learning rate α by trial and error, it is usually best to find the most suitable one by monitoring learning curves that plot the cost function as a function of time. Slowing down the learning rate when gradient descent approaches a minimum in the cost curve can prevent the parameter values from oscillating around the minimum without reaching it. Algorithms with adaptive learning rates use a separate value for each parameter. For example, the update equation for AdaGrad [Duchi et al., 2011] is as follows:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} \boldsymbol{g}_\tau^2}} \odot \boldsymbol{g}_t \tag{6.6}$$

where $\boldsymbol{g}$ is the gradient, and division and square root are applied element-wise. The denominator computes the L_2 norm of all previous gradients on a per-dimension basis. This equation differs from 3.7 in that for each parameter the global learning rate α is modified by the denominator: it is larger if the past gradients are small and smaller if they are large. However, since the previous gradients are squared and

added together, the denominator keeps increasing during training. This might cause the learning rate to shrink to a point in which the algorithm is no longer able to acquire additional knowledge.

AdaDelta [Zeiler, 2012] addresses this weakness by restricting the window of accumulated past gradients to a fixed size. Since storing previous squared gradients is inefficient, this method implements this accumulation as an exponentially decaying average of the squared gradients $E[g^2]_t$. At time step t, we have

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \tag{6.7}$$

where γ is a decay constant analogous to that used in the momentum method. It is then possible to compute the root mean squared of the gradient (RMS):

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \tag{6.8}$$

where the constant ϵ is a smoothing term that will avoid division by zero. The update rule for AdaDelta is as follows:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\alpha}{RMS[g]_t} \odot \boldsymbol{g}_t \tag{6.9}$$

One further step is necessary in order to ensure that the update

$$\Delta\theta_t = \frac{\alpha}{RMS[g]_t} \odot \boldsymbol{g}_t \tag{6.10}$$

has the same unit as the updated parameter. This step consists in defining another exponentially decaying average for the squared updates:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \tag{6.11}$$

The root mean square of the updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \tag{6.12}$$

The constant ϵ serves two purposes: it starts off the first iteration where $\Delta\theta_0 = 0$, and it ensures progress continues to be made even if previous updates are of small

entity. In the final AdaDelta update equation we replace the learning rate α with the RMS computed in Equation 6.12, referred to the previous step:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \odot \boldsymbol{g}_t \tag{6.13}$$

For my implementation of AdaDelta, I used the Blocks default values ($\gamma = 0.95$, $\epsilon = 1e-06$). Since preliminary experiments with SGD with momentum yielded worse results, I decided to build the final NMT models with the AdaDelta optimization algorithm.

6.4.2.2 Regularization

One strategy that is commonly adopted to avoid overfitting, that occurs when the model fits to well the training data and is not able to generalize to unseen data, is regularization. While there are several regularization techniques that can be utilized, the one that I have considered for this task is dropout [Srivastava et al., 2014]. The basic intuition is that a probability is set for some units of the network to be "switched off" whenever a different training example is processed, so that all the incoming and outgoing links to that unit are eliminated, and training and back-propagation occur in a simpler and reduced network. At test time, it is not feasible to explicitly average the predictions from exponentially many reduced models. However, in practice, a single neural network without dropout can be used at test time. The weights of this network are scaled-down versions of the trained weights. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time. The function that results from dropout has a decision boundary that is less complex and less attentive to the idiosyncrasies of the training set, and closer to the linear separation of linear regression. A unit will be discouraged to give too much importance (i.e. weight) to a unit in the previous layer, given the risk that the latter is randomly switched off. As a consequence, the weights will be uniformly distributed over all the units in the previous layer and will tend to maintain low values. According to Equation 3.12, if the weights are small, the weighted sum z will be close to 0, which, if we apply an activation function like the hyperbolic tangent (Figure 2), will correspond to the part of the curve that resembles a linear function, again similar to linear regression, that is less prone to overfitting. Dropout regularization can also be viewed as some sort of ensemble technique, where for each training example a different network is set-up and trained, resulting in better overall performance. I have run experiments with the regularization parameter p set at the default 1.0 (no regularization) and 0.7.

6.4.2.3 Parameter Initialization and Additional Hyper-parameters

With respect to parameter initialization, the biases have been initialized to 0 and the recurrent weights to an isotropic Gaussian distribution, with a mean of 0 and a standard deviation of 0.01. The following hyper-parameters have been used in addition to those related to optimization algorithm and regularization:

- **Input and hidden dimensions.** In NMT, input symbols, in our case characters, are represented as vectors (embeddings) whose number of dimensions is a hyper-parameter of the model that commonly varies from 100 to 1000. I opted for a number of 300 for the input dimensions of both encoder and decoder, as well as for the hidden units of the encoder and decoder and for the attention vector.
- **Number of layers.** A further option in RNNs is to stack multiple layers, thus adding depth to the network architecture. Since this increases computational cost substantially, I opted for a single layer in both the encoder and decoder.
- **Mini-batch size.** I set the mini-batch size to 20. This means that the parameters are updated by the optimization algorithm each time after 20 training examples have been processed.
- **Epochs.** One epoch corresponds to processing once all training examples. I set the number of epochs to 30.
- **Gradient clipping.** Gradient clipping is used to set a maximum threshold for the gradients, in order to prevent the problem of exploding gradients. I set the threshold at 1.
- **Beam size.** Beam size for the beam search has been set to 3 for the decoding phase, and to 1 for the training phase.

7 Results

7.1 Results

In this thesis I carry out text normalization of Swiss German WhatsApp messages with two different approaches: character-level statistical machine translation and neural machine translation with sequence-to-sequence models. The methods and settings used for the two approaches, and for establishing a baseline and ceiling, are illustrated in detail in Chapter 6. The results of the experiments are shown in Table 7.

System	Corpus	
	Original	**Modified**
cED + LMsms:char + LMwus+sms:word	**87.85**	87.48
cED + LMsms:char + LMwus:word	87.20	87.16
cED + LMwus+sms:word	87.72	87.48
cED + LMsms:char	87.18	87.16
cED + LMwus:words	87.20	87.27
cED ensemble 5	86.41	86.43
cED average 5	84.15	83.81
CSMT LMwus+sms:char	86.35	86.43
CSMT LMwus:char	85.30	85.85
Baseline	84.45	84.45
Ceiling	93.19	93.19

Table 7: Text normalization accuracy scores. Original: corpus with hyper-links and emojis as description of the symbol. Modified: corpus without hyper-links and with emojis as symbols. cED: character based encoder-decoder model. CSMT: character-level statistical machine translation. LM: language models on words or characters. cED average 5: average over five encoder-decoder models. cED ensemble 5: ensemble of five encoder-decoder models (all other cED models, except the average, are extensions of this ensemble). wus: corpus of WhatsApp messages. sms: corpus of sms messages.

Both the CSMT and the ED models outperform the baseline in all settings, except for the averaged ED models. We do not observe a clear and systematic predominance of one particular corpus version (original vs. modified). With respect to the ED models, the integration of each additional word-level language model, the first trained on the WUS corpus, the second on the SMS corpus, results in better performance. Adding a character-level language model trained on the SMS corpus produces an improvement, though not on the ED model already augmented with the word-level WUS language model.

When the most basic configurations are used (average of 5 ED models vs. CSMT with one LM), the CSMT approach achieves better results. This seems to confirm the premise made in Chapter 1: in settings with a small training set, neural methods do not outperform CSMT. However, the performance of the ED approach improves when an ensemble of 5 models is used. Moreover, the capability of ED models to overcome certain limitations of the CSMT approach becomes evident when we exploit the possibility of augmenting them with word-level language models.

The best accuracy overall (87.85% for the original corpus) is reached by the ED model augmented with the SMS character-level LM and both word-level LMs. The best CSMT model, when the original corpus is used, achieves a score of 86.35%. This result shows that NMT does have the potential to outperform CSMT even when little training data is available. A detailed comparative error analysis between CSMT and NMT can be found in Section 7.2.

Table 8 shows the same breakdown that was computed for the baseline and ceiling in Section 6.1, this time with respect to the best models of the two approaches, when the original corpus is used.

		Accuracy		
Word category	**Proportion**	**Baseline**	**CSMT**	**NMT**
Unique	55.05	98.43	96.72	98.46
Ambiguous	32.70	81.22	79.81	80.26
New	12.25	30.27	57.23	60.39
Total	100	84.45	86.35	87.85

Table 8: Proportion of words and breakdown of accuracy score for each category, in percentage, for the best model of CSMT and NMT.

Both models have a performance that is comparable to the baseline on unique and ambiguous words, but show a substantial improvement on new words, i.e. words unseen during training that, according to the baseline criteria, are simply copied

to generate a normalized form. These are the words that would pose a problem in a word-level framework, since they would be regarded as unknown by the model. Instead, in our character-level framework the models can normalize them by applying the transformation patterns learned during training on other words. The improvement on new words, though dampened by their low frequency in the test set (12.25%), is reflected in the overall accuracy. The margin between the accuracy score achieved by the models and the 93.19% threshold that represents the ceiling is primarily due to the wrongly normalized new words. Still, even if the models were able to correctly normalize all new words, the inevitable error due to ambiguity in the source text would prevent accuracy to go beyond the ceiling. We also observe that the best accuracy on ambiguous words is achieved by the baseline.

We shall recall that the French part of the WUS corpus has been automatically normalized by its creators with the MElt sequence labeler (see Section 5.1). Manual evaluation, performed by checking a sample of 1,314 tokens from 160 randomly selected messages, resulted in an accuracy of 95.74%. This is a good performance, if compared to the score of 87.85% achieved by the best NMT model on Swiss German. Since both tasks are performed on CMC texts, it seems reasonable to attribute this substantial score discrepancy to a higher degree of variation in the Swiss German messages, due to regional differences and to the lack of an orthographic standard, both of which do not affect the French language. It is highly remarkable that the performance on the French messages is even better than the 93.19% ceiling computed for Swiss German.

In Section 6.4.2.2, I described dropout regularization and mentioned that I have run experiments with the dropout parameter p (which controls the probability of a neural unit to be retained) set at 0.7 and at the default value 1.0 (no regularization). The results in Table 7 refer to the latter setting. When dropout is used, the accuracy scores for all the NMT models are lower than when it is not used, with the only exception of the average of 5 models (84.44% with regularization vs. 84.15% without).

7.2 Error Analysis

In order to perform error analysis, I have written the Python code `eval_csmt_nmt.py` (L.2). This code outputs three .txt files, with the errors made by the CSMT model, the NMT model and both models respectively. In addition, the code prints a list of predictions that is displayed as follows:

```
5418 de - dann - dann - der
5419 vodafone - vodafone - vodafone - vodafone
5420 woni - wo ich - wo ich - wo ich
5421 mech - mich - mich - mich
5422 glaubs - glaube es - glaube es - glaube es
```

The first column is the line number, which denotes the position of the normalization unit in the test set consisting of 5422 units. The other columns represent the source text (Swiss German), the CSMT normalization, the NMT normalization and the reference text (manual normalization). The actual output uses colors for the third and fourth columns, to emphasize correct (green) and wrong (red) predictions. The list snippet above shows that unit 5418 has been wrongly normalized by both models, which make the same error. All the other units have been correctly normalized by both models.

The code also outputs details about the number of correct and wrong predictions, especially with focus on emojis. Table 9 shows a summary of the errors made by the best models of both approaches on both versions of the WUS corpus.

	Original		**Modified**	
	Units	**Emojis**	**Units**	**Emojis**
Total in test set	5422	166	5422	166
Only CSMT wrong	194	0	191	0
Only NMT wrong	113	0	135	7
Both wrong	546	0	544	0
- Same error	409	0	396	0
- Different errors	137	0	148	0

Table 9: Comparison of errors made by the best CSMT and NMT models.

When the original version of the corpus is used, of a total of 166 emojis, all of them are correctly normalized by both models. We shall recall that in the original corpus emojis are represented by a description than can be a very long character sequence. The absence of errors indicates that the models are able to effectively process them, thus avoiding the need for solutions that could be cumbersome in terms of framework engineering, such as copying emojis at decoding time. However, when the modified corpus is used, where each emoji is a single Unicode character, the ED best model makes 7 wrong predictions. In 4 cases, the test set emoji is not in the training set, so that the model could not learn it in the first place. The fact that CSMT makes no errors in normalizing unknown emojis is easily explained, since Moses simply copies

characters (words in WSMT) that have not been seen during training. In the case of emojis, this turns out to be the right strategy, because they are always the same in the source and target language.

In the next sections I analyze in detail the normalization errors made by the best model of each of the two approaches (CSMT and NMT). The analysis is based on the original corpus, since the best accuracy score overall is achieved with this version. I have classified errors according to what, according to my judgment, is the most plausible factor that might have caused them. There are of course borderline cases that might fit in two or more different categories. When a field in a table is left blank, it means that the normalization for that particular unit is correct. This makes it easier to visualize which system is wrong, and provides a better overview on the general performance on each specific category of errors. The tables show a selection of the errors, whereby the first column indicates the position of the unit in the test set of 5422 total units/lines. The list of errors should not be considered exhaustive.

7.2.1 Weakness of the Model

In this section I analyze errors that, according to my judgment, can be ascribed to weaknesses of the models (Table 10).

- **Construction article + noun.**
  Lines 3940 and 5403 of Table 10 show examples of wrong normalization in constructions where an article is merged with the following noun in the source text. These constructions are rare in the corpus. One possible cause, at least with reference to the feminine article *die*, could be an inconsistency in the manual normalization, where *dselina* is left unchanged instead of being normalized as *die selina*. At least one similar case occurs in the training set (*tcorinne*), whereas other occurrences are correctly normalized, such as *tnummere* → *die nummer*.

- **Hyper-correction.**
  Line 9 shows a possible case of hyper-correction affecting the NMT model. The source word *kriegt* (reference *kriegt*) is normalized as *gekriegt*. The source is here used as present tense form (third person singular) of the infinitive *kriegen* ('to get', 'to receive'), but is erroneously interpreted as a past participle, as indicated by the added prefix *ge-*. The model is seemingly misled by the fact that Swiss German has a tendency to omit the prefix, when this clashes phonetically with the root verb. For example, the past participle of

cho ('come') is *cho*, instead of the expected *gcho*. Line 4971 is another case of hyper-correction, this time affecting the CSMT model, which sees a merged construction verb + pronoun where there is in fact none.

- **Construction word + pronoun.**
  The other examples in Table 10 show cases in which the models have difficulty identifying constructions in which a lexical element is merged with the following pronoun in the source text. These constructions are fairly common in the corpus.

Line	Source	CSMT	NMT	Reference
9	kriegt		gekriegt	kriegt
3234	dasmir es	dass wir es		dass man es
3940	sgfühl	sgefühl	sgefühl	das gefühl
4897	schnidich	schneidich	schnidich	schneide ich
4971	unriifi	unreife ich		unreife
5090	schribim	schreibeim	schreibe ich	schreibe ihm
5180	söuis	sollte es		soll ich es
5279	selli	sollte		soll ich
5403	tzia	tzia	tzia	die zia

Table 10: Examples of errors due to a weakness of the model. Blank fields are correct predictions.

7.2.2 Idiosyncrasies of the Source Text

- **Typos, slang and irregularities in the source text (Table 11).**
  Typos and peculiarities of CMC increase the degree of variation of the source text and can in some cases be misleading for the model. I have already argued that typos are hard to define in a language that lacks an orthographic standard. However, Table 11 shows examples in which a source word is written in a way that goes beyond the variation that we would expect to be caused by dialect differences or arbitrary spelling choices, such as in line 5109 (*ado* instead of the more plausible *aso*) and in line 5136 (*stattgrunde* instead of *stattgfunde*). Similar cases are found in lines 5123, 5329 and 5364. Line 4648 is a clear case of unconventional abbreviation. Since the abbreviation *we* does not exist, the person who did manual normalization had no other option but to normalize the source word *we* with the full form *wochenende* ('weekend'), thus generating a reference that is very dissimilar from the source and harder to normalize for

an automatic system. Line 5060 shows a case of vowel reduplication. At a first analysis, this category of source text features seems to pose challenges to both CSMT and NMT.

- **Foreign words and brand names (Table 12).**
  The model has difficulty normalizing foreign words and brand names, possibly because they contain character sequences that are rare both in the source and in the target language. Line 114 is an interesting case of an English word in which both systems erroneously force normalization. The CSMT model wrongly identifies the common pattern $i \rightarrow ei$, whereas the NMT model changes the sequence *sh* to the more plausible German sequence *sch*. In line 2560, the identification of the common pattern $ch \rightarrow k$ – the same that is responsible for *cho* $\rightarrow$ *kommen* ('come') and *chöne* $\rightarrow$ *können* ('can') – also leads to unnecessary normalization by the NMT system.

Line	Source	CSMT	NMT	Reference
4235	oberk	oberk	oberk	oberkörper
4648	we	wenn	wenn	wochenende
5060	verdaaaaamti	verdammte ich	verda	verdammte
5109	ado	ada	an do	also
5123	verchlimmert	verklimmert	verklimmert	verschlimmert
5136	stattgrunde	stattgrunden	stattgrunden	stattgefunden
5213	zzzz	tztztz	zuzz	zzzz
5329	hany	haben	hany	handy
5364	psychatrischi	psychatrische	psychatrische	psychiatrische

Table 11: Examples of errors related to typos and irregularities of the source text.

Line	Source	CSMT	NMT	Reference
114	nightwish	nightweist	nightwisch	nightwish
2560	chorizo		korizo	chorizo
5282	feedback	feichtback		feedback
5288	cream	kream		cream
5386	breakers	breakeres		breakers

Table 12: Examples of errors related to foreign words. Blank fields are correct predictions.

7.2.3 Choices in Manual Normalization

Some errors are possibly the consequence of normalization choices that may be debatable (Table 13). For example, it should be considered if abbreviations in the source language should be left unchanged or normalized to the corresponding full form, in those cases where the abbreviated form is conventional in Standard German. The fact that *bzw* is normalized as *beziehungsweise* and *wg* as *wohngemeinschaft* may be misleading for the model.

Line	Source	CSMT	NMT	Reference
2459	öv	ev	öffentlichen verkehr	öffentlicher verkehr
3945	hdl	hdl	halt	habe dich lieb
3957	bzw	bzw.		beziehungsweise
4515	omg	umg		oh mein gott
5389	wg	wg		wohngemeinschaft

Table 13: Examples of errors related to choices in manual normalization. Blank fields are correct predictions.

Moreover, some words are not consistently normalized: for example, the word *druf/druff* is normalized 19 times as *darauf* and 15 times as *drauf*; the word *gärn/gern* 37 times as *gerne* and 17 times as *gern*; *omg* 11 times as *oh mein Gott* (German) and 5 times as *oh my God* (English); *bf* 3 times as *bf* and 2 times as *best friend*; the word *ok/oki*, 14 times as *ok* and 133 times as *okay*. It is true that in German some of the normalized pairs above are not always interchangeable, as the two forms may express different nuances and the choice of one form or the other by a speaker might be deliberate. However, based on my observations, an analysis of the normalization choices, carried out by a native speaker, and a review of the guidelines would be advisable to improve consistency and could lead to an improvement in the performance.

This appears to be the task in which the additional word-level language models integrated into the NMT framework produce the most noticeable improvements. Most cases in which the reference is very dissimilar from the source are correctly normalized by the NMT system. In line 2459, the error is an inflection subtlety and is of very small entity.

7.2.4 Known Ambiguities

Some errors are due to ambiguities which are inherent in the mapping between Swiss German and its normalized form. Swiss German has a tendency to use simplified forms and to drop phonetic and orthographic elements, which causes two or more standardized forms to be conflated in one single source form. Table 14 shows a selection of source words that in the corpus have been manually normalized in different ways due to their ambiguous nature.

Source	Normalization forms			
abe	abend (11)	hinab (7)		
bi	bei (184)	bin (166)		
cho	gekommen (29)	kommen (20)		
dass	dass (89)	das (55)		
de	der (227)	dann (183)	den (171)	denn (42)
di	dich (83)	die (68)		
e	eine (131)	ein (36)	einen (25)	
es	ein (168)	es (137)		
i	ich (674)	in (69)		
mir	mir (111)	wir (96)		
s	das (193)	es (77)		
si	sie (134)	sind (81)	sein (58)	

Table 14: Ambiguities in the source text. For each source word, the most frequent normalization forms are shown in order of frequency.

This type of error is the primary cause of the ceiling of 93.19%, representing the maximum accuracy achievable. For example, there is no way for the system to know whether the source word *bi* is used to signify *bei* ('by') or *bin* ('am'). Therefore, when the system outputs a prediction, there is a high likelihood that it makes an error.

7.2.5 Correct Normalization

In this section I present interesting cases of correct predictions (Table 15). For example, in many cases both models are able to resolve merged constructions such as *woni* → *wo ich* and *isches* → *ist es*. In line 3443, both systems have identified and correctly normalized a feature of some Swiss German dialects, such as the Bernese dialect, according to which *l* is written *u* to reflect a pronunciation that differs from other dialects. The CSMT model makes however an error in the inflection. In

line 3508, both models correctly normalized a source word that, according to the arguments that I put forward in Section 7.2.2, represents an instance of misspelled word. The wrongly written *sh* sequence has been changed to *sch*. Line 3516 shows a case which is similar to line 9 in Table 10. In this case, the source has been correctly identified as a past participle. In line 5073, both systems can correctly normalize a sequence reproducing a laugh, even though the normalized form makes what appears to be an arbitrary adaptation of the source.

Line	Source	CSMT	NMT	Reference
3443	autä	alten		alter
3508	shriibe			schreiben
3516	klappt			geklappt
5073	ahaha			haha

Table 15: Examples of correct normalization. Blank fields are correct predictions.

7.2.6 Improvements Due to the Word-Level LM

One last analysis, and a very important one, is aimed at assessing the impact of the integrated WUS word-level language model on the basic NMT model consisting of the ensemble of 5 models without any augmentation (Table 16). This integration, applied by Ruzsics and Samardžić [2017] to morphological segmentation, represents the primary innovation proposed in this thesis in the attempt to build NMT models that can achieve and improve state-of-the-art performance in automatic text normalization. Table 7 shows that this device produces a 0.79% improvement in the accuracy score.

Line	Source	Ensemble	Integrated LM	Reference
371	wiedsr	wiedser		wieder
891	hanise	hanise		habe ich sie
1949	hb	hauptb		hauptbahnhof
3956	normaau	norma		normal
4071	o	auc		auch
4088	eig	eigentlic		eigentlich
5042	kg	kilogr		kilogramm

Table 16: Examples of NMT errors corrected by the integrated word-level language model. Blank fields are correct predictions.

Line 371 is a case in which the integrated LM can correct a typo, since the source word *wieder* has been misspelled as *wiedsr*. Moreover, it correctly normalizes a source word that is normalized as three target words (line 891), a source word with a peculiarity of the Bernese dialect (line 3596, with *u* instead of *l*), and source words whose normalized form is very dissimilar (all the other examples).

8 Conclusion

In this thesis I have described methods for training models aimed at automatic text normalization of WhatsApp messages written in Swiss German. Text normalization is performed with the goal of converting non-canonical text to a standardized form that can be better processed by NLP tools and applications. The task of text normalization shares many properties and principles with machine translation, and has been carried out in this work by applying two different approaches: statistical machine translation (SMT) and neural machine translation (NMT) with sequence-to-sequence models. The two models have been trained on a parallel corpus of manually normalized WhatsApp messages, in which most normalization units consist of one-to-one word alignments, with some cases of one-to-many and rare cases of many-to-one and many-to-many alignments. I have described the factors that contribute to the high degree of variation that characterizes Swiss German WhatsApp messages. Firstly, the lack of an orthographic standard and the regional variation due to the differences between the dialects that fall under the definition *Swiss German*. Secondly, the fact that we are dealing with computer-mediated communication (CMC), a domain in which texts are characterized by a number of irregularities that increase the variation in the text.

Although in machine translation tasks the neural approach has nowadays replaced the statistical one as the new state-of-the-art method, in text normalization it still lags behind. One of the main reasons for this unevenness in two tasks that are apparently so similar lies in the limited size of the corpora that are available for training normalization models. Machine translation can rely on vast amounts of data and resources coming from translation memories and freely available parallel corpora. In our globalized world, texts are manually translated by humans every day for cultural, commercial and diplomatic purposes. One example is that of translations of laws and proceedings in institutions related to the European Union, which are done by professional translators and become available to the NLP community. By contrast, data sets for training normalization systems are usually created *ad hoc* by experts and researchers and are therefore inevitably small. Even if the amount of texts written in Swiss German increases, they still have to be manually normal-

ized in order to create parallel corpora, an expensive and time-consuming step that represents a major obstacle. Since neural frameworks need large training sets in order to express their full potential, their performance in text normalization tasks is still not satisfactory and SMT methods still represent the state of the art in this particular task.

The central research question posed as motivation for this work asked whether it is possible to train neural models able to achieve state-of-the-art performance in text normalization. Given that, at least for the moment, the possibility to collect large corpora is ruled out, one fundamental contribution of the present work in this endeavor is to exploit the flexibility that enables neural architectures to be augmented with additional components that can overcome certain limitations of statistical models. In both the SMT and NMT approach, the basic character-level frameworks can be augmented by training language models on additional corpora. The expectation I had for the task described in my thesis is that neural character-level methods can benefit greatly from the integration of language models trained on a different level of granularity (i.e. words), a method that has already been applied successfully to other tasks, such as morphological segmentation ([Ruzsics and Samardžić, 2017]).

The results of my experiments indicate that both approaches outperform the strong baseline of 84.45% that has been established for this task. The best neural model achieves an accuracy score of 87.85%. Though this result is only slightly better than the 86.35% achieved by the SMT system, it represents a very promising indication, since it shows that the neural model can outperform the SMT model thanks to the integration of the additional word-level LMs. This method has a positive impact on the normalization of units in which the target sequence consists of more than one word or is very dissimilar from the source, two patterns that are hardly captured by a character-level model. The results suggest that the proposed adaptation can be successfully extended to integrating multiple potential components into a single neural framework. This possibility can be exploited for further improvements of text normalization methods.

8.1 Future Work

As mentioned above, the future of neural text normalization heavily depends on the availability of training material. At least as far as CMC is concerned, it seems reasonable to expect that more and more texts will be written in Swiss German. This will solve the problem of scarce training material only partially, since the source

texts must be manually normalized and this process, as I have argued, is currently conducted on a very limited scale.

Based on my observations, an improvement in the manual normalization (correction of inconsistencies, review of the normalization guidelines if necessary) of the corpus used as data set for this work could contribute to a better performance by automatic normalization systems.

With reference to the pipeline of automatic normalization, further developments may rely on the use of the context to resolve language ambiguities. This entails using normalization units that consist in text segments instead of isolated words. However, it is important to bear in mind that there is always a trade-off between sequence length and computational efficiency. While longer sequences are able to capture a wider context, thus resolving ambiguity and resulting in a better performance, they pose a great challenge to systems for automatic normalization: in CSMT, they make character alignment particularly cumbersome and increase the size of phrase tables; in NMT, they increase the dimensions of tensors and thus the number of necessary operations.

The problem of ambiguity could be also addressed by making use of linguistic information, such as POS tags. This method has been already applied to SMT in a framework known as factored translation. Hollenstein and Aepli [2014] have trained a POS tagger for Swiss German that could be used to tag the source text. The additional information could then be exploited in case of ambiguity, in that the right normalization is chosen according to the POS of the source word. For example, the source *bi*, which in the WUS corpus is normalized, with almost equal occurrences, as both the preposition *bei* and the verb form *bin*, could be correctly normalized if the POS tag were known.

Both CSMT and NMT are characterized by a number of parameters that must be carefully set and tuned. There are no clear-cut criteria that prescribe the best parameters to choose, since the optimal setting depends on many factors, such as the type of task and the features of the data set. Finding the parameters that yield the best performance is an empirical process of trial and error that might be very time-consuming, since the same process might have to be repeated several times with a different set of parameters. For example, in my experiments, following related work done in other research projects, I have used 7-gram character language models. Future experiments may be tried with higher order models, that might be able to capture a larger context. One further solution that could be explored is to use a different neural architecture, for example one based on the LSTM instead of the GRU that I have employed.

References

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

A. Baron and P. Rayson. VARD 2: A tool for dealing with spelling variation in historical corpora. In *Proceedings of the Postgraduate Conference in Corpus Linguistics*, Aston University, 2008.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *JOURNAL OF MACHINE LEARNING RESEARCH*, 3: 1137–1155, 2003.

M. Bollmann. (Semi-)automatic normalization of historical texts using distance measures and the Norma tool. In *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities (ACRH-2)*, pages 3–14, Lisbon, Portugal, 2012.

P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=972470.972474`.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D14-1179`.

O. De Clercq, B. Desmet, S. Schulz, E. Lefever, and V. Hoste. Normalization of Dutch user-generated content. In *Proceedings of RANLP 2013*, pages 179–188, Hissar, Bulgaria, 2013.

P. Denis and B. Sagot. Coupling an annotated corpus and a lexicon for state-of-the-art POS tagging. *Language Resources and Evaluation*, 46(4):

721–736, 2012. doi: 10.1007/s10579-012-9193-0. URL https://hal.inria.fr/inria-00614819.

E. Dieth. *Schwyzertütschi Dialäktschrift. Leitfaden einer einheitlichen Schreibweise für alle Dialekte.* Orell Füssli, Zurich, 1938.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021068.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017. URL http://arxiv.org/abs/1705.03122.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, pages III–1319–III–1327, 2013.

C. Gulcehre, O. Firat, K. Xu, K. Cho, and Y. Bengio. On integrating a language model into neural machine translation. *Computer Speech and Language*, 5 2016. doi: 10.1016/j.csl.2017.01.014.

L. Hansen and P. Salamon. Neural network ensembles. 12:993 – 1001, 11 1990.

K. Heafield. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom, July 2011. URL https://kheafield.com/papers/avenue/kenlm.pdf.

S. Hochreiter and J. Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

N. Hollenstein and N. Aepli. Compilation of a Swiss German dialect corpus and its application to PoS tagging. In *Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects (VarDial)*, COLING 2014, Dublin, Ireland, 2014. Association for Computational Linguistics.

P.-E. Honnet, A. Popescu-Belis, C. Musat, and M. Baeriswyl. Machine Translation of Low-Resource Spoken Dialects: Strategies for Normalizing Swiss German. *ArXiv e-prints*, 1710.11035, Oct. 2017. URL https://arxiv.org/abs/1710.11035.

N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language*

Processing, pages 1700–1709, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D13-1176.

K. Kann, R. Cotterell, and H. Schütze. Neural morphological analysis: Encoding-decoding canonical segments. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 961–967, Austin, Texas, November 2016. Association for Computational Linguistics. URL https://aclweb.org/anthology/D16-1097.

P. Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of the MT Summit 2005*, pages 79–86, 2005. URL http://www.iccs.inf.ed.ac.uk/~pkoehn/publications/europarl-mtsummit05.pdf.

P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (Volume 1)*, pages 48–54, Edmonton, Canada, 2003.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the ACL 2007 demonstration session*, pages 177–180, Prague, Czech Republic, 2007.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, feb 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

N. Ljubešić, T. Erjavec, and D. Fišer. Standardizing tweets with character-level machine translation. In *Proceedings of CICLing 2014*, Lecture notes in computer science, pages 164–175, Kathmandu, Nepal, 2014. Springer.

D. Matthews. Machine transliteration of proper names. Master's thesis, University of Edimburgh, 2007.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia,

Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL http://www.aclweb.org/anthology/P02-1040.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, pages III–1310–III–1318, 2013.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

E. Pettersson, B. B. Megyesi, and J. Nivre. Normalisation of historical text using context-sensitive weighted Levenshtein distance and compound splitting. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (Nodalida 2013)*, pages 163–79, Oslo, Norway, 2013.

E. Pettersson, B. B. Megyesi, and J. Nivre. A multilingual evaluation of three spelling normalisation methods for historical text. In *Proceedings of the 8th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH)*, pages 32–41, Gothenburg, Sweden, 2014.

F. Rash. *The German language in Switzerland: multilingualism, diglossia and variation*. Lang, Bern, 1998.

B. Ruef and S. Ueberwasser. The taming of a dialect: Interlinear glossing of Swiss German text messages. In M. Zampieri and S. Diwersy, editors, *Non-standard Data Sources in Corpus-based Research*, pages 61–68, Aachen, 2013.

T. Ruzsics and T. Samardžić. Neural sequence-to-sequence learning of internal word structure. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 184–194, Vancouver, Canada, 2017. Association for Computational Linguistics.

T. Samardžić, Y. Scherrer, and E. Glaser. Normalising orthographic and dialectal variants for the automatic processing of Swiss German. In *Proceedings of The 4th Biennial Workshop on Less-Resourced Languages*. ELRA, 2015.

T. Samardžić, Y. Scherrer, and E. Glaser. ArchiMob - a corpus of spoken Swiss German. In N. C. C. Chair), K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis, editors, *Proceedings of the Tenth International Conference on Language*

Resources and Evaluation (LREC 2016), Paris, France, may 2016. European Language Resources Association (ELRA). ISBN 978-2-9517408-9-1.

F. Sánchez-Martínez, I. Martínez-Sempere, X. Ivars-Ribes, and R. C. Carrasco. An open diachronic corpus of historical Spanish: annotation criteria and automatic modernisation of spelling. Research report, Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, Alicante, 2013.

Y. Scherrer and T. Erjavec. Modernising historical Slovene words. *Natural Language Engineering*, 22(6):881–905, 2016.

Y. Scherrer and N. Ljubešić. Automatic normalisation of the Swiss German ArchiMob corpus using character-level machine translation. In *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016)*, pages 248–255, 2016.

H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK, 1994.

M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, Nov. 1997. ISSN 1053-587X. doi: 10.1109/78.650093. URL `http://dx.doi.org/10.1109/78.650093`.

H. Schwenk. Continuous space translation models for phrase-based statistical machine translation. In *COLING*, 2012.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

F. Stahlberg, E. Hasler, D. Saunders, and B. Byrne. SGNMT - A flexible NMT decoding platform for quick prototyping of new models and search strategies. *CoRR*, abs/1707.06885, 2017. URL `http://arxiv.org/abs/1707.06885`.

E. Stark, S. Ueberwasser, and B. Ruef. Swiss SMS Corpus. University of Zurich. `https://sms.linguistik.uzh.ch`, 2009–2015.

E. Stark, S. Ueberwasser, and A. Göhring. Corpus "What's up, Switzerland?". University of Zurich. `www.whatsup-switzerland.ch`, 2014.

A. Stolcke. Srilm — an extensible language modeling toolkit. 2, 07 2004.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

J. Tiedemann. Character-based PSMT for closely related languages. In *Proceedings of 13th Annual Conference of the European Association for Machine Translation*, pages 12–19, Barcelona, Spain, 2009.

J. Tiedemann. Character-based pivot translation for under-resourced languages and domains. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 141–151, Avignon, France, April 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/E12-1015`.

J. Tiedemann and P. Nabende. Translating transliterations. *International Journal of Computing and ICT Research*, 3(1):33–41, 2009. URL `http://www.ijcir.org/Special-Issuevolume3-numbe1/article4.pdf`.

S. Ueberwasser. The Swiss SMS Corpus. Documentation, facts and figures. `https://sms.linguistik.uzh.ch`, 2015.

S. Ueberwasser and E. Stark. What's up, Switzerland? A corpus-based research project in a multilingual country. *Linguistik Online*, 84(5), 2017. ISSN 1615-3014. URL `https://bop.unibe.ch/linguistik-online/article/view/3849`.

B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowskia, and Y. Bengio. Blocks and fuel: Frameworks for deep learning. 2015.

D. Vilar, J.-T. Peter, and H. Ney. Can we translate letters? In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 33–39, Prague, Czech Republic, 2007.

I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0120884070.

M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL `http://arxiv.org/abs/1212.5701`.

Lebenslauf

Persönliche Angaben

Massimo Lusetti
Hädrichstrasse 9
8047 Zürich
massimo.lusetti@uzh.ch

Schulbildung

2015-2018	Master-Studium Multilinguale Textanalyse an der Universität Zürich
2011-2015	Bachelor-Studium Fremdsprachen und Literaturwissenschaft an der Universität Bologna

Berufliche und nebenberufliche Tätigkeiten

Seit Februar 2018	Hilfsassistent Romanisches Seminar an der Universität Zürich
Oktober-Dezember 2017	Tutorat Einführung in die Multilinguale Textanalyse an der Universität Zürich

A Foundations of Machine Learning

A.1 Linear Algebra Operations with Tensors

Tensors are mathematical objects that can be described in many different ways, depending on whether we look at them from the point of view of algebra, geometry or physics. For the purposes of this work, I believe a simple and intuitive explanation will suffice. Tensors of rank 0 are called scalars, such as real numbers. Tensors of rank 1 are called vectors. We can think of a vector as an ordered set of n real number coordinates, where each coordinate refers to a dimension in a n-dimensional space. A vector $\boldsymbol{v}$ can be represented as

$$\boldsymbol{v} = [v_1, v_2, \ldots, v_n] \tag{A.1}$$

If $n = 4$, we could have for example the following vector:

$$\boldsymbol{v} = [3, -2, 7, 6] \tag{A.2}$$

Several mathematical operations can be applied to vectors. We define the dot product of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ as

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i \tag{A.3}$$

which is the same as writing

$$\boldsymbol{u} \cdot \boldsymbol{v} = [u_1 v_1 + u_2 v_2 + \cdots + u_n v_n] \tag{A.4}$$

The result of the dot product between two vectors is a real number. The element-wise (also called Hadamard) product of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is computed by

$$\boldsymbol{u} \odot \boldsymbol{v} = [u_1 v_1, u_2 v_2, \ldots, u_n v_n] \tag{A.5}$$

The result of the Hadamard product of two n-dimensional vectors is itself a n-dimensional vector.

A tensor of rank 2 is called a matrix. We can think of a matrix as an array of m rows and n columns, in which each row is a vector. The product of a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$ is the dot product of $\boldsymbol{v}$ with each of the rows of $\boldsymbol{A}$. The number of columns in $\boldsymbol{A}$ has to be equal to the number of components in $\boldsymbol{v}$, so if we have a $m \times n$ matrix and a $n \times 1$ vector, the result of the product will be a $m \times 1$ vector.

A.2 Data Instances as Vectors

In many machine learning frameworks, including neural networks, instances of a data set are viewed as vectors in a multi-dimensional space, in which each dimension corresponds to a feature of the instances. Viewing instances of a data set as vectors requires a drastic mindset change in the way we see and represent the world around us. In order to introduce some important concepts of machine learning, I will use an example that, though not closely related to the task of text normalization described in this thesis, can help make such necessary change of mindset.

In a text classification task, text instances that I have collected to form a data set are defined by features, and each feature adds a dimension to the vector space. For example, we could use occurrences of POS tags to classify text fragments belonging to two different domains: *news* and *fiction*. Each instance (i.e., each text fragment) is defined by three features, that have been singled out in a process known as feature engineering. In this example, the features are the number of pronouns, simple past verbs and personal pronouns normalized over 1000 words. In addition, each instance belongs to one of the two classes, *fiction* or *news*, which also represents the label of that particular instance. Table 17 shows a sample of five instances extracted from the whole data set of 77 fragments.[1]

The text fragment with index 1 can be represented as a vector in a three-dimensional space, where the x coordinate has value 224.206349, the y coordinate has value 53.571429, and the z coordinate also has value 53.571429. The three coordinates define the point that the instance occupies in the vector space. If a second instance,

[1]The data, results and graphs presented in this appendix are taken from a work that I personally conducted on text classification.

	Features			
Index	**1: noun**	**2: simple past**	**3: personal pronoun**	**Class**
1	224.206349	53.571429	53.571429	Fiction
2	337.028825	13.303769	17.738359	News
3	180.198020	63.366337	71.287129	Fiction
4	296.066253	18.633540	24.844720	News
5	261.744966	11.185682	26.845638	News

Table 17: Text fragments as vectors. Each row is a 3-dimensional vector representing a text fragment. Each feature is a dimension in the vector space.

according to its three coordinates, happens to end up close to the first instance in the vector space, the two instances are similar. There are various methods to compute the distance, and therefore the degree of similarity, between data points in space. One method consists in taking the dot product of two vectors. Instances in a data set could be defined by hundreds or thousands of features, which require a multi-dimensional space that is difficult for humans to visualize.

The conversion of a data instance into a vector can produce a sparse or dense representation. As an example, we might have a text and want to convert each of its words into a vector. First we assign to each word type an integer index that identifies that specific word in the text vocabulary (the list of all word types appearing in the text). Next, one possible vector representation is the one-hot encoding, in which each vector has as many dimensions as the number of words in the text vocabulary, with 1 in the position corresponding to the word index and 0 in all other positions. If the text has a vocabulary of only ten words, the one-hot representation of two of them would be:

```
w1 = [0 0 0 0 0 0 0 1 0 0]
w2 = [0 1 0 0 0 0 0 0 0 0]
```

The two vectors *w1* and *w2* represent the words of the vocabulary with indexes 8 and 2, respectively. This is a sparse representation that can become very long, depending on the size of the vocabulary, and might be problematic for linear algebra operations, as the dot product of the two vectors is 0. For this reason, the one-hot encoding is not suitable for expressing similarity between words. A valid alternative is a dense representation, that requires fewer dimensions and can be obtained, for example, by exploiting the distributional similarity of words in a text. According

to the principle of distributional similarity, words that appear in the same context are similar. Each word is expressed as a real-valued vector, whose entries can be adjusted in a machine learning process, so that two similar words are represented by two vectors with a high dot product. The two words of our example might appear as follows in a dense representation:

```
w1 = [0.331 0.178 -0.561]
w2 = [-0.387 0.411 -0.102]
```

The notion of dense vectors is important in neural network models, which represent one of the methods for text normalization that I describe in this thesis, where input characters are encoded into dense vectors, whose components are then updated during training in order to produce the most accurate output.

Converting a symbol (a word or a character) into a dense vector representation is referred to as *embedding*. Word2Vec [Mikolov et al., 2013], trained on data from Google News and GloVe [Pennington et al., 2014], trained on data from Wikipedia and Common Crawl, offer a set of pre-trained word vectors that can be downloaded online and used as input for various machine learning tasks.

A.3 Supervised Learning and Logistic Regression

Supervised learning is an application of machine learning in which the data from which the system learns is labeled. This means that each instance in the dataset has been manually annotated and classified, so that the learning algorithm can learn from the correct labels during training. The portion of the dataset from which the system learns represents the training set, and the portion that contains the new data on which the system has to make predictions is the test set. In addition, a portion of the dataset, the development set, is used by the system to adjust certain parameters in order to find those that give the best results in terms of correct predictions.

In the text classification task described in Appendix A.2, each text fragment has been manually labeled with its domain (*fiction* or *news*), and the model learns which features are characteristic of the class *news* and of the class *fiction* in order to classify new text fragments. Figure 6 shows an example of classification for such task using logistic regression, an algorithm that constitutes an important building block for the implementation of neural networks. For the sake of a two-dimensional representation, the vector space is defined by only two features (personal pronouns and simple past). The values on the x and y axes are not frequency values, since

the data has been standardized.[2] The black dots represent *news* texts and the white dots *fiction* texts in the training set. The model learns the weights w associated with each of the two features as well as an intercept value b and uses them in a linear function to draw a boundary line that separates the two classes, whereby each class occupies a region of the vector space with a different color. The weights w and the intercept value b are defined as the parameters of the algorithm. At prediction time, if a test set instance, according to its features, lies in the bright area, it will be assigned the label *fiction*; if it lies in the dark area, it will receive the label *news*.

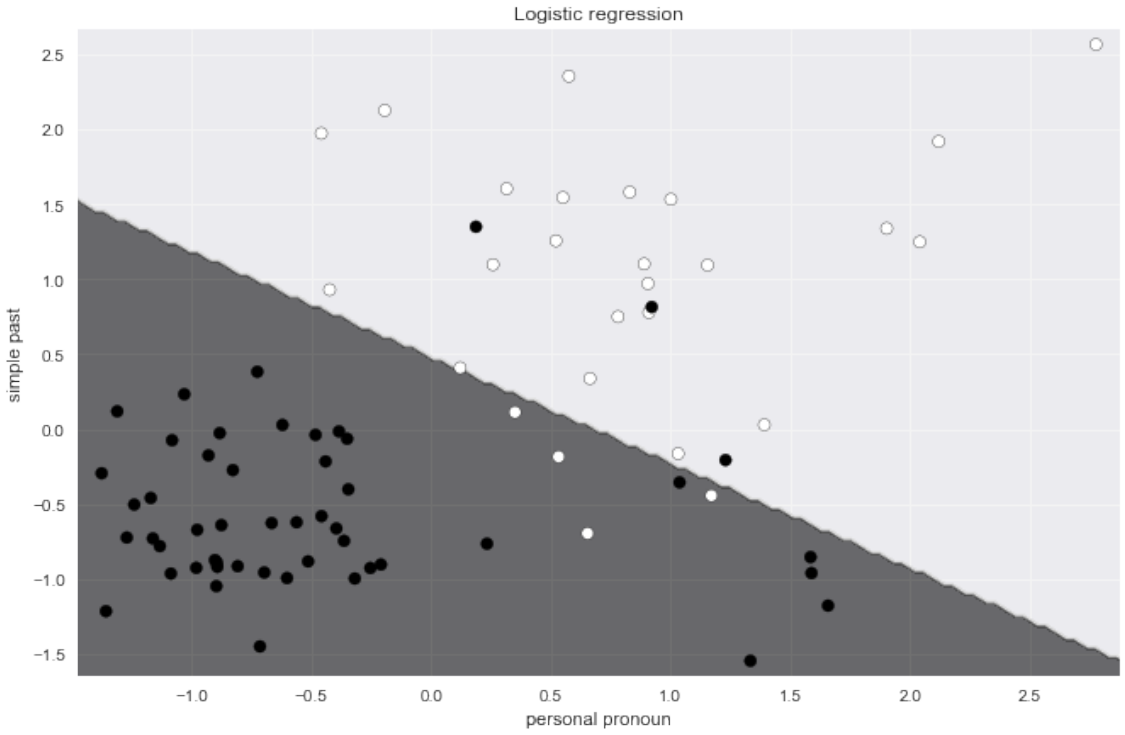

Figure 6: Example of logistic regression, in this case applied to the whole dataset of text fragments. The white dots are instances of the *fiction* register, the black dots of the *news* register. The model learns a linear decision boundary between the two classes. New instances are then placed in the vector space according to their features, and receive a label according to the area in which they fall (light gray = fiction, dark gray = news).

[2]Some machine learning algorithms are based on the distance between instances in the vector space. Even when all features are expressed in the same unit of measurement, we must take into account that, for example, nouns are overall more frequent than personal pronouns, which could have an impact on the distribution of the instances in the vector space. For this reason, I performed standardization using the z-score, which in turn relies on the notion of standard deviation. The standard deviation is a measure of how the values of a data set are distributed around the mean. A low value indicates that the data is concentrated around the mean, whereas a high value indicates that the data is scattered over a wide range of values. The z-score measures the distance between a data point and the mean, using the standard deviation as unit of measurement, and is a common method to standardize data expressed in different scales or characterized by different orders of magnitude.

Two important concepts in supervised learning are underfitting and overfitting. Underfitting occurs when the model has an inherent weakness that causes difficulty in interpreting and classifying training set instances, resulting in a poor overall performance. Overfitting occurs when the model fits the training data extremely precisely and learns meticulously how to account for all its idiosyncrasies, but is unable to generalize to unseen data, so that the performance is good on training data, but poor on test data.

If we convert the labels *news* and *fiction* to the integers 0 and 1, given an input feature vector $\boldsymbol{x}$, logistic regression outputs a prediction $\hat{y}$, which is an estimate of the real label y:

$$\hat{y} = p(y = 1|\boldsymbol{x}) \tag{A.6}$$

Once we have the probability p of class 1, we can easily compute the probability of class 0, because these two values must add up to 1. In order to compute $\hat{y}$, the model combines the parameters $\boldsymbol{w}$ (the vector of the weights of each feature) and b (the intercept) in a linear function as follows:

$$z = \boldsymbol{w}^T\boldsymbol{x} + b \tag{A.7}$$

The symbol T indicates a transpose operation. The result of the linear function is then fed into a non-linear function (the sigmoid function σ, shown in Figure 7) to produce probability values between 0 and 1:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{A.8}$$

After outputting a prediction $\hat{y}$, a function $\mathcal{L}$ measures how good the prediction is with reference to the true label y, with respect to a single training example. Equation A.9 shows the cross-entropy loss function, which is commonly used in logistic regression for this purpose:

$$\mathcal{L}(\hat{y}, y) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y})) \tag{A.9}$$

If i represents a single training example and m is their total number, it is then possible to compute the cost function J of the parameters w and b, referred to the entire training set:

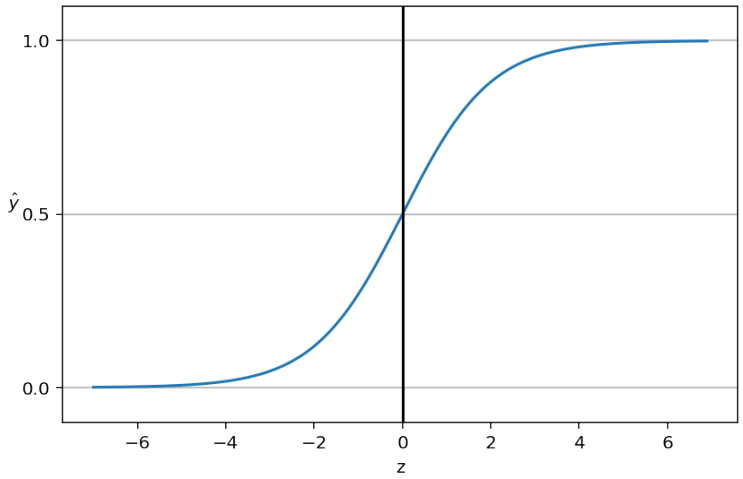

Figure 7: The sigmoid function.

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{A.10}$$

The goal is then to find the parameters w and b that minimize the cost function J. This process, known as optimization, is achieved by employing the gradient descent algorithm.

Figure 8 can help understand how gradient descent works. For visualization purposes, the graph only shows one weight on the x axis, as if the training examples only had one feature, and the intercept b is omitted. The y axis represents the cost function J. The algorithm starts by assigning a random value to w, which corresponds to point 1 on the curve and to a certain cost J. Since the goal of gradient descent is to minimize J, after one iteration over all the training examples (i.e., the cost function has been computed by adding the loss function of each one of them) the algorithm will take one step in the direction of the steepest descent. This operation is repeated until convergence, i.e., until reaching the point that corresponds to the minimum value of J. Equation A.11 shows how the parameters θ (w and b) are updated at each iteration in this search for the minimum value of J:

$$\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \tag{A.11}$$

The symbol := indicates that θ is updated with the right part of the equation, and $\frac{\partial J(\theta)}{\partial \theta}$ is the partial derivative of $J(\theta)$ with respect to θ. The term α, the learning rate,

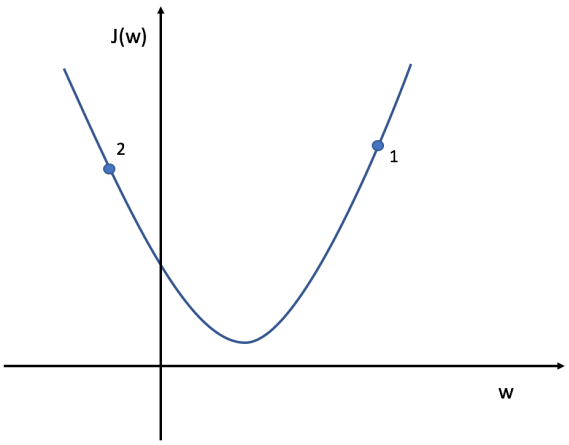

Figure 8: Gradient descent for a simplified problem in which data instances have only one feature, and therefore one parameter w on the horizontal axis. The cost function is on the vertical axis. The bias, which represents an additional parameter, is here omitted for visualization purposes. If w has a value that corresponds to point 1 on the curve, the derivative at that point is positive, whereas at point 2 the derivative is negative.

indicates the length of each step toward the minimum, and must be chosen carefully: if too large, there is risk of overshooting the minimum, if too small, convergence may take a very long time. Looking at Figure 8, it becomes clear how gradient descent achieves the goal of minimizing the cost. The derivative is the slope of a curve at a point, and at point 1 the derivative of $J(w)$ with respect to w is positive. Therefore, in Equation A.11, we subtract a positive quantity from w and the new value of w will be lower, so that the step taken by gradient descent will be toward the left. If w had been randomly initialized at point 2, the derivative would be negative and the updated value of w would be greater, thus pointing to the right. The same operation is done for the intercept b and for all the weights associated with other features. The process of computing the loss function $\mathcal{L}$ is called forward propagation, whereas back-propagation denotes the computation of the derivatives that are then used to update the parameters.

B Scripts

This appendix contains a list of scripts that I used for this thesis.

- The scripts used in the Moses pipeline can be found at `https://github.com/moses-smt/mosesdecoder`:

  M.1 `lowercase.perl` lowercases the corpus.

  M.2 `train-model.perl` trains the translation model.

  M.3 `mert-moses.pl` is used for tuning.

  M.4 `moses` is used for decoding.

- The scripts related to the KenLM language model can be found at `https://github.com/kpu/kenlm`:

  K.1 `lmplz` trains the language model.

  K.2 `build_binary` binarizes the language model.

- Some authorless scripts used for pre-processing, post-processing and evaluation can be found at `https://drive.google.com/open?id=1BWD9KmUSUFwXpvViIG2Cidd01YE7WvHT`:

  U.1 `addUnderscores.py`

  U.2 `removeUnderscores.py`

  U.3 `disableDistortion.py`

  U.4 `accuracy.py`

- The scripts related to the NMT framework are mainly parts of the Theano, Blocks and SGNMT libraries described in Section 5.2.3, with some adjustments and extensions kindly made available by Tatyana Ruzsics of the University of Zürich, to whom I am sincerely grateful. They can be found at `https://github.com/tatyana-ruzsics/uzh-corpuslab-morphological-segmentation`

- I have written the following scripts, that can be found at `https://drive.google.com/open?id=1BWD9KmUSUFwXpvViIG2Cidd01YE7WvHT`:

  L.1 `split_corpus.py`
  Used in the CSMT framework to divide the corpus into source and target side.

  L.2 `eval_csmt_nmt.py`
  Compares the output of different models. Used for error analysis.

  L.3 `baseline_cat.py`
  Awkwardly named script, originally written to compute the baseline, then adapted to break down accuracy and word proportion into the three word categories *unique*, *ambiguous* and *new*.

  L.4 `run_one.sh`
  Launches Moses only once on the corpus split used throughout the whole project. Originally written to launch multiple runs of Moses on different shuffles of the corpus.

  L.5 `run_moses_one_split.sh`
  Handles the Moses pipeline.

  L.6 `run-all-for-all-hyper.sh`
  Launches the NMT process. Originally meant to launch NMT multiple times as a loop over different hidden and embedding dimension settings, then set to fixed values once the optimal setting was established.

  L.7 `Main-Norm-lm-ch-2-hyper.sh`
  Handles the NMT pipeline. I thank Tatyana Ruzsics for her valuable help in writing this code.

Universität Zürich[UZH]

Philosophische Fakultät
Studiendekanat

Universität Zürich
Philosophische Fakultät
Studiendekanat
Rämistrasse 69
CH-8001 Zürich
www.phil.uzh.ch

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Masterarbeit von mir selbst ohne unerlaubte Beihilfe verfasst worden ist und ich die Grundsätze wissenschaftlicher Redlichkeit einhalte (vgl. dazu: http://www.uzh.ch/de/studies/teaching/plagiate.html).

ZÜRICH, 15.06.2018 Monsieur Durett

Ort und Datum Unterschrift